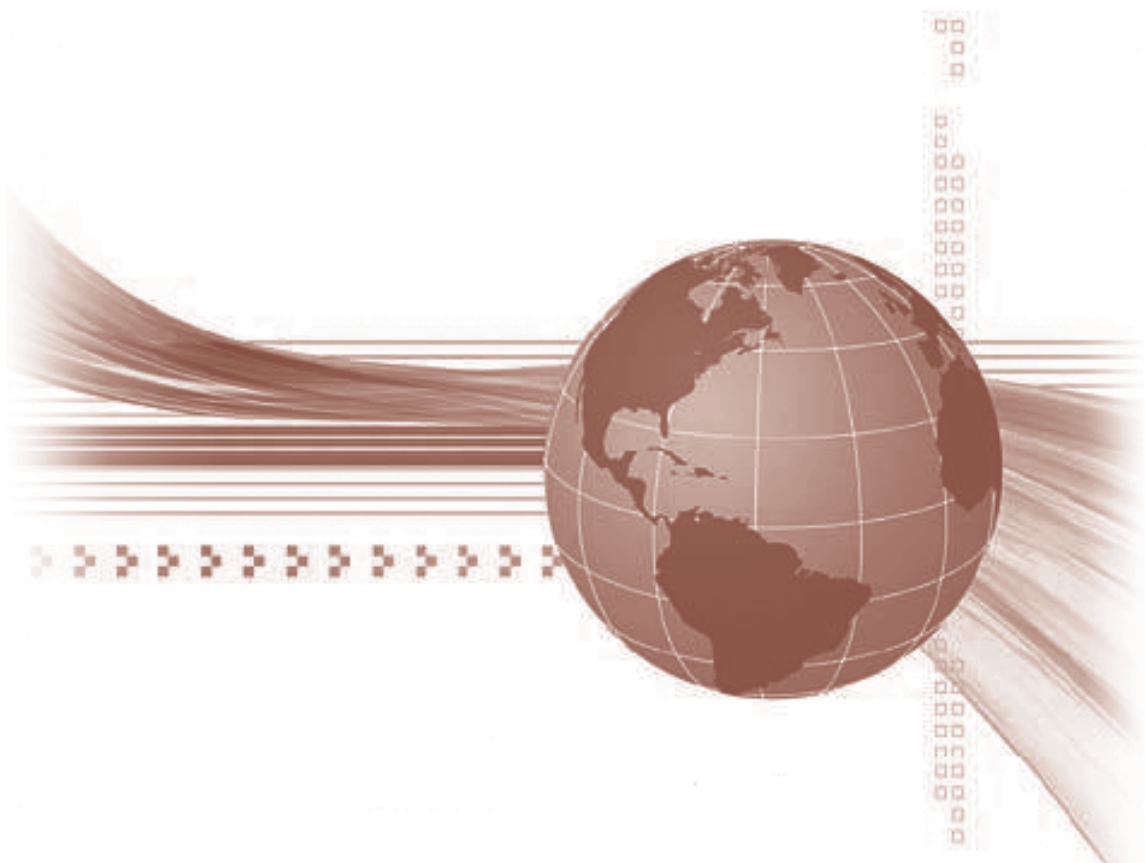
STUDIA UNIVERSITATIS
BABEŞ-BOLYAI



INFORMATICA

2/2019

STUDIA

UNIVERSITATIS BABEŞ-BOLYAI
INFORMATICA

No. 2/2019

July - December

EDITORIAL BOARD

YEAR **Volume 64 (LXIV) 2019**
MONTH **DECEMBER**
ISSUE **2**

S T U D I A

UNIVERSITATIS BABEȘ-BOLYAI

INFORMATICA

2

EDITORIAL OFFICE: M. Kogălniceanu 1 • 400084 Cluj-Napoca • Tel: 0264.405300

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXIV**, Number 2, 2019
DOI: 10.24193/subbi.2019.2.01

DETECTION OF PEDESTRIAN ACTIONS BASED ON DEEP LEARNING APPROACH

DĂNUŢ OVIDIU POP[1,2,3]

Abstract. The pedestrian detection has attracted considerable attention from research due to its vast applicability in the field of autonomous vehicles. In the last decade, various investigations were made to find an optimal solution to detect the pedestrians, but less of them were focused on detecting and recognition the pedestrian's action. In this paper, we converge on both issues: pedestrian detection and pedestrian action recognize at the current detection time (T=0) based on the JAAD dataset, employing deep learning approaches. We propose a pedestrian detection component based on Faster R-CNN able to detect the pedestrian and also recognize if the pedestrian is crossing the street in the detecting time. The method is in contrast with the commonly pedestrian detection systems, which only discriminate between pedestrians and non-pedestrians among other road users.

1. INTRODUCTION

Pedestrian detection is a significant problem for computer vision, which involves several applications, including robotics, surveillance, and the automotive industry. It is one of the main interests of transport safety since it implies reducing the number of traffic collisions and the protection of pedestrians (i.e., children and seniors), who are the most vulnerable road users.

There are almost 1.3 million persons die in road traffic collisions each year, and nearly 20-50 million are injured or disabled due to human errors inherited in the usual road traffic. Moreover, the clashes between cars and pedestrians are the leading cause of death among young people, and it could be effectively reduced if such human errors were eliminated by employing an Advanced Driver Assistance System (ADAS) for pedestrian detection.

Received by the editors: May 16, 2019.

2010 *Mathematics Subject Classification.* 68T40, 68T45.

1998 *CR Categories and Descriptors.* I.2.9 [**Artificial intelligence**]: Robotics – *Autonomous vehicles*; I.2.10 [**Artificial intelligence**]: Vision and Scene Understanding – *Video analysis*; I.5.4 [**Artificial intelligence**]: Applications – *Signal processing*.

Key words and phrases. Pedestrian Detection, Pedestrian Action Recognition, Deep Learning.

Over the last decade, the scientific community and the automobile industry have contributed to the development of different types of ADAS systems in order to improve traffic safety. The Nissan company has developed a system which recognizes the vehicle's environment, including pedestrians, other vehicles, and the road. Lexus RX 2017 has a self-driving system which is linked up to a pedestrian detection system. More recently, the Audi ADAS system accumulates the data of the camera and/or radar sensor to determine the possibility of a collision by detecting pedestrians or cyclists and warns the driver with visual, acoustic and haptic alerts if a crash is coming.

These current ADAS systems still have difficulty distinguishing between human beings and nearby objects. In recent research investigations, deep learning neural networks have frequently improved detection performance. The impediment for those patterns is that they require a large amount of annotated data.

This paper proposes a pedestrian detection system which not only discriminates the pedestrians among other road users but also able to recognize the pedestrian actions at the current detection time (T=0).

The contribution of this paper concerns detecting and classifying pedestrian actions. To do so, we develop the following methodology relying on a deep learning approach:

- Train, all pedestrian samples with the Faster R-CNN-Inception version 2 for pedestrian detection, proposes [1];
- Train all pedestrian samples also using the pedestrian actions tags (cross/not cross) with the CNN as mentioned above for detection and action recognition based on the Joint Attention for Autonomous Driving (JAAD) [2] dataset;

The paper is organized as follows: Section 2 outlines sever existing approaches from the literature and supplies our main contribution. Section 3 presents an overview of our system. Section 4 describes the experiments and the results on the JAAD dataset. Finally, Section 5 presents our conclusions.

2. RELATED WORK

A wide variety of methodologies have been proposed with optimization in performance, resulting in the development of detection methods using deep learning approaches [3, 4, 5] or combination of features followed by a trainable classifier [4, 6].

A deep, unified pattern that conjointly learns feature extraction, deformation handling, occlusion handling, and classification evaluated on the Caltech and the ETH datasets for pedestrian detection was proposed in [7]. An investigation focused on the detection of small scale pedestrians on the Caltech data

set connected with a CNN learning of features with an end-to-end approach was presented in [8].

A combination of three CNNs to detect pedestrians at various scales was introduced on the same monocular vision data set [9]. A cascade Aggregated Channel Features detector is used in [10] to generate candidate pedestrian windows followed by a CNN-based classifier for verification purposes on monocular Caltech and stereo ETH data sets.

In [11] is presented a pedestrian detection based on a variation of YOLO network model, (three layers were added to the original one) in order to join the shallow layer pedestrian features to the deep layer pedestrian features and connect the high, and low-resolution pedestrian features.

A mixture of CNN-based pedestrian detection, tracking, and pose estimation to predict the crossing pedestrian actions based on JAAD dataset is addressed in [12]. The authors utilize the Faster R-CNN object detector based on VGG16 CNN architecture for the classification task, use a multi-object tracking algorithm based on Kalman filter, apply the pose estimation pattern on the bonding box predicted by the tracking system and finally handle the SVM/Random Forest to classify the pedestrian actions (Crossing /Not Crossing).

This approaches mentioned above use standard pedestrian detection pattern which only discriminates the pedestrian and non-pedestrian among other road users. To our knowledge, there is no prior research linked to pedestrian detection, which involves various pedestrian behavior tags.

We investigate the pedestrian detection issue in two ways, using the pedestrian and non-pedestrian tags, we call the Classical Pedestrian Detection Method (CPDM), and various pedestrian tags the: pedestrian, pedestrian crossing which we call Crossing Pedestrian Detection Method (CrPDM).

3. PEDESTRIAN DETECTION METHOD

This paper concerns solve the problem of pedestrian detection and pedestrian recognition actions using deep learning approach.

For developing a pedestrian detection system is mandatory to take into account three main components: the sensors employed to capture the visual data, the modality image processing elements, and the classification parts. In general, all these components are synchronically developed to achieve a high detection performance, but seldom specific item could be investigated independently according to the target application. We concurrently exam in the detection part by applying a generic object detector based on the public Faster R-CNN [13]. We handle the Inception CNN architecture versions 2 for the classification task with the TensorFlow public open-source implementation

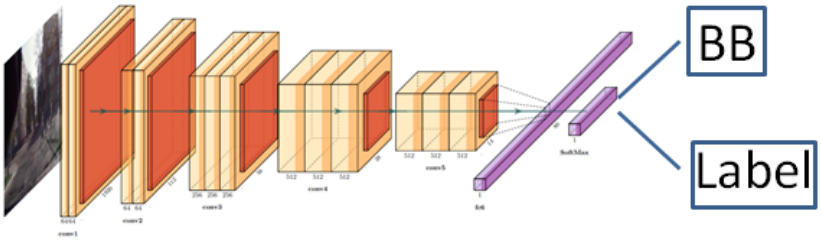


FIGURE 1. Pedestrian detection system architecture. Label represents the pedestrian tags which could be pedestrian, pedestrian crossing; BB cord represents the bounding box coordinates.

described in [1]. All the training process is based on JAAD [2] dataset. This dataset has different pedestrian tags. It has an annotation of pedestrians with behavioral tags and pedestrians without behavior tags.

4. EXPERIMENTS

This section presents the set of experiments, including setups and performance assessment of our approaches.

4.1. **Data Setup.** The experiments are completed on the JAAD dataset [2] because of its data set in typical urban road traffic environments for various locations, times of the day, road and weather conditions. This dataset supplies pedestrian bounding boxes (BB) for pedestrian detection, tagged as non-occluded, partially occluded and heavily occluded BBs. Moreover, it includes the pedestrian actions annotations for several of them, the pedestrian attributes for estimating the pedestrian behavior and traffic scene elements.

The experiment employs all the pedestrian samples, including the partially and heavily occluded pedestrians for all training and testing process.

4.2. **Training, Testing and Evaluation Protocol.** We used the first 70% of samples from the whole JAAD dataset for the learning process. The training set consists of first 190 video sequence training samples, and includes even the partially occluded and heavily occluded BBs, in contrast with [2] where the authors used just a part of the dataset, omitting samples with partially and heavy occlusion. Moreover, the authors do not give a detailed explanation of how the datasets were divided into training and testing sets; hence, it does not allow a fair comparison.

FIGURE 2. Pedestrian detection using the pedestrian tag for all pedestrians.

FIGURE 3. Pedestrian detection using multiple tags.

The validation set represents 10% of the learning set. We used the holdout validation method, which held back from training the model. The evaluation of a model skill on the training dataset would result in a biased score. Therefore

TABLE 1. Our detection performances using one or multiple output labels. One label represents that all samples are tagged as a pedestrian. Multiple labels represent: P=Pedestrian, PCr=Pedestrian Crossing.

Method	Train on	Output	mAP% ± CI
Faster R-CNN Inception v2	All pedestrian samples tagged as P	Pedestrian BB Label	70.91 ± 1.61
Faster R-CNN Inception v2	All Pedestrian with P and PCr Tags	Pedestrian BB+ Action Label	64.31 ± 1.70
SSD Fusion Inception [14]	RGB, Lidar, Distance	Pedestrian BB Label	51.88

the model is evaluated on the holdout sample to give an unbiased estimate of model skill.

The JAAD dataset has three main pedestrian actions tags:

- pedestrian completes crossing the street;
- pedestrian does not cross the street;
- pedestrian does not have any intention of crossing.

The first pedestrian action we tag as pedestrian cross and others as pedestrian considering his/her intention is ambiguous or does not cross the street.

We adopt two approaches for the training stage (the main architecture is described in Fig 1):

- using the Classical Pedestrian Detection Method (CPDM) where we consider all pedestrian samples without any specific tag (see Fig 2);
- using the Crossing Pedestrian Detection Method (CrPDM) where we use various pedestrian tags: Pedestrian is crossing the street (PCr), and Pedestrian (P) for all other pedestrians who do not cross the street or their action is obscure (see Fig 3).

We perform the CNN training process on 200000 iterations, using an initial learning rate value to 0.00063 with ADAM learning algorithm and momentum at 0.9 [2]. We used the pre-trained weights from COCO dataset with the default Faster RCNN loss function which is optimized for a multi-task loss function.

The testing set used to assess the CNN model performance is independent of the training dataset. It contains 110 video samples, the last 30% of video samples from the whole JAAD dataset.

The evaluation process for all the CNN models is performed with Tensorflow Deep Neural Network Framework. The performances are assessed by the mean

average precision (mAP) for the detection part using the TensorFlow metrics tool.

We calculate the Confidence Interval (CI) to evaluate whether one model is statistically better than another one.

$$CI = 2 * 1.96\sqrt{\frac{P(100-P)}{N}}\%. \tag{1}$$

In this formulation, P represents the performance system (e.g., mAP) and N represents the number of video testing.

4.3. **Results and Discussions.** The detection results are presented in Table 1. We observed that the detection performance achieved with the CPDM method (using all samples as pedestrians) a good performs on Jaad dataset since it has to identify the pedestrian among other road users. The CrPDM approach (using multiple pedestrian tags), although it detects the pedestrians, cannot be associated with the first method because it also instantly classifier the action of the pedestrians during the detection step. Therefore its performance is less than the first detection approach. On the other hand, pedestrian detection using the multiple tags approach could be a start point for a deep investigation. This approach estimates the pedestrian actions on the current time (T=0) and could be beneficial for developing a pedestrian prediction system. We can not compare our detection models with JAAD approaches [2] as our results are not directly comparable since the authors made a classification for a specific pedestrian action based on pedestrian attention information and only used the non-occluded pedestrian samples [2]. Their approach is based on a variation of AlexNet-imagenet CNN where the input data are cropped beforehand.

Contrariwise we should evaluate our model using approximately the same non-occluded pedestrian training samples as in [2] for a fair comparison, but we did not do that in this previous work since we assume that in the all traffic congestion even exist pedestrians who are partially and heavily occluded.

Nevertheless, we compare our detection models with another method [14] which is close to our first one. This approach is based on a variation of SSD-Inception CNN based on SvDPed dataset. The method merged the RGB images, low-resolution Lidar and the distance between the camera and object detected. Notwithstanding the [14] approach used many sensors for pedestrian detection, our methods outperforms this approach significantly (please see Table 1).

5. CONCLUSIONS

This paper presents a classical pedestrian detection system and a pedestrian detection system able to recognize even the pedestrian actions based on deep learning approaches using JAAD dataset.

We evaluated the pedestrian detection approach (we called Classical Pedestrian Detection Method (CPDM)), where all sample are tagged as pedestrian and not pedestrian and a pedestrian detection approach using multiple tags (pedestrian, pedestrian cross), which we call Crossing Pedestrian Detection Method (CrPDM). The first method achieved better performance since it has only to distinguish the pedestrians among other road users in contrast with the second one who has to recognize even the pedestrian actions. The second detection approach returned a weaker performance than the classical one. Contrariwise, we deem this approach could be a start point for a deep investigation. The experiments were carried out on the common object detector based on the public Faster R-CNN merged with Inception version 2 architecture for the classification part.

Future work will be concerned with improving and benchmarking of pedestrian detection using multiple pedestrian action tags and extending the method to a pedestrian prediction system.

References

[1] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. *CoRR*, abs/1611.10012, 2016.

[2] Amir Rasouli, Iuliia Kotseruba, and John K. Tsotsos. Are they going to cross? a benchmark dataset and baseline for pedestrian crosswalk behavior. In *The IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017.

[3] Dănuţ Ovidiu Pop, Alexandrina Rogozan, Fawzi Nashashibi, and Abdelaziz Bensrhair. Incremental cross-modality deep learning for pedestrian recognition. *In 28th IEEE Intelligent Vehicles Symposium (IV)*, pages 523–528, June 2017.

[4] Shanshan Zhang, Rodrigo Benenson, Mohamed Omran, Jan Hendrik Hosang, and Bernt Schiele. How far are we from solving pedestrian detection? *CoRR*, abs/1602.01237, 2016.

[5] J. Schlosser, C. K. Chow, and Z. Kira. Fusing lidar and images for pedestrian detection using convolutional neural networks. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2198–2205, May 2016.

[6] Rodrigo Benenson, Mohamed Omran, Jan Hosang, and Bernt Schiele. Ten years of pedestrian detection, what have we learned? In Lourdes Agapito, Michael M. Bronstein, and Carsten Rother, editors, *Computer Vision - ECCV 2014 Workshops*, pages 613–627, Cham, 2015. Springer International Publishing.

[7] W. Ouyang, H. Zhou, H. Li, Q. Li, J. Yan, and X. Wang. Jointly learning deep features, deformable parts, occlusion and classification for pedestrian detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP(99):1–1, 2017.

[8] R. Bunel, F. Davoine, and Philippe Xu. Detection of pedestrians at far distance. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2326–2331, May 2016.

[9] M. Eisenbach, D. Seichter, T. Wengefeld, and H. M. Gross. Cooperative multi-scale convolutional neural networks for person detection. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 267–276, July 2016.

[10] Xiaogang Chen, Pengxu Wei, Wei Ke, Qixiang Ye, and Jianbin Jiao. *Pedestrian Detection with Deep Convolutional Neural Network*, pages 354–365. Springer International Publishing, Cham, 2015.

[11] W. Lan, J. Dang, Y. Wang, and S. Wang. Pedestrian detection based on yolo network model. In *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*, pages 1547–1551, Aug 2018.

[12] Z. Fang and A. M. López. Is the pedestrian going to cross? answering by 2d pose estimation. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1271–1276, June 2018.

[13] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[14] T. Kim, M. Motro, P. Lavieri, S. S. Oza, J. Ghosh, and C. Bhat. Pedestrian detection with simplified depth prediction. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2712–2717, Nov 2018.

(1) Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu Street, 400084 Cluj-Napoca, Romania

(2) INRIA Paris, RITS team, 2 Rue Simone IFF, 75012 Paris, France

(3) INSA Rouen, LITIS, 685 Avenue de l'Université, 76800 Saint-Étienne-du-Rouvray, France

Email address: danutpop@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXIV**, Number 2, 2019
DOI: 10.24193/subbi.2019.2.02

ROAD CONDITION CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

GEORGE-BOGDAN MACA

Abstract. *Autonomous driving is an increasingly important theme nowadays. One of the reasons behind this is the evolution of hardware components in the last years, which made possible both research and implementation of much more complex deep learning techniques. An interesting direction in the vast field of autonomous driving is the discrimination of the condition of the road, with respect to weather. This paper presents a supervised learning based approach to road condition classification. Specifically, we take advantage of the power of Convolutional Neural Networks (CNNs) in the context of image classification. We describe several CNN architectures that use state of the art deep learning techniques and compare their performance. In addition to the simple CNN-based learners, we propose a CNN-based ensemble learner able of a better predictive performance compared to the single models.*

1. Introduction

In this paper we refer to *road condition* as the state of the road in terms of weather (i.e. if the road is bare, or covered by water or snow). By the *classification* of road condition we refer to the task of deciding what is the state of the road from a given image. It can be argued that this task can be quite complex, since there might arise situations difficult to distinguish even by the human brain. These are the cases that fall somewhere at the *limit* between two classes, when it is not obvious to which of those classes does the image belong. Additionally, since the data comes from real traffic scenes, the images might contain *noise* in the form of other objects present in the scene, that can obturate the view (e.g. cars, pedestrians, etc.).

Why do we need road condition classification? The concept of *self-driving cars* has been around for some years. Nowadays it is becoming increasingly

Received by the editors: June 25, 2018.

2000 *Mathematics Subject Classification.* 68T05,91E45.

1998 *CR Categories and Descriptors.* I.2.6 [**Artificial intelligence**]: Learning – *Concept learning.*

Key words and phrases. Autonomous driving, Road condition classification, Supervised learning, Convolutional Neural Networks, Ensemble learner.

popular and we can already see it in real world scenarios. In the context of *driver assistance*, or potentially even *autonomous driving*, the braking system of a vehicle can be improved. Specifically, the parameters of the ESP (electronic stability program) system can be adjusted depending on the condition of the road. Another scenario, especially helpful for 2-wheeled vehicles, would be sending warnings to the driver in case of dangerous road state, so that he can reduce the speed in time. For example, the road could be wet and thus slippery, even though it is not obvious from the condition of the weather at that particular moment.

However, the necessity for road condition classification was first introduced not in the context of autonomous driving, but rather in the not so "luxurious" task of dealing with *extreme weather conditions* (especially during winter). In northern states like Norway, Sweden, Canada, etc. winters can be very harsh and usually have very bad effects on the roads and the traffic, implicitly. Therefore, these situations must be taken care of very responsibly and efficiently (by "efficient" we mean both fast and, if possible, with the smallest consumption of resources).

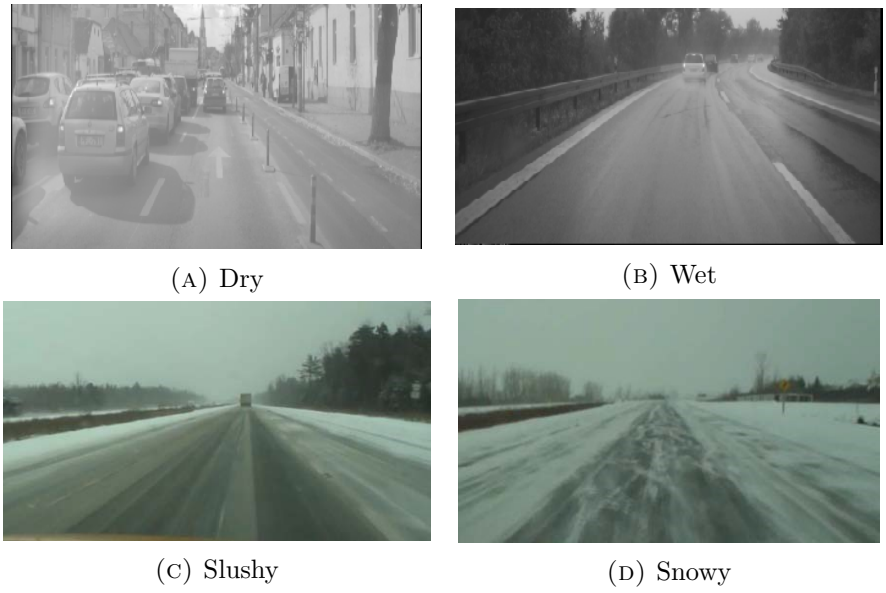
Currently, monitoring of winter road conditions is mainly done either using road weather information systems (RWIS) installed at fixed positions or through visual observation and manual recording by maintenance personnel. The former is limited in spatial coverage while the latter is limited in repeatability, objectivity and details [16]. Therefore, the need of automatic classification of the state of the road in a particular place (e.g. in one of the following classes: dry, wet, snowy, slushy, etc. - Figure 1) came up. The classification will be done *automatically* using *machine learning* techniques. More specifically, a direct approach of the form:

$$input - classifier - label$$

will be used, rather than the classical approach based on feature extraction:

$$input - features - classifier - label$$

In this problem we approach the task of classifying real world traffic scenes. The data consists of images obtained from video cameras mounted on cars. During the training process, each image (frame) is labeled with one of three classes (dry, wet, snow). The choice of output classes was conditioned by the state of our dataset, which was constructed and labeled based on images from these particular classes, that we were able to gather. Since the images are labeled, a *supervised learning* algorithm is more suited then an unsupervised one, mainly because the supervised approach would be more robust and certain of success. We have therefore chosen to use an approach based on *Convolutional Neural Networks* [12, 10].



(A) Dry (B) Wet

(C) Slushy (D) Snowy

FIGURE 1. Possible states of the road.

In this paper we present a supervised learning based approach to road condition classification, specifically using Convolutional Neural Networks. To the best of our knowledge regarding the state of the literature, we introduce a novel approach on this task, mainly due to the fact that we propose original architectures. There is also a very recent similar work [15] in which CNNs are used for Road Condition Classification. However, the authors of the paper [15] use state of the art CNN architectures [5, 21] pretrained on the ImageNet dataset [2]. Our works also differ in terms of chosen output classes and the conditions in which the images are recorded. The main contributions of this paper come from the fact that we propose multiple CNN architectures, built using various techniques. Also, we combine some of these architectures and obtain different models, including an ensemble of three learners. The models are evaluated experimentally and the obtained results are discussed.

The rest of the paper is structured as follows. In Section 2 we present another approaches to the same problem, found in literature. Section 3 offers a detailed explanation of our supervised learning approach, as well as a presentation of the architectures used. In Section 4 we present the performance of each of the proposed models. The results shown are then discussed in Section 5. Section 6 ends the paper with conclusions and potential improvements.

2. Related work

In this section we present a few approaches that are related to our work in different aspects. Such aspects are: the Machine Learning algorithms used, the information used as input (obtained from sensors or images), the type of input information (raw or processed). We focus on scientific works dedicated to road condition classification in scenarios similar to the ones available in our dataset.

2.1. **Friction measurement.** A solution rather related to physics, based on continuous friction measurement (CFM), was proposed in [3]. Friction is a measure of the resistive force to movement between the tires of a vehicle and the road surface and thus it represents an accurate approximation of the quality of driving on that road. In the study, they used the idea that the value of friction is inversely proportional with the quantity of snow on the road.

Friction measurements on a section of Highway 417 in eastern Ontario were obtained using a special equipment called Traction Watcher One (TWO). In addition to the sole friction measurement, the authors used some new parameters obtained from probability density pattern (e.g. variance and skewness) and from spectral density pattern of CFM. For this problem, 6 different classes were used: bare dry (Type 0), bare wet (Type 1), thin snow cover (Type 2), slushy snow cover (Type 3), partially snow covered (Type 4) and mostly snow covered (Type 5). After observing that some pairs of their chosen classes were very difficult to distinguish, the authors of the article decided to use a multilayer nested structure as their model. This architecture comprised five binary logistic regression models.

Split number	**Validation accuracy**
1	0.9383
2	0.9405
3	0.7957
4	0.8876
5	0.58063

Table 1. Results of the CFM approach. [3]

The results obtained in this paper, on the validation data set, are summarized in Table 1. It shows the performance of the five logistic regression models in turn. For each one of them, the authors decided what features to use and how to combine them, from the following list: *F* - friction, *Std* - standard deviation, *Skew* - skewness, *HighFreq* - high frequency power of CFM, *LowFreq* - low frequency power of CFM.

2.2. **Neural Networks approach.** Another approach to road condition classification is described in [11]. It is an image based approach that uses Neural Networks which receive as input feature vectors, carefully chosen from images. The purpose is to classify images in one of the following 5 classes: dry, wet, snowy, icy and snowy with tracks. Their motivation is to supplement already existing measurements (such as wind speed and temperature) with new information obtained solely from images. Also, the stage is set for this approach, in the sense that some measurement stations are already equipped with video cameras. The knowledge of the road state is used in order for the authorities to decide what kind of maintenance to perform and in what places.

After a careful analysis of what kind of information from an image is needed for this particular problem, the authors of the article end up with 6 features (e.g. mean gray level, standard deviation of the gray level, ratio of the standard deviation of the red image to the standard deviation of the blue image, etc.).

One of the downside of the proposed solution is the dimensionality of the architecture. More specifically, the neural network used had no more than 9 neurons (3 or 4 on the input layer; 3, 4 or 5 on the single hidden layer and one on the output layer – depending on the architecture and feature combination chosen). This is understandable, due to the fact that the dataset was very small – 69 RGB images in total . Thus, if the neural network had been bigger, it would have been very prone to overfitting. Mainly because of the two unfavorable aspects discussed above, the results obtained also make room for improvement: their best model and combination of features produces about 50% accuracy. However, as the authors of the article noted, there are several possibilities of improvement in this direction.

In addition to the work presented above, the authors of [15] introduce another approach based on Neural Networks, specifically CNNs. However, they use ResNet [5] and Inception [21] CNNs pretrained on the ImageNet dataset [2]. Also the output classes addressed in their problem are different from the ones used for our images.

2.3. **SVM-based approach.** In order to address the limitations of the currently existing methods of monitoring road surface condition (either using RWIS, or maintenance personnel) discussed in the first section, a new solution was proposed in [16]. It is based on using general purpose vehicles (like police cars or public transport) equipped with GPS systems and video cameras. Each image is GPS – tagged upon being taken and whenever there is an available internet connection, the tagged data is sent to a central server, where the actual computing takes place. This particular idea is very efficient in terms of time and space coverage.

As for the image classification task, a *support vector machine* (SVM) was used. The classes chosen for this problem are winter oriented: bare (dry), snow covered and track bare (with the center covered). Since the classification problem has more than two classes, the authors used a *one vs. all* multiclass technique. The data consisted of more than 500 RGB images (207 bare, 109 covered and 200 track bare). Each image was resized to 500x150 pixels before the feature extraction process. The dataset was split in the following way: 70% training images and 30% for testing purposes.

Results obtained with this approach are reasonably good, offering an average accuracy of 85% for the correct mapping of each of the three classes. These achievements are limited by various factors, like: the quality of the images, the amount of data and the illumination conditions, which tend to have a big influence in the color composition of an image.

There are several other approaches worth presenting, but many of them use methods based on features extracted from images (e.g. the one described in [17]). However, we want to direct our attention towards an approach that performs straightforward processing of data.

3. Proposed approach

3.1. **Theoretical background.** In this paper we present an approach to road condition classification based on *Convolutional Neural Networks* (CNNs) [12]. CNNs are the current state of the art in image classification tasks [20] and in the last decade have been used intensely for this kind of problems [10, 18, 21, 5]. Therefore, they represent an obvious direction to follow in the context of road condition classification. In addition, this technique has been also used in the context of autonomous driving, providing some good results (e.g. the approach described in [1], where the authors present an end-to-end learning method that maps an input image to a steering angle for the car).

CNNs are very similar to classical neural networks, in the sense that their ultimate purpose is to approximate a function, by training a set of weights. The main difference between them is that CNNs were created for the specific case, where the input to the network is an image. However, other types of inputs can also be fed to the network (e.g. audio, or even text) as long as they respect the *translational invariance* property (i.e. when it is not important exactly where something is located in an image). By making this assumption, convolutional networks can benefit from certain useful optimizations, which drastically reduce the otherwise huge number of parameters. The main advantage of a convolutional network over an ordinary neural network (when

working with images) is that they can obtain almost similar results with a much lower computational cost.

3.2. **Proposed architectures.**

3.2.1. *Overall architecture.* Before getting to the architectural details of the models used in this paper, we will first describe shortly each type of layer present in a convolutional neural network.

Convolutional layers are the core constructs of a CNN. Their behaviour can be understood as applying a filter operation on an image (i.e. each neuron on a convolutional layer computes a weighted sum of a group of pixels located at a particular position in the input volume). The parameters of a convolutional layer are the links between the neurons and the pixels they sum up.

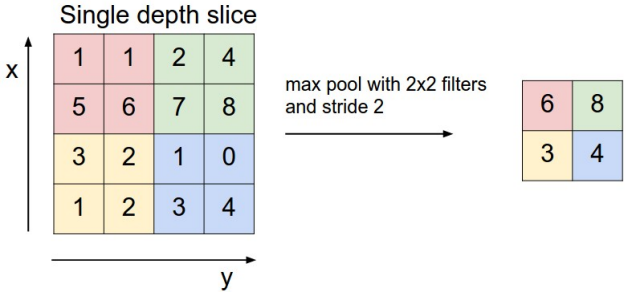


FIGURE 2. An example of max pooling [7].

Pooling layers are very similar to convolutional layers, the main difference being that pooling layers do not have learnable parameters. Their only purpose is to reduce the size of the input by performing *max* or *average* operations (no more weighted sums), as can be seen in Figure 2. *Dense* layers are nothing more than the *Fully connected* layers from the ordinary feed-forward neural networks. The neurons in this kind of layers are connected to all the neurons in the previous layer. It is a popular practice to use Dense layer towards the end of a CNN, just before the output layer.

A *Softmax* layer is used as the output layer of a CNN. It has the same number of neurons as the number of classes in the classification problem. The value in each of the neuron is interpreted as the normalized probability of the class corresponding to that neuron. The so called *softmax function* [8], which is able to compute those probabilities from the neurons on the previous layer, has the following form:

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{1}$$

where: j is the current output neuron; the sum over k iterates over all the output neurons; z_k is the value inside the k^{th} output neuron after computing the weighted sum of its inputs.

3.2.2. *Proposed models.* In the following lines we will present the architectures used and discuss about the choices made. The first model we tried - let's call it **M1** - was inspired from the architectures used in [10] or [22]. More specifically, we started with bigger filters in the early layers (e.g. 7 x 7 in the first layer) and gradually reduced their dimension. This reduction happened after pooling layers or convolutional layers with strides of 2 x 2, so when the dimensions of the output volumes was reduced. In other words, this means that bigger filters would look at bigger "images" and this is intuitive since they can assimilate more information at a time, which seems to be desirable in the case of higher resolution images. Because the size of the filters decreases with the size of the image, we can say that their dimensions are directly proportional. However, it has been argued in the literature that this approach isn't actually practical (or at least, there exist better techniques) [18], but this will be discussed in more detail later, when we present another architecture we tried.

The second model used - call it **M2** - is a small adjustment to **M1** in terms of size. However, the overall architecture structure is similar. The two main differences between the models are the number of features maps at each layer (which is higher in the second model) and the number of parameters (which was also higher in **M2**). Although it wasn't highlighted in the description of **M1**, it is worth mentioning that both models contain a *Dense* layer at the end of the architecture, preceding the *Softmax* layer. The number of parameters in **M2** was drastically increased by using more neurons in that Dense layer.

In the third model used - call it **M3** - we incorporate and combine some more interesting concepts described in scientific papers. The authors of several papers in the literature [18, 21, 5] argue that the depth of a CNN is of central importance, especially for more complex tasks. This insight is actually intuitive, since the level of *abstraction* of the features extracted from a given input increases with the number of *consecutive (stacked)* layers in the network. Having in mind these assumptions, we decided to implement also a deeper architecture for the road condition classification problem. However, deep models come with several difficulties in the training process and we had

to make use of the ideas presented in the scientific papers in order to address these issues.

Firstly, we took advantage of the concept of *residual blocks* (see Figure 3), inspired from [5]. The intuition behind *residual connections* (or skip connections) is that they help the network to preserve information from the input, while going deeper through convolutional layers. Mathematically, the output of a residual block is of the form:

$$y = F(x) + x \tag{2}$$

where x is the input of the block, y is the output and $F(x)$ is the so called *residual mapping* (which is actually what an ordinary CNN would compute, given an input x). The " + " operator mentioned above is just an element-wise addition between the two operands involved, which means that their sizes must be equal along all the three dimensions (width, height and depth). This technique addresses the problem of *degradation* (i.e. the incapacity of very deep neural networks to converge even on the training set), as described in [5].

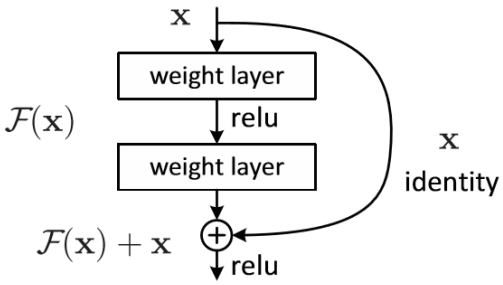


FIGURE 3. A building block of residual learning [5].

Secondly, we made extensive use of the of the *inception modules* introduced in [21], which can be visualized in Figure 4. The usage of this concept has at least two beneficial implications. The first advantage of inception modules is related to computational efficiency, since they introduce a parallelized manner of applying convolutions and pooling. The second insight is not as obvious as the first one and refers to the way information is extracted by the network. Specifically, at a particular layer in the network (i.e. given a fixed configuration of the input volume), information is extracted in more ways: by doing 1 x 1, 3 x 3 and 5 x 5 convolutions (the blue ones in Figure 4) and also a max-pooling (the red rectangle in the same image). This is not the case for ordinary CNNs, where only a single type of operation (e.g. 3 x 3 convolution, or max pooling,

etc) can be performed on **exactly** the same input (i.e. exactly the same values).

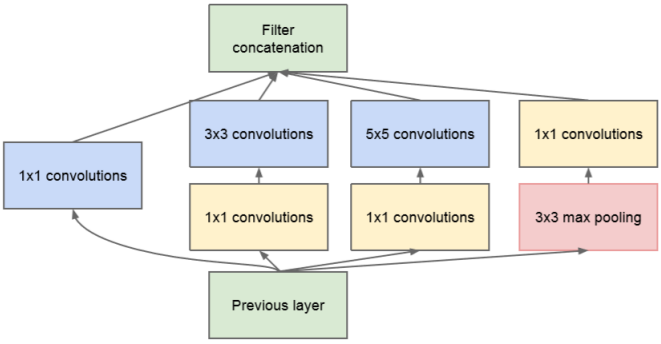


FIGURE 4. Inception module with dimension reductions [21].

The 1 x 1 convolutions from the yellow rectangles in Figure 4 are used for *dimension reduction*, as the authors of [21] argue. They say that performing convolutions with big filter sizes (e.g. 5 x 5, 7 x 7) is computationally expensive, so it would be better if they are performed on a small amount of feature maps in the input volume. Since the number of feature maps is given by the output of the previous layer and cannot be chosen arbitrarily, it was necessary to find a way to control that number. This is done by applying 1 x 1 convolutions on the input feature maps and choosing the desired number of output feature maps. In the case of 1 x 1 yellow convolution following the max pooling, its purpose is to control the number of feature maps that are concatenated with the convolutions; that number would otherwise be the same as the number of input feature maps, which would get higher and higher while going deeper through the network. It is also worth mentioning that in order to concatenate two or more volumes of feature maps, their spatial dimensions (width and height) must have the same size. So, these 1 x 1 convolutions in the yellow rectangles are used only for the reduction of the *depth* dimension, the other two dimensions remaining intact.

Last but not least, another interesting insight was gathered from [18]. As we mentioned earlier, we will describe another approach to choose the size of the kernels at each layer, rather than having big kernels in the first layers of the network and smaller ones towards the end. The authors of [18] mention the fact that two consecutive 3 x 3 convolutional layers perceive the same amount of information as a single 5 x 5 convolutional layer (a 7 x 7 layer is equivalent with three stacked 3 x 3 layers and so on). This implies that it is sufficient to use only filters of size (at most) 3 x 3 and this would have the

following advantages: reduce the computational cost induced by larger kernels and obtain CNN architectures that are deeper (which is desirable, as discussed earlier).

By combining the three ideas mentioned above (residual blocks [5], inception modules [21] and 3 x 3 kernels [18]) we end up with a network architecture made of building blocks looking like the one shown in Figure 5. There, one can observe the following aspects:

1 Only convolutions with filter sizes of at most 3 x 3 are used.
2 The parallelized block has also the role of the *residual mapping* $F(x)$ presented in Figure 3.
3 In the parallelized block, the following convolutions are performed, from left to right, on exactly the same input: a 1 x 1 convolution, a 3 x 3 convolution and two stacked 3 x 3 convolutions (equivalent to a single
5 x 5 convolution).
4 After the concatenation of the three branches, a 1 x 1 convolution is used in order to obtain a volume with the same number of feature maps as in the input, so that the element-wise addition between the input x and the residual $F(x)$ should make sense.

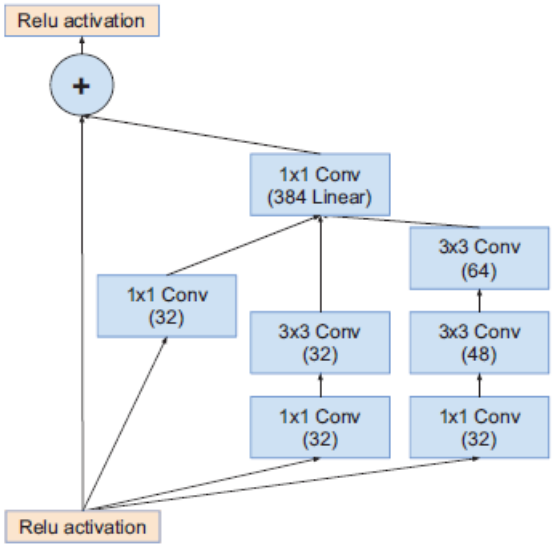


FIGURE 5. Example of Inception-ResNet building block [20].

An approach that combined these ideas is also presented in [20]. There, the authors also introduce the concept of *scaling* with respect to the addition

between the input and the residual mapping. They observed that computing a weighted sum that favours the input more would provide better results (at least in terms of convergence speed, if not actual accuracy). Thus, the " + " operator visible in Figure 5 would reduce to the following formula:

$$y = x + \gamma * F(x) \tag{3}$$

where γ is the scale parameter that weighs the value of the residual and it should take values between 0.1 and 0.3, as the authors of [20] argue. We have also conducted some attempts to test the **M3** model with different values and combinations for the scale parameter.

A highlight of the main aspects which are different in the three models proposed is illustrated in Table 2. The last column in the table refers to the number of neurons on the Dense layer that precedes the Softmax layer (**M3** doesn't have such a layer). It can be seen that, even though **M3** is a much deeper architecture than **M2**, their number of parameters is comparable (**M3** has even fewer parameters). The reason behind this is the lack of Dense layers, which is affordable for **M3**, but not also for **M2**.

Model	**Total number of layers**	**Number of conv. layers**	**Number of params.**	**Neurons on dense layer**
M1	16	3	608,659	64
M2	21	5	840,986	256
M3	86	28	812,298	-

TABLE 2. Comparison between the architectures used.

Another approach we tried was to create a separate network for each one of the classes involved in the problem and make only binary predictions. Thus, each network would solve a classification problem having only two classes: positive and negative (e.g. the *"dry-CNN"* would answer to the following question: "Is the road dry?" with either "yes" or "no"). The idea behind this approach was to obtain a specialized network for each of the possible road condition categories, rather than a more general network. In the end, this ensemble of networks would provide its answer in the following way: given an input image, feed it through each of the specialized networks and obtain the scores $s_1, s_2, \ldots, s_N$ (N being the number of classes), which are vectors with two elements representing the probabilities of the negative and positive classes respectively (these values sum to 1, according to *Softmax*); then take

maximum of the values of the positive classes from these vectors: $\max_{1 \leq k \leq N}(s_k[p])$ (where p represents the position where the probability for the positive class is stored in the score vectors); finally, conclude that the image introduced is of the type of the network that produced the maximum positive probability.

3.2.3. *Learning methodology - details.* For the learning process we use a gradient descent algorithm, specifically *Adam* [9]. Also, *relu* [13] activation functions are used after every convolutional layer, in every model.

In order to prevent overfitting we use the simple and effective concept of a *Dropout layer*, introduced in [19]. The intuition behind this idea is that some random neurons are ignored (eliminated, *dropped out*) during training so that they are not able to influence the process and especially their neighbour neurons. In this way, the neighbours that would normally depend on this neurons (which are now dropped out) are obliged to deal with the situation on their own and thus learn a more general context. This approach reduces the "co-adaptation" between neurons, as mentioned by the authors of the paper.

In order to address the famous problem *vanishing and* ***exploding*** *gradients* [4], a technique called *Batch Normalization* [6] is used. However, the idea was initially introduced in order to deal with *covariate shift* (i.e. the distribution of the inputs changes from layer to layer). Batch Normalization is applied at a particular layer and performs a *standard deviation* (stdev) normalization, thus disallowing the values of the parameters to reach undesirably high values. The term *"Batch"* from the name of the method refers to the fact that normalization is applied to a group of input data at a time and not to each training example separately. The authors of [6] argue that this has two advantages: the gradient of the cost function over a batch of inputs estimates better the gradient over the entire data set; and normalizing one batch at a time is more computationally efficient due to the possibility of parallelization.

3.3. **Evaluation metrics.** As far as the evaluation process is concerned, two possible metrics are useful in the context of image classification (or mapping an input to a label in general): the *loss* and the *accuracy*. The loss value is more relevant from the machine learning perspective, while the accuracy is more intuitive and easier to visualize for users. Our architectures use the *log-likelihood cost* [8, 14], since it comes together with the Softmax classifiers. The value of the loss is computed with the following formula:

$$L = -\log\left(\frac{e^{s_y}}{\sum_k e^{s_k}}\right) \tag{4}$$

where s_k is the k^{th} element of the vector of scores (class probabilities), y is the index of the correct label for the given input in the vector of scores s and s_y is the probability value of the correct class, outputted by the network. This result follows from the more general formula of the cross-entropy between two probabilistic distributions:

$$H(p,q) = \sum_x p(x)\log(q(x)) \tag{5}$$

where p is the "true" distribution, where the probability is concentrated on the correct class and q is the distribution of the estimated class probabilities (computed by the softmax function) [8]. The intuition behind the cross-entropy loss is that it quantifies exactly how far an estimated distribution outputted by the network is from the ideal case when the probability for the correct class is 1 and all the others are 0 (so, the farther the distributions are from each other, the greater the loss). In contrast, the accuracy measurement is much less realistic, since it just verifies if the maximum probability obtained corresponds with the actual correct class. So, for example, in a configuration with three classes, where the correct class for a particular example is class #3, an output with the following values: [0.33, 0.33, 0.34] would be classified as correct; this is far from ideal. However, the cross-entropy loss would heavily penalize this output, since it is not close at all to the correct output of: [0, 0, 1].

4. Numerical results

4.1. **The Dataset.** The classification problem we approached consists of three classes: *dry*, *wet* and *snow*. Our data set is split into three parts: training set, validation set and test set. Each of them is balanced in the sense that their images are equally spread between the three classes. The training set contains about 75000 images (with $\sim$25000 from each class) and the test and validation sets both contain approximately 15000 images (with $\sim$5000 from each class). The images are grayscale and have a 640 x 245 resolution.

The images from the dataset are obtained from video sequences filmed in real world traffic with a camera mounted on a car, which means that consecutive frames dumped from a sequence are very similar. Therefore, one additional aspect becomes relevant to the diversity of the input images: from how many different video sequences a particular bunch of images was obtained. This comparison would make sense only in a "per-class" context; a visual representation is shown in Table 3. The labeling was done per-sequence, which means that all the frames in a particular sequence have the same label. This aspect has some negative implications which will be discussed in Section 5.

Dataset	Dry sequences	Wet sequences	Snowy sequences
Training	82	52	12
Validation	11	5	2
Test	9	5	4

TABLE 3. Number of different sequences that lead to their per-class corresponding image sets (for example, the 25000 images labeled as "dry" from the training set were obtained from 82 different sequences, while the 5000 images labeled with "snow" from the validation set were obtain from only 2 sequences).

In order to train the specialized binary classifier models, we created three more datasets from the original one. The labels of these new datasets were adjusted in the following way: for each of the three networks, the corresponding class would have a *positive* label (e.g. value 1) and the other two classes would have a negative label (value 0). After this change in the labels, the negative class images became two times more than the positive ones. Thus, the dataset had to be modified such that the number the two numbers became equal. This was done by sampling half of the images with a negative label.

4.2. **Results.** For evaluation purposes, let's call by **M4** the model obtained from the ensemble of three specialized models (one for each class), each doing a binary classification. All the three models have the same architecture as **M3**, the only difference being that their softmax layers contain two neurons instead of three. Table 4 summarizes the results obtained by each of the four models discussed on both the validation and test sets. The accuracy metric is used in this illustration for simplicity and visibility. However, the model parameters have been chosen in terms of loss actually, even though there could have been configurations with higher accuracy (but also higher loss - which is not desirable).

In addition to the accuracy comparison of the four models, Table 5 illustrates the results provided by the three specialized binary classifiers in turn. We are going to provide next some technical data regarding our models. The training time ranges between 40 minutes for **M2** to 70 minutes for **M3**, while the validation and testing times range between 2 minutes and 5 minutes respectively, for the same two models. In addition, a prediction with **M2** yields approximately 7 frames per second (fps), while a prediction done using **M4** is much slower, providing only $\sim$2 fps.

Model	Validation accuracy	Test accuracy
M1	81.16%	$76.61 \pm 0.653\%$
M2	92.41%	$84.76 \pm 0.555\%$
M3	86.68%	$82.55 \pm 0.586\%$
M4	95.88%	$87.70 \pm 0.507\%$

TABLE 4. Accuracy comparison of the four discussed models in the 3-way classification problem. For the test accuracy metric we expressed the values using a *Confidence Interval* (CI) of 95%

Model	Validation accuracy	Test accuracy
dry-CNN	90.25%	87.81%
wet-CNN	96.37%	89.94%
snow-CNN	99.00%	90.01%

TABLE 5. Performance of the three specialized models in the 2-way classification problem.

5. Discussions and challenges

The **M1** model isn't particularly efficient, mainly due to its reduced size. Nonetheless, exactly this issue provided us with an interesting insight concerning the size of a CNN. Specifically, the **M1** model was able to converge on the training set, but it performs poorly on the validation and test sets. Our first intuition led us to believe it was an overfitting scenario caused by a high number of parameters. However, we went on to realize it was actually the reduced size of the network that was causing problems. An explanation of this situation could be that the small network made its best effort to fit the training data but, due to its reduced size, it was restricted to finding only a local minimum, being unable to generalize. We managed to prove this insight in practice with our second model, **M2**, which only had more layers and more neurons on the Dense layer compared to **M1**, the overall architecture style being the same.

The second model, **M2**, achieved surprisingly good results despite being only a regular *sequential* model (i.e. it doesn't contain parallelized building blocks or skip connections). It even outperforms the much deeper **M3** architecture, as can be seen in table 4. The reason behind this is still under study, but we are inclined to believe that either the model wasn't trained for an enough amount of epochs, or the lack of the Dense layer is affecting the

performance (although this doesn't happen in the case of the 2-class models, which after all are also of type **M3**).

One can observe in Table 4 that the ensemble of learners (**M4**) provides the best results in terms of accuracy. The result is strengthened by the fact that the confidence interval for the accuracy of this model does not overlap with the CI of any other model. This situation would also be predictable by looking at the good results obtained by each of the three specialized models, available in Table 5. However after analyzing the test accuracy of the 2-class models, we would expect a test accuracy of around 89% for the ensemble of models (**M4**), by taking the average value. This is not the case, since the actual test accuracy is $87.70 \pm 0.507\%$, as can be seen in Table 4. This happens because the *max* operation used to combine the outputs of the three models is not ideal.

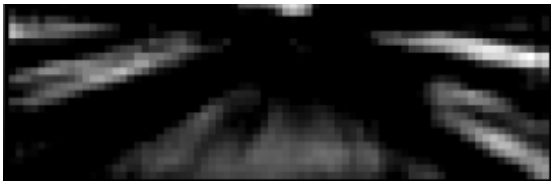
The main challenges encountered in our approach are related to the dataset. Firstly, some of the sequences contain images that lie at the limit between two classes. Such images are sometimes difficult to classify even by a human, not to mention an artificial neural network. Secondly, as discussed in Section 4.1, all the images obtained from a sequence share the same label. Obviously, some portions of the road in a sequence may not agree with the overall chosen label for that sequence and thus an inconsistency is introduced.

Other challenges we met are linked to the difficulty of understanding the behavior of a convolutional neural network. As also mentioned in Section 1, CNN is a direct approach that doesn't need manual feature extraction performed by the programmer. We can say that a CNN is able to decide by itself what features are needed in the learning process. In order to get a look inside the *black-box* of a CNN, we plotted the values of the neurons at different layers in the networks (see Figure 6). The white pixels in the subfigures 6b and 6c represent points of interest for the network. In this way one can get in insight regarding to where does a CNN "look" when it is fed with an input image.

We encountered difficulties in finding a relevant comparison between our results and the results obtained in the other approaches to road condition classification, presented in Section 2. There are multiple reasons behind this and we are going to briefly describe them. First of all, the datasets used for evaluation in other approaches from the literature are not published, so we couldn't evaluate our model on the same data. Even if the data was available, the problems approached are not identical, since the classes used for classification differ slightly in each paper. In addition, as far as we know there isn't any public benchmark containing data for the road condition classification problem. Moreover, the implementation details present in the papers from

(A) Input image.

(B) The plotted values of neurons of a feature map situated after the second convolutional layer (and its activation) of the **M2** model



(C) The plot of another feature map, situated after the third convolutional layer and its activation, in the same model

FIGURE 6. Looking inside a CNN

Section 2 are insufficient, so we were not able to reproduce the models and evaluate them on our data. Of course that we can analyze the accuracy values and compare them directly, but this is far from ideal and irrelevant.

6. Conclusions and future work

In this work, we presented a CNN-based road condition classification. We performed an in-depth analysis of several flavours of CNN-based single classifiers and we developed a CNN-based ensemble classifier that involves a bagging strategy with a max voting operator. We obtained a significant increase in the quality of classification (in terms of accuracy) by the developed ensemble

learning method. Our approach, albeit it is still in research phase, could potentially be used as a building block of an autonomous driving system, especially in terms of prediction efficiency.

After observing the initial success of our approach, we are convinced that *future work* should follow. Up to this point, the main improvement that we have in mind is related to the main challenge to our approach: the *dataset* (as we discussed in Section 5). The existence of images with uncertain labels isn't necessarily a strong estimate of the quality of the network, since some system might need to know only certain states of *dry, wet or snow*. Therefore we are planning to split the dataset into two, difficulty oriented sets: an easy one and a more difficult one. In this way we will be able to asses the performance of our architecture in a more realistic environment and also prove that the network performs bad mostly in uncertain conditions.

In addition to this, we also aim at improving the architectures, by trying more combinations of layers and hyperparameters with the purpose of increasing the classification quality even more. In order to obtain better results in terms of time and memory usage, we will try to adapt our CNN implementation to the specific properties of the board computers. Last but not least, we would like to consider a different method for splitting the dataset into train, validation and test sets. Currently the images are randomly split into the three datasets and we could use for example an algorithm like *K-Fold*.

Acknowledgment

The technical support of Lucian Cristea, my mentor during internship, is highly appreciated.

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[3] F. Feng, L. Fu, and M. S. Perchanok. Comparison of alternative models for road surface condition classification. In *TRB Annual Meeting*, 2010.

[4] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[7] A. Karpathy. Convolutional neural networks: Architectures, convolution and pooling layers. 2016. last access (july, 2018).
[8] A. Karpathy. Softmax classifier. 2016. last access (july, 2018).
[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
[11] A. Kuehnle and W. Burghout. Winter road condition recognition using video image classification. *Transportation Research Record: Journal of the Transportation Research Board*, (1627):29–33, 1998.
[12] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
[13] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
[14] M. A. Nielsen. Neural networks and deep learning. 2015. last access (july, 2018).
[15] M. Nolte, N. Kister, and M. Maurer. Assessment of deep convolutional neural networks for road surface classification. *arXiv preprint arXiv:1804.08872*, 2018.
[16] R. Omer and L. Fu. An automatic image recognition system for winter road surface condition classification. In *Intelligent transportation systems (itsc), 2010 13th international ieee conference on*, pages 1375–1379. IEEE, 2010.
[17] Y. Qian, E. J. Almazan, and J. H. Elder. Evaluating features and classifiers for road weather condition analysis. In *Image Processing (ICIP), 2016 IEEE International Conference on*, pages 4403–4407. IEEE, 2016.
[18] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
[20] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.
[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.
[22] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu Street, 400084 Cluj-Napoca, Romania
Email address: mgic1759@scs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXIV**, Number 2, 2019
DOI: 10.24193/subbi.2019.2.03

ANALYSING THE ACADEMIC PERFORMANCE OF STUDENTS USING UNSUPERVISED DATA MINING

GEORGE CIUBOTARIU AND LIANA MARIA CRIVEI

Abstract. *Educational Data Mining* is an attractive interdisciplinary domain in which the main goal is to apply data mining techniques in educational environments in order to offer better insights into the educational related tasks. This paper analyses the relevance of two *unsupervised learning* models, *self-organizing maps* and *relational association rule mining* in the context of students' performance prediction. The experimental results obtained by applying the aforementioned unsupervised learning models on a real data set collected from Babeş-Bolyai University emphasize their effectiveness in mining relevant relationships and rules from educational data which may be useful for predicting the academic performance of students.

1. Introduction

Extracting relevant patterns from the educational processes is the main topic in the *Educational data mining* (EDM) field, as it could provide effective methods for understanding students and their learning process and, subsequently, improving the learning outcomes. EDM has lately been under great consideration from the research community since extracting hidden valuable knowledge from educational data is of major interest for the academic institutions and also effective for reviewing and improving their teaching techniques and learning procedures [13].

There is a continuous interest in applying *machine learning* (ML) techniques in the educational field [3]. Within the ML domain, *unsupervised learning* (UL) techniques are broadly applied nowadays in numerous domains including software engineering, medicine, bioinformatics, the financial domain, in order to extract relevant hidden knowledge from the data in the form of rules

Received by the editors: October 28, 2019.

2010 *Mathematics Subject Classification.* 68T05, 68P15.

1998 *CR Categories and Descriptors.* H.2.8[**Database management**]: Database Applications – *Data Mining*; I.2.6[**Computing Methodologies**]: Artificial Intelligence – *Learning*;

Key words and phrases. Educational data mining, Unsupervised learning, Self-organizing map, Relational association rule.

or patterns. Diverse applications using data mining and machine learning algorithms have been implemented, so far, in the EDM domain. Machine learning methods are applied, both from a *supervised* and *unsupervised* perspective, as data mining techniques for developing systems for course planning, predicting the students' progress and academic performance, detecting students' learning type, grouping students in similar classes, supporting instructors in the educational process [8].

The study performed in this paper is aimed to highlight the potential of applying two UL techniques (*self-organizing maps* (SOMs) and *relational association rule mining*) (RARs) in analysing students' academic performance. The main research question we are investigating in this paper is regarding the ability of unsupervised learning models (SOMs and RARs) to detect hidden relationships between the grades received by the students during the semester and their final examination grade category at a certain academic discipline. In addition, we aim to analyse if the unsupervised learning models may reveal some information regarding the quality of the educational processes.

A study on the EDM literature reveals various approaches using unsupervised learning for mining student data in educational environments. Various clustering algorithms, including partitional and hierarchical clustering were applied by Ayers et al. [2] for determining groups of students who have simlar skills. Dutt et al. [6] present a survey on applying *unsupervised learning* techniques for various tasks from the educational setting. *K-means* clustering is applied by Parack et al. [14] for grouping students according to their learning patterns. The identified groups are then used for determining the cognitive styles for each cluster. SOMs were used by Kurdthongmee [11] to group students in clusters according to their academic results. Khadir et al. [10] performed a study based on clustering and SOMs for students' academic performance prediction. Saxena et al. [15] have also applied SOMs for classifying students in categories according to their academic performance. To the best of our knowledge, a study similar to ours has not been performed in the literature.

The rest of the paper is organized as follows. Section 2 presents the *self-organizing maps* and *relational association rule mining* models used in our study. Section 3 introduces our experimental methodology, while Section 4 discusses about the experimental results. The conclusions of our study together with several directions for future research are summarized in Section 5.

2. Unsupervised machine learning models used

Unsupervised learning models are known in the ML literature as *descriptive* models, due to their ability to detect how data are organized. Unsupervised learning algorithms receive only unlabeled examples and learn to detect hidden patterns from the input data based on their features. UL methods are useful for discovering the underlying structure of the data.

2.1. **Relational association rule mining.**

Relational association rules (RARs) [4, 16] were introduced in the data mining literature as an extension of the classical *association rules* with the goal of uncovering various types of relationships, both ordinal and non-ordinal, between data attributes.

The concept of *RARs* will be further presented. We consider $\mathcal{O} = \{o_1, o_2, \ldots, o_n\}$ a set of *instances* (objects) and $\mathcal{A} = (at_1, \ldots, at_m)$ a sequence of relevant attributes characterizing the instances from $\mathcal{O}$. Each attribute at_i takes values from a non-empty domain D_i. The value of attribute at_i for instance o_j is denoted by $\eta(o_j, at_i)$ and by $\mathcal{T}$ is denoted the set of all possible relations which can be defined between the domains D_i and D_j.

Definition 1. *A* relational association rule [16] *is an expression* $(at_{i_1}, at_{i_2}, at_{i_3}, \ldots, at_{i_h}) \Rightarrow (at_{i_1} \tau_1\ at_{i_2} \tau_2\ at_{i_3} \ldots \tau_{h-1}\ at_{i_h})$, *where* $\{at_{i_1}, at_{i_2}, at_{i_3}, \ldots, at_{i_h}\} \subseteq \mathcal{A}$, $at_{i_k} \neq at_{i_p}$, $k, p = 1, \ldots, h$, $k \neq p$ *and* $\tau_k \in \mathcal{T}$ *is a relation over* $D_{i_k} \times D_{i_{k+1}}$, D_{i_k} *representing the domain of the attribute* at_{i_k}.

a) *If* $at_{i_1}, at_{i_2}, \ldots, at_{i_h}$ *are non-missing in* l *instances from* $\mathcal{O}$ *then we call* $s = \frac{l}{n}$ *the* support *of the rule.*

b) *If we denote by* $\mathcal{O}' \subseteq \mathcal{O}$ *the set of instances where* $at_{i_1}, at_{i_2}, at_{i_3}, \ldots, at_{i_h}$ *are non-missing and all the relations* $\eta(o_j, at_{i_1})\ \tau_1\ \eta(o_j, at_{i_2})$, $\eta(o_j, at_{i_2})\ \tau_2\ \eta(o_j, at_{i_3})$, $\ldots$, $\eta(o_j, at_{i_{h-1}})\ \tau_{h-1}\ \eta(o_j, at_{i_h})$ *hold for each instance* $o_j \in \mathcal{O}'$ *then we call* $c = \frac{|\mathcal{O}'|}{n}$ *the* confidence *of the rule.*

The notion of *interestingness* was introduced in [16] as the property of RARs to have their *support* and *confidence* greater than or equal to certain thresholds. The algorithm *DRAR* (*Discovery of Relational Association Rules*) for uncovering interesting RARs was introduced in [5]. *DRAR* is an Apriori-like algorithm consisting of a RAR generation process that starts from the 2-length rules which are filtered such that to preserve only the interesting rules. The iterative process continues with 3-length rules, then with 4-length rules and so on. At a certain iteration, the RARs of length n are generated by joining [16] interesting RARs of length n-1. The obtained set is then filtered for preserving only the interesting n-length rules. When no new interesting RARs are identified at a certain iteration, the generation process stops.

2.2. **Self-organizing maps.** *Self-organizing maps* (SOMs), also known in the literature as *Kohonen maps*, were introduced by Teuvo Kohonen and are *unsupervised learning* models widely used as tools for visualizing high-dimensional data. SOMs are connected to the *artificial neural networks* (ANNs) in literature and to *competitive learning*. In *competitive learning*, the output neurons compete themselves to be activated. A *self-organizing map* [17] is trained using an unsupervised learning algorithm (Kohonen's algorithm) to map, using a non-linear mapping, the continuous input space of high-dimensional instances into a discrete (usually two-dimensional) output space called a *map* [7]. The *topology preserving mapping* is the main characteristic of the mapping provided by a SOM. This means that the input samples which are close to each other in the input space will be also close to each other on the map (output space). Thus, a SOM is able to provide clusters of similar data instances [12].

3. Methodology

As previously stated, our study aims at investigating the relevance of unsupervised SOM and RAR models in analysing the academic performance of students.

We are introducing the following theoretical model. We denote by $\mathcal{S}tud = \{stud_1, stud_2, \ldots, stud_n\}$ a data set in which the instances $stud_i$ describe the performance of students during an academic semester, at a given academic discipline $\mathcal{D}$. Each instance $stud_i$ is characterized by a set of *grades* received during the semester $\mathcal{G} = \{g_1, g_2, \ldots, g_m\}$ representing attributes for measuring the performance of the student for the given discipline. Thus, each $stud_i$ is represented as an m-dimensional vector $stud_i = (stud_{i1}, stud_{i2}, \ldots, stud_{im})$, $stud_{ij}$ representing the value of attribute g_j for student $stud_i$.

The goal of the current study is to investigate if two unsupervised data mining models, *self-organizing maps* and *relational association rule mining*, are able to discover some rules and relationships which would be useful for predicting the *final performance* for the students, based on their grades obtained during the academic semester. Since predicting the exact final examination grade for a student is a difficult task, considering the uncertainty in the learning and evaluation processes, we are considering in this paper four categories of final grades: (1) *excellent* (denoted by E and representing the final grades 9 and 10); (2) *good* (denoted by G and representing the final grades 7 and 8); (3) *satisfactory* (denoted by S and representing the final grades 5 and 6); and (4) *fail* (denoted by F and representing the final grade 4). Let us denote by $\mathcal{C} = \{E, G, S, F\}$.

We note that our unsupervised analysis does not include the grades of the students' at the written exam (obtained in the examination session), which

are also part of their final examination grade. Thus, we aim to analyse if only the grades received by the students during the semester are enough to discriminate their written exam grade and, accordingly, the students' final examination grade category.

The problem investigated in this paper, from an *unsupervised learning* perspective, is that of assigning to each student (characterized by its grades received during an academic semester) the category corresponding to its final grade. This assignment may be formalized by a mapping $f : \mathbb{R}^m \to \mathcal{C}$.

3.1. **Data set.** In our experiment we are considering a real data set [1], denoted by D, containing the grades obtained by students at a Computer Science undergraduate course offered at Babeş-Bolyai University collected during six academic years (2011-2017) at the regular and retake examination sessions. The data set consists of 905 students characterized by 4 attributes, denoted by a_1, a_2, a_3, a_4. Attributes a_1, a_2, a_3, a_4 represent scores obtained by the students at several evaluations during the academic semester: seminar score (a_1), project score (a_2), project status score (a_3) and written test score (a_4). For each student $s \in D$, its *final examination grade* ($f(s)$), received at the end of the academic semester after the final examination is known. In our experiments, the final examination grade of a student is transformed into a category (E, G, S, F), as previously shown. However, in our unsupervised learning based experiments, the students' final grade will be used only for evaluating the learning performance, without using it for building the SOM and RAR models. Figure 1 depicts a histogram of grades (4-10) built on the data set D.

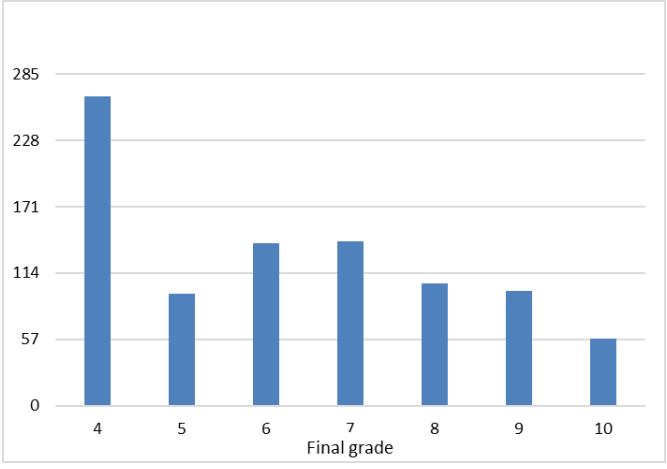


FIGURE 1. Histogram of grades from the data set D.

The histogram from Figure 1 reveals a distribution of passing grades (grades without 4) close to the normal one. For analysing how correlated are the attributes a_1, a_2, a_3, a_4 with the target output (category corresponding to the final examination grade), the Pearson correlation coefficients are computed. The correlation values are shown in Table 1.

a_1	a_2	a_3	a_4
0.631	0.676	0.461	0.607

TABLE 1. The Pearson correlation coefficient between attributes a_1, a_2, a_3, a_4 and the target output.

From Table 1 we observe that there is a good enough correlation between the attributes a_1, a_2, a_4 and the category corresponding to the final examination grade. The *project score* (attribute a_2) shows the maximum correlation with the final category. The smallest correlation is observed for attribute a_3.

3.2. **Experiments.** The experiments described in this section are aimed to test the ability of SOMs and RARs, as unsupervised learning models, to detect relevant relationships in the students' grades (received during the semester) which are well correlated with their final grade category. For a certain grade category (class) $c \in \{E, G, S, F\}$ we denote by $D_c \subset D$ the subset of students from D whose final grade category is c. We note that $\bigcup_{c \in \mathcal{C}} D_c = D$.

The *first experiment* is conducted for obtaining, using a SOM, a two dimensional representation of the data set. Two SOM visualizations will be provided: one for the entire data set (characterized by all attributes a_1, a_2, a_3, a_4) and the second for the data set without attribute a_3 (i.e the data set characterized only by the attributes a_1, a_2, a_4). After the SOM was unsupervisedly built, the U-Matrix method [9] will be used for visualization. For the SOM, a torus topology is used, with the following parameters: 200000 training epochs and a learning rate of 0.1. The *Euclidian distance* is used as a distance metric between the input instances.

The goal of our *second experiment* is to uncover in each subset D_c, using the $DRAR$ algorithm, a set RAR_c of interesting RARs. We aim to verify the hypothesis that the sets RAR_c are able to discriminate between the classes of students having different final grades.

For the RAR mining experiment, five additional attributes were added to the data sets D_c ($a_i = i, \forall i, 5 \leq i \leq 9$) and we used the following parameters for the mining process: 1 for the minimum support threshold, 0.6 for the minimum confidence threshold and two possible binary relations between the

attributes ($<$ and $\geq$). Attributes a_5, a_6, a_7, a_8, a_9 represent some thresholds for the grades (i.e. $5, 6, 7, 8, 9$) and were added with the goal of enlarging the set of uncovered RARs, allowing the discovery of binary RARs such as $a_i \leq 5$.

For evaluating how well the set RAR_c characterizes the set of students from the set D_c we use the average confidence of all the subrules from the rules from RAR_c, denoted by $Prec_c = \frac{\sum_{r \in RAR_c} \sum_{sr \in \mathcal{S}_r} conf(sr)}{|RAR_c|}$. By $conf(r)$ we denote the confidence of the rule r in the data set D_c and $\mathcal{S}_r$ represents the set of subrules of r (including itself). We note that $Prec_c$ ranges from 0 to 1, higher values for $Prec_c$ indicating that the set RAR_c better characterizes the data set D_c.

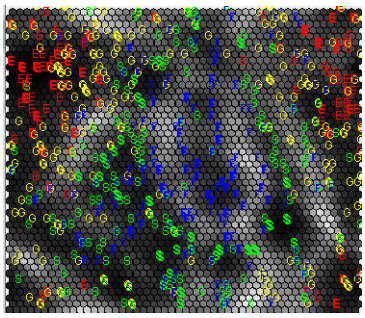
4. Results and discussion

This section presents the experimental results obtained following the experimental methodology introduced in Section 3.2 and discusses about the patterns unsupervisedly discovered using the SOM and RAR models.

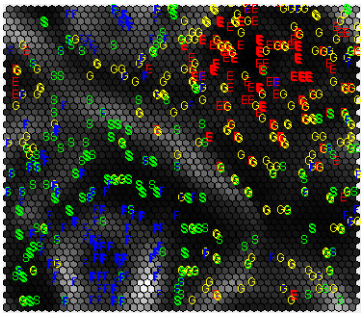
4.1. **Experiments using SOMs.** The left hand side image from Figure 2 illustrates the SOM obtained on the data set from Section 3.1 using attributes a_1, a_2, a_3, a_4, while the right hand side image from Figure 2 depicts the SOM trained on the instances characterized only by attributes a_1, a_2, a_4. On both images, the students with the same final class (final grade category which is depicted on the map) are marked with the same colour: red for the E labels, yellow for G, green for S and blue for F. As expected, Figure 2a depicts a good enough mapping, but still there is no clear separation between the grades. It seems that a slightly better mapping and separation between the grades is provided by the map from Figure 2b when attribute a_3 (the project status score) has not been considered.

The SOMs from Figure 2 reveal the difficulty of the task for predicting the final examination grade category for the students, based on the grades they received during the semester. However, the unsupervisedly built SOMs are able to uncover some patterns regarding the students' final grade category. We observe two main areas on both maps, a cluster of students with the final categories F and S, which is well enough delimited and one containing the categories G and E. Inside the first cluster, we observe a well distinguishable subclass containing students with the final category F.

For evaluating the quality of the SOMs built, the *average quantization error* (AQE) is computed during the unsupervised training process. The AQE [18] is computed by averaging the mean Euclidean distance between the input samples and their best-matching units. Figure 3 comparatively illustrates how AQE decreases during the training of the maps built for the entire data set



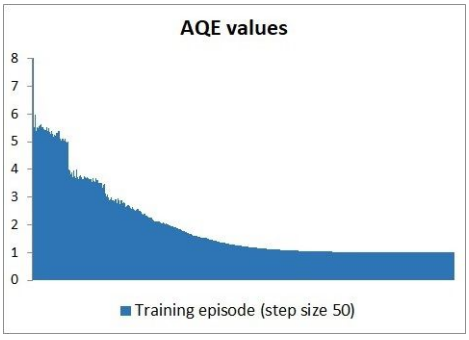
(A) SOM visualization considering attributes a_1, a_2, a_3, a_4.



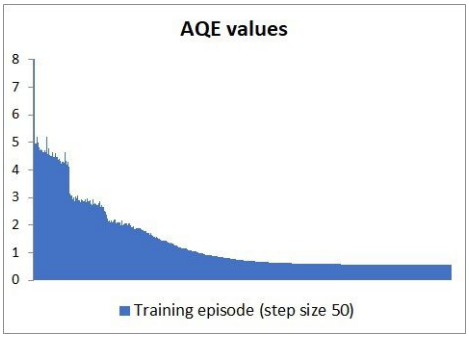
(B) SOM visualization considering attributes a_1, a_2, a_4.

FIGURE 2. U-Matrix visualization of the SOM built on the data set D using attributes a_1, a_2, a_3, a_4 (left) and a_1, a_2, a_4 (right).

(left side image) and for the data set without attribute a_3 (right side image). We note that the AQE reached after the training was completed is 0.997 for the SOM from Figure 3a and 0.559 for the SOM from Figure 3b. The final AQEs which are close to 0 confirm the accuracy of the trained SOMs. In addition, the SOM built on the data set without attribute a_3 has the smallest AQE, indicating a better mapping.



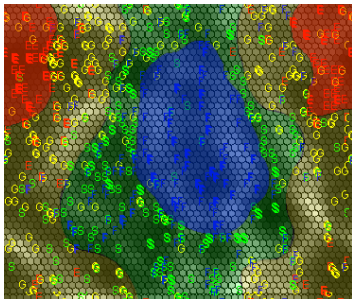
(A) AQE visualization for the SOM built using attributes a_1, a_2, a_3, a_4.



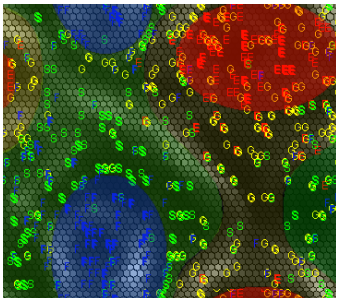
(B) AQE visualization for the SOM built using attributes a_1, a_2, a_4.

FIGURE 3. Evolution of AQE values during training the SOMs built on the data set D using attributes a_1, a_2, a_3, a_4 (left) and a_1, a_2, a_4 (right).

Figure 4 illustrates a detailed visualisation of the SOMs from Figure 2, considering the torus topology used for building the SOMs. The left side image from Figure 4 corresponds to the SOM from Figure 2a, while Figure 4a corresponds to the visualization from Figure 2b. On both SOMs, the distinguishable classes of students are highlighted and coloured according to their class label (E - red, G - yellow, S - green, F - blue).



(A) Detailed visualisation of the SOM from Figure 2a.



(B) Detailed visualisation of the SOM from Figure 2b.

FIGURE 4. Detailed visualisation of the SOMs from Figure 2.

A comparative analysis of the two images from Figure 4 and the highlighted areas reveal the following. In Figure 4a we observe that the F labeled cluster is well disgtinguishable. However, there are a few outliers that go beyond the cluster's border, entering in the S class zone. Moreover, the E labeled cluster also interferes with the outer regions and creates noisy zones on the map. The flaw with the image from Figure 4a is that, even though the centres of the E and F labeled groups are compact and solid, the margins of each one tend to be more fuzzy, exchanging different grades with neighbouring regions. As we can see, there are overlapping grades, especially regarding the F labels, that incline towards a defiance of the boundaries, which suggests that the unsupervised classification model is not capable of clearly discriminating all the students and, consequently, misinterprets some of the patterns of their performance. Regardless these misclassifications that may be due to the small number of attributes characterizing the instances and the presence of outliers, the SOM model is confident enough to make correct prediction most of the time. In Figure 4b there are two contrasting, well separated, areas of high/low grades with a sharp gap between them. The regions corresponding to the average grades surround tightly these two clusters. Few exceptions still occur when separating the grades.

On the other hand, the SOM from Figure 4b seems to provide a better classification of the data. The two opposed areas (of students with high/low grades) now are more compact and clearly separated. Their margins tend to be smoother, especially for those labeled F that are now more compact than in Figure 4a, with less perturbations from the other grades, as a virtual median barrier keeps them apart. However, the class of average grades is still difficult to be separated, and, there has been a trade-off between size and accuracy, since the higher grades are now more compact, but less separated from each other.Nonetheless, if we would combine the two previous interpretations, we may analyse and classify the data better by using both of their strengths, as in each model the data is more or less scattered across the map, which would be of use in cases when we desire a greater confidence on a particular class of grades. While the model from Figure 4a may offer us a better understanding of the students with passing grades (as they belong to a rather compact group), the SOM from Figure 4b may show us an antithetical approach of the highest grades and the lowest ones. Moreover, the SOM model built without using attribute a_3 is particularly good at classifying lower grades with greater accuracy, and, even if there still is noise in the data categorised as E or G, the model can confidently predict the performance of a good student. What makes it so difficult to classify all the students is the fact that there is a discrepancy mainly among the F labeled class, as there is an inconsistent progress for each one, that would result in a more scattered pattern that interferes with better classified data.

Analysing both maps from Figure 2 we also note that most of the students belonging to category F (i.e. having the final grade 4) are, on both maps, well enough delimited from the students from other categories (S, G, E). Overall, we observe as a main pattern that neighboring students belong to near categories (F-S, G-E). But several outliers may be observed on the map: neighboring students having very different categories (e.g. E and F). A possible explanation for such incorrect mappings may be that the data set includes the examination results not only for the normal session, but also for the retake session. Thus, it is very likely to have the same instance but with different final labels (i.e. the categories from the normal and retake session) which may be very different (e.g. F and S). Besides the previously mentioned cause for outliers, another possible one is given by the intrinsic uncertainty of the educational processes. The data set includes instances for which there is a visible uncorrelation between the grades received during the semester and the final examination grade. Such discordances may appear due to a bias evaluation or some unexpected events in the students learning process. To avoid introducing noise in the data set which will affect the performance of

the learning, we will further investigate preprocessing techniques for detecting such outliers.

4.2. **Experiments using RARs.** The results of the RAR mining experiment are further presented, with the aim of highlighting the relevance of the relational association rules mining process in distinguishing between classes of students having different final grade category. For each category $c \in \mathcal{C}$ we present in Figure 5 the set RAR_c of maximal RARs mined from D_c using the experimental methodology from Section 3.2. In addition, we indicate the value of $Prec_c$ which evaluates the quality of the set RAR_c.

c	Length	Rule	Confidence	Prec
E	2	$a_1 > 8$	0.898	0.716
	2	$a_2 > 9$	0.713	
	2	$a_4 > 8$	0.771	
	3	$a_1 \leq a_2 > 8$	0.707	
	3	$a_1 \leq a_3 > 8$	0.650	
	3	$a_1 \leq a_4 > 7$	0.643	
	3	$a_2 \leq a_3 > 9$	0.618	
G	2	$a_1 > 5$	0.87	0.689
	2	$a_1 \leq 8$	0.675	
	2	$a_2 > 7$	0.728	
	2	$a_2 \leq 9$	0.679	
	2	$a_3 > 7$	0.683	
	2	$a_3 \leq 9$	0.630	
	2	$a_4 > 6$	0.630	
	2	$a_4 \leq 8$	0.626	
	3	$a_1 \leq a_2 > 6$	0.614	
	3	$a_1 \leq a_3 > 6$	0.606	

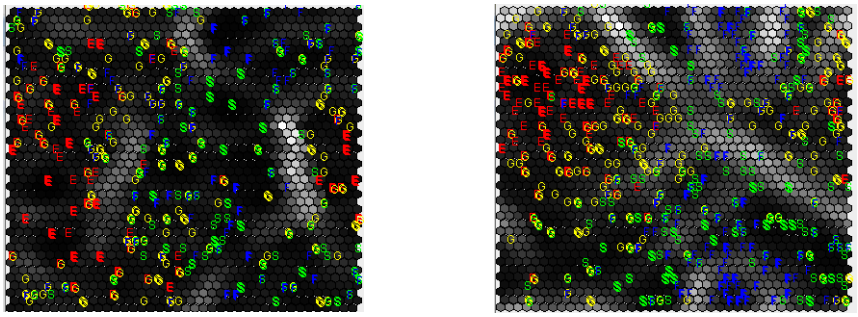
c	Length	Rule	Confidence	Prec
S	2	$a_1 \leq a_3$	0.686	0.69
	2	$a_1 \leq 6$	0.737	
	2	$a_2 \leq 6$	0.661	
	2	$a_3 > 5$	0.623	
	2	$a_3 \leq 7$	0.619	
	3	$a_1 \leq a_2 \leq 9$	0.640	
	3	$a_1 \leq a_3 > a_4$	0.636	
	3	$a_1 > a_4 \leq 5$	0.653	
	3	$a_2 > a_4 \leq 5$	0.640	
	3	$a_3 > a_4 \leq 7$	0.631	
	4	$a_1 \leq a_2 > a_4 \leq 7$	0.619	
F	2	$a_1 \leq a_3$	0.643	0.695
	2	$a_1 \leq 5$	0.624	
	2	$a_2 \leq a_3$	0.654	
	2	$a_2 \leq 5$	0.680	
	2	$a_3 \leq 6$	0.661	
	3	$a_1 > a_4 \leq 5$	0.628	
	3	$a_3 > a_4 \leq 8$	0.617	

FIGURE 5. The sets of maximal interesting RARs mined for each category of grades: E, G, S, F.

From Figure 5 we observe that the sets of RARs characterizing the classes of students with different final grade category are disjoint, in general. For example, the fourth RAR of length three from the left side table indicate that for 61.8% of the students who have received a final examination grade of 9 or 10 (category E), the project score is less than or equal to project status score, which is greater than 9. We note that this RAR does not characterize the other categories of students, thus it is very likely to be useful for discriminating students according to their final category.

For facilitating the interpretation of the RARs, we decided to build a SOM for having a visual representation of the rules and highlighting how well they characterize the classes of students. Let us denote by *Rules* the sequence of all distinct mined RARs given in Figure 5, including all their subrules. If a RAR

of a certain length appears in more than one category, it will appear in *Rules* only once (e.g. the RAR $a_1 \leq a_3$ of length 2 appear as an interesting rule for both categories S and F). Thus, an additional data set D_{RAR} is created by characterizing each student $stud_i$ from the original data set D by a 48-length binary vector $V_i = (v_1^i, v_2^i, \ldots, v_{35}^i)$, where 48 is the size of the sequence *Rules*, i.e the number of distinct RARs mined. An element v_j^i from V_i is set to 1 if the j-th RAR from *Rules* is verified in $stud_i$ and 0 otherwise. Figure 6 depicts the SOMs built on the data set D_{RAR} using all attributes a_1, a_2, a_3, a_4 (left) and using only attributes a_1, a_2, a_4 (right). On each SOMs, the instances are labeled with their final grade category (E, G, S, F).



(A) SOM visualization considering attributes a_1, a_2, a_3, a_4.

(B) SOM visualization considering attributes a_1, a_2, a_4.

FIGURE 6. U-Matrix visualization of the SOM built on the data set D_{RAR} using attributes a_1, a_2, a_3, a_4 (left) and a_1, a_2, a_4 (right).

From the interesting RARs depicted in Figure 5 and visualized in Figure 6a we also observe that there is an overlapping, in general, between the set of RARs characterizing near categories (F/S, G/E). This is expectable, as previously shown in Section 4.1 where the SOM mapping highlighted that there are instances, mainly from near categories, that are hard to discriminate. For instance, the rule $a_1 \leq a_3$ appears for both S and F categories with highly similar confidences (0.686 for S and 0.643 for F). Another example is the rule $a_2 > 7$ from G and $a_2 > 9$ from E which is also explainable due to some instances that are on the border between the two categories. Certainly, such overlapping rules are not useful for discriminating between classes. A post-processing step would be useful for detecting and removing such rules from the mining process and will be further investigated. However, we observe interesting RARs, such as the 4-length rule from the S category, which characterizes only this category of students. On the other hand, the

RARs expressing the E category have the higher precision (0.716) and this is also observable on the SOM, as this category is easily distinguishable from other classes, sustaining the conclusions from Section 4.1.

Regarding the usefulness of attribute a_3 in mining relevant RARs, the following were observed by analysing the RARs depicted in Figure 5 and visualized in Figure 6b. On the one hand, a_3 creates some misleading relationships between the features, such as the rule $a_3 > a_4$, creating overlapping values with other grades (i.e. it is interesting for both F and S categories). That would cause some misclassifying when interpreting border cases, students that are in between two classes. When removing the a_3 feature, the RAR model is improved, as there are less overlapping areas, even though there are not as many relationships between the features (but the number of RARs may be increased by reducing the minimum confidence threshold). This can also be seen in the experiment from Section 4.1, when comparing the two SOMs (built with and without attribute a_3). On the SOM from Figure 6b built without considering a_3 feature, there is a tendency of the higher and lower grades to create distinct clusters that have little or no noise. The class of F labeled instances is more clearly distinguishable in Figure 6b than in Figure 6a.

The results previously presented highlight the potential of the sets RAR_c to differentiate the students according to their final examination grade category, based on their grades received during the academic semester. As previously shown, RARs are able to express interesting patterns in academic data sets and are useful for providing a better insight into the problem of students' academic performance prediction. However, we have a small number of attributes in our case study. By increasing the number of relevant attributes, it is very likely that more informative and meaningful RARs would be mined.

It is worth mentioning that the results obtained using both SOM and RAR models conducted to similar conclusions, which were detailed in Section 4.1 and 4.2. For obtaining a more accurate representation of the input instances (students) using both unsupervised learning models investigated in this paper (SOMs and RARs), the attribute set characterizing the students must be enlarged with other relevant characteristics. It would be useful to have multiple attributes in the mining process and to extend the set of relations used in the mining process in order to obtain much more informative and relevant RARs as well as a better separation using the SOM model.

A more in depth analysis of the outlier instances provided by the SOM and RAR models may provide valuable information regarding the improvement of the educational processes. For instance, the results of the unsupervised learning processes may reveal the following: (1) the examination grades for some of the evaluations received during the academic semester may be incorrect due to

the variations within the instructors evaluation criteria or standards, as well as possible cheating methods used by a few students; (2) some of the partial examinations may be redundant; (3) a change of the computation method for the partial grades may be required; (4) it could be necessary to increase the number of the examinations performed during the academic semester.

5. Conclusions and future work

This paper examined two unsupervised learning models, *self-organizing maps* and *relational association rule mining*, in the context of analysing data sets related to students' academic performance. Experiments performed on a real data set collected from Babeş-Bolyai University, Romania highlighted the potential of unsupervised learning based data mining tools to detect meaningful patterns regarding the academic performance of students.

We may conclude that the grades received by the students during the semester may be relevant in predicting their final performance. However, several outliers were observed in the data set. Such anomalous instances may be due to: (1) a small number of students' evaluations during the semester (attributes); (2) the students' learning process which is not continuous during the academic semester; (3) the difference between the evaluation standards of the instructors from the laboratory and seminar activities. As a consequence, an increased number of evaluations during the academic semester would be useful, for stimulating students to study during the semester and not only for the final examination.

Future work will be performed in order to extend the experiments and the analysis of the obtained results. For increasing the performance of the unsupervised learning process, methods for detecting anomalies and outliers in data will be further investigated. In addition, a post-processing phase for filtering the set of mined RARs will be analysed for removing rules which overlap with multiple classes.

References

[1] Academic data set, 2018. http://www.cs.ubbcluj.ro/∼liana.crivei/AcademicDataSets/ThirdDataSet.txt.

[2] Elizabeth Ayers, Rebecca Nugent, and Nema Dean. A comparison of student skill knowledge estimates. In *Educational Data Mining - EDM 2009, Cordoba, Spain, July 1-3, 2009. Proceedings of the 2nd International Conference on Educational Data Mining.*, pages 1–10, 2009.

[3] Alejandro Bogarín, Rebeca Cerezo, and Cristóbal Romero. A survey on educational process mining. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 8(1), 2018.

[4] Alina Câmpan, Gabriela Şerban, and Andrian Marcus. Relational association rules and error detection. *Studia Universitatis Babes-Bolyai Informatica*, LI(1):31–36, 2006.

[5] Gabriela Czibula, Maria-Iuliana Bocicor, and Istvan Gergely Czibula. Promoter sequences prediction using relational association rule mining. *Evolutionary Bioinformatics*, 8:181–196, 04 2012.
[6] Ashish Dutt, Saeed Aghabozrgi, Maizatul Akmal Binti Ismail, and Hamidreza Mahroeian. Clustering algorithms applied in educational data mining. *Intern. J. of Information and Electronics Engineering*, 5(2):112–116, May 2015.
[7] N. Elfelly, J.-Y. Dieulot, and P. Borne. A neural approach of multimodel representation of complex processes. *International Journal of Computers, Communications & Control*, III(2):149–160, 2008.
[8] Syed Tanveer Jishan, Raisul Islam Rashu, Naheena Haque, and Rashedur M. Rahman. Improving accuracy of students' final grade prediction model using optimal equal width binning and synthetic minority over-sampling technique. *Decision Analytics*, 2(1):1, Mar 2015.
[9] S. Kaski, and T. Kohonen. Exploratory data analysis by the self-organizing map: Structures of welfare and poverty in the world. *Neural Networks in Financial Engineering. Proceedings of the Third International Conference on Neural Networks in the Capital Markets*, pages 498–507, World Scientific, 1996.
[10] S.A. Khadir, K.M. Amanullah, and P.G. Shankar. Student's academic performance analysis using SOM. *International Journal for Scientific Research and Development*, 3(02):1037–1039, 2015.
[11] Wattanapong Kurdthongmee. Utilization of a self organizing map as a tool to study and predict the success of engineering students at Walailak University. *Walailak Journal of Science and Technology*, 5(1):111–123, 2008.
[12] J. Lampinen and E. Oja. Clustering properties of hierarchical self-organizing maps. *Journal of Mathematical Imaging and Vision*, 2(3):261–272, 1992.
[13] Siti Khadijah Mohamad and Zaidatun Tasir. Educational data mining: A review. *Procedia - Social and Behavioral Sciences*, 97:320–324, 2013.
[14] Suhem Parack, Zain Zahid, and Fatima Merchant. Application of data mining in educational databases for predicting academic trends and patterns. *22012 IEEE International Conference on Technology Enhanced Education (ICTEE)*, pages: 1–4, 2012.
[15] K. Saxena, S. Jaloree, R.S. Thakur, and S. Kamley. Self organizing map (SOM) based modelling technique for student academic performance prediction. *Intern. Journal on Future Revolution in Computer Science and Communication Engineering*, 3(9): 115–120, 2017.
[16] Gabriela Şerban, Alina Câmpan, and Istvan Gergely Czibula. A programming interface for finding relational association rules. *International Journal of Computers, Communications & Control*, I(S.):439–444, June 2006.
[17] Panu Somervuo and Teuvo Kohonen. Self-organizing maps and learning vector quantization for feature sequences. *Neural Processing Letters*, 10: 151–159, 1999.
[18] Yi Sun. On quantization error of self-organizing map network. *Neurocomputing*, 34(1-4): 169–193, 2000.

Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu Street, 400084 Cluj-Napoca, Romania
Email address: cgir2476@scs.ubbcluj.ro, liana.crivei@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume LXIV, Number 2, 2019
DOI: 10.24193/subbi.2019.2.04

A COMPARATIVE STUDY OF SOFTWARE ARCHITECTURES IN MOBILE APPLICATIONS

DRAGOŞ DOBREAN AND LAURA DIOŞAN

Abstract. The mobile market grows larger year by year and at the core of those devices, we have the mobile applications that push the technological advancement forward continuously. Due to the increased hardware performance and the popularity of those devices as well as advancements in their operating systems, mobile applications have grown to be complex projects with many dependencies and large teams working on them. As the application becomes bigger and more complex, the problem of choosing the right software architecture arises. This study focuses on an analysis of the most commonly used architectural patterns on mobile applications highlighting their features and flaws. Moreover, it also presents a comparison between them when implementing a medium-sized application. The usage of the appropriate architecture can simplify the work of developers and enable the creation of sustainable applications and the improvement of the software?s capacity to endure and evolve over time.

1. Introduction

The market of operating systems for mobile devices is shared between Android (Google) and iOS (Apple), together they cover over 95% of it [23]. Since they cover so much of the market, those operating systems have to run on both high end devices and low end ones. Mobile applications nowadays have short release cycles, they pack the latest technologies and trends for the high end devices and they also maintain the support for the old or low end devices.

In order to be able to design a software system that can work well and can be implemented efficiently under the given circumstances, there is a need for an architectural solution that matches the scope of the desired system.

There are some architectural patterns pushed by the creators of those platforms; Apple promotes MVC in their iOS framework while Google promotes

Received by the editors: October 22, 2019.

2010 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.11 [**Software engineering**]: Software architectures – *Domain-specific arhitectures*.

Key words and phrases. mobile applications, iOS, software implementation, software architecture, programming techniques.

MVP on Android, but they do not scale well and are not suited for all classes of applications. Moreover, most of the time they are wrongly used resulting in massive classes that heavily violate the single responsibility principle, have low cohesion and high coupling. Those kinds of wrongly designed applications are really hard to test and while their codebase increases, the development time needed for adding new features increases drastically as well. Furthermore, the whole development process becomes laborious as the codebase becomes hard to understand. It usually lacks testing and it is very difficult to implement a new feature or modifying one without breaking another part of the application.

This article will focus on analysing the most common software architectures used on the iOS platform by taking into account criteria like reusability, flexibility, testability, dependency among components, development costs. The strong and weak features of each architecture will be discussed, while answering three important questions, which lay the foundation for this study:

- Q1: Why is the software architecture important in mobile applications?
- Q2: What does good software architecture mean in the context of mobile applications?
- Q3: How can a mobile platform software architecture be analysed and benchmarked?

Section 2 talks about the most commonly used mobile architectures on iOS platform highlighting their strong and weak points; section 3 presents our process of analysing those architectures, while section 4 showcases our findings in regard to the questions above enunciated.

2. Details of patterns & comparative analysis

In [12], one of the first papers that described and compared two presentation patterns for designing mobile application — MVC and Presentation-Abstraction-Control (PAC), the authors have emphasised the conditions facing mobile application and they have concluded that the selection of a particular software pattern for the user interface architecture depends on the class of mobile application.

Another MVC-based architecture, called balanced MVC architecture, has been proposed in [7] for service-based mobile applications. The proposed architecture is aimed to divide the kernel application optimally between the client and the server. Again, the authors have remarked the specificity of the proposed architecture for different types of applications, but no other design patterns have been taken into account and analysed.

A unified architecture model adapted to Android development called Extended MVC has been proposed in [19]. The adaptation regards the specificity of the mobile applications. The authors have tested their approach by considering ten devices of various physical specifications (versions of Android system, screen resolutions, internal storage, CPU, RAM, etc.) and they have concluded that their pattern improves the flexibility of the mobile application (without using some metrics for evaluating it).

In [21] and [22] the authors have surveyed several widely used architectural design patterns (MVC, PAC, HMVC [1], MVP, MVVM) involved in the development of mobile applications. Furthermore, the authors have proposed an MVC-based design pattern particularly adapted to the Android system (called Android Passive MVC) and they have evaluated its quality in terms of maintainability, extensibility and reusability with scenario-based software architecture evaluation method. The authors have remarked (and somehow quantified) the reduced complexity of the mobile application developed by integrating the proposed architecture.

VIPER is another alternative architecture proposed by MutualMobile in [10] and comes as an alternative to Clean Architecture from Android on iOS. In [16] the authors highlight the importance of using specialised architectures rather than the "default" ones and analyse the re-architecting process of Coursera's mobile application, where they have chosen VIPER as their architectural solution.

Other technical surveys about architectural patterns can be identified by taking into account the industrial/technical blogs [24], [20].

2.1. **MVC.** The Model View Controller is one of the most versatile and used software architectural patterns. It has been firstly used in Smalltalk and was later adopted by Objective-C and other programming languages such as Java and Ruby [15], [14], [2]. It is used for developing desktop, web and mobile applications [17].

This pattern is the one that is promoted by Apple with its iOS platform, encouraging the development of the applications that pursue it. Many frameworks available from Apple for development purposes follow this pattern and, when using them, custom objects are required to play an MVC role. However, the simple usage of these frameworks do not guarantee that all MVC principles are respected. Many frameworks available from Apple for development purposes follow this pattern [8] and when using them custom objects are required to play an MVC role.

Apple's MVC is a little bit different from the classic MVC as shown in Figure 1. In the classic MVC implementation, the model communicates with

[1]Hierarchical Model View Controller

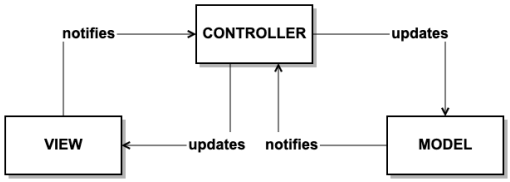


FIGURE 1. Model-View-Controller architectural overview

the view, while in this flavour of MVC the controller acts as a mediator between the model and the view, is responsible for updating the model as well as the view and reacting to notifications from both the model and the view.

The controller has a more active role than in the classic pattern being the bridge between the view and the model. Because this type of object is concerned with how and when to display certain data on the screen and how to react to user interaction it has been named "ViewController". In addition to this, the data and event flow in this flavour of MVC is linear, while in the classic architecture the flow is circular.

2.2. **MVP.** The Model-View-Presenter architecture has been long used in other software development areas not only on mobile platforms. The principles behind this pattern were not designed from scratch and it came as flavour of MVC bringing in some advancements. This pattern can also be adapted to a large set of applications such as client/server or multi-tier applications [13].

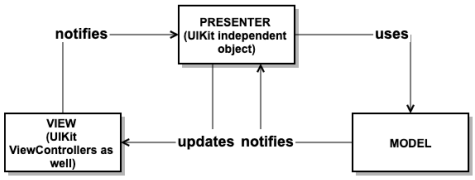


FIGURE 2. Model-View-Presenter architectural overview

The whole pattern is built with the idea that the actions in the application should be driven by the user interaction, by the view layer rather than by the controller. It is composed of three major types of components: the model that handles all the data, the view that takes care of the interaction with the user, the presenter that is responsible for connecting elements, as can be seen in Figure 2.

MVC and MVP look very much alike and the differences are subtle, that is why MVP comes as a flavour of MVC and not as a new concept, both of them being presentational patterns. While they might look alike, there are differences and advantages in using one or another.

In MVP the presenter is responsible for manipulating the views and they communicate through interfaces, the views being decoupled from the presenters and vice versa. In the world of MVC, all the communication between the views and models is done through controllers, the elements are more tightly coupled. The controller receives an event from the view layer, it does some processing, it might manipulate the models and updates the views accordingly. Another difference is the fact that in MVC the views are dumb objects, they do not contain processing code as contrary to what happens in the MVP pattern where the views have to communicate with the presenter.

2.3. **MVVM.** Model-View-View-Model is another architectural pattern from the MV family, which is heavily used on mobile applications. This pattern has gained a lot of attention and has been implemented in many applications because it addresses the problem of massive view controllers [6] and, it also works well with reactive programming [4].

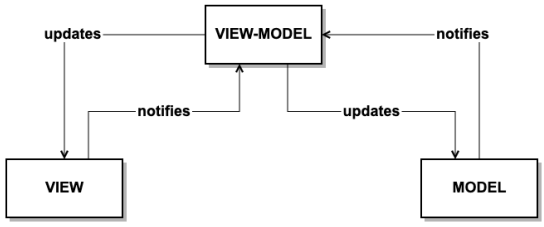


FIGURE 3. Model-View-View-Model architectural overview

MVVM tries to solve some of the problems that might lead to the massive view controllers by using a new layer between the Model and the Controller called View Model as shown in Figure 3. The purpose of this View Model is to take a model object and apply all the transformations and presentation logic to its attributes such that those can be easily presented by the view, for instance, transforming a date into a formatted string. By using this approach the controllers become less bloated with UI configurations and mappings, becoming lighter.

This architectural pattern works very well with reactive programming because the idea behind this architectural concept is that every change done to the model should be automatically reflected in the View through the View Model. The task of propagating this information is easily achieved with the use

of reactive programming or by language features such as Key-Value-Observing [1]. MVVM is also compatible with the MVC as it just adds an extra layer that is responsible for configuring the View by mapping various values and applying some business logic on the Model.

2.4. **VIPER.** VIPER stands for View Interactor Presenter Entity Routing and it is a software architecture used in large mobile applications. VIPER does not come from MV family [10], [16], [5], [18]. As shown in Figure 4 it uses five layers of abstraction for separating concerns in the application. It does that for solving problems that come with using a classical MVC architecture where there is no clear layer where the business logic should be placed. VIPER respects the principles of a Clean Architecture [9] and it can be considered a pattern for the whole application (not only a presenter pattern, like the previously described ones).

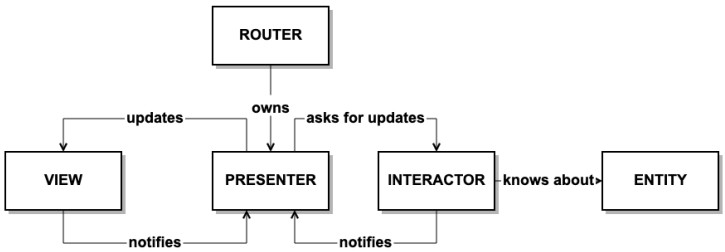


FIGURE 4. VIPER architectural overview

The software systems built using this architecture resemble a game of LEGO. A complete application is built from multiple VIPER modules, the size of those modules depending on the granularity sought. Each component has a well defined and single concern, this architecture is built on the Single Responsibility Principle. The view is only responsible for displaying the items it receives from the presenter.

The presenter works closely with the interactor and prepares the content it receives from the interactor for the view so that this component can display it. The presenter is also responsible for reacting to events from the view and requesting new data from the interactor.

The business logic is contained in the interactor; its responsibility is to manipulate the entity objects. All the logic should be independent on any UI components and all its behaviour should be portable to other platforms.

The entity layer contains the items with which the business logic works and it is related to the model in the MVC. The navigation from one view to another is shared between the presenter and an object which handles the navigation

	MVC	MVP	MVVM	VIPER
Model	Handles data & encodes the model objects			
	Is unaware of anything except itself			
	Handles business logic			Doesn't handle business logic
	Is manipulated by Controller	Is manipulated by Presenter	Is manipulated by ViewModel	Is manipulated by Interactor
View	Presents the interface			
	Handles user input actions			
	Is not Model aware			
	Is passive	Is active	Is passive	Is passive
	Does not have any knowledge about Controller	Has a reference of the Presenter	Has a reference of the ViewModel	Has a reference of the Presenter
Controller / Presenter / ViewModel / Presenter	Handles application navigation			Does not directly handle application navigation
	Contains logic to react to user inputs			
	Updates the views			
	Handles application logic			Does not handle application logic
	Mediator (Model-View)			Mediator (Interactor-View)
	Directly interacts with the Model			Does not interact directly with Model (goes trough Interactor)
	Handles business logic for the view		Contains some business logic (controller logic)	
	Is aware of the View		Knows nothing about the view	Is aware of the View
	Calls methods on the View to notify it to update itself	Calls methods on the View to notify it to update itself	Exposes change events when the state changes	Calls methods on the View to notify it to update itself
	Draws the view trough direct interaction	Drives the view with an interface	Draws the view trough some data binding	Draws the view trough direct interaction
	Tightly coupled with the view	Is decoupled from the view	Not tied to a specific view	Tightly coupled with the view
Interactor				Handles business logic
Router				Handles navigation logic

TABLE 1. Findings after analysis

stack. The presenter is responsible for deciding when to navigate to another view while the routing object is creating the actual transition.

2.5. **Findings after analysis.** We have already seen that the differences among these patterns are relatively small, but they are significant. To better emphasize the most important characteristics of these patterns, we resume them in Table 1.

3. Analysis and benchmark

After the retrospective of the most used software architectures on the iOS platform, we have implemented them in a medium sized iOS application. The application has eight different screens all of which have custom UI components such as lists, buttons, animations, views as shown in Figure 5.

The purpose of the application is to highlight the codebase complexity and the potential problems which might arise from the usage of those architectures in a mobile application. The application is a simple game and one of its most important functional requirements is the ability to send a messages between players, while maintaining a stopwatch which should be synchronised with the

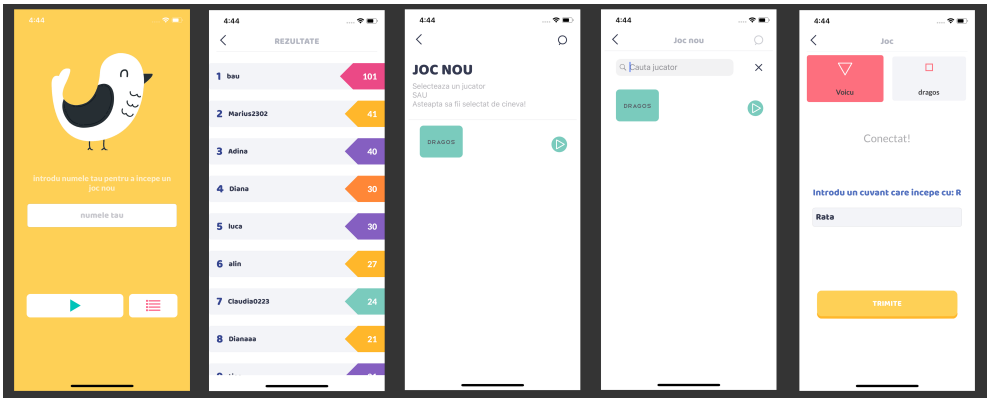


FIGURE 5. Implemented iOS application

one from the peer player. The actual functionality of the application is less important, its choice being made in order to better exemplify the analysed concepts.

After analysing the top 50 free applications from the iOS App Store on various categories, this type of application was chosen as representative for the following reasons:

- it heavily relies on network operations;
- it has a user interface which needs to be adapted constantly and dynamically based on the events its receives from the peer;
- it uses open source libraries which have different architectures and, sometimes, even a different programming language than the ones used in the development of the application.

After implementing the application with all the software architectures described in section 2 we have observed that the MVC is one of the most common architectures as it is easy to follow the pattern imposed by the iOS framework and develop the applications based on the blueprint they provide, as shown in Table 2 (column 2).

However, this can lead to the problem of massive view controllers, as usually developers are not using multiple view controllers for the same user interface page (screen). The main cause of this misusage is that there are not many resources in the literature in which this idea is promoted and most of the beginner developers are unaware of this feature.

In the implemented application we have respected the MVC pattern as described and while we did not produce any massive view controller classes, it was clear for us why the problem might occur. The requirements for the implemented application were clear from the beginning and we have not started

to implement extra features on an old codebase. The introduction of new features and changing the development team can lead to the above stated problem, because it is easier to add a something on top of an fully working system than to refactor it and do things properly, in concordance with its architecture. This is especially true for small features such as adding an extra button or a new label.

The MVP offers another flavour of MVC and, while the problem of massive view controllers is somehow resolved, it is easy to create massive presenters classes just like in the case of MVC. While MVP-based approach is a little better than creating massive view controllers, because those presenters are plain objects and usually do not inherit from a superclass or receive callbacks regarding the user interaction actions as view controllers do, this makes them more testable. However, without proper separation and design, we could easily end up with massive presenters, which take the responsibility of a massive view controller, where we also require extra code for passing the user interaction events from the view to the presenter, which is another task that could introduce bugs and increase the development time.

MVVM comes and adds another layer of abstraction to the classic MVC architecture. It binds the models to the views and vice-versa by using another layer of abstraction, the view-model. This approach takes some of the complexity away from the view controllers. We found out that it felt like we are over-engineering when using this approach for small view components, which have only a label or some minimal information. This approach surely makes sense for components that are complex and they need to display easily large amounts of information or for the views with multiple states; however, for light components the amount of work necessary for implementing it does not always justify its advantage.

Another important aspect of MVVM regards the usage of a third party library (Table 2, column 3), which is usually required for implementing the Observer pattern in complex applications. In the case of iOS, when developing an application in Objective-C, this mechanism was already built in the language, however on Swift we can no longer use this approach without heavily relaying on the Objective-C runtime. That is why most of the applications which use this pattern relay on a third party library for implementing the synchronised behaviour and, thus, adding extra complexity to the overall project.

In addition to this, we can see that the MVVM pattern is the only one that implements the synchronicity between the Model and the View layer (Table 2, column 5) which means that when a change occurs to the Model layer it will be automatically forwarded to the View layer and that usually results in an alternated UI as well.

ARCHITECTURE	Easy of use / follow ↑	Requires extra libraries	No. of layers	Synchronicity between MV layers	Complexity no. of lines ↓	Development costs ↓	Granularity	Strong feature	Weak feature
MVC	*****	NO	3	Async	Base	**	Coarse-grained	simple concept	massive view controllers
MVP	****	NO	3	Async	Base * 1.078	***	Coarse-grained	extra flexibility over MVC	massive classes increased complexity
MVVM	****	YES - usualy	4	Sync	Base * 1.065	****	Coarse-grained	increased testability	complex external dependencies
VIPER	**	NO	5	Async	Base * 1.185	*****	Fine-grained	increased granularity and testability	very complex

TABLE 2. iOS Software architectures comparision (↑ - maximum criterion, ↓ - minimum criterion)

The pattern with the most granularity the ones we have analysed is VIPER. The separation of concerns done in this approach tries to ensure that the architecture of the application will erode at a slower pace (see column 8 of Table 2). It provides the greatest flexibility of all the presented architectures and it also the easier and most testable one. Nonetheless, the flexibility and granularity come at the cost of more code written, more layers and the most complex concepts. We also found out that for applications that have a short lifecycle this approach can be costly from a development perspective, being the most time consuming and the implementation which required the most skilful developers.

In regards to development costs Table 2 (column 7) shows that more commonly used architectures such as MVC or MVP have a smaller development cost than the ones which are more complex such as VIPER or MVVM. While ranking the architectures, we have looked at the following aspects: the architectural skills required by the developers to work on a codebase that implements a certain architecture, the ease of implementing new features (how many layers and components would have to be adapted or created), the cognitive complexity in respects to the layers and the flows of the application and the time needed to develop new features.

The implementation also revealed the complexity of each pattern in the number of lines written for each architecture (see column 6 of Table 2). We took MVC as a baseline, and we have observed that each architecture increases the number of lines of code. MVP showed an increase of 7.8%, MVVM 6.5%, VIPER 18.5%. While those numbers apply to our benchmark application, those could vary depending on the way the model and services layers are written, depending on the number of views and the overall complexity and features of the applications.

Increasing the granularity of the architecture automatically increases the number of layers and dependencies between components (see column 4 of Table

2). This not only introduces more code in order to link those but it also requires a higher level of skills from those who develop the application in order to be able to respect the overall architecture. The cost of granularity is more code written, more interfaces and more classes that are responsible for linking the layers.

The MVP architecture has shown an increase in the number of classes and interfaces from the base application (implemented with MVC). Based on the complexity of the UI elements for every initial View Controller we had at least one extra class and one extra interface, but this number can greatly vary based on the complexity of the application.

MVVM is very similar from this benchmark point of view with MVP; there was an increase of at least one class and one interface from baseline, but as in the case of MVP, this number considerable varies based on the complexity of the application and its architectural granularity. It is not uncommon in large enterprise projects to have more than 5 view-model classes for a View Controller.

In the case of VIPER, we have noticed that for every View Controller from the base application (implemented with MVC) the architecture required at least another 2 classes and 3 interfaces (protocols in Swift).

The number of dependencies increases with the complexity of the application and it is heavily influenced by the chosen architectural pattern. In the application we have benchmarked there were not so many complex views which could be implemented with the MVVM, however in applications with multiple view states and complex UI elements the percentage of code written and the number of dependencies would increase drastically from an MVC baseline point of view.

After the experiment, we have also calculated the Weighted Methods per Class (WMC) and the Coupling Between Objects (CBO) classes for the initial View Controllers of the MVC implemented application in order to reveal the complexity, reusability and the coupling of each architectural pattern. WMC counts the number of methods associated with a class, a high value indicates increased complexity and low reusability [3]. CBO it is a values which indicates the dependencies between classes, counting the relationships between classes without taking inheritance into account. A high CBO value indicates an increased dependency among classes and restricted reusability [3].

The codebases with the lowest WMC and CBO scores represented the ones which allowed a higher flexibility and testability, however they usually have a higher number of layers and components – which means higher development costs. It is important to mention that in the case of large and complex codebases those values are much more important than in the case of small and

medium codebases as they accurately indicate the degree of flexibility, extensibility and testability. In smaller codebases, usually those metrics are not so important since the logic of the applications is less complex and the cognitive complexity of the flows is more easily to comprehend.

In the case of MVC, for one of the most complex View Controller of the application, the WMC was 28 and the CBO value was 2. MVP has shown a drastically decreased value of 14 for WMC and 1 for CBO for the same class, MVVM had its WMC value of 26 and the CBO 2. VIPER has as well shown a decrease in complexity with a WMC of 19 and a CBO value of 1.

The most testable architecture of them is VIPER as it provides great granularity and, with its concept of router classes, the navigation between views can also be unit tested. In the case of the other architectures the navigation between views is harder to test based on the way this is implemented (segues or programmatically modifying the navigation stack). MVVM provides increased testability for the user interaction components over MVC, while the MVP is as well more testable than MVC as presenters are usually plain objects and the interactions with those elements can be manually mocked and the events are not controlled by the iOS SDK.

4. Conclusions

We have analysed some of the most common software architectures used in mobile applications software development on a medium sized application on the iOS platform. The focus was on showcasing the strong and weak features of the evaluated architectural patterns on a real case example. We have considered the architectures from the MV family as these are the ones advocated for by the creators of the mobile operating systems, as well as different flavours of those. We have also included in our research VIPER, which is a relatively new architectural pattern that has gained a lot of popularity on the iOS mobile applications development scene.

The basis for our study was the implementation of the same iOS application with every one of the evaluated software architectures (MVC, MVP, MVVM, VIPER). After the implementation, we have examined each code base from the following points of view: flexibility, testability, dependencies between components and development costs.

After the experiment, we reinforced the assumption formed in years of commercially developing those kinds of applications regarding the importance of the software architectures. Software architecture has a critical role in the lifecycle of a mobile application and can strongly impact the cost of an application.

4.1. **Q1 - Why is the software architecture important in mobile applications?** For most companies that develop mobile applications, the cost is one of the most important factors in developing it. Having that in mind, choosing the right architecture for the application based on the functional requirements and the roadmap of the application can have a strong impact on the overall cost of the project.

For instance, it would not make sense for a proof of concept application (whose lifecycle is only a few months or a year) to be over-engineered. Spending time on implementing an architecture, which will provide flexibility for the future development of the application, has no sense from an economic point of view. By choosing a sophisticated architecture such as MVVM, the code base would increase dramatically and, based on the exact requirements of the application, the number of classes and line of codes could potentially double. In order to achieve those, the team which develops the application would have to be more skilled and the time for development and the cost will be directly proportional to the size of the codebase.

However not all mobile applications have the cost as one of the most important factors in their development. If we think about companies such as Snapchat, Tinder and Uber, all those companies are built around a mobile application and while they provide Web or Desktop applications as well, most of their revenue and user base comes from the mobile platforms. Those companies are not that concerned with the cost of the development and are more concerned with the extensibility of the application, its flexibility to adapt to new technologies, the range of devices on which the application can run, the ability to monitor the way their users interact with the application and to implement A/B testing for new features. They are also more concerned with the security and scalability of their application as well as the ability to provide new and interactive user interfaces and experiences and the ability to easily change these.

For mobile product companies, it makes a lot of sense and it is absolutely mandatory to have a software architecture which helps them achieve all their requirements. Failure to do so at the beginning of the project results in a technical debt which, most of the times, can only be leveraged by rewriting the application, or adding more resources and spending more time and money on the development.

Choosing the right software architecture for the product you are building while it is a hard task and, usually, involves people from all the layers of the company; product team, developers, business analysts etc., it is one of the most important task which you have to achieve and which will pay dividends in the long term. The importance of the task and the results of implementing it is closely related to the purpose of the application and its lifecycle. Nevertheless,

it is important that at the beginning of the development of a product, after a thorough analysis, to make this decision.

4.2. **Q2: What does good software architecture mean in the context of mobile applications?** The purpose of the architecture is to provide a blueprint that is easy to follow and hard to break. It has to constrain the developer so that even the most inexperienced ones have to respect its principle and to avoid architectural erosion [11].

Good mobile software architecture should also be more flexible to change than other software architectures used in other types of software products such as web applications or embedded systems software. The reasons for this are the fact that mobile platforms are in continuous expansion and the field of the mobile application is reshaped every year with new kind of devices which have newer and more powerful hardware or they introduce completely new hardware which allows the developers to add unpredicted functionalities to their applications.

The most important aspect which heavily affects the architecture of an application is the development cost. The cost is influenced by the time of development, the skill of hired developers and the technology stack used. All these elements put their fingerprint on the final architecture of the product, that is why when starting a new project and choosing an architecture, it is really important to see if it is feasible from an economic point of view. Deciding for the wrong kind of architecture and not being able to correctly implement it while also delivering the required functionalities could cripple the project or badly erode the architecture in a way that might make it impossible to deliver new features or keep expanding it without massive refactoring.

4.3. **Q3: How can a mobile platform software architecture be analysed and benchmarked?** Mobile software architectures can be analysed as any other software architecture, from a complexity and granularity point of view, as well as for its flexibility and testability. What we have noticed is the fact that the mobile application architectures erode pretty fast as there are technological advancements in this field every year.

In order to benchmark an architecture, one has to first define the scope of the application, its lifecycle, release cycle and feature set. While there are many architectures that can be used for a certain application, its scope and budget usually dictate what architecture will be chosen. Given the budget and the scope of the application, we can filter the architectures and find out which one best fulfil the purpose.

Taking into consideration the scope of the application, the budget and the lifecycle, we can benchmark the architectures by their degree of flexibility, resistance to erosion, testability and ease of implementing.

Another important aspect in choosing the architecture is the number of its layers. While a greater granularity makes the whole architecture more flexible and testable and allows more developers to work on the same flow without creating conflicts, it also creates the need to write more code in order to link those components. Choosing the degree of granularity is as well as other features of an architecture strongly bounded to the purpose of the application heavily influences the lifecycle as well as the development costs.

The synchronicity between the model and what the user sees on the screen is another aspect that should be taken into consideration when choosing an architecture for a mobile application. There are architectures in which the view is automatically synchronised with the model the latter changes. Usually, the purpose of the application steers this kind of feature and synchronicity. For instance, a stock trading application would require that the model should always be in-sync with the view in order for users to see accurate prices. A photo browser application would not require this kind of synchronicity as it would not make sense from a network consumption and a user interaction point of view to refresh the feed of all the users when a new photo is added.

Nonetheless, choosing the right architecture is only the first step in building an application. The task of implementing an architecture is as important as choosing the right one and the skills of the development team usually affect the final product more than the chosen architecture.

4.4. **Future work.** As the next steps, based on this work, we plan to approach an auto-adaptation of an architectural pattern to the application; this approach should fit both medium sized applications and large, enterprise ones.

References

[1] Apple. Key value observing. https://developer.apple.com/library/archive/ documentation/Cocoa/Conceptual/KeyValueObserving/KeyValueObserving.html, 2018. Accessed date: 2018-06-30.

[2] S. Burbeck. Applications programming in smalltalk-80 (tm): How to use model-view-controller (MVC). *Smalltalk-80 v2*, 5:1–11, 1992.

[3] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

[4] R. Garofalo. *Building enterprise applications with Windows Presentation Foundation and the Model View View Model Pattern*. Microsoft Press, 2011.

[5] J. Gilbert and C. Stoll. Architecting iOS apps with VIPER. https://www.objc.io/issues/13-architecture/viper/, 2014. Accessed date: 2018-04-02.

[6] S. Khanlou. Massive View Controller. http://khanlou.com/2015/12/massive-view-controller/, 2015. Accessed date: 2018-06-30.

[7] H. J. La and S. D. Kim. Balanced MVC architecture for developing service-based mobile applications. In *e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference on*, pages 292–299. IEEE, 2010.

[8] D. Mark, J. LaMarche, and J. Nutting. *More iPhone 3 Development*. Springer, 2010.
[9] R. C. Martin. *Clean architecture: a craftsman's guide to software structure and design*. Prentice Hall Press, 2017.
[10] MutualMobile. Meet VIPER: Mutual mobile's application of clean architecture for iOS apps. https://mutualmobile.com/posts/meet-viper-fast-agile-non-lethal-ios-architecture-framework, 2014. Accessed date: 2018-04-02.
[11] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, 17(4):40–52, 1992.
[12] D. Plakalovic and D. Simic. Applying MVC and PAC patterns in mobile applications. *arXiv preprint arXiv:1001.3489*, 2010.
[13] M. Potel. MVP: Model-view-presenter the taligent programming model for C++ and Java. *Taligent Inc*, page 20, 1996.
[14] T. Reenskaug. The model-view-controller (MVC): its past and present. *University of Oslo Draft*, 2003.
[15] T. M. H. Reenskaug. The original MVC reports. Technical report, Xerox Palo Alto Research Laboratory, PARC, 1979.
[16] F. J. A. Salazar and M. Brambilla. Tailoring software architecture concepts and process for mobile application development. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, pages 21–24. ACM, 2015.
[17] P. Sauter, G. Vögler, G. Specht, and T. Flor. A MVC extension for pervasive multi-client user interfaces. *Personal and Ubiquitous Computing*, 9(2):100–107, 2005.
[18] M. A. Sayed. VIPER design pattern for iOS application development. https://medium.com/@smalam119/viper-design-pattern-for-ios-application-development-7a9703902af6, 2017. Accessed date: 2018-04-02.
[19] F. E. Shahbudin and F.-F. Chua. Design patterns for developing high efficiency mobile application. *Journal of Information Technology & Software Engineering*, 3(3):1, 2013.
[20] A. Sinhal. MVC, MVP and MVVM design pattern. https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad, 2017. Accessed date: 2018-04-02.
[21] K. Sokolova, M. Lemercier, and L. Garcia. Android passive MVC: a novel architecture model for android application development. In *International Conference on Pervasive Patterns and Applications*, pages 7–12, 2013.
[22] K. Sokolova, M. Lemercier, and L. Garcia. Towards high quality mobile applications: Android passive MVC architecture. *International Journal On Advances in Software*, 7(2):123–138, 2014.
[23] Statcounter. Mobile operating system market share worldwide. http://gs.statcounter.com/os-market-share/mobile/worldwide/monthly-201705-201805, 2018. Accessed date: 2018-06-30.
[24] C. Trevino. Flux and presentation architectures. https://blog.atsid.com/flux-and-presentation-architectures-91283f7ef94b, 2016. Accessed date: 2018-04-02.

Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu Street, 400084 Cluj-Napoca, Romania
Email address: dobrean@cs.ubbcluj.ro, lauras@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume LXIV, Number 2, 2019
DOI: 10.24193/subbi.2019.2.05

PREDICTING RELIABILITY OF OBJECT-ORIENTED SYSTEMS USING A NEURAL NETWORK

ALISA BUDUR, CAMELIA ŞERBAN, AND ANDREEA VESCAN

Abstract. One of the most important quality attributes of computer systems is reliability, which addresses the ability of the software to perform its required function under stated conditions for a stated period of time.

The paper aim is twofold. Firstly, the proposed approach explores how to define a metric to qualify the sub-aspects comprised in ISO 25010 regarding reliability as maturity and availability. Secondly, we investigate to what extent the internal structure of the system quantified by the Chidamber and Kemerer (CK) metrics may be used to predict reliability.

The approach for prediction is a feed-forward neural network with back-propagation learning.

The results indicate that CK metrics are promising in predicting reliability using a neural network method.

1. Introduction

Quality of a system can be described by different attributes such as reliability, maintainability, usability, etc. Among these attributes, reliability has an important role because it reveals how stable a system is or, in other words, how often it fails.

The definition of reliability is based exclusively on the software external behaviour, although it is well known that the internal structure has an important impact on a quality attribute such as reliability. For example, a reliable system has a complexity minimized as much as possible. Also, coupling in a reliable system is reduced at maximum because it facilitates testing. Considering this, we can say that the better we assess the internal structure of the system, the more accurate will be the prediction of its external behavior. Several studies [4], [2], [12], [11] reveal that Chidamber and Kemerer (CK) [5]

Received by the editors: October 24, 2019.

2010 *Mathematics Subject Classification.* 68T05,68M15.

1998 *CR Categories and Descriptors.* I.2.6 [**Computing methodologies**]: Artificial Intelligence – *Learning*; D.2.8 [**Software engineering**]: Metrics – *Complexity measures*.

Key words and phrases. Reliability, prediction, neural network.

metrics have a strong impact on software reliability. These metrics are briefly presented in what follows [5].

Definition 1. Depth of Inheritance Tree (DIT) [5] is defined as the length of the longest path of inheritance from a given class to the root of the tree.

Definition 2. Weighted Methods per Class (WMC) [5]metric defined as the sum of the complexity of all methods of a given class. The complexity of a method is the cyclomatic complexity.

Definition 3. Coupling Between Objects *(CBO) [5] for a class c is the number of other classes that are coupled to the class c, namely that Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.*

Definition 4. Response for a Class *(RFC) [5] metric is defined as the total number of methods that can be invoked from that class.*

Definition 5. Lack of Cohesion in Methods *(LCOM) [5] is defined by the difference between the number of method pairs using common instance variables and the number of method pairs that do not use any common variables.*

Definition 6. Number of children of a class (NOC) [5] *is defined as the number of all direct sub-classes of a given class.*

Considering the reliability definition and the fact that the internal structure is very important for a reliable system, the goal of this paper is twofold. Firstly, to compute the reliability attribute based on the software external behavior, i.e number of faults occurred during the testing phase, as well as the number of faults discovered during the usage of the software - for this step, the Quality Model, ISO25010 [10] was considered because it addresses two of the four sub-characteristics related to reliability: *Availability*, *Maturity*, *Fault Tolerance* and *Recoverability*. Secondly, to investigate whether the neural networks can predict reliability attribute based on the previously defined attribute and having as predictors object-oriented design metrics.

The second step is very useful because it allows us to know the reliability quality attribute as early as possible in the development life cycle. This helps because it can suggest what classes are more likely to have bugs and the testing team can focus on testing features that use those classes. In this way, we identify bugs earlier and the cost of fixing them is lower.

The structure of the paper is the following: Section 2 presents the software reliability, Goal Question Metric (GQM) approach and how it is used to determine reliability, neural networks and a short related work section. Section 3 presents in more details how to achieve the two objectives of this paper:

how to compute reliability attribute and then how to predict it using a neural network. Section 4 describes the data sets used for validation, the conducted experiments and obtained results. Finally, the conclusions and future directions are emphasized in Section 5.

2. Setting the context

This section presents the theoretical aspects used in this research investigation, i.e. reliability, the GQM approach to quantify reliability, neural networks and a short related work.

2.1. **Reliability as an important aspect of safety-critical systems.** The official definitions of reliability are: "The ability of the software to perform its required function under stated conditions for a stated period of time" (*IEEE Standard Glossary of Software Engineering Terminology* [17]) and "The probability of failure-free operation of a computer program for a specified period of time in a specified environment" (ANSI [14]).

A safety-critical system is a system whose failure might lead to life loss, financial loss, and/or environmental damage. Many everyday systems can be dangerous for us and therefore, the software architects and developers should design and create systems that are very safe. An important question that raises here is *"How can we test that the system is safe?"*.

The first approach is to prove that there are no faults in it. This can be accomplished using formal mathematical methods in the design and proofs of the design correctness. The disadvantage of this approach is that it works well only for small systems.

The second approach is to accept that mistakes can appear and to consider error prediction methods. This is more generally adopted and can be done by quantifying reliability quality attribute of the system.

2.2. **Goal-Question-Metric approach.** Software metrics are very important in understanding, controlling and improving software quality. Fenton's theory of measurement [9] explains that any measurement must have a well defined goal, but in practice, a lot of measurements are not goal oriented and therefore, the data collected is not useful at all. Goal Question Metric approach [3] was introduced to ensure that the measurements are goal oriented. It has three steps:

(1) Define goals with respect to various points of view: quality, time etc.
(2) Define questions to characterize how the goals defined above could be achieved.
(3) Determine which metrics must be collected in order to answer the above questions.

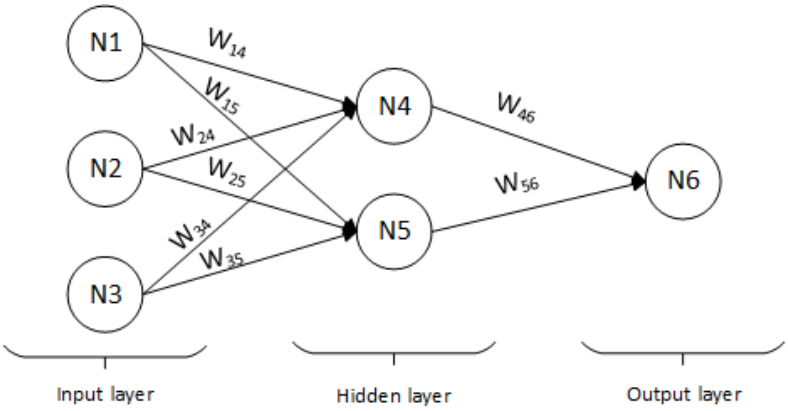


FIGURE 1. Structure of a feed-forward neural network.

2.3. **Neural networks.** A neural network is a supervised learning algorithm [13] that is inspired by biological neural networks. It learns how to perform tasks by taking into consideration already labeled data. For example, if we supply a neural network with weather data from the last month, it will learn how to predict the weather for the next days.

A neural network [13] consists of multiple nodes connected by links. A numeric weight is associated with each link. The neural network communicates with the environment through input and output nodes. A layered feed-forward network is a neural network in which every node is linked only to nodes in the next layer, for example in Figure 1 node $N5$ is linked to node $N6$ but not back to nodes in the previous layers.

Each node performs the following computation: it takes the input values from its input links (for example, input values for node N4 in Figure 1 are N1, n2, N3) and computes a new value (activation value), sending it along each of its output links. The computation consists of two parts. Firstly, it computes the weighted sum of the input values of the node. The weighted sum of a node is the sum of all its input values times their respective weights. Secondly, it computes the activation value of the node by applying the activation function to the weighted sum previously computed.

All the above steps are graphically presented in Figure 2.

Usually, learning in a feed-forward neural network is done using the back-propagation algorithm. Back-propagation learning works in the following way: the network is supplied with inputs and if it computes an output vector that matches the expected output, the algorithm terminates. Otherwise, an error is computed (the difference between the expected output and the actual output),

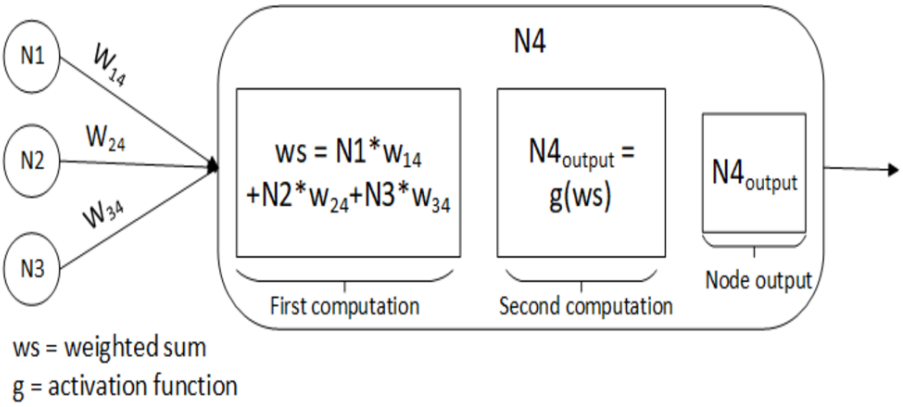


FIGURE 2. Node computation.

```
1  Network NeuralNetworkLearning(trainingData)
2
3    network <- a network with random assigned
4               weights;
5
6    while(prediction is incorrect or
7    termination condition not reached)
8      foreach item in trainingData
9        networkOutput =
10           network.ComputeOutput(item);
11       expectedOutput =
12           item.GetExpectedOutput();
13       UpdateWeightsBasedOn(networkOutput,
14                expectedOutput);
15     end foreach
16   end while
17
18   return network
```

LISTING 1. Generic learning in a neural network

then this error is used to adjust each weight in the network such that the next error to be smaller than the current one. The back-propagation uses gradient-descendant for dividing the error among all weights. The generic learning in a neural network is presented in Listing 1.

2.4. **Related work.** Reliability is one of the most important quality attributes when we describe safety-critical systems. It is so important because a fail in such a system could produce significant losses. This subject was of major interest in last years and several studies investigated its impact on software safety, as well as searched for methods through which we can predict and accomplish a reliability value from the earliest development stages.

How reliability prediction can increase trust in reliability of safety-critical systems was studied in paper [15]. The author determines a prediction model for different reliability measures (remaining failure, maximum failures, total test time required to reach a given number of remaining failures, time to next failure), concluding that they are useful for assuring that software is safe and for determining how long to test a piece of software.

Another approach [6] defined a classifier (with 37 software metrics) and use it to classify the software modules as fault-none or fault-prone. They compared their works with others and concluded that their model has the best performance.

The work described in [8] investigates how to solve the problem of determining the error rate of the electronic parts of a track circuit system (which is a safety critical system) by using Markov chains in order to predict the reliability of the fault-tolerant system.

An approach for assessing and predicting reliability of an object oriented system, taking a statistical approach by using multiple linear regression was proposed in [16].

In relation to existing approaches, ours investigates how we can use CK metrics to predict reliability and relates to approach [6], with the difference that we use CK metrics instead of cyclomatic complexity, decision count, decision density, etc. and we predict a reliability value for each class in the project, instead of classifying the modules in two categories.

3. Proposed approach for predicting reliability

This section presents our approach for reducing the time necessary to compute reliability of a software system. The approach is based on GQM and has the following structure:

- **Goal**: To reduce the time necessary to compute reliability of a software system.
- **Question**: Can the internal structure of a software affect the reliability?
- **Metrics**:
  (1) Collect CK metrics.

(2) Collect bugs (with severity and priority) from testing, operation and maintenance phases for a period of time.
(3) Predict reliability using CK metrics.

The data collected in the measurement phase (i.e. CK metrics and bugs data) are processed in two steps: *Reliability Assessment* and *Reliability prediction*. These two steps are graphicaly detailed in Figure 3.

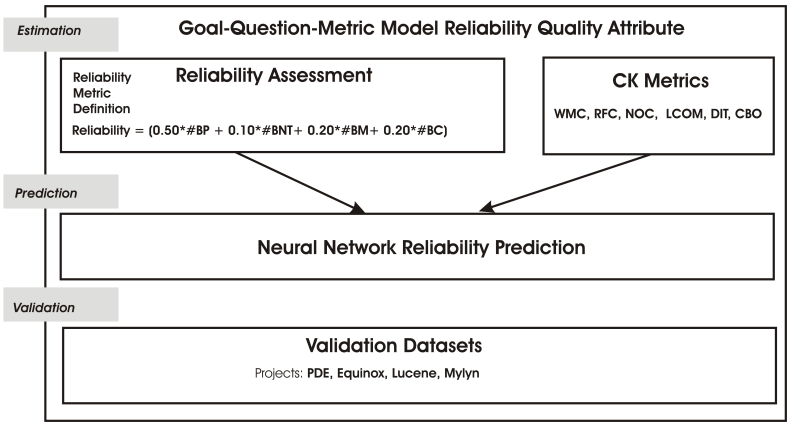


FIGURE 3. The two steps applied in processing the data collected using the GQM approach

The first step, named *Reliability Assessment*, aims to find a formula for computing the reliability quality attribute taking into account the information collected about bugs. It is known that the bugs found the in system are classified by severity and priority in the following way: bugs considered to be priority, bugs being non trivial, bugs considered to be critical and bugs that have a major importance. So, it is known the number in each category of bugs, for each class. For this, the ISO25010 Quality Model [10] was used. Four reliability sub-characteristics are related to this model: *Maturity*, *Availability*, *Fault Tolerance* and *Recoverability*. The main aspects for *Maturity* and *Availability* are related to the post release faults/bugs found in the analyzed system. We claim that these sub-characteristics of reliability could be correlated to bugs discovered during testing and maintenance phase of the development, considering their severity and priority. number of bugs found in the source code of a class. An important remark that should be emphasized here, is that various aspects should be considered to assess the reliability of a class, not only those aspects related to bugs. Finding a perfect metric is a very difficult problem, thus the proposed metric does not claim an equality

relation between bugs and reliability. Others aspects related to reliability can be added to improve the metric effectiveness.

Having this in mind, we establish weights for each of the above four categories of bugs having into account the priority in treating these faults/bugs. Thus, we considered assigning a greater impact for high priority bugs #BHP, major bugs #BM and for critical ones, #BC, with weights of 0.25. Common bugs are the lowest priority and we consider the weights of 0.15 for non-trivial bugs and 0.10 for common bugs. The pairs of (weight, bug category) are the following: {(0.50,#BP), (0.20,#BM), (0.20,#BC), (0.10,#BNT)}, where (#BP) represents the number of bugs viewed as being priority, (#BNT) is the number of bugs considered to be non trivial, (#BM) denotes the number of bugs with a major importance, and (#BC) is number of bugs treated as being critical. The reliability of a class is defined as an aggregate measure by means of Equation 1 that linearly combines the number of different categories of bugs.

$$Reliability = 0.5 * \#BP + 0.10 * \#BNT + 0.2 * \#BM + 0.2 * \#BC \tag{1}$$

The reliability measured in above described way can be only computed in the latest stages of development, when we have a functional piece of software. The goal of the GQM approach is to find a way to compute reliability as early as possible, so in the second step of processing the data collected, named *Reliability Prediction*, we investigate the potential of system's internal structure, expressed by CK metrics, to predict reliability.

To predict reliability, a feed-forward neural network with back-propagation learning is used, with the following structure (see Figure 4): six nodes on the input layer, one node on the output layer and two hidden layers, each of them having four nodes. Each node uses the Bipolar Sigmoid activation function, given by the following formula:

$$g(h_i) = \frac{1 - e^{-h_i}}{1 + e^{-h_i}} \tag{2}$$

This investigation uses a neural network for the following reasons: it has the ability to learn non-linear and complex relationships, after learning it can generalize [13], which means that it shows good results even for unseen data and the way that it works is simple and understandable.

When the algorithm training terminates, we want to know how well the algorithm will work on an unseen dataset. In some cases, it is quite difficult to achieve this because of insufficient data. The concept of cross-validation [7] is used in order to help us to solve this problem.

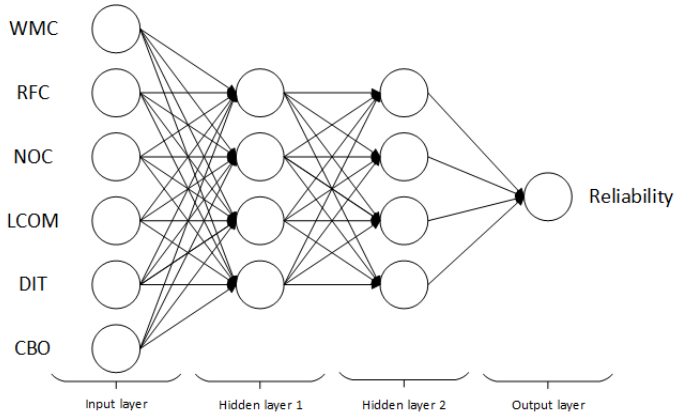


FIGURE 4. Structure of the feed-forward neural network used.

The idea of cross-validation is to put aside a part of the training data and use it later to test the trained model. In this way, the model is validated on an unseen dataset. This technique causes another problem: by removing a part of the training data, we may lose some patterns. In order to solve the second problem, the concept of k-fold cross-validation is used. K-fold cross-validation involves that the entire dataset to be split into k subsets. We will train the model k times and each time we will use a different subset for testing and the rest k-1 subsets for training. The total error of the model will be the average error of all k trials. Our investigation considered splitting the entire data set into ten subsets: nine of them are used for training and the remaining one is used for testing. The model is trained ten times and each time the testing set is changed. That means that each subset is used once for testing and nine times (k-1) for training. Figure 5 presents how cross-validation was applied in our experiments. Black rectangles represent the testing subsets and white rectangles represent the training subsets.

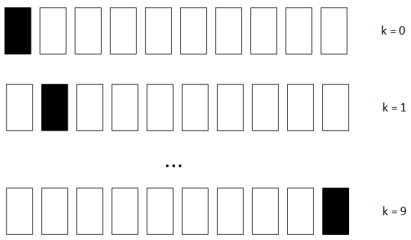


FIGURE 5. Cross-validation process used.

4. Experiments description

This section expose the datasets used to develop the reliability neural network prediction model. In order to validate our model, we used the Root Mean Square Error.

4.1. **Datasets.** The data set used is "Bug prediction dataset" and is described in [1]. For this research, the chosen data are collected from the last version of five different software systems: JDT (Java development tool - release 3.4, version 91), PDE (Plug-in Development Environment - release 3.4.1, version 97), Equinox (release 3.4, version 91), Lucene (release 2.4.0, version 99), and Mylyn (release 3.1, version 98). Table 1 compares the characteristics of each project: JDT includes an index-based search infrastructure used for refactoring, PDE yields solutions for Eclipse plug-ins, Equinox is an implementation of the OSGi R6 framework, Lucene implement an indexing and search technology, and Mylyn is a task management tool for developers.

Table 1. Characteristics of investigated Projects

Metrics	**Characteristics of projects**				
	UI	***Framework***	***Indexing and search***	***Plug-in manag.***	***Task manag.***
JDT	**Y**	**N**	**Y**	**N**	**N**
PDE	**Y**	**N**	**N**	**Y**	**N**
Equinox	**Y**	**Y**	**N**	**N**	**N**
Lucene	**Y**	**N**	**Y**	**N**	**N**
Mylyn	**Y**	**N**	**N**	**N**	**Y**

These data contain CK metrics and number of bugs categorized (with severity and priority) for each class of the system. Data are collected during the testing, operational and maintenance phases.

More information about the number of classes in each project and number of bugs may be visualized in Table 2 (C=number of total classes with no bugs, CB=number of classes with bugs, #B=number of bugs, #BNT=number of bugs Non Trivial, #BM=number of bugs Major, #BC=number of bugs Critical, #BHP=number of bugs High Priority)

4.2. **Experiments methodology.** This investigation used five experiments, using the five projects/datasets. In each experiment, a neural network-based prediction model was trained using 9/10 data from a single dataset (each experiment used a different dataset for training). Each prediction model was

TABLE 2. Data sets information

Metrics	Data sets information						
	$\#C$	$\#CB$	$\#B$	$\#BNT$	$\#BM$	$\#BC$	$\#BHP$
JDT	**44**	**997**	**11605**	**10119**	**1135**	**432**	**459**
PDE	**426**	**1497**	**5803**	**4191**	**362**	**100**	**96**
Equinox	**120**	**324**	**1496**	**1393**	**156**	**71**	**14**
Lucene	**197**	**691**	**1714**	**1714**	**0**	**0**	**0**
Mylyn	**701**	**1862**	**14577**	**6806**	**592**	**235**	**8004**

then validated in two steps. The first step was to validate it using the cross-validation technique, which means that we used for validation the remaining 1/10 data from the dataset that was used for training. The second step was to validate the model using data from the other four projects/datasets. More information about each experiment is listed in Table 3.

TABLE 3. Training and testing data for each experiment

Experiments	Training data	Validation data
Experiment 1	9/10 of JDT	1/10 of JDT, PDE, Equinox, Lucene, Mylyn
Experiment 2	9/10 of PDE	1/10 of PDE, JDT, Equinox, Lucene, Mylyn
Experiment 3	9/10 of Equinox	1/10 of Equinox, JDT, PDE, Lucene, Mylyn
Experiment 4	9/10 of Lucene	1/10 of Lucene, JDT, PDE, Equinox, Mylyn
Experiment 5	9/10 of Mylyn	1/10 of Mylyn, JDT, PDE, Equinox, Lucene

4.3. **Results.** The mean reliability values computed using the bugs-based formula for each project, are listed in Table 4. The mean reliability values computed using the neural network based prediction model for each project and for each experiment are listed in Table 5. The bolded values are obtained in

the cross-validation step, while the others are obtained in the second step of validation. A visual representation of the results is listed in Figure 6.

The experiments explored how the obtained neural network model differs in terms of performance when using projects with different characteristics.

TABLE 4. Mean reliability value for each project computed with the bugs based formula

Projects	**Reliability values by bugs**
JDT	0.048582
PDE	0.023806
Equinox	0.062020
Lucene	0.030623
Mylyn	0.049389

TABLE 5. Mean reliability value for each project predicted by neural network prediction model

Projects	**Reliability values**				
Experiment	***1***	***2***	***3***	***4***	***5***
JDT	**0.046973**	0.026686	0.029756	0.037569	0.068397
PDE	0.054409	**0.020266**	0.046260	0.049615	0.061642
Equinox	0.070716	0.043553	**0.061428**	0.073883	0.118110
Lucene	0.043030	0.0022160	0.031426	**0.025880**	0.066846
Mylyn	0.023523	0.018856	0.015811	0.031991	**0.038243**

To validate our model we use the Root Mean Squared Error (RMSE) metric. It is computed as the square root of the average of squared differences between prediction and actual observation. The metric represents the standard deviation of prediction errors (the residuals).

The RMSE formula is:

$$RMSE = \sqrt{\overline{(f-o)^2}}, \tag{3}$$

Where:
f = forecasts (expected values or unknown results),
o = observed values (known results).

The model is better in its predictions when RMSE is lower, thus the predicted values are close to the observed values.

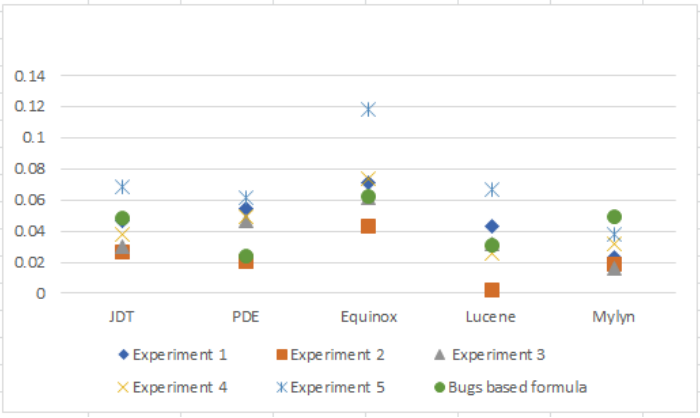


FIGURE 6. Mean reliability value for each project predicted by neural network prediction model.

The RMSE analysis for each experiment is listed in Table 6. Also, a visual representation of the results is listed in Figure 7.

TABLE 6. RMSE analysis for each experiment

Projects	**RMSE**				
Experiment	***1***	***2***	***3***	***4***	***5***
JDT	0.047940	0.075931	0.096312	0.075865	0.090505
PDE	0.060582	0.043147	0.155933	0.063116	0.091675
Equinox	0.113011	0.110130	0.109097	0.097998	0.174369
Lucene	0.068518	0.051839	0.102805	0.043320	0.112893
Mylyn	0.069323	0.072348	0.103535	0.070011	0.063391

Our findings on predicting reliability using CK metrics considering various projects with different characteristics as a basis for the neural network model construction identified best RMSE for the PDE project, thus with UI and plug in management characteristics. Worst value is obtained with Equinox project, thus with UI and Framework characteristics. Overall these findings are in accordance with findings reported in [16] where a statistical approach by using multiple linear regression was used for the same set of data.

5. CONCLUSION

We have proposed in the current paper an approach to measure and investigate reliability of an object oriented system, employing two steps: estimating

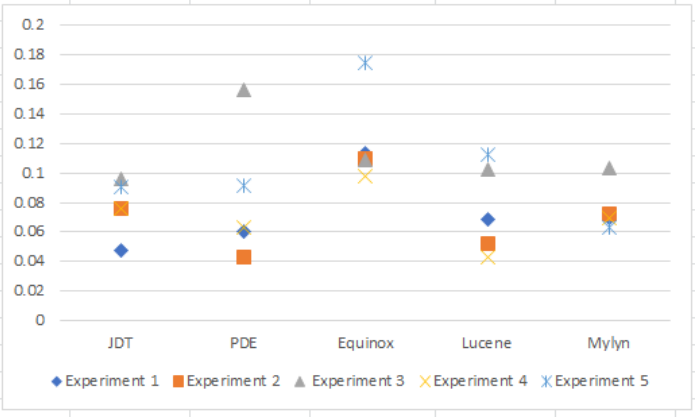


FIGURE 7. RMSE analysis for each experiment.

reliability using the numbers of bugs and predicting reliability using a neural network model based on CK metrics values. The present study confirmed the findings about the relevance and impact of CK metrics to quantify the reliability quality attribute.

The neural network model obtained to predict reliability is validated using a date set containing over 5000 instances/classes, grouped in 5 projects. The experiments revealed that a UI and indexing and search (JDT) project obtain the "best" neural network reliability prediction model.

Future work will investigate the reliability prediction problem by aggregating the results obtained using regression equation and neural networks prediction model. Another future direction refers to apply the equation prediction for other quality attributes.

REFERENCES

[1] M. D'Ambros, M. Lanza, R. Robbes,"An Extensive Comparison of Bug Prediction Approaches",Proceedings of MSR, 2010, pp. 31–41.

[2] V.R. Basili, L.C. Briand, W.L. Melo, *A Validation of Object-Oriented Design Metrics as Quality Indicators.* Technical Report, Univ. of Maryland, 1995. p. 1-24.

[3] V. Basili, D. Rombach. The TAME project: Towards Improvement-Oriented Software Environments. IEEE Transactions on Softw. Engineering, 14(6), jun 1988.

[4] F. Brito e Abreu and W. Melo, *Evaluating the impact of object-oriented design on software quality*, Proceedings Third Int. Software Metrics Symposium, 1996., 90–99

[5] S. R. Chidamber, C. F. Kemerer, *A Metric Suite for Object- Oriented Design*, IEEE Transactions on Software Engineering. 20 (6), 476–493 (1994)

[6] S. Chitra, K. Thiagarajan, M. Rajaram: Data collection and Analysis for the Reliability Prediction and Estimation of a Safety Critical System Using AIRS. International Conference on Computing, Communication and Networking, (2008)

[7] Cross-Validation in Machine Learning, https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f. Last accessed 17 Feb 2019
[8] M. Danhel, Prediction and Analysis of Mission Critical Systems Dependability, PhD Thesis, Faculty of Information Technology, Czech Technical University (2018)
[9] N. Fenton. *Software Measurement: A Necessary Scientific Base*. IEEE Transactions on Softw. Engineering, 20(3), 1994.
[10] ISO25010 description information, https://www.iso.org/standard/35733.html, https://iso25000.com/index.php/en/iso-25000-standards/iso-25010
[11] B. Kitchenham, S. L. Pfleeger, N. E. Fenton, *Towards a Framework for Software Measurement Validation*, IEEE Trans. on Software Engineering, 21(12), 929–944 (1995)
[12] W. Li, S. Henry, Object-oriented metrics that predict maintainability. Journal of Systems and Software, 23(2):111–122, 1993
[13] S. Russel, P. Norvig, : Artificial intelligence: a modern approach. Alan Apt, Englewood Cliffs, New Jersey 07632 (1995)
[14] A. Quyoum, UdM. Din Dar, S.M.K. Quadr: Improving software reliability using software engineering approach—a review. I.J. Comput. Appl. 10(5), 0975– 8887 (2010).
[15] N. Schneidewind: Reliability Modeling for Safety-Critical Software. IEEE Transactions on Reliability 46(1), 88–98 (1997)
[16] C. Serban, A. Vescan, "Predicting Reliability by Severity and Priority of Defects", Proceedings of the 2Nd ACM SIGSOFT International Workshop on Software Qualities and Their Dependencies, 2019, pp. 27–34.
[17] Standards Coordinating Committee of the IEEE Computer Society, IEEE Standard Glossary of Software Engineering Terminology, IEEE-STD-610.12-1990 (1991)

Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu Street, 400084 Cluj-Napoca, Romania

Email address: {camelia, avescan}@cs.ubbcluj.ro, abudur@riasolutionsgroup.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume LXIV, Number 2, 2019
DOI: 10.24193/subbi.2019.2.06

QUANTITATIVE ANALYSIS OF STYLE IN MIHAI EMINESCU'S POETRY

ANAMARIA BRICIU

Abstract. Quantitative stylistic methods aim to express certain aspects of a text in numeric form, thus allowing the introduction of fast, powerful and accurate computational approaches for analysis. While in the case of literature, the validity and usefulness of such studies is highly controversial, one cannot deny the opportunities brought forward by computational methods: first, the exploration of large sets of documents in search of patterns otherwise difficult to discover by human readers; second, the possibility of opening up new perspectives by uncovering latent features of texts. In this study, we investigate the poetic work of one of the most important Romanian poets, Mihai Eminescu, through a variety of quantitative methods addressing lexical, morphological, semantic and emotional aspects of text. We propose a comparison between the results of the computational approach and established interpretations of Eminescu's work in order to assess the viability of computational methods in poetic style studies.

1. Introduction

Computational studies of literary style have enriched the domain of literary criticism by quickly and efficiently analyzing large text corpora, and presenting readers with useful representations and visualizations. Poetry, in particular, has been the subject of a number of recent articles that explore perspectives such as poetic style, computational aesthetics or means of expressions regarding certain topics. The majority of studies, however, explore English and American literature.

In the present paper, we propose a quantitative analysis of style for one of the most famous Romanian poets, Mihai Eminescu, with elements of novelty in the study of the unique relationship between well-defined, unambiguous

Received by the editors: November 15, 2019.

2010 *Mathematics Subject Classification.* 68T10, 68T50.

1998 *CR Categories and Descriptors.* I.2.7 [**Computing Methodologies**]: Artificial Intelligence – *Natural Language Processing*; I.7.m [**Document and Text Processing**]: Miscellaneous.

Key words and phrases. text processing, quantitative analysis, poetry.

statistics and subjective, nuanced interpretations of poems in question. Having a large body of works available for reference with respect to Mihai Eminescu's writing style, we consider this analysis a worthwhile starting point for the investigation of the utility of computational methods in analyzing complex Romanian literature.

Word count methods, or quantitative measures of writing style are seen as features on the surface structure of the literary text that create certain aesthetic effects that provoke a reaction from the reader [13]. They have been widely contested by literary experts for being overly simplistic, artificial, unsubtle and incapable of generating any meaningful results. While it is certainly true that statistical methods lack in nuance, and hardly have the power of capturing artistic expression in its multifaceted form, they must be interpreted as guiding tools in informed literary endeavors rather than methods to search for a ground truth [14]. Moreover, they can be viewed as methods to take advantage of the large amount of digital literary texts available, thus enabling new modes of "reading" and analysis that summarize distinguishing features of very large corpora, something difficult to do in the case of traditional reading. Some experts even argue that computational approaches could bring added rigor and a degree of objectivity to the open question of interpretation and generalization [8]. John Burrows, in a novel study that used stylometry to analyze Jane Austen's works, famously stated that "...it is a truth not generally acknowledged that, in most discussions of works of English fiction, we proceed as if a third, two-fifths, a half of our material were not really there" [3, p. 1]. Consequently, the present computational analysis of style is not aimed at seeking absolute truths about Mihai Eminescu's work or simple, concise answers regarding his style - definitive solutions to the stated problem, as in most algorithmic approaches - but to prove that statistical methods can open up interesting, two-way debates between literary scholars and computer scientists by exploring and describing a representative corpus of Romanian poetry in an efficient and concise way.

This paper is structured as follows. Section 2 provides a brief overview of related works in the field. Section 3 includes a short description of Mihai Eminescu's writing style and the characteristics of each of his creation phases, while Section 4 details the methodology of the study, including information about the dataset used, the resources and tools employed, as well as the challenges met. Section 5 presents and discusses the results obtained, while the last section highlights conclusions and outlines directions for future work.

2. Related work

Quantitative studies of literature have a long history: first modern methods were pioneered in the 1850s, and, since then, the field has known periods of intense activity and evolution, especially in the last decades, when computational approaches were introduced. This includes broad topics, from simple information extraction from literary works to character trait identification and affect modeling in narratives. As far as computational poetry is concerned, there are a series of overlapping research directions: analysis of poetic style, creation of visualization tools, and poetry generation. We will cover the existing works in the first two cases as they relate to the present work.

2.1. **Computational analysis of poetic style.** SPARSAR [5] is a comprehensive system for automatic analysis of poetry style that makes use of computational linguistic methods. The developed system outputs syntactic, semantic and structural information about a poem, as well as affect and phonetic models, ultimately summarizing this data in seven complex indices that allow visual comparison between multiple works. Comparisons between linguistic styles of different poets are made in [10, 11, 22]. Kao and Jurafsky, for instance, examine elements of poetic craft such as imagery, sound devices, emotive language, and diction features to analyze differences between contemporary professional and amateur Imagist poets [10]. Interesting results have also been obtained when poems have been translated into a vector space [11],[22]. Kaplan and Blei represent a poem through a vector of stylistic features that include ortographic, syntactic and phonemic measures, while Zhang and Gao use the sum of the word embedding vectors for the most frequent terms in a poem as its representation. Such vector representations can further be used to either attempt classification, generally with the aim of distinguishing authors [12] or clustering [22].

2.2. **Poetry visualization.** Tools for text visualization can be extremely useful, especially in literary works, where some aspects of the work may not be immediately evident, but might emerge in carefully chosen visual representations.

The tool in [17], for example, offers a wide range of visuals like assessing unique, informative words in each poem, places, time periods, figures of speech, and sentiment expressed in each term and verse.

There are also works that propose similar tasks to ours, namely visualizations of author style through certain periods of time [10, 5]. In the same sense, there are studies that focus on Mihai Eminescu's writing, but they either address different aspects of his style (such as specific semantic units [20]) or target other types of writing, such as journalistic articles [4].

3. Mihai Eminescu's work: Brief Theoretical Overview

Mihai Eminescu was a Romantic poet, novelist, and journalist. He is generally regarded as the most famous and influential Romanian poet, and considered the first modern poet in Romanian literature. His work is unique through the ways of artistic expression that do not necessarily conform to rigid norms, but explore a vast space between the communication of social and political messages and the intrinsic, reflexive state of individual reality transposed in art.

These means of artistic expression are extensively analyzed by literary critics, with entire books dedicated to overviews of Eminescu's poetic style [7] and to his use of language [9]. In this study, we will address poetic style analysis from a computational perspective, investigating quantitative measures of language. In particular, we will study the ability of computational methods to synthesize important facets of the stages in Mihai Eminescu's creation.

3.1. **Linguistic style.** The appeal of Mihai Eminescu's poetic language resides in its novelty and naturalness, which blends folkloric and familiar forms with expressions of high-order, intellectualised language. His entire poetic work is, in actuality, a vast composite model that in many ways transcends literary genres. L. Galdi, in his analysis of Eminescu's poetic style, argues that "a meditation written by Eminescu means more than any other meditation of the era; it comes so far from any epigonism [...] even a teen love poem like "De-aș avea" has personal touches that, from an affect perspective, would be looked for uselessly in some of Alecsandri's works that served as model" [7].

3.2. **Phases of creation.** Most literary critics separate Eminescu's work in three phases of creation, or three poetical and ontological visions. In fact, these phases can be interpreted as three types of realities proposed by the author, each with its own specific themes, motifs and means for poetic expression. We will focus on the three main time periods of 1866-1870 (Phase 1), 1870-1876 (Phase 2) and 1877 and later (Phase 3) [18]. This temporal separation, however, is not a rigid one. Some experts argue in favor of some intermediary, transitional stages [18, p. 433], and contextualize certain poems within a different phase that the date of its creation would recommend it for [18, p. 451].

In this study, we will assess quantitative measures of style in conjunction with the three main stages of Mihai Eminescu's poetic expression, making the membership of a poem to a specific phase group a crisp one, but take into consideration the classification in [18], overwriting automatic assignment of phase based on poem year where it is needed.

3.2.1. *Phase 1.* Chronologically speaking, the first phase refers to poems written between 1866 and 1870. In the majority of the poems written during this phase, a strong influence of the forty-eighters poets can be observed (e.g. Vasile Alecsandri, Ion Heliade Rădulescu, Dimitrie Bolintineanu), both in terms of topic and poem genre. Eminescu writes odes, satires and folklore-inspired poems where he makes use of both folkloric and highly intellectual poetic means, integrating archaic terms with neologisms seamlessly. Moreover, a predilection for comparisons and longer, ornery epithets is observed. This is meaningful in contrast to later works in which the dominant figure of speech is the metaphor, and epithets are simplified in favor of creating sharper, more emotional-heavy visual images.

3.2.2. *Phase 2.* Literary critics approximate the second phase of creation to range from 1870 to 1876. During this stage, the admiring poetic tone from the previous phase is abandoned, the author negating the fabulous, transcendental motifs such as the music and heart of the universe in favor of a more disillusioned, realistic perspective. The present becomes empty of meaning and essence, and there are only two ways in which the poet can battle this realization: first, the return to the idyllic time of childhood, characterized by a series of obsessively referenced nature related motifs (e.g. forest) and the use of verbs in imperfect tense which reflect his nostalgia; and second, a rebellion against the senseless universe. The vocabulary employed to outline such themes is a somber one - there are frequent references to shadows, darkness, blackness, the void, demonic sides, sadness, detachment and alienation, death, and oneiric colors like deep green, navy blue or off-white.

In this phase, the author creates something called compensatory universes, picturesque and pristine poetic worlds that alleviate the pain of living in the real one. There are four ways in which these illusory universes can be created: dreams, love, poetic art and history. Overall, the second phase is characterized by more abstract, suggestive language, in contrast to the rational discourse in the first phase. There are a number of poems in which emotional states are expressed with higher frequency than in the first, works that seem to be deeply personal, and invoke a series of sentiments and emotions: negativity, anger, disgust, sadness, anticipation, love.

3.2.3. *Phase 3.* The last phase of Eminescu's work is defined by a conscious approach to creation in which he abandons the romantic vision. Verses become simpler, poems lack figures of speech but involve more complex structure. At this stage, the main subject of the poems becomes the human condition. This is conveyed through the introduction of terms on both sides of the

"thought/action" semantic pair and temporal references with a similar contrast ("always","infinite" vs. "suddenly"), in the syntax of the use of verbal tenses that suggest an undetermined time (imperfect tense) and verb times that accentuate the pain of the present (indicative present).

4. Methodology

4.1. **Data.** For the data considered, we collected 339 poems from an available online source[1]. Of these, only works excluded were those that could not be definitively associated with a publication year. The number of poems published in each year can be observed in Table 1.

Number of poems per year							
Year	# poems	Year	# poems	Year	# poems	Year	# poems
1866	10	**1872**	14	**1878**	27	**1884**	2
1867	8	**1873**	25	**1879**	36	**1885**	1
1868	4	**1874**	16	**1880**	18	**1886**	2
1869	19	**1875**	5	**1881**	14	**1887**	3
1870	9	**1876**	62	**1882**	12		
1871	12	**1877**	11	**1883**	29		

Table 1. Number of poems per year

4.2. **Tools and Resources.** The tools and resources used in this study will be described in this section.

4.2.1. *RoEmoLex.* RoEmoLex (Romanian Emotion Lexicon) [2, 15, 16] is a resource developed for text-based emotion detection in Romanian language and it contains 9177 terms annotated with eight primary emotions (*Anger, Anticipation, Disgust, Fear, Joy, Sadness, Surprise, Trust*) and two polarity tags (*Positivity, Negativity*). Moreover, each term has part-of-speech information associated. Of the 9177 terms in the lexicon, we take into account only 8486, eliminating multiple word idioms from consideration due to the difficulty of matching highly irregular poetic language to template expressions. We use the RoEmoLex database to compute emotion features for each poem.

4.2.2. *RoWordNet.* RoWordNet (Romanian WordNet) [21, 6] is a semantic network for the Romanian language that mimics Princeton WordNet, a large English lexical database. The basic unit in WordNet is a synset, which expresses a unique concept and contains a number (a "set") of synonym words known as literals. All synsets have additional properties such as part-of-speech,

[1]https://ro.wikisource.org/wiki/Autor:Mihai_Eminescu

definition, conceptual category and relationship information with regard to other synsets (i.e. semantic relations like hypernymy ("is-a"), meronymy ("is-part"), antonymy etc). We use the RoWordNet python API[2] to search for a term's number of senses and relation to other synsets (hypernymy hierarchy).

4.2.3. *NLP Cube.* NLP-Cube [1] is an open source natural language processing framework that supports tasks such as sentence segmentation, tokenization, part-of-speech tagging, lemmatization and dependency parsing for a variety of languages. We use the NLP-Cube python API[3] with a Romanian language model for the following tasks: sentence segmentation, tokenization and part-of-speech tagging.

4.2.4. *UAIC Romanian Noun Phrase Chunker.* The UAIC Romanian Noun Phrase Chunker [19] is a complex tool which recursively detects and annotates noun phrase chunks for Romanian text. Noun phrase (NP) chunking is defined as a partial parsing task that generates an output of the nominal groups in a text, i.e. the units for which the principal word (head) is a noun. We use the web service[4] provided by the UAIC Natural Language Processing Group for our noun phrase related features: average length of noun phrases, number of noun phrases in a poem, types of noun phrases in a poem.

4.3. **Features.** We propose a number of features distributed across semantic, lexical, syntactic and affective perspectives as follows.

(1) **Number of tokens in poem.** Represents the total number of tokens in a poem. This includes stopwords[5] but excludes punctuation.
(2) **Average word length.** Represents the average length of a word in character units. Approximates vocabulary complexity on the simple assumption that longer words are more difficult ones.
(3) **Type-token ratio.** A metric conventionally used as proxy for vocabulary richness [10], defined as the number of unique words in a text divided by the number of all words in the text.
(4) **Hapax Legomena.** Some researchers argue that the type-token ratio is not a sufficient metric for assessing vocabulary richness, and introduce features that count hapax legomena [5]. A hapax legomenon is a word that occurs only once within a context, either in the written record of an entire language, in the works of an author, or in a single text[6]. It speaks to the importance of rare words in a corpus, and

[2]https://github.com/dumitrescustefan/RoWordNet
[3]https://github.com/adobe/NLP-Cube
[4]http://nlptools.info.uaic.ro/WebNpChunkerRo/NpChunkerRoWS?wsdl
[5]very common words in a given language, usually connectives
[6]https://en.wikipedia.org/wiki/Hapax_legomenon

can be used to measure vocabulary growth across different stages of creation [5].

(5) **Part-of-speech density.** The frequencies of parts of speech reflect a poet's mode of discourse [11]. These measures are defined as the number of terms belonging to a certain part-of-speech category divided by the total number of words in the poem. In this study we considered the following categories: nouns, adjectives, verbs, adverbs, and pronouns. For pronouns, we have also investigated the ratio of first, second, and third person pronouns with respect to the total number of terms in the pronoun category for greater specificity. As for verbs, we look at the use of different tenses (present, past, imperfect).

(6) **Abstract and concrete ratios.** Poetry is a type of text dense in imagery and suggestion. It makes sense, then, to define features that assess these aspects [10]. The abstract to concrete feature value is defined as the ratio between the number of abstract concepts and the number of terms referring to concrete terms. To compute this value, we traverse the RoWordNet hypernym hierarchy of each word in a poem [12]. If a hypernym with a semantic category of 'Abstract' is found, then the number of abstract concepts is incremented. Conversely, if we find a hypernym with semantic class of 'Physical' or 'Object', we increment the concreteness count. The ratio is computed by dividing these two values.

(7) **Valence and emotion ratio.** For the two valences and eight emotions for which tags exist in RoEmoLex, we have counted the number of words in each emotion and valence class in each poem and divided it by the length (in words) of the poem.

(8) **Epithets: Noun phrase information.** With the help of the UAIC Romanian Noun Phrase chunker, we compute the number of noun phrases, their average length and the count of different part-of-speech associations that make up a noun phrase for each poem.

4.4. **Archaic terms and unusual word forms.** Mihai Eminescu's poetic language is unique in that it combines literary language with popular one seamlessly. In a given poem, both neologisms and archaic terms can be found - the latter a significant challenge for natural language processing tools developed within the context of contemporary language. The main issue concerns word spellings determined by out of date language rules ("î" instead of "â" inside a word), which, more often than not, are not recognized while processing.

What is more, the author takes many liberties in prefixing and suffixing nouns, adjectives and verbs to suit the desired poetic construction [9]. These

unusual inflectional forms also pose a problem for the processing tools in the sense of lemmatization and part-of-speech tagging. Lastly, we note that some hapax legomena encountered in the text might not be true entities of this kind, due to interchangeable spellings. There may be situations where the tool used was unable to count the different versions of the same word - though in this case, it could be argued that the intentionality in choosing a particular form on the poet's part should not be ignored.

4.5. **Sentence segmentation.** In analysing a corpus of poetry, one must answer the question of the basic unit to be studied: a verse, a sentence or a stanza, taking into consideration the way the poet intended to delineate meaning, which is not always clearly deductible. Additionally challenging is the fact that there may be ambiguous sentence delimitation in such artistic constructs.

4.6. **Word order.** Syntactical parsing of poem content faces the challenge of unusual word order - inversions, repetitions and different constitutive arrangements for elements of a sentence from those in common language are frequent, employed as artistic devices for building musicality of verses, maintaining rhyme and meter, and nuancing the emotional tone of the poem. While they certainly count for aesthetic effect, these specificities of poetic language might lead to incorrect results in syntactic parsing.

4.7. **Word Disambiguation.** In working with tools like RoWordNet or RoEmoLex, which distinguish between word senses, one would wish also identify sense in poem terms for accurate results. However, simple to implement methods like the Lesk algorithm do not yield satisfying results, since the decision is based on surrounding word context. In poems, it is difficult to assess the window size for this context and to obtain any matches, for reasons previously detailed: unusual structural expressions in poetic language and uncommon word forms.

5. Results and Discussion

Stylistically-wise, Mihai Eminescu's work can be separated in three phases of creation, albeit in a slightly fuzzy manner [18]. We have covered theoretical aspects of these stages in Section 3.2. In the following sections, we will present results for a quantitative analysis of the corpus in relation with these three phases of creation.

5.1. **Vocabulary.** We have approached the task of vocabulary examination by taking into account only content words (nouns, adjectives, adverbs, verbs). Table 2 presents the number of poems and the total number of content tokens discovered in each phase.

	# poems	# tokens
Phase 1	44	5843
Phase 2	158	40669
Phase 3	137	20919

TABLE 2. Number of poems and content tokens in creation phases

It can be observed that Phase 1 contains the least poems and smallest number of tokens, while Phase 2 contains the most poems and tokens. It is interesting to note that while the difference in number of poems between Phase 2 and Phase 3 is relatively small, the number of tokens is almost double in Phase 2. This is due to the higher number of long poems (over 2000 tokens) in Phase 2 (9 poems) versus Phase 3 (2 poems).

As far as frequent words are concerned, we take the percentage represented by the total count of a given word in a phase with regards to the total number of tokens in that phase. We find a more frequent occurrence of the words *suflet/soul*, *floare/flower*,*inimă/heart*, *dor/longing*, *amor/love*, *dulce/sweet* and *alb* in the first phase of creation, which references the high number of joyous love poems written in this stage. Conversely, *umbră/shadow* and *negru/black* have a slightly larger presence in the second phase, suggesting darker emotional states and emotionally heavier subjects. For the third phase, we draw attention to the terms *trece/to pass*, *ști/to know* and *lung/long*. In this phase, Mihai Eminescu's poems address the passing of time and the place of a meditative man within a hostile historical context, concepts built upon terms like "to pass", "to know", "long". Lastly, there are terms which we interpret as motifs, such as *lună/moon*, *cer/sky*, *val/wave*, *stea/star*. They appear with the same density throughout the years, under different meanings.

Equally interesting can prove to be the analysis of vocabulary richness. Inspired by the work in [5], we choose three measures to examine this aspect: VR1, equal to the mean type-token ratio for each phase, HA1, the percentage of hapax legomena in a phase with respect to the total number of unique content tokens in that phase, and HA2, the number of hapax legomena unique to the phase in question (i.e. not present as hapax legomena in other phases) divided by the total number of hapax legomena in the phase.

As it can be observed in Table 3, the vocabulary richness measures have similar average scores throughout the three phases: the mean type-token ratio is at around 75% in all stages, while the percentage of hapax legomena with

	VR1	HA1	HA2
Phase 1	0.74	0.67	0.70
Phase 2	0.73	0.61	0.84
Phase 3	0.75	0.63	0.75

TABLE 3. Vocabulary richness measures for each phase

respect to the total number of unique content tokens varies between 60% and 70%. Lowest values are recorded for Phase 2, which is explained by the length of the poems written during these years, considerably greater than in the other phases. However, it must be noted that the second hapax legomena measure (HA2) - the percentage of terms that appear only once in a phase, and only in that phase - is highest for Phase 2. This points to the exploration of the different poetic visions at this creation stage. While in Phase 3, Eminescu's work shows maturity and is more a concentrated synthesis of his ideas, the second phase feels more like a search for answers. The compensatory universes he creates might all have the same aim, but they vary in vocabulary and tone depending on founding idea (dreams, love, history, artistic expression).

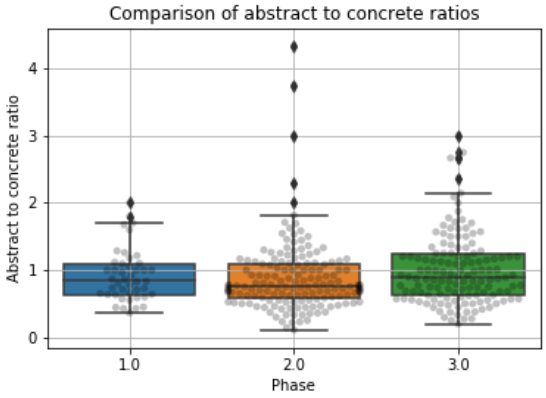


FIGURE 1. Comparison of abstract to concrete ratios in different phases of creation

As for the semantic measure of abstract to concrete concept ratio, we found very small differences between phases, as can be seen in Figure 1. While the median ratio is slightly higher for the third stage, it is a too small difference to draw any definitive conclusions in support of the theoretical observation that third-phase poems approach more abstract topics like *thought versus action*, *human condition* or *the role of the divine*. Abstract terms such as *dream*,

longing, *glory*, *sigh*, *self*, *future*, *flight* are recurrent concepts throughout the whole corpus, with the poet building a number of different meanings around them. Alternatively, concrete references such as *gold*, *wind*, *butterfly*, *sun*, *face*, *hand*, *eye* are symbols and elements of imagery that no phase lacks.

5.2. **Morphology and syntax.** With respect to morphology and syntax, we look at the average part-of-speech densities per phase. For pronouns, we also compare ratio of first, second and third person pronouns with respect to all pronouns, while for verbs, we examine the use of different tenses.

	1st person	2nd person	3rd person
Phase 1	0.17	0.15	0.68
Phase 2	0.23	0.19	0.58
Phase 3	0.25	0.15	0.60

TABLE 4. Average ratios of first, second and third person pronouns per phase

As far as part-of-speech density is concerned, the average ratio of nouns, adjectives, verbs, adverbs and pronouns is almost identical in each phase. Small differences can only be observed in the case of nouns (average of 0.28 in Phase 1 versus 0.25 in Phase 2 and 0.24 in Phase 3), which can be interpreted as a product of the high number of epithets that is present in earlier works. As for pronouns, the number and frequency of third person forms far surpasses that of first and second person uses, even though there is a high number of love poems where the relationship between the poet and the loved one is built on a "I"-"you" linguistic relation [9]. As it can be seen in Table 4, the distribution of percentages differs slightly in the first phase, which can be attributed to the more personal type of poetry of the later phases, as opposed to an overwhelming amount of scenery descriptions in the earlier works.

For verbs, results match theoretical observations as well, but in a similarly subtle manner. Overall, verbs in the present tense are the most frequent (60%), followed by forms in past tense (9-15%) and imperfect (3-7%), in that order. The highest average ratio of imperfect forms can be found in Phase 2 (7%), which suggests the idea of undetermined time, that of the idyllic, compensatory worlds Eminescu constructs in this phase, while past tense can be found most often in the first phase (15%), which may be explained by the fact that this stage of creation contains a number of odes and history-themed poems.

Therefore, in the case of morphological and syntactical features, results are often less conclusive, and more challenging to interpret. However, some subtleties of the author's writing can be translated even in a quantitative

analysis, as shown by the differences between first and third person pronouns in early and later phases, and the values obtained in the case of verbs at imperfect tense.

5.3. **Emotions.** Analysis of emotions in Mihai Eminescu's work with respect to phases yields no clear emotional profile for a creation stage, with mean percentages of emotion almost identical throughout. However, there are some interesting results with regards to the maximum value for percentage of emotion in each phase, as it is shown in Table 5.

	Anger	Anticipation	Fear	Disgust	Joy	Sadness
Phase 1	0.08	0.17	0.13	0.04	0.14	0.14
Phase 2	**0.17**	0.18	**0.17**	**0.09**	0.16	**0.22**
Phase 3	0.13	**0.30**	0.21	0.08	**0.23**	0.18

TABLE 5. Maximum value for percentage of emotion in each phase

The poem with highest percentage of Anger is *"Venin și farmec"* (Phase 2), while *"E ceasul cel de taină"* (Phase 3) has both the highest percentage of Anticipation and of Fear. As for Disgust and Sadness, the highest percentages can be found in Phase 2, in the short poem *"De ce mă-ndrept ș-acum"*, and *"Îngere palid"*, respectively. Finally, the maximum value for the emotion Joy is obtained for a Christmas poem in Phase 3, named *"Colinde, colinde"*.

In our analysis, emotion and valence as measured by frequency of occurrence do not seem to be distinguishing features for the three phases of creation in Mihai Eminescu's work. Therefore, we propose extending the investigation into this aspect in future works by considering trajectories of emotion in poems and other finer-grained emotional features.

5.4. **Stylistic devices.** As far as figures of speech like epithets are concerned, on a simple count of their occurrence, we find that the phase with most such stylistic devices is Phase 2. However, this result should be interpreted in the context of poem length at this stage, given that the number of figures of speech is proportional to the length of the poem. A more informative measure may be the average length of noun phrases, which is highest in the first phase and recording a decrease in later stages, from 2.3 words to under 2. This is in line with the observed presence of long, ornery epithets in Phase 1, and a simplification of poetic expression at stages of maturity.

Finally, we examine the five most common types of noun phrases in each phase, shown in Table 6. They are the same in each phase, but the hierarchy differs slightly. For instance, the most frequent composite phrase in each phase is made up of two nouns (N+N), but a noun followed by a pronoun (N+P) is second most common only in Phase 1.

	N+N	N+P	N+ADJ	ADJ+N	N+ADP+N	ART+N
Phase 1	207	159	156	60	122	97
Phase 2	1370	889	1097	391	916	673
Phase 3	691	383	472	178	480	332

TABLE 6. Most common types of noun phrases in each phase

What is interesting to notice is that there is a high number of nouns followed by adjectives (N+ADJ) in all phases, and, if we also consider the reversed order (ADJ+N), types which are closest to a formal definition of epithets, the total surpasses any other values for Phase 1 and 2. The exception is Phase 3, where a phrase consisting of two nouns is still more common than one with an adjective and a noun. There is also a considerable amount of nouns linked by adpositions (N+ADP+N), especially in later phases. While using adjectives in proximity to nouns is a device used for characterization and intrinsic poetic expression, the second type, which incorporates adpositions, has a more functional role in building and structuring imagery.

6. Conclusions

In this paper, we have presented a computational analysis of a representative Romanian poetry corpus, the poetic work of Mihai Eminescu. We have examined a series of features addressing vocabulary richness, language complexity and emotional content in each phase of artistic creation. Results show that for a series of measures, theoretical observations from the literary criticism field find correspondence in quantitative analysis, indicating that this approach, and visualization of results, in particular, could be used as support tool for interpretation.

However, while computational analysis of literary works and especially poetry is a promising research direction, there are a few challenges to be considered. First, the choice of features must be informed by the domain to be truly relevant, and so must interpretations. The latter, in particular, require knowledge from both the literary linguistics domain, and a familiarity with the considered author's body of work. This is the reason we mark the task of making visualization of results public for future work; we consider that by inviting debate from informed readers of poetry, new, interesting interpretations could arise, or, on the contrary, features taken into account could draw attention to aspects not considered before.

Finally, the original contribution of this paper to the field was an in-depth quantitative analysis of the works of Mihai Eminescu and a close examination of the relationship between the phases of the author's artistic expression

and quantifiable aspects of language. In particular, the emotional and valence features we defined are unique to this study, and while results in this regard were not very conclusive with respect to considered task, we propose two research directions to further examine this exciting aspect: first, the standalone exploration of emotion in the corpus, and second, finer-grained measures that capture the nuances of expression throughout a poem, such as trajectories of emotion in a text.

References

[1] T. Boroș, S. D. Dumitrescu, and R. Burtica. NLP-cube: End-to-end raw text processing with neural networks. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 171–179, Brussels, Belgium, October 2018. Association for Computational Linguistics.

[2] A. Briciu and M. Lupea. RoEmolex - a Romanian Emotion Lexicon. *Studia Universitatis Babeș-Bolyai Informatica*, 62(2):45–56, 2017.

[3] J. F. Burrows. *Computation into criticism: A study of Jane Austen's novels and an experiment in method*. Clarendon Press, 1987.

[4] M. Dascalu, D. Gîfu, and S. Trausan-Matu. What makes your writing style unique? Significant differences between two famous Romanian orators. In *International Conference on Computational Collective Intelligence*, pages 143–152. Springer International Publishing, 2016.

[5] R. Delmonte. Computing poetry style. In *Proceeding of Emotion and Sentiment in Social and Expressive Media: Approaches and perspectives from AI (ESSEM 2013), CEUR Workshop*, pages 148–155, 2013.

[6] S. D. Dumitrescu, A. M. Avram, L. Morogan, and S.-A. Toma. RoWordNet–a Python API for the Romanian WordNet. In *2018 10th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–6. IEEE, 2018.

[7] L. Gáldi. *Stilul poetic al lui Mihai Eminescu*. Editura Academiei Republicii Populare Române, 1964.

[8] A. Hammond, J. Brooke, and G. Hirst. A tale of two cultures: Bringing literary analysis and computational linguistics together. In *Proceedings of the Workshop on Computational Linguistics for Literature*, pages 1–8, 2013.

[9] D. Irimia. *Limbajul poetic eminescian*. Junimea, 1979.

[10] J. T. Kao and D. Jurafsky. A computational analysis of poetic style. In *LiLT (Linguistic Issues in Language Technology)*, volume 12, pages 1–33, 2015.

[11] D. M. Kaplan and D. M. Blei. A computational approach to style in American poetry. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 553–558. IEEE, 2007.

[12] V. Kesarwani. *Automatic Poetry Classification Using Natural Language Processing*. PhD thesis, University of Ottawa, 2018.

[13] M. Kestemont and L. Herman. Can machines read (literature)? *Umanistica Digitale*, 3(5), 2019.

[14] M. G. Kirschenbaum. The remaking of reading: Data mining and the digital humanities. In *The National Science Foundation Symposium on Next Generation of Data Mining and Cyber-Enabled Discovery for Innovation, Baltimore, MD*, 2007.

[15] M. Lupea and A. Briciu. Formal Concept Analysis of a Romanian Emotion Lexicon. In *13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 111–118, 2017.
[16] M. Lupea and A. Briciu. Studying emotions in Romanian words using Formal Concept Analysis. *Computer Speech & Language*, 57:128 – 145, 2019.
[17] L. Meneses and R. Furuta. Visualizing poetry: Tools for critical analysis. *The Journal of the Initiative for Digital Humanities, Media, and Culture*, 3(1):1–14.
[18] I. E. Petrescu. *Studii eminesciene*. Casa Cărții de Știință, 2007.
[19] R. Simionescu. Romanian deep noun phrase chunking using Graphical Grammar Studio. In *Proceedings of the 8th International Conference Linguistic Resources And Tools For Processing Of The Romanian Language*, pages 135–143, 2012.
[20] D. Tatar, M. Lupea, E. Kapetanios, and G. Altmann. Hreb-like analysis of Eminescu's poems. *Glottometrics*, 28:37–56, 2014.
[21] D. Tufiș and V. Barbu Mititelu. *The Lexical Ontology for Romanian*, volume 48 of *Text, Speech and Language Technology*, pages 491–504. Springer, 2014.
[22] L. Zhang and J. Gao. A comparative study to understanding about poetics based on natural language processing. *Open Journal of Modern Linguistics*, 7:229–237, 2017.

Babeş-Bolyai University, Department of Computer Science, 1 M. Kogălniceanu Street, 400084 Cluj-Napoca, Romania
Email address: anamaria.briciu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume LXIV, Number 2, 2019
DOI: 10.24193/subbi.2019.2.07

AUTOMATION AND GAMIFICATION OF COMPUTER SCIENCE STUDY

IMRE ZSIGMOND

Abstract. A large part of the job of a university teacher in computer science is to verify student exercises. The task of verifying correctness, coding style, conventions, and grading is laborious and if done correctly leaves little time for questions. Automating aspects of this work helps all parties involved by freeing time up for questions. The solution detailed in the paper automates a number of these tasks and provides near instant feedback for the students, together with a platform to gamify student learning.

1. Introduction

Computer science teachers at a university level face laborious work verifying the correctness, coding style, conventions, and predefined technical details of student exercises [2]. Very little time is left to answer questions, help struggling students, to tackle complex coding situations, and adapt new teaching techniques. In addition, the institution needs experienced and trained personnel to do the job. Extra care must also be taken to check for plagiarism. All the while, the study of computer science lends itself to automation in core areas.

In the last decade there has been increased research into the gamification of learning, as a response to the ever-increasing challenge to motivate students and increase their engagement [14]. The current best accepted definition of gamification is, the use of game design elements in non-game contexts [4]. Literature reviews suggest that these techniques tend to have a positive effect most of the time [9]. Some of these elements in their turn come from

Received by the editors: November 15, 2019.

2010 *Mathematics Subject Classification.* 68Q60.

1998 *CR Categories and Descriptors.* D.2.11 [**Software engineering**]: Software Architectures – *Domain-specific architectures*; D.2.4 [**Software engineering**]: Software/Program Verification – *Model checking*.

Key words and phrases. system architecture, education, automated verification, gamification .

psychology, entertainment [18]. The goal is to maximize engagement through capturing the interest of learners, and inspiring them to continue learning.

If parts of the work arguably should be automated and there is an increasing need for gamification elements, the question remains how much to automate? how many game design elements to add? and how to imbue the two?

In the solution detailed in the paper the approach to these questions was to make an assignment validator platform with configurability and expandability in mind. The various options and the easy expandability allow for fine tuning of features for a given course material. The platform also serves as a gamification workbench to aid experimentation.

This paper is organized as follows: related work on assignment automation and gamification in section 2, main contribution in section 3, conclusion in section 4, discussion in section 5 and bibliography in section 6.

2. Related work

Automated grading of computer science assignments at college level goes back to 1965 [8]. Back then it was used for numerical analysis courses, written in BALGOL a dialect of ALGOL, and handed in on punch cards. A decade later physicists still use porta-punch cards but assignments are generated randomly and evaluated in batch [12]. A decade after that we see automated grading of multiple-choice tests in [17]. By the early 1990's complex assignment graders for programming and Matlab are developed and used [19]. As late 2000's arrive we have student websites graded with the help of Ruby scripts, as well as requirement assignment, although it relies on exact HTML naming on student's side [5]. The base problem is still being researched and different solutions are being developed [15].

JFDI Academy's solution is approaching ours, they were able to provide students with timely feedback. It had auto-grading, in which once a student submitted the answer to a question, (s)he was automatically given feedback on whether it was correct. Instructors were also able to receive feedback on a student's progress. Students also had the opportunity to raise questions or concerns regarding the assignment, by posting comments which enabled instructors to help them in a timely manner [3].

In parallel, although much recently gamification experiments are being conducted on students in various settings. For example, the promotion of risk taking in language learning [10]. Sometimes the two are combined, for example in the case of [6] a computer engineering master's level college course was gamified with automatic grading. This resulted in forum engagement increase of 511% to 845% versus the previous year.

3. GamifyCS

3.1. **Solution description.** The question of, how much teaching assisting tools, as well as gamification elements to add, is a non-trivial one. The answer tends to depend on the needs of various courses and the aims of the professors teaching said courses. With the goal to provide an automating tool and gamification workbench we identified the following priorities to implement first: individualized exercise assignment, correctness check, semi-automated anti-plagiarism test, near instant feedback, story elements. These features allowed for large scale experimentation, and more time helping students, the details of which will be presented in a future paper. The following features were prototyped and remain to be used in the next semester: fully-automated anti-plagiarism test, static code analysis checks, achievements. While the prototypes worked a subset of them were prioritized to experimentally validate them before more were added.

While the tool is both an assignment validator and a gamification workbench, the core validation is detailed in Figure 1, and there is a more technical view in the next section. While the processes shown are all optional, they do represent the typical needs of a teacher. The process in general starts with an upload on the student's part, the archive is saved locally, extracted, cleaned, and repackaged for MOSS. [1]. If the language warrants compilation the sources are compiled and linked to executables. Then separate instances are run for each predefined test. Test scenarios are evaluated, and the results are saved to be displayed later. Meanwhile on the coding style side StyleCop analyzer is invoked, the resulting report being saved to the database. In parallel the repackaged source files are uploaded to MOSS. for anti-plagiarism checks, and the results saved to the database.

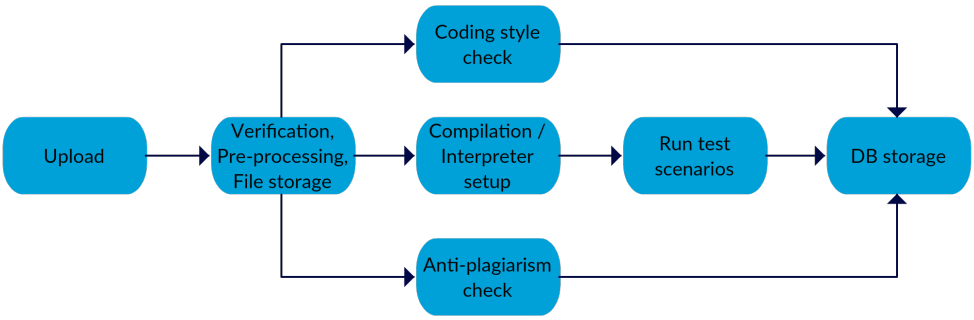


Figure 1. Validation flow

For this tool to be useful it had to support multiple programming languages, so it can be applied for many courses and needs. Supporting any number of programming languages is quite a challenge in itself. The solution was to be language agnostic from the code's point of view and just swap in different modules that had the job of dealing with specific programming language features. The module can decide to try to resolve the specifics with code, or calling external tools. Thus, support for any new language has to be coded in separately.

Correctness check for any given problem is its own separate branch of computer science, and entirely out of scope for this paper and solution. To help with solving this problem each assignment has a well-defined input and output sequence. This sequence is run on each hand-in of a specific assignment, thus testing it. This limits the type of assignments that can be tested though. It is an acceptable first step since a significant number of assignments, in an academic setting have console-based UIs. The proposed solution is useful for full courses, and ultimately can be expanded for more specialized correctness checks, while retaining their other benefits.

Static code analysis of programming assignments has been the focus of computer science research, all the while the specifics of code quality is not a universally agreed upon subject [13]. The debate becomes more varied when different languages are concerned, and it also changes as time goes on. Even on a principle level there are various disagreements, and few universal rules are agreed by most people. For example, memory leaks are universally considered bad while variable naming changes wildly even within one language. What rules to require remains the discretion of the teacher of a specific course.

Plagiarism check is one of the basic functionalities that is also one that cannot be completely automated [11]. An example of technical difficulty is the project files that are generated by their IDE, that should not be checked. The tool sometimes leads to false positives which would be detrimental to learning if students were falsely accused. There are many special cases and situations in practice that would lead to false positives. Our solution was to take the reports generated by Stanford's MOSS API and check them manually. It should be mentioned that sometimes teachers can and do discover certain forms of plagiarism. While software based solutions may not work better then experienced staff, together with the increased time per student, because of the lack of need to check for correctness, increases the change for cheaters being caught.

3.2. **Architecture.** For an illustration of the architectural decisions a UML communication diagram is used in Figure 2. The main application is a website developed using ASP.NET MVC in C#. It runs on a windows server with an

SQL Server installed locally. Structurally it can be considered a standard MVC site, at around 14000+ lines of code. Logging was twofold, with various activities either saved to the database or to logfiles. The hub for the more interesting parts is the TestRunnerService. As mentioned in section 3.1 the process of verification starts with an upload on the student's part, the archive of each attempt is saved locally to a unique path, then extracted, cleaned of non-source files, and repackaged for the MOSS API's convenience.

From this point on the verification thread, source code is compiled and linked if compilable, or supplied to the interpreter otherwise. Separate instances of the resulting executable are run for each test to ensure clean runs. Communication with the executable on StdIO redirects had to be run on separate threads. Then test scenarios are evaluated, the instances closed, and results are saved to be displayed later. The running executables are monitored by a windows service called Monitor Service. The reason for a separate service is that in certain scenarios the executables would not terminate from the context of the verification thread, and remain active in the system. Such programs if caught in an infinite loop would quickly consume large amounts of processor resources, and had to be killed externally. The monitor service does just that by comparing process StartTimes to local time, for running processes of a given name.

On the coding style thread, the StyleCop analyzer is invoked asynchronously, the resulting report being saved to the database, and displayed as part of the test results. Meanwhile on the anti-plagiarism thread the repackaged source files are uploaded to the MOSS API, and the results saved to the database.

3.3. **Features.** Compilation of the uploaded project was one of the central issues of automation on the project while supporting several potential languages. External tools were necessary because real life programming languages evolve as do their compilers. The solution decides how to prepare the executables based on the language in question. The switch is done through the strategy design pattern. Supporting C# required only .NET framework calls and it produced the required executable from code. Python being an interpreted language required the execution of the interpreter with the " -i + pathToSource" supplied as parameters. The resulting execution was the target for testing. By far the most complicated was support for C++, where minor differences between compiler parameters between student side compilation and sever side compilation lead to subtle but significant differences in test output. After much experimentation the following parameters yielded the best results:

```
"cl /permissive- /GS- /analyze- /w /Zc:wchar_t /ZI /Gm- /Od /sdl
/Zc:inline /fp:precise /D \"WIN32\" /D \"_DEBUG\" /D \"_CONSOLE\"
/D \"_UNICODE\" /D \"UNICODE\" /D \"_CRT_SECURE_NO_WARNINGS\"
```

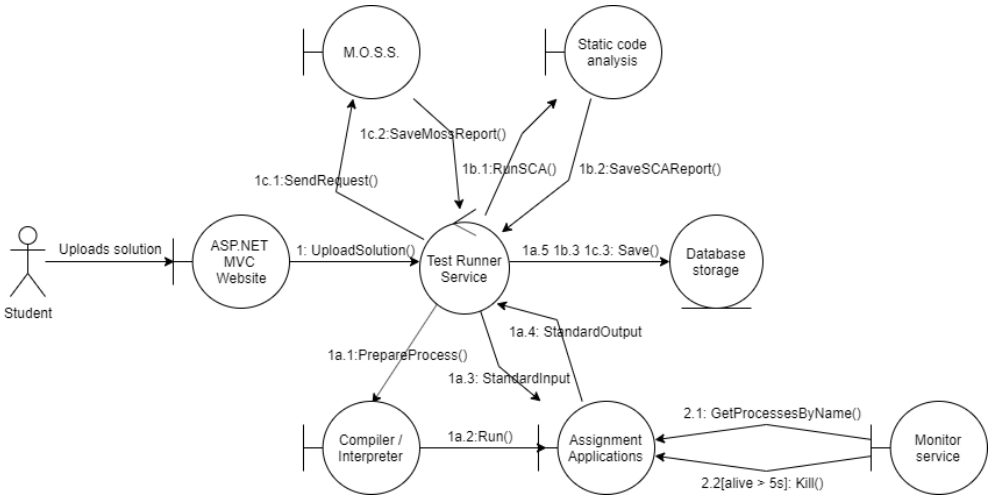


FIGURE 2. Communication diagram

```
/errorReport:none /WX- /Zc:forScope /RTC1 /Gd /Oy- /MDd /FC /EHsc
/nologo /diagnostics:column [listOfFullPathFilesInProject]
/Fa.\\ /Fe:.\\program.exe /Fo.\\"
```

When executing the compiled code, the main application redirected the standard input and output of the assignment to the main program, which in turn fed, then read the predefined sequences. While redirection is a just a flag at startup, the two being streams separate threads needed to be started to communicate with the new process. To facilitate the situations where the program being tested stopped to respond or hang in an infinite loop, timeouts on the threads were used. In addition to timeouts the new process's state was monitored both from the testing program itself and a separate windows service. The monitor service was to query running processes with certain names and terminate those with more than 5 s runtimes. To give more leeway on correctness checks for the students, regular expressions were used by default on their output that has been compared to the expected output on the I/O test. An unexpected and welcome side effect was that students could upload their solutions multiple times during the practical exams serving as working, testable backups in case of software/hardware failure.

To illustrate with an example: one of the assignments had among other requirements the need to catalogue old maps, the "add" and "display" console commands were specified to take the form:

add mapCatalogueNumber, stateOfDeterioration, type, yearsOfStorage
list

One test aiming to check the "add" command with a valid input, started with sending the application: "add 1234, new, political, 15", then sent: "list", finally the regex that ran against the output was: ".*(1234)*.*new.*political.*15".

Code quality checks were prototyped with StyleCop an open source static code analysis tool which has been used to success with .NET projects [16]. The StyleCop.Analyzers project was used for the analysis. It may be called with a configurable list of rules and it generates a report of violations. Example of a rule would be: "FieldNamesMustBeginWithLowerCaseLetter" the violation of this rule happens when the name of a field in a class begins with an upper-case letter. The various rules can be configured per course.

User interface was specifically designed to be minimalist and responsive. The goal was to remove distractions and appeal to current generation's standards. It also focused the work to deliver higher quality rather than more features. The UI consisted mainly of data manipulation screens for the administrators. An example of test results the students saw can be found in Figure 3.

Tests run:

Name	Result
add+list	Passed
add+list simple	Passed
delete	Passed
update	Passed
filter	Passed

FIGURE 3. Test Run Example

Among the possible gamification mechanics, two were chosen on the basis of what experiments were scheduled. The first two were instant feedback and narrative. Instant feedback is one of the most basic gamification mechanics. It shortens the work-reward cycle and it is easier to do on modern computers. In computationally intensive situations the feeling of instant feedback can be achieved by making time for static code analysis to complete by showing the I/O test results one by one with small, artificial delays. If the instant feedback option was used on a group then they could upload solutions any time of the day, otherwise they could only upload during their individual laboratory times.

Narrative is one of the strong suits of R.P.G.s [7], which is where some of the gamification mechanics originate from. It aids in immersion and gives a sense of purpose to solving exercises. The aspect of tying all exercises together in a narrative was one of the interests while working on this system. With the aim of finding out, if adding cross assignment narrative adds to the enjoyment of solving programming puzzles. If a group was selected for narrative than they saw an alternate text for their assignments that took part of an overarching story. Regular assignments and examinations both took part of the story. There was also support for different languages for the exercises for the same course thought in a foreign language.

For login and authorization by role, standard ASP.NET MVC controls were used, which are deemed safe enough. Any Id used in the various URLs are GUIDs as to make it impossible to guess any other ids. Students had the possibility to re-upload new exercises solutions until they were happy with the results. All code variants were saved on the server both code and test result for data-mining purposes but after the semester the student names and emails were anonymized.

4. Discussion

Taking into account previous research [2] the issue of a secured environment came up. Allowing random code to run without a sandbox would expose the server to vulnerabilities. A sandboxed environment would be ideal but this remains a future development. Another potential issue mentioned in the same research that was noticed is the temporary overwhelming of the hardware when too many students upload at the same time. Their solution is to add waiting queues which would be a sensible choice here as well, but it has to be prototyped and UI has to be changed to manage student expectations.

An unfortunate side effect of having the server accessible to the internet was a persistent, distributed attempt to gain control from at least one botnet. While unsuccessful the several tens of thousands of requests did tie down a portion of the server's resources.

5. Conclusions and further work

The system developed has great potential for experimentation, since it allows for features to be switched off and can be freely extended. Various experiments can be designed around the tool. Since we store code and usage data, data mining methods can be used in lockstep with traditional experimental setups.

The tool was used for a semester for an object oriented programming course. Aside from bug-fixes the tool was unchanged during this period. The gamification experiments conducted is the subject of an upcoming paper. The automated correctness check did noticeably increase the code review time per student. In the future this will be the default tool for that course.

Assignment validator features should be expanded with full blown sandboxes to run the executables in, the support for running queues to balance load and improve measurements, the support for more programming languages and UI languages, unit test code coverage calculator, and also data mining tools would be a useful addition. There are a number of small changes that should be made for example messaging students directly.

Since the field of gamification is vast, and so far under researched, there are many techniques that should be added and tried. Mechanics for pacing might be added for example leveling, and achievements. Also, mastery might be targeted with leaderboards.

References

[1] A. Aiken. Measure of software similarity: Plagiarism detection system. Technical report, Computer Science Division of University of California, 2002.

[2] A. L.-W. O. B. Cheanga, A. Kurniaa. On automated grading of programming assignments in an academic institution. *Computers & Education 41*, pages 121 —- 131, 2003.

[3] Y. L. B. Leong. Application of game mechanics to improve student engagement. In *International Conference on Teaching and Learning in Higher Education*. Citeseer, 2011.

[4] S. Deterding, D. Dixon, R. Khaled, and L. Nacke. From game design elements to gamefulness: Defining gamification. *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, MindTrek 2011*, 11:9–15, 09 2011.

[5] X. Fu, B. Peltsverger, K. Qian, L. Tao, and J. Liu. Apogee: automated project grading and instant feedback system for web based computing.

[6] J. J.-D. G. G. Barata, S. Gama. Engaging engineering students with gamification. *2013 5th International Conference on Games and Virtual Worlds for Serious Applications, VS-GAMES 2013*, pages 24–31, 09 2013.

[7] D. A. G Gygax. *Dungeons and dragons*. TACTICAL STUDY RULES, 1974.

[8] N. W. G.E. Forsythe. Automatic grading programs. *Communications of the ACM, 8(5)*, pages 275–529, 1965.

[9] J. Hamari, J. Koivisto, H. Sarsa, et al. Does gamification work? - A literature review of empirical studies on gamification. In *HICSS*, volume 14, pages 3025–3034, 2014.

[10] S. Itamar. Using gamification and gaming in order to promote risk taking in the language learning process. *MEITAL National Conference*, pages 227 — 232, 2015.

[11] N. v. V. J. Hage, P. Rademaker. A comparison of plagiarism detection tools. *Technical Report UU-CS-2010-015*, 2010.

[12] D. D. J. Taylor. Constructed-response, computer-graded homework. *American Journal of Physics, 44*, pages 598–599, 1976.

[13] M. M. Striewe. A review of static analysis approaches for programming exercises. *Computer Assisted Assessment. Research into E-Assessment.*, pages 100–113, 2014.

[14] K. J. Majuri, Jenni and J. Hamari. Gamification of education and learning: A review of empirical literature. In *Proceedings of the 2nd International GamiFIN Conference, GamiFIN 2018*. CEUR-WS, 2018.

[15] M. Poženel, L. Fürst, and M. Viljan. Introduction of the automated assessment of homework assignments in a university-level programming course. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 761–766. IEEE, 2015.

[16] B. A.-H. Q. Zoubi, I. Alsmadi. Study the impact of improving source code on software metrics. *2012 International Conference on Computer, Information and Telecommunication Systems (CITS)*, 2012.

[17] H. H. R. M. Rottman. Computer grading as an instructional tool. *Journal of college science teaching, 12*, pages 152–165, 1983.

[18] B. F. Skinner. Two types of conditioned reflex and a pseudo type. *Journal of General Psychology 12*, pages 66–77, 1935.

[19] U. von Matt. Kassandra: The automatic grading system. *ACM Special Interest Group on Computer Uses in Education Outlook*, 22, 03 2001.

Faculty of Mathematics and Computer Science, Babeș-Bolyai University, 1 Mihail Kogălniceanu, RO-400084 Cluj-Napoca, Romania

Email address: imre@cs.ubbcluj.ro