I.J. Engineering and Manufacturing, **2018, 4, 40-55**
Published Online July 2018 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijem.2018.04.05

Available online at http://www.mecs-press.net/ijem

Classification of Small Sets of Images with Pre-trained Neural Networks

Biserka Petrovska[a], Igor Stojanovic[b], Tatjana Atanasova-Pacemska[b]

[a] *Military Academy "General Mihailo Apostolski", University "Goce Delcev",1000 Skopje, RMacedonia*
[b] *Faculty of Computer Science, University "Goce Delcev", 2000 Stip, RMacedonia*

Received: 12 March 2018; Accepted: 19 April 2018; Published: 08 July 2018

Abstract

Nowadays the rise of the artificial intelligence is with high speed. Even we are far away from the moment when machines are going to make decisions instead of human beings, the development in some fields of artificial intelligence is astonishing. Deep neural networks are such a filed. They are in a big expansion in a new millennium. Their application is wide: they are used in processing images, video, speech, audio, and text. In the last decade, researches put special attention and resources in the development of special kind of neural networks, convolutional neural networks. These networks have been widely applied to a variety of pattern recognition problems. Convolutional neural networks were trained on millions of images and it is difficult to outperform the accuracies that have been achieved. On the other hand, when we have a small dataset to train the network, there is no success to do it from a scratch. This article exploits the technique of transfer learning for classifying the images of small datasets. It consists fine-tuning of the pre-trained neural network. Here in details is presented the selection of hyper parameters in such networks, in order to maximize the classification accuracy. In the end, the directions have been proposed for the selection of the hyper parameters and of the pre-trained network which can be suitable for transfer learning.

Index Terms: Pre-trained neural networks, deep learning, transfer learning, accuracy, hyper parameters, small datasets.

1. Introduction

Artificial intelligence (AI), deep learning (DL), and neural networks (NN) are powerful machine learning-based techniques. These techniques are incredibly exciting and used to solve many real-world problems.

* Corresponding author.
E-mail address:

On one hand, human-like deductive reasoning and decision-making by a computer is still a long time away. But on the other hand, there have been remarkable gains in the application of AI techniques and associated algorithms of AI. Popular examples of an AI solution includes IBM's Watson, Apple's Siri and Amazon's Alexa. Watson was made famous by beating the two greatest Jeopardy champions in history. It is now being used as a question answering computing system for commercial applications [29].

So far AI has been used for speech recognition and natural language applications (processing, generation, and understanding). It is also used for other recognition tasks (pattern, text, image, video, audio, facial …), autonomous vehicles, medical diagnoses, gaming, search engines, robotics, spam filtering, crime fighting, marketing, remote sensing, transportation, classification, etc.

There are many different goals of AI as mentioned, with different techniques used for each. The primary topics of this article are deep neural networks, especially one certain kind of them – convolution neural networks and their use for transfer learning.

At the time of this writing, there are a lot of pre-trained convolutional neural networks, developed by scientists or big corporations. The main purpose of these networks is to solve image classification problems. Even that the above mentioned CNN were trained on certain image sets, they can be used for image classifications on other sets of images. This is so called 'transfer learning'.

Transfer learning is a technique of optimization of pre-trained CNN in order to classify a set of images on which it was not trained before. Optimization actually consists selection of the hyper parameters of the neural network.

Nomenclature

AI	Artificial Intelligence
DL	Deep Learning
NN	Neural Network
CNN	Convolutional Neural Network
BP	Backpropagation
SL	Supervised Learning
UL	Unsupervised Learning
RL	Reinforcement Learning
RNN	Recurrent Neural Network
MLP	Multilayer Perceptron
KSH	Krizhevsky, Sutskever, and Hinton
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
GPU	Graphic Processor Unit
RGB	Red Green Blue
ReLU	Rectified Liner Unit

2. Neural Networks and Deep Learning

A standard neural network (NN) consists of many neurons. They are simple, connected processors, each producing a sequence of real-valued activations. Input neurons differ from the other neurons of the neural network. They get activated through sensors perceiving the environment. Other neurons (hidden and output) get activated through weighted connections from previously active neurons. As input neurons are affected by the surrounding, also some neurons may influence the environment. Learning is about finding weights that make the NN behaves in the desired way, such as driving a car. Such behavior may require long causal chains of computational stages. It depends on the problem and how the neurons are connected. Each stage transforms the aggregate activation of the network. As mentioned in reference [1] Deep Learning is about accurately assigning weights across many such stages.

In the history of neural networks first, shallow NN-like models appeared. Shallow NNs with several

successive nonlinear layers of neurons date back at least to the 1960s and 1970s. Backpropagation (BP) algorithm is an efficient gradient descent method for teacher-based Supervised Learning (SL) indiscrete, differentiable networks of small depth. It was developed in the 1960s and 1970s and applied to NNs in 1981. But despite the case of the shallow NNs, training of deep NNs with many layers with BP-based algorithm had been found to be not so successful by the late 1980s. It had been put a lot of efforts in an extensive research on that subject by the early 1990s. Unsupervised Learning (UL) enabled DL to some extent. In the 1990s and 2000s also many improvements of exclusively supervised DL were achieved. In the last fifteen years, as explained in [2] and [3], deep NNs have finally attracted wide-spread attention. It has been mainly due to the fact that they outperformed the alternative machine learning methods such as kernel machines in numerous important applications. In fact, since 2009, supervised deep NNs have won many official international pattern recognition competitions. Deep NNs achieved the level of superhuman visual pattern recognition results in limited domains. These networks also have been set up for a good solution for Reinforcement Learning (RL). In Reinforcement Learning there is no supervising teacher.

Two types of deep neural networks, NNs (FNNs) and recurrent NNs (RNNs) have won contests. Reference [4] says that RNNs are the deepest of all NNs. Their main purpose is to create and process memories of arbitrary sequences of input patterns. RNNs can learn programs that mix sequential and parallel information processing in a natural and efficient way. Authors agree with [1] that RNNs are exploiting the massive parallelism, which is crucial for the rapid decline of computation cost over the past 75 years.

Deep learning is about two “deep” things - computational models and representation of data. Computational models are composed of multiple processing layers and they learn representations of data with multiple levels of abstraction. Deep learning has dramatically improved the state-of-the-art in speech recognition, visual object recognition, and object detection. Face recognition (FR), which has numerous practical applications in the area of biometrics, information security, access control, law enforcement, smart cards, and surveillance system can be done with DL neural network [31]. DL is beneficial in many other domains such as drug discovery and genomics. BP algorithm discovers intricate structure in large data sets. It shows how a machine should change its internal parameters (weights) that are used to compute the activation in each layer from the activation in the previous layer. Convolutional networks–CNN, have brought breakthroughs in processing images, video, speech, and audio, for example, recognition of car number plates on images with a complex background [32]. RNN has performed better results on sequential data such as text and speech [5].

3. Related Work

For classification of small sets of images, there is a benefit of transfer learning. It helps to overcome the lack of training samples. Transfer learning technique is related to the pre-trained network which is used to transfer the knowledge from. In our case, it is a type of convolutional neural network AlexNet [6]. CNNs are the pretty much novel concept of neural networks. They are successors of feed forward neural networks and are especially useful for classification of images. AlexNet made a revolutionary step with its unique architecture and achieved accuracy on the large international contest in visual pattern recognition in 2012. It leaved well behind itself the other competitors. Here we optimize a part of the neural network hyper parameters during transfer learning from AlexNet [6], in order to maximize the classification accuracy of small sets of images.

3.1. Convolutional Neural Networks

A convolutional neural network (CNN) is one kind of a deep neural network. It has vast application to a variety of pattern recognition problems, such as image recognition, speech recognition, etc. The CNNs were first developed by Hubel & Wiesel [7]. Then other researches followed. Some successful implementations of CNN are NeoCognitron [8], LeNet5 [9], HMAX [10], AlexNet [6], GoogLeNet [11], ResNet [12], etc.

There is a problem that often occurs in the feed forward neural networks. The problem is all Multilayer Perceptron (MLP) layers are fully connected to each other. Such a network architecture does not take into

account the spatial structure of the images [13], which is such that the parts of the image which are close to each other are dependent on each other as well. The basic idea of CNN is to build invariance properties into neural networks by creating models that are invariant to certain inputs transformation.

The architecture of CNN usually is composed of a convolutional layer and a sub-sampling layer as shown in Fig.1 [27]. The convolutional operation is implemented in the convolutional layer, and a sub-sampling operation is implemented in the sub-sampling layer (pooling layer). In the root of CNN are basically three main concepts, i.e., local receptive fields, weight sharing, and pooling.

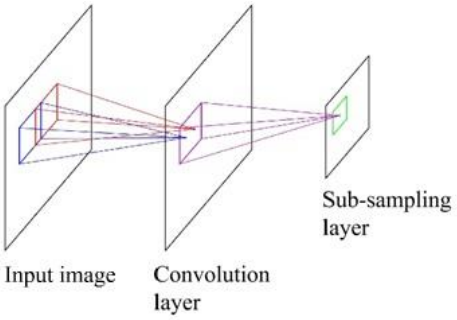

Fig.1.The Architecture of a CNN

Unlike in MLP, each neuron in a hidden layer in CNN will be connected to a small field of the previous layer. This filed is called a local receptive field. If the local receptive field has a 3×3 area, a neuron of the first convolutional layer is referred to 9 pixels of the input layer. The local receptive field is shown in Fig.2 (a) [27]. The lines represent where the neuron is connected to. Each connection from the local receptive filed of the input layer to the hidden neuron learns a weight and the hidden neuron itself learns an overall bias as well.

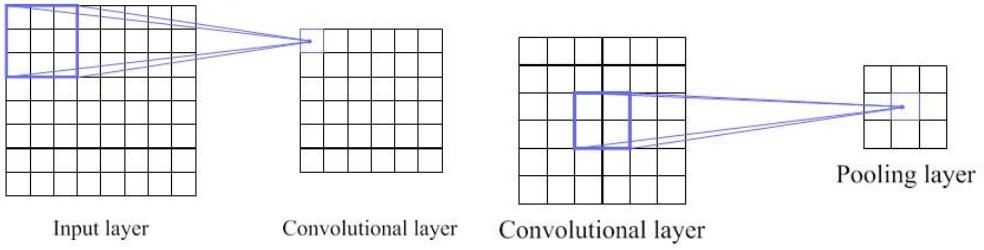

Fig.2. (a) Convolutional Layer - Local Receptive Fields; (b) A Pooling Layer In A Feature Map

Feature maps are multiple parallel hidden layers in the convolutional layer, where the neurons are organized into. Each neuron in a feature map is connected to a local receptive field. It is important to emphasize that all neurons for every feature map share the same weight parameter that is known as filter or kernel. The output for the *j*,*k*-th hidden neuron is:

$$\sigma\left(b+\sum_{l=0}^{2}\sum_{m=0}^{2}w_{l,m}a_{j+l,k+m}\right) \tag{1}$$

Where σ is the neural activation function, b is the value of the bias, $w_{l,m}$ is a 3×3 array weights and $a_{x,y}$ is the

input activation at position *x,y*. As it is mentioned previously, the values of the bias and weights are shared for the neurons of the local receptive field.

Beside convolutional layers, CNN contains pooling layers, as well. It is usually used immediately after a convolutional layer. Pooling layer generates translation invariant features by computing statistics of the convolution activations from a small receptive field that corresponds to the feature map. Fig.2 (b) illustrates an example of how it works for each feature map [27]. Usually, there are more than a single feature map. The example of that case is depicted in Fig.3 [27].

Fig.3 illustrates one concrete example for pooling procedure, so-called max-pooling. When it comes to max-pooling, a pooling unit outputs the maximum activation in the input region, in our case 2x2 region.

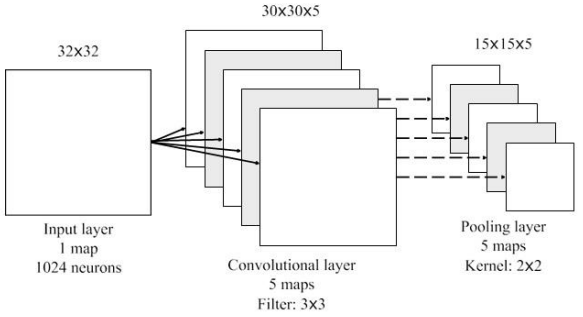

Fig.3. Overlapping Pooling with size 2 ×2 for Each Feature Map

Another kind of pooling is known as L2 pooling. Here, despite the maximum pooling where we take the maximum activation of an input region of neurons, it is taken the square root of the sum of the squares of the activations in the input region [13].

3.2. AlexNet

A deep convolutional neural network was trained and tested by Krizhevsky, Sutskever, and Hinton (KSH) [6]. They trained and tested the network using a restricted subset of the ImageNet data [14]. This subset came from (ILSVRC) - the ImageNet Large-Scale Visual Recognition Challenge. The good thing about using the ImageNet data is that it is a way of comparing the capabilities of the studied NN to other leading techniques. The ILSVRC-2012 training set contained about 1.2 million ImageNet images. The images were categorized into 1,000 categories. The validation and test sets had 1,000 categories of images as well and they contained 50,000 and 150,000 images, respectively.

The ILSVRC competition has one disadvantage, many ImageNet images contain multiple objects and because of this, they are ambiguous. So, an algorithm was considered correct if the actual ImageNet classification was among the 5 classifications the algorithm considered most likely (top-5 criterion). The deep convolutional network developed by Krizhevsky, Sutskever, and Hinton (which was later named AlexNet) achieved an accuracy of 84.7 percent, according to this top-5 criterion. It was better than the next-ranged CNN, which achieved an accuracy of 73.8 percent. KSH's network achieved an accuracy of 63.3 percent, strictly speaking for the restrictive metric of accuracy.

The KSH network has 7 layers of hidden neurons - 5 hidden layers which are convolutional layers, and 2 layers which are fully-connected layers. Some of the convolutional layers are with max-pooling. The output layer is a 1000-unit softmax layer. The architecture of the network is shown on Fig.4 [6]. The network was trained on 2 GPUs. That’s why many layers are split into 2 parts. Half of the kernels of the KSH network were put on the first GPU and the other half of the kernels were put on the second GPU. The GPUs communicate

only in certain layers.

The input layer of KSH network contains 3×224×224 neurons. They represent the RGB values for a 224×224 image. Krizhevsky, Sutskever, and Hinton expanded the training data in order to reduce the overfitting. AlexNet is a large network, so expanding the training data is particularly helpful in such networks. Because ImageNet contains images of varying resolution, KSH rescaled each image in a way shorter side had length 256. After this, they cropped out a 256×256 area in the centre of the rescaled image. Krizhevsky, Sutskever, and Hinton extracted random 224×224 sub images from the 256×256 images. They extracted horizontal reflections as well. These 224×224 images were used as inputs to the network.

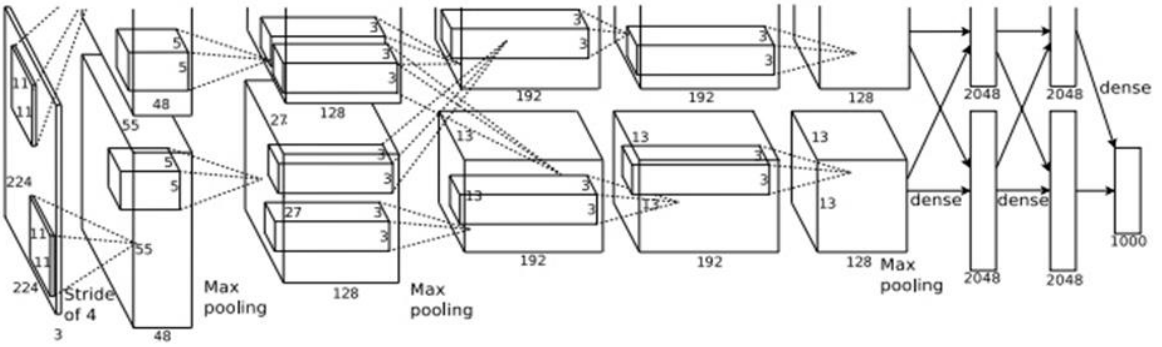

Fig.4. Architecture of AlexNet

The first hidden layer in AlexNet is a convolutional layer, with local receptive fields of size 11×11, and a stride length of 4 pixels. There are a total of 96 feature maps. They are split into two groups of 48 each. The first group of 48 feature maps is processed on one GPU, and the second 48 feature maps are processed on the other GPU.

The second hidden layer is also a convolutional layer. Its parameters are 5×5 local receptive fields, a total of 256 feature maps. The feature maps are split into two groups of 128 each and are processed on each of the GPUs separately.

KSH network has three more convolutional layers: the third, fourth and fifth hidden layers. The third hidden layer has 384 feature maps, with 3×3 local receptive fields, and 256 input channels. The fourth hidden layer has 384 feature maps, with 3×3 local receptive fields, and 192 input channels. The last convolutional layer has 256 feature maps, with 3×3 local receptive fields, and 192 input channels.

Next two hidden layers, the sixth and seventh, are fully-connected layers. Each of them has 4096 neurons.

The output layer is a softmax layer with 1000 neurons.

As can be noticed on Fig.4, GPUs communicate only in the third convolutional layer, its kernels are connected to all kernel maps in the second layer. The kernels of the other convolutional layers, the first, second, fourth and fifth, are connected only to those kernel maps in the previous layer which reside on the same GPU. The sixth and seventh layer are fully connected, their neurons are connected to all neurons in the previous layer.

Response-normalization layers follow the first and second convolutional layers. They are followed, the same as the fifth convolutional layer, by max-pooling layer with 3×3 regions. The pooling regions are allowed to overlap and are just 2 pixels apart. In order to speed up training, the ReLU (Rectified Linear Unit) non-linearity is applied to the output of every convolutional and fully-connected layer; the activation function is $f(z) \equiv \max(0, z)$ [13].

The KSH network takes advantage of many techniques. KSH network was susceptible to overfitting because it had roughly 60 million learned parameters. To solve this problem Krizhevsky, Sutskever and Hinton expanded the training set using the random cropping of the image set, they also used a variant of L2

regularization and dropout. AlexNet was trained with a BP algorithm, momentum-based, mini-batch stochastic gradient descent.

3.3. Transfer learning

The purpose of transfer learning is to transfer knowledge between the source and target domains. For image classification, the main benefit of transfer learning is to overcome the deficit of training samples for some categories. It is done by adapting classifiers trained for other categories. A neural network gains knowledge from the training data. Knowledge is contained in weights of the network. In a case when there is a lack of training data, training a network from a scratch is not useful, nor the accuracy is satisfying. In these situations, we extract the weights of the original network and "transfer" the learned features [28].

Transfer learning has been successfully applied to text sentiment classification [15], image classification [16, 17], human activity classification [18], software defect classification [19], and multi-language text classification [20]. Pan [21] in 2010 published a transfer learning survey paper. Since then there have been over 700 academic papers written about this topic. It surveys transfer learning on different levels: new algorithm development, improvements to existing algorithms, as well as algorithm deployment in new application domains.

ImageNet dataset [14] is huge and diverse. This is the reason why existing networks, pre-trained on ImageNet, demonstrate a good ability to classify images outside this dataset via transfer learning. What is actually done is fine-tuning the pre-existing model. In reference [28] is explained how it is performed in three ways:

1. **Feature extraction** – This is the situation when a pre-existing model is used as a mechanism for feature extraction. Here the output layer is removed and then for the new data set the entire network can be used as a fixed feature extractor.
2. **Use the Architecture of the pre-trained model –** In this scenario, all we use from the pre-trained model is the architecture. All the weights are initialized randomly. The network is trained according to the dataset of interest.
3. **Train some layers while freezing others** – Third way to use a pre-trained network is to train it partially. In these cases, the weights of the lower layers of the network are kept unchanged, while the higher model layers are retrained. It is a matter of trial and test as to how many layers to be kept unchanged and how many layers to be trained.

Fig.5 represents diagram how to make a decision to proceed with using the pre trained model in a specific case [28].

Case 1- **Size of the data set is small and the data similarity is very high –** Here there is no need to retrain the network. It is due to the similarity of data set, which is high. The pre-existing model is used as a mechanism for feature extraction. But certainly pre-existing model needs some changes. The output layers should be customized and modified according to our problem.

Case 2- **Size of the data is small and data similarity is very low** – In this scenario, a certain number of lower layers (k for example) are frozen. The remaining (n-k) higher layers of the pre-existing network model have been trained again. Because of the fact that new data set has low similarity with the data set the network was originally trained on, it is important to retrain the higher layers of the pre-existing model.

Case 3- **Size of the data set is large and the data similarity is very low** – The first condition for the effectiveness of a neural network training is the size of the data set to be sufficient. In this scenario the dataset is large and it is very different compared to the data set on which the network was originally trained, so the best solution is to train the neural network from a scratch.

Case 4- **Size of the data is large and there is high data similarity –** Since the data set is large, there is a

possibility to retrain the whole network. It is needed to keep the architecture of the model and the initial weights of the model. After this, the neural network should be re-trained with the weights as initialized in the pre-trained mode.

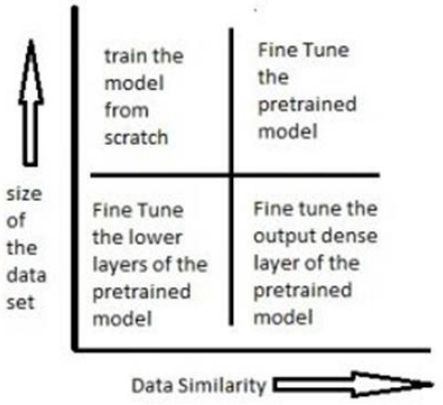

Fig.5. Transfer Learning According to the Dataset

3.4. Hyper Parameters

Different learning algorithms involve different sets of hyper-parameters [30]. Here we are discussing the hyper parameters which are optimized in our simulation scenario.

The initial learning rate (η_0) - is the most important hyper parameter. Researches put efforts to tune it up to approximately a factor of 2. When the inputs of a neural network are standardized (they are in the interval (0, 1)), the values of initial learning rate moves in the range (10^{-6}, 1). This should not be taken so strictly. During the network training, there is a possibility to adapt the learning rate according to a given schedule. In equation (2) τ is time constant, which when $\tau \rightarrow \infty$ means that the learning rate is constant over whole network training. The constant value of learning rate gives satisfying results in most cases of neural network training. An example of *O (1/t)* learning rate schedule, used in [22] is

$$\eta_t = \frac{\eta_0 \tau}{\max(t, \tau)} \tag{2}$$

Here learning rate is kept constant for the first τ steps and then decreases it in *O (1/tα).*

The mini-batch size B –the value 32 is a good default value. Usually, B takes a value between 1 and few hundreds. When the value of B is a couple of tens, network training takes advantage of matrix-matrix products. Selection of the mini-batch size has strictly computational meaning. With larger B computations are faster, but in those scenarios, we need a larger training set. It is because network training requires visiting more data in order to reach the same accuracy since there are fewer updates per epoch [30].

Number of training iterations T - this hyper-parameter is optimized using the principle of early stopping. As neural network training progresses (every N updates of a mini-batch), we keep track on accuracy estimated on a validation set. The rule for termination the training concerns best classification accuracy: if it doesn't

improve for quite some time (number of training iterations), then training stops.

Momentum β- the idea with momentum is to remove some of the noise and oscillations that gradient descent has. A long time thesis for temporally smoothing out the stochastic gradient samples obtained during network training has been present [23]. A moving average of the past gradients is given with $\bar{g} \leftarrow (1-\beta)\bar{g} + \beta g$. Parameter *g* is the gradient in a moment $\frac{\partial L(z_t, \theta)}{\partial \theta}$ or a mini batch average. The other parameter β controls how much the old examples influence the moving average. It typically takes values between 0 and 1. The function of momentum is to make the updates proportional to smoothed gradient estimator $\bar{g}$. Without momentum β updates are proportional to instantaneous gradient *g*.

Layer-specific optimization hyper parameters –sometimes researches use different values for the hyper-parameters (such as the learning rate) on different layers of a deep neural network. This is a case especially when there is a big difference in a number of neurons from layer to layer. Also, there is a possibility to use different learning rates for the different types of parameters one finds in the model. In deep learning networks such an example are biases and weights. The importance of different learning rate for separate parameters is bigger when parameters such as precision or variance are also included [24].

A number of hidden neurons (layers) – during designing the model of a neural network, we also need to set the number of neurons in each hidden layer. We are free to choose the size of layers in a neural network as big as we need because there are many techniques that combat overfitting - early stopping, weight decay, etc. Variation of this parameter is a number of hidden layers.

Weight decay regularization coefficient λ – a way to reduce overfitting is to add a regularization term to the cost function, in order to make network prefers to learn small weights. There are two types of regularization L2 and L1. First one adds a term $\lambda \sum_i \theta_i^2$ to the cost function, while the second one adds a term $\lambda \sum_i |\theta_i|$. They act differently: L2 penalizes large weights strongly; L1 tends to concentrate the weights of the network in a small number of high-important connections, the others go to zero [30]. Some researches during neural network training regularize only the weights *w* and not the biases *b* associated with the hidden neurons activations.

Neuron non-linearity- if *x* is the vector of inputs into the neuron, *w* the vector of weights and *b* the bias parameter, then neuron output is *s(a) = s(w'x+b)*. Following non-linear functions, *s* can be used for hidden neurons: the sigmoid $1/(1+e^{-a})$, the rectifier max (0, a), the hyperbolic tangent and the hard tanh [25].

The above mentioned hyper parameters are generic choices. There are also a number of other hyper parameters which can be optimized with different architectures and learning algorithms, among them: sparsity of activation, weights initialization scaling coefficient, random seeds, pre-processing etc.

4. Simulations

This section describes the efforts of the authors in order to achieve their goal, that's it to optimize pre-trained neural network for classification of small data sets on which it was not trained before, with acceptable accuracy. The following issues are explained: a pre-trained neural network which is used, along with the training and test data sets, transfer learning scenario, software, and hardware simulation platform. It should be mention that due to lack of the original training and test images, here the images from CIFAR-10 data set are used. But, it is good to evaluate transfer learning technique with training and test images from our own data set. Simulation results give an overview of the achieved classification accuracy in case of different values for each of the proposed hyper parameters. It should be mention that the choice of the optimized hyper parameters is not unique and there is always a possibility of another selection.

4.1. Simulation Scenario

As it is mentioned before, the aim of our work is to optimize (fine-tune) the hyper parameters of a pertained convolutional neural network, in order to use it for classification of small datasets on which it was not trained before. Here it is used AlexNet [6] as pre-trained model architecture. AlexNet was trained on a restricted subset of the ImageNet data [14] by its creators. The goal is to classify a set of images on which the neural network was not trained before. Used images are a subset of CIFAR-10 dataset [26]. The subset consists images which belong to 5 categories.

The CIFAR-10 dataset consists of 10 classes of images with 6000 images per class or 60000 32x32 colour images in total. From 60000 images, 50000 are training images and 10000 are test images. There are five training batches and one test batch, each consisted of 10000 images. Test batch contains 1000 images from each class, while training batches contain 5000 images from each class. Images in a test batch are randomly selected while remaining images belong to training batches. There is no overlapping of categories between different classes of images.

In section **III.Related work**, subsection *3.3 Transfer learning*, there are briefly explained four possible scenarios during transfer learning. Which scenario should take place, it depends on two variables: the size of the dataset and the similarity between the dataset on which the convolutional neural network was trained at first and the dataset which should be classified. The CIFAR-10 dataset has 60000 training and test images, but we will use only a part of them. So, the dataset is definitely small. ImageNet data is consisted of 1000 object categories, for example, keyboard, mouse, pencil, and many animals. Compared to images in the CIFAR-10 dataset, we can say that the similarity between the two datasets is big. This situation can be resolved by scenario 1 – we fine tune the output layers of the pre-trained neural network.

Actually, here we use AlexNet as a feature extractor. We then use these features and send them to dense layers which are trained according to our dataset. The last three layers form the original AlexNet architecture: a fully connected layer with 1000 neurons, a softmax layer, and the classification output layer are removed. They are replaced with new layers relevant to our problem: a fully connected layer with 5 hidden units (because we will classify images from 5 categories), softmax layer and classification layer of 5 categories. Optionally, we added an additional fully connected layer with 75 neurons before the last fully connected layer with 5 hidden neurons.

Because the similarity of images which belong to ImageNet data and CIFAR-10 is high, during training we wanted to keep the weights of the lower layers of AlexNet pretty much unchanged or changed a little. Despite this, the weights of the last layers we introduced in the network should learn faster than the others. That's why weight learn rate factor and bias learn factor are from order 10^1.

During training the network, we used train dataset and test dataset, which size was 20% the size of the training dataset. We were plotting the training accuracy in order to visualize the process and to realize when the network starts severly overfitting. We performed the simulations in MATLAB with GeForce GTX 960 GPU (Graphic Processor Unit).

In the next subsection we will present and discuss the simulation results from optimization of the following hyper parameters: size of a training set, learning rate, regularization parameter λ, weight and bias learn rate factor, size of a mini batch, number of training epochs (iterations), number of hidden layers (units).

4.2. Simulation Results

Size of a training data is not a hyper parameter actually. But, it is interesting to see which is the scale of the data set, for which the pre-trained convolutional network gives acceptable accuracy. Fig.6 shows the overall accuracy which is accomplished by CNN with transfer learning from AlexNet, as a function of the number of training images. It can be seen that with only a couple of thousands of images, the accuracy is nearly 85%. One has always to have on his mind that compared to this, AlexNet was originally trained on nearly 1.2 million images.

Fig.7 (a) depicts the training accuracy of the neural network as a function of the learning rate. We used a single constant value for the learning rate. For bigger learning rates 0.1 and 0.01 our classification accuracies

are no better than chance and our network is acting as a random noise generator. The best accuracies are achieved for learning rates 0.001 and 0.0001. Learning rate 0.00001 is too small and it slows down stochastic gradient descent. Fig.7 (b) shows the process of training CNN with transfer learning from AlexNet with different learning rates. It can be seen that with $\eta = 0.001$ the network suffers from severe overfitting, so $\eta = 0.0001$ is a better option.

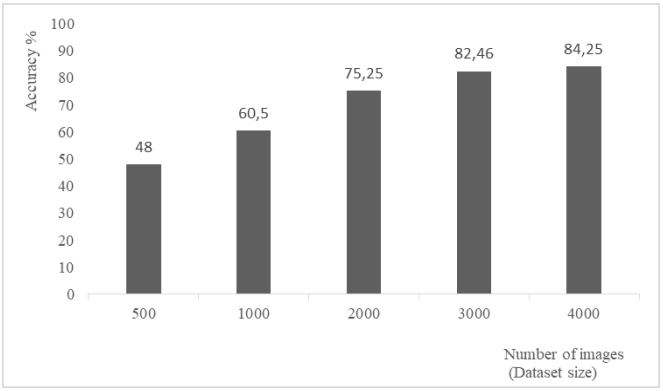

Fig.6. Accuracy of Pre-trained CNN as a Function of Dataset Size

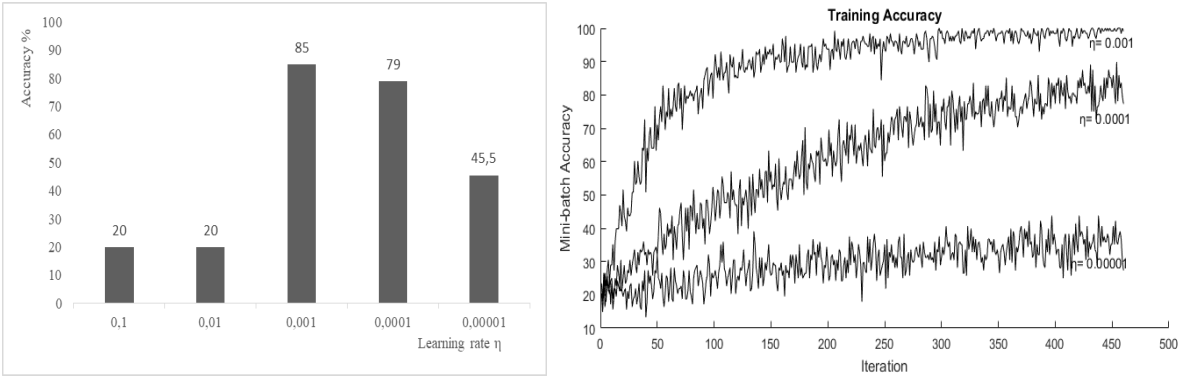

Fig.7. (a) Accuracy of Pre-trained CNN as a Function of a Learning Rate η; (b) Process of Training with Different Learning Rates η

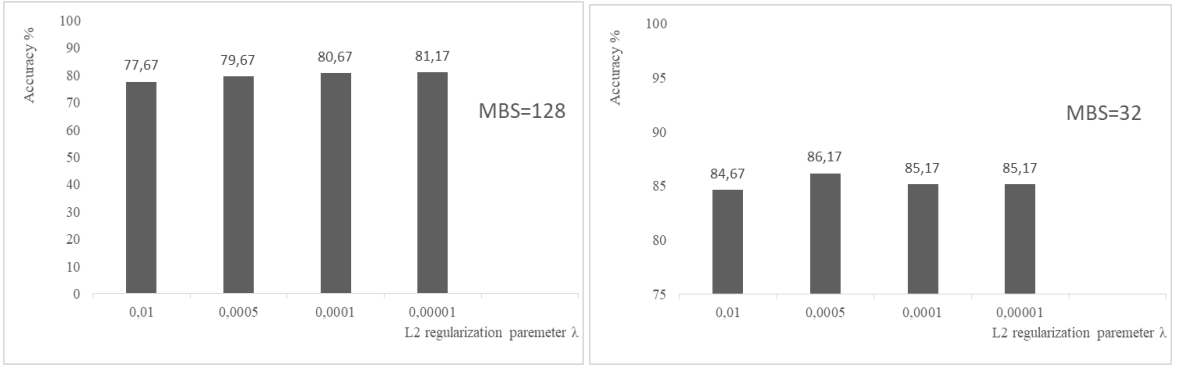

Fig.8. Accuracy of Pre-trained CNN as a Function of L2 Regularization Parameter λ with (a) MBS=128; (b) MBS=32

Next parameter we tried to optimized is L2 regularization parameter λ. AlexNet was originally trained with λ=0.0005. We searched for the appropriate value of λ in the surrounding of that value, decreasing or increasing with factor 5 or 10. For λ=0.0001 we got a little bit better accuracy, Fig.8 (a), and after that the improvement was negligible. Here, it should be mentioned that value of the MBS (mini batch size) was selected to be 128. Fig.8 (b) depicts the achieved accuracy in the process of transfer learning for different values of λ and MBS=32. One can notice that the values in the second case are fluctuating and a conclusion can't be made. This is due to the severe overfitting, which is explained in details in the following paragraph.

Choosing a size of a mini batch is a compromise. If mini batch size is too small, then neural network training won't benefit from fast hardware and matrix libraries optimized for it. But if it is too large, the weights are not updated often enough. We were training our network for 20 epochs with a mini batch size 8, 16, 32, 64 and 128 respectively. Since the number of iterations was biggest for MBS (mini batch size) 8 –more than 4000, and it was smallest for MBS 128 – below 500, CNN experienced enormous overfitting for smaller mini batches and less overfitting for larger mini batches. During training of the network, we introduced an accuracy threshold of 99.5, in order to stop the training process because of the overfitting. The training was stopped in cases when the size of a mini batch was 8 and 16. Fig.9 (a) illustrates the process of training the network with MBS=16 and the severe overfitting it introduced. Fig.9 (b) shows the accuracy which was achieved with different values of MBS.

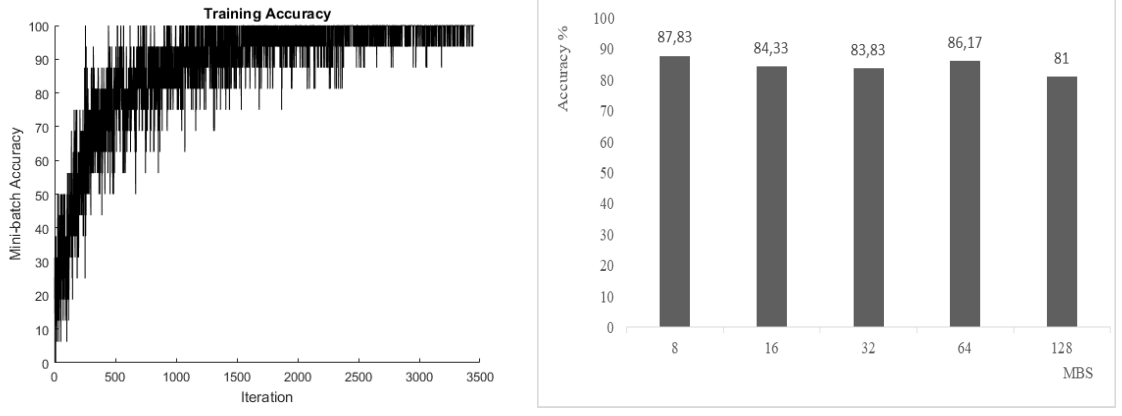

Fig.9. (a) Training Accuracy with the Mini Batch size of 16 Images (b) Accuracy of Pre-Trained CNN as a Function of MBS

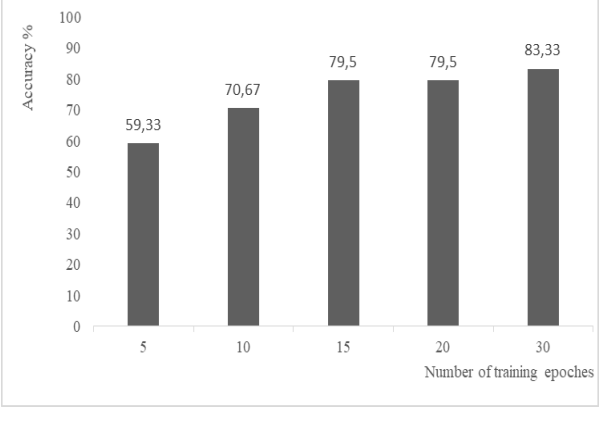

Fig.10. Accuracy of Pre-trained CNN as a Function of a Number of Training Epochs

In order to achieve the same accuracy with MBS=128 as with smaller mini batch sizes, network training requires visiting more data since there are fewer updates per epoch.

As we mentioned above, the influence of the size of the mini batch is closly related to the number of epoches (iterations) the network was trained. Fig.10 depicts the accuracy of the pretarined network as a function of the tarining epoches. In this case, the mini batch size was 128 images. It can be seen that after a couple of tens of epochs, the accuracy reaches over 83%. With transfer learning technique, it is not necessary to train the network for hundreds of epochs. Acctually, it is not recommended because of the small dataset and chances for overfitting.

Fig.11 presents the dependence of the accuracy of our trained network upon the learning rate factor of the weights and biases of the last fully connected layer. In our simulation scenario, we had a tendency to keep the weights and biases of the layers which we took from AlexNet in a great measure. On the other hand, we intended to make the new fully connected layer to learn more, compared to the layers of the original network. That's why we introduced layer-specific optimization hyper parameter: the learning rate for weights and biases of the neurons of the last fully connected layer. One can notice that the accuracy grows up with the growth of the weight and bias learn rate factor.

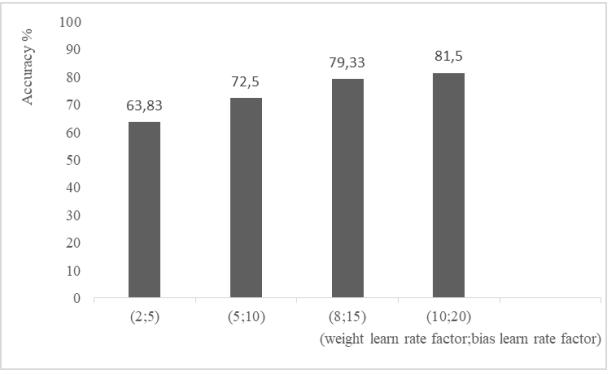

Fig.11. Accuracy of Pre-trained CNN as a Function of Weight and Bias Learn Factor

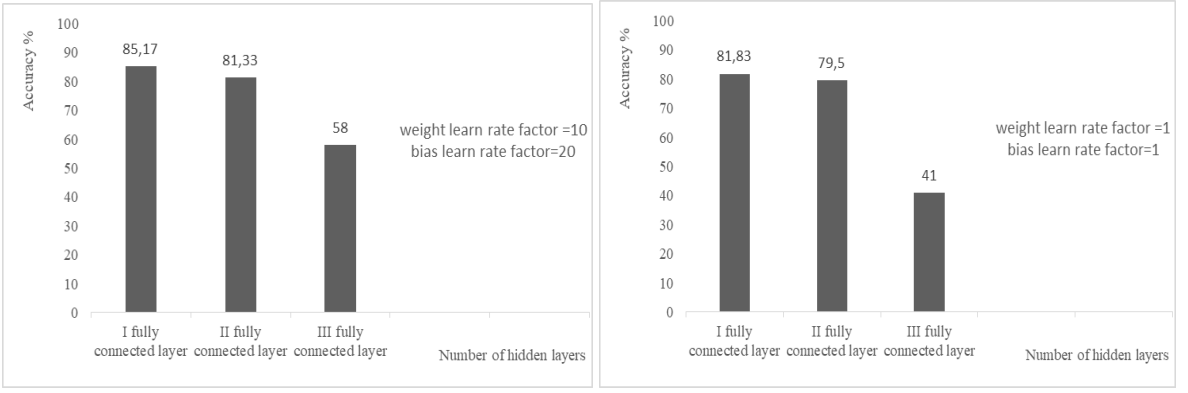

Fig.12. Accuracy of Pre-trained CNN as a Function of the Number of Hidden Layers with (a) Weight Learn Rate Factor = 10 and Bias Learn Rate Factor = 20 of the Last Fully Connected Layer; (b) Weight Learn Rate Factor and Bias Learn Rate Factor of the Last Fully Connected Layer Equal то 1

The last hyper parameter we have optimized is a number of hidden layers (units). As explained in the previous section, we imported the layers from AlexNet, except the last three ones. Then we added layers on our

own. In the first case, we added one fully connected layer with 5 neurons, softmax layer, and classification layer. Then we added another fully connected layer with 75 neurons before the first one. And in the last case, we added an additional third fully connected layer with 500 neurons and dropout = 0.5 before the second fully connected layer. Neurons in all three cases were ReLU (Rectified Linear Units). Fig.12 (a) represents the case when weight learn rate factor and bias learn rate factor of the last fully connected layer is 10 and 20 respectively. Fig.12 (b) represents the case when weight learn rate factor and bias learn rate factor of the last fully connected layer are 1. It can be concluded that as much as we go away from the structure of the original AlexNet, we get worse accuracy.

5. Conclusions

In our simulation scenario, we used a transfer learning from AlexNet, in order to classify a small dataset of images on which the network was not trained before. Actually, we were optimizing the hyper parameters of our network with the purpose to maximize the accuracy achieved on the test data. We got an accuracy of over 85% in specific cases of a simulation scenario.

We achieved the accuracy above with the training set of a couple of thousands of images and a short time spent for training the network. We didn't modify the weights too soon and too much. Also, we kept the architecture of AlexNet in a great measure, as far as the weights of the imported layers of the pre-trained network.

Sometimes there is no possibility to gain a big training and testing set of images, so training a neural network from a scratch is not an option (because of the small accuracy that would be achieved). Then comes to a situation when one should take transfer learning from a pre-trained neural network in to consideration. We should be very careful when it comes to decision what pre-trained model to be used in a specific scenario. If the image set we have to classify differs a lot from the one on which the pre-existing neural network was trained, then the classification would be very inaccurate. One must be sure that the pre-trained model selected has been trained on a similar data set as the one that he wishes to use it on. There are various architectures people have tried on different types of data sets.

References

[1] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview", *Neural Networks*, Volume 61, January 2015, Pages 85–117.

[2] Sch¨olkopf, B., Burges, C. J. C., and Smola, A. J., editors (1998), "Advances in Kernel Methods Support Vector Learning", *MIT Press, Cambridge*, MA.

[3] Vapnik, V. (1995), "The Nature of Statistical Learning Theory", *Springer*, New York.

[4] Siegelmann, H. T., and Sontag, E. D. (1991), "Turing Computability with Neural Nets", *Applied Mathematics Letters*, 4(6):77–80R.

[5] Y. LeCun, Y. Bengio, G. Hinton, "Deep Learning", *Nature 521*, 436–444 (28 May 2015).

[6] A. Krizhevsky, I. Sutskever, G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", pp. 1106–1114, 2012.

[7] D. H. Hubel, T. N. Wiesel, "Receptive Fields, Binocular Interaction and Functional Architecture in The Cat's Visual Cortex", *The Journal of Physiology*, Vol. 160, No. 1, pp. 106–154.2, Jan. 1962.

[8] K. Fukushima, "Neocognitron: A Self-Organizing Neural Network Model for A Mechanism of Pattern Recognition Unaffected by Shift in Position", *Biological Cybernetics*, Vol. 36, No. 4, pp. 193–202, April 1980.

[9] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-Based Learning Applied to Document Recognition", *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278–2324, Nov. 1998.

[10] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, T. Poggio, "Robust Object Recognition with Cortex-

Like Mechanisms", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 29, No. 3, pp. 411–426, March 2007.

[11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, "Going Deeper with Convolutions", *In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, June 2015.

[12] K. He, X. Zhang, S. Ren, J. Sun, "Deep Residual Learning for Image Recognition", arXiv:1512.03385 [cs], Vol. 1, Dec. 2015. arXiv: 1512.03385.

[13] Michael A. Nielsen, "Neural Networks and Deep learning", *Determination Press* 2015.

[14] Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database", In CVPR09, 2009.

[15] Wang C, Mahad-evan S, "Heterogeneous Domain Adaptation Using Manifold Alignment", *Proceedings of the 22nd international joint conference on artificial intelligence*, vol. 2. 2011. p. 541–46.

[16] Duan L, Xu D, Tsang IW, "Learning with Augmented Features for Heterogeneous Domain Adaptation", *IEEE Trans Pattern Anal Mach Intell 2012*; 36(6):1134–48.

[17] Zhu Y, Chen Y, Lu Z, Pan S, Xue G, Yu Y, Yang Q, "Heterogeneous Transfer Learning for Image Classification", *Proceedings of the national conference on artificial intelligence*, vol. 2. 2011. p. 1304–9.

[18] Harel M, Mannor, "Learning from Multiple Outlooks", *Proceedings of the 28th international conference on machine learning 2011*, p. 401–8.

[19] Nam J, Kim S, "Heterogeneous Defect Prediction", *Proceedings of the 2015 10th joint meeting on foundations of software engineering*, 2015. p. 508–19.

[20] Zhou JT, Pan S, Tsang IW, Yan Y, "Hybrid Heterogeneous Transfer Learning Through Deep Learning", *Proceedings of the national conference on artificial intelligence*, vol. 3. 2014. p. 2213–20.

[21] Pan SJ, Yang Q, "A Survey on Transfer Learning" *IEEE Trans Knowl Data Eng 2010*; 22(10):1345–59.

[22] Bergstra, J. and Bengio, Y. (2012), "Random Search for Hyper-Parameter Optimization", *J. Machine Learning Res.*, 13, 281–305.

[23] Hinton, G. E. (2010), "A Practical Guide to Training Restricted Boltzmann Machines", *Technical Report UTML TR 2010-003*, Department of Computer Science, University of Toronto.

[24] Courville, A., Bergstra, J., and Bengio, Y. (2011), "Unsupervised Models of Images by Spike-And-Slab RBMs", *In ICML'*2011.

[25] Collobert, R. and Bengio, S. (2004a), "Links Between Perceptrons, MLPs and SVMs", *In ICML'*2004.

[26] Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", 2009.

[27] http://socs.binus.ac.id/2017/02/27/convolutional-neural-network/

[28] https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/

[29] https://www.innoarchitech.com/artificial-intelligence-deep-learning-neural-networks-explained/

[30] Bengio, Y. (2012), "Practical Recommendations for Gradient-Based Training of Deep Architectures", arXiv: 1206.5533V2 [cs.LG] 16 Sep 2012.

[31] Priya Gupta, Nidhi Saxena, Meetika Sharma, Jagriti Tripathi,"Deep Neural Network for Human Face Recognition", *International Journal of Engineering and Manufacturing(IJEM)*, Vol.8, No.1, pp.63-71, 2018.DOI: 10.5815/ijem.2018.01.06

[32] Kalid A.Smadi, Takialddin Al Smadi,"Automatic System Recognition of License Plates using Neural Networks", *International Journal of Engineering and Manufacturing(IJEM)*, Vol.7, No.4, pp.26-35, 2017.DOI: 10.5815/ijem.2017.04.03

Authors' Profiles

Biserka Petrovska received BSc degree in 2002 and MSc degree in 2010, both in Faculty of electro technique and information technologies, University "St. Cyril and Methodius" Skopje. Currently, she is on Ph.D. studies in the field of image processing and deep learning. She works in Ministry of Defense of the Republic of Macedonia in Skopje and also works as Teaching Assistant at Military Academy "General Mihailo Apostolski" in Skopje, in the Department of natural and military-technical sciences.

Igor Stojanovic earned M.Sc. degree in 2002 and a Ph.D. degree in 2011, both from Ss. Cyril and Methodius University in Skopje. He also received a Ph.D. degree in 2014 from University of Nis, Serbia. Currently, he is an associate professor at the Faculty of Computer Science at University "Goce Delcev" - Stip, Macedonia. His research interests are Information Systems, Business Process Modelling, Digital Video and Image Processing.

Tatjana Atanasova - Pachemska earned M.Sc. degree in 2002 and a Ph.D. degree in 2006, both from Faculty of natural science and mathematics at Ss. Cyril and Methodius University in Skopje. Currently, she is a full professor at the Faculty of Computer Science, University "Goce Delcev" - Stip, Macedonia. Her research interests are theoretical mathematics, applied mathematics and mathematical modelling and math education.

How to cite this paper: Biserka Petrovska, Igor Stojanovic, Tatjana Atanasova-Pacemska,"Classification of Small Sets of Images with Pre-trained Neural Networks", International Journal of Engineering and Manufacturing(IJEM), Vol.8, No.4, pp.40-55, 2018.DOI: 10.5815/ijem.2018.04.05