THE 5th INTERNATIONAL CONFERENCE
MECHANICAL ENGINEERING IN XXI CENTURY

December 9-10, 2020
Faculty of Mechanical Engineering in Niš, Serbia

Classification of COVID-CT Images Utilizing Four Types of Deep Convolutional Neural Networks

Lara LABAN, Mitra VESOVIĆ

Control Engineering, Faculty of Mechanical Engineering, Kraljice Marije 16, Belgrade, Serbia
laralaban@mas.bg.ac.rs, mvesovic@mas.bg.ac.rs

***Abstract*— In this paper a method is presented for the classification of COVID-CT (CT_COVID, CT_NonCOVID) image data set. Four different types of deep convolutional neural networks are proposed, two with the architecture resembling the VGGNet, one resembling the LeNet-5 and one using transfer learning. In addition, neural networks utilized the following techniques: decay, dropout and batch normalization. Since we needed to combat a significantly small dataset, we used data augmentation in order to transform and expand our dataset. Moreover, juxtapositions were made when observing the results given by these four neural networks, as well as the affect made by two different optimizers. The training of the neural networks was done using small batches with a binary cross entropy loss function, in order to achieve an up to scratch classification accuracy.**

***Keywords*— deep learning; convolutional neural networks; image classification; data augmentation; batch normalization; COVID-CT dataset; dropout; transfer learning.**

I. INTRODUCTION

Convolutional neural networks (CNNs) are a subset of deep neural networks, which are used for classifying images. The main idea is to take a set of images correctly labeled as the input data and used them to train our neural network so as to achieve an output with an appropriate categorization [1]. The inspiration for CNNs comes from the observation of the animal visual cortex. Conversely, the flourishing of these networks only came recently due to the increase of computational power and the development of many possible libraries that could be used to battle complex mathematically based problems, such as back propagation. The first paper [2] that introduced the convolutional neural networks as we have come to know them today has demonstrated that a model which consists of a multilayered network can be successfully used for recognition of stimulus patterns according to the differences in their shapes. However, there is some debate that the true begging was when a paper in 1990 [3] demonstrated that a CNN model which aggregates simpler features into progressively more complicated features can be successfully used for handwritten character recognition. In 2012 the ImageNet Large Scale Visual Recognition Challenge [4], at that moment consisting of he 1000 categories and 1.2 million images received a submission that would propel the CNNs development once again. AlexNet [5] achieved a top-5 error of 15.3% , which at the moment surpassed by an astonishing 10% all of the other submissions, and had a much faster training time as it was implemented on a GPU. The following year, the same challenge, now with a larger dataset was won by ZFNet [6]. It had the top-5 error of 14.8%, however even more so important is that it was able to reduce the first layer filter size from to and had a stride of 2, rather than 4 in the pooling layer.

VGG16 is a convolutional neural network model proposed in the paper [7]. This model achieved 92.7% top-5 test accuracy. The main contribution of this model was that it used kernel sized filters, instead of the . It was trained for weeks using GPUs, and had a huge computational cost. However, it introduced a new idea using the same kernels throughout the entire architecture, this aided in generalization for classification problems outside of what they were originally trained on. If for a second we go back to LeNet [8] that was the foundation for all of these previously mentioned CNNs we can observe the main sequence of three layers convolution, pooling and non-linearity still play the key part, and sometimes it is beneficial not to import to many layers when training a smaller dataset [9]. Finally, in recent years transfer learning [10], which addresses cross domain learning problems by extracting useful information from data in a related domain and transferring them for being used in target tasks, has been demonstrating a significant impact.

Coronavirus disease 2019 (Covid19) is an infectious disease caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2) [11]. It was first identified in December 2019 in Wuhan, Hubei, China, and has resulted in widespread pandemic. At the moment Covid19 has caused over one million deaths all around the world and counting. There are several methods that include quick testing, however in order to grasp the full scope of the problem some of the most important ways to battle this disease is to examine the computed tomography (CT) scan images. Chest CT scanning in patients with Covid19 has shown ground-glass opacification, possibly with consolidation, as well as cases of pleural effusion, pleural thickening and lymphadenopathy. Data is collected daily and it is still scarce, howbeit one of the first datasets was proposed and constructed for a marvellous paper [12], that is still

pending publication at the moment, in order to try and aid the ongoing battle against the pandemic. In it multi-task learning and contrastive self-supervising learning were used and achieved an accuracy of 89% in distinguishing between CT_COVID and CT_NonCOVID images. In addition to all that was stated beforehand the pivotal goal of this paper is to try and implement various CNNs to combat this classification problem and if possible obtain a slightly better classification accuracy.

This paper is organised in the following manner: section 2 represents a description of the dataset. In section 3 the main methods which are used are explained in detail, as well as the architecture of the CNN. As a result, in section 4 we discuss the results and compare the methods, based on accuracy and loss functions. In section 5, following a short summary a conclusion is made and future work and possible directions are stated.

II. DATASET AND ITS IMPLEMENTATION

The dataset which is used in this paper consists of 672 CT scan images from patients [13], including 349 CT_COVID and 323 CT_NonCOVID. We took the approach of data augmentation, where we increase the diversity of data by altering the original samples using translation, rotation, shearing, flips and adding them to the training set. Data augmentation covers a wide range of techniques used to generate new training samples using the original input images, by applying random jitters and perturbations in such a manner as to not change the class labels. The main idea here is to decrease the generalization error of the testing (sometimes at the expense of the training error) so as to achieve an increase of generalizability of the model. The neural network is then using slightly modified versions of the input data and it is able to learn more robust features.

However, we introduced scaling of the data, as well, by computing a weight for each class during the training and as an outcome amplifying the loss by a larger weight when we approach the smaller dataset. Even though the difference is small in this example, this benefited the training process. During the preprocessing of images we resized all the images to a fixed size $32x23$, and in doing so we also maintained the aspect ratio. The reasoning behind this being that all the images in a dataset need to have a fixed feature vector size. This means all the images will have identical widths and heights, making it easier to quickly load and preprocess a dataset and briskly move through our convolutional neural network. The aspect ratio will enable us to resize the images along the shorter dimension, be it width or height, and in cropping it, will maintain the ratio. It is important to note that this step is not necessary if you are not working with a difficult dataset. Notwithstanding its benefits, it was implemented in this particular dataset.

A. *ImageNet dataset*

ImageNet is a dataset consisting of over 14 million images, which belong to one thousand classes. It was used as the dataset in the highly respected convolutional neural network model VGG16 which was proposed by Oxford scientists. In this paper the VGG16 network was used as a pre-trained convolutional neural network, in order to incorporate transfer learning.

III. METHODS DESCRIPTION

In order to try and reduce overfitting and increase our classification accuracy on the CT_COVID dataset we endeavour in performing three types of neural network training techniques:

- dropout and decay
- batch normalization
- transfer learning (neural networks as feature extractors)

The first technique that is used in order to improve the generalization error in the convolutional neural network is dropout [14]. Dropout is nothing more than a form of regularization, which succours us in controlling the model capacity. The dropout layers are arranged in the network in such a manner that we have randomly disconnected nodes by a probability of 0.3 in the first few layers; and 0.6 probability in the last layer. The reason for this is that if the first layers are dropped by a higher probability, then that will later affect the training. The dropout is implemented after the pooling layer, and before the next convolutional layer (or last flatten and dense layers). This was used for the neural networks resembling the VGG with data augmentation (DA). The network resembling the VGG without DA used a dropout with a probability 0.25 in the first few layers and double the increase in the last layer, while the LeNet network did not utilize this method. Decay that is used in this neural network is a standard decay that can be obtained using the Keras library in Python. Since the learning rate controls the step that is made along the gradient, larger steps are usually used in the beginning to make sure that we do not stagnate in the local optima, while smaller steps are used deeper in the network and near the end of the convolution in order to converge to a global minimum. We have initialized the learning rate to be 0.01 (for the networks with DA) and 0.05 (for the network without DA), and applied the following formula to adjust it after each epoch,

$$\alpha_{i+1} = \frac{\alpha_i}{1+k \cdot i} \tag{1}$$

where α is the current learning rate, i is the epoch and k is the decay calculated as the division between the learning rate and the number of epochs. This type of adjustment of the learning rate each epoch, can increase accuracy, as well as reduce the loss function and the time necessary to train a network. Batch normalization [15] is used to normalize the activations of a given layer's inputs by applying mean and standard deviation before passing it onto the next layer. In addition, the covariate shift refers to a change in the distribution of the input variables which are present in the training and validation data. Since it has been proven that the training of the neural network is the most coherent when the inputs to each layer are alike, the main intention is that even when the explicit values of inputs layers to hidden layers change, their mean and standard deviation will still remain relatively the same, thus reducing the covariate shift. Batch normalization has demonstrated an immensely effective approach to reducing the number of epochs necessary for training by allowing each layer to learn independently. Here the idea that differs from the original paper and is first proposed in [16] states that the batch

normalization should be implemented after the activation layer. The main reasoning behind this is that we want to avoid setting the negative values coming out of the convolution layer to zero. Instead we pass them through the batch normalization layer, right after the activation (ReLU) layer, and assure that some of the features that otherwise would not have made it do. This yields a higher accuracy and lower loss, and is to this day a debate amongst the creators of Keras.

Finally, the second technique is transfer learning [17], a machine learning technique where networks can behave as feature extractors. Transfer learning is nothing more than the ability to use a pre-trained model to learn patterns from data, on which the original network was not trained on. As previously stated deep neural networks trained on a large scale dataset ImageNet have demonstrated to be superb at this task.

When treating networks as feature extractors we choose a point, in this case before the fully connected layer and remove it. Subsequently, in this particular example while using the VGGNet pre-trained on the ImageNet we removed the fully connected layer and stopped at the last pooling layer where the output shape is $7 \times 7 \times 512$, 512 filters with the size 7×7. Now, our feature vector has $7 \times 7 \times 512 = 25088$ values and it will be used to quantify the contents of the images, which were not included in the original training process. The format which allows us to extract these features is the hierarchical data format version 5 (hdf5), which is used to store and organize large amount of data.

Transfer learning is an optimization, which has been proven to yield a better performance and drastically save time. This is precisely why we used it in this paper, to see if we could obtain a higher classification, and perform faster. Transfer learning relaxes the hypothesis that the training data must be independent and identically distributed with the test data, which we clearly stated as a must in the beginning of this chapter. Moreover, transfer learning is able to solve the problem of insufficient training data. Furthermore, there is the option to remove the fully connected layers of the existing network in order to add a new fully connected layer to the CNN and fine tune the weights to recognize object classes. However, here it was not implemented since treating networks as arbitrary feature extractors was enough.

A. *Convolutional Neural Network*

Into the bargain all that was explained, we picked the following CNN architecture shown in Fig. 1. It is consisted of multiple convolutional and pooling layers, as well as the fully connected layers. The first two convolutional layers learn 32 filter each with a size 3×3.

Sequentially, the fourth and the fifth layers learn 64 filters with the size 3×3 and the last two learn 128 filters with the size 3×3. The pool layer is used to reduce the computational load and the number of parameters, thus reducing the risk of overfitting. We used a max pooling layer with a pool size 2×2 and a stride 2. Finally, we have the fully connected layer which consists of 2048 parameters, input values which learn 512 nodes. The activation layers which were used are Rectified Linear Unit (ReLU) defined as,

$$f(x) = \max(0, x) \tag{2}$$

where x is the input into the neuron. Softmax or the normalized exponential function assigns normalized class probabilities for each prediction, and is represented by,

$$S(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{k} e^{y_j}} \tag{3}$$

for $i = 1, \dots, k$ and $\mathbf{z} = (z_1, \dots, z_k) \in \mathbb{R}^k$.
Softmax takes an input vector and normalizes it into a probability distribution between $[0,1]$. Therefore the sum of all output values is equal to 1, which in turn makes the training converge more quickly. In order to achieve this, before training we must include one hot encoding in order to convert the labels from integers to vectors.

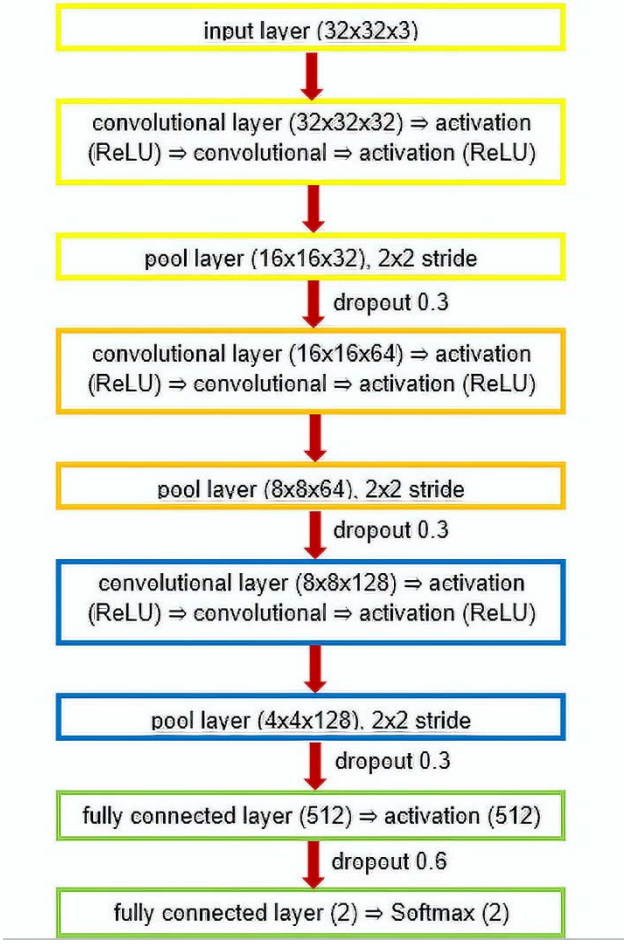

Fig. 1 A schematic of the convolutional neural network without batch normalization, that resembles the VGGNet. All of the convolutional layers that precede the fully connected layers have filters 32, 64, 128 that are the same size . The probability distribution is applied in the last layer using Softmax and the output yields two class labels CT_Covid and CT_NonCovid.

In addition, later when we want to add the batch normalization layer, we can apply it after each activation layer, as discussed previously.

B. *Implementation and training of a simpler version of the LeNet*

Taking into the bargain all that was explained before, the implementation of this CNN was done by using the Python programming language. We used Keras [18] which is mainly used for implementing of activation

functions, optimizers, convolutional and pooling layers, and is actually able to do backpropagation automatically.

Right after we load and preprocess our images dataset it is necessary to use one hot encoding. This is done by using a part of the Sklearn library LabelBinarizer. However beforehand we must split the training data and the validation data, here we opted to split it 75% and 25%, sequentially. The next step is the implementation of an optimizer, here we used the Adam optimizer. The Adam optimizer is short for Adaptive Moment Estimation optimization algorithm [19]. Its main purpose is to attempt to rectify the negative effects of a globally accumulated cache by converting the cache into an exponentially weighted moving average, just like the Root Mean Square Propagation (RMSProp). The Adam optimizer is essentially a combination of momentum and RMSProp. Momentum is implemented into the neural network, by adding a temporal element to the update vector of the past time step to the current update vector,

$$\Delta \boldsymbol{w}(k) = -\alpha \nabla E(k) + \gamma \Delta \boldsymbol{w}(k-1) \quad (4)$$

where γ is usually set between 0.8 and 0.9 and function E is the index of performance.

This network resembles the architecture of the LeNet in such a way that we have 5×5 filters with a stride of 20 in the first convolution layer, and 50 in the second convolutional layer. The mini batch method were the neural network selects a part of the training data and updates the weights, but trains the network with the average weight update. Usually the smallest standard batch size which is used is 32, however we opted to use 24, as it complemented our data. The reasoning behind this is that present research confirms that using small batch sizes achieves the best training stability and generalization performance, for a given computational cost, across a wide range of experiments. The loss function which was used is the binary_crossentropy function. This was done because we only had two classes, if there were more we would have had to use categorical_cross_entropy, but have in mind we could have used categorical as well, but studies show that binary is much more efficient in this case. The training of the CNN was done in 30 epochs.

C. *Implementation and training of a simpler version of the VGGNet*

We constructed two different neural networks resembling the VGG, the first one had a dropout of 0.25 in the first few layers and no data augmentation or batch normalization layers. The training data and the validation data were split 75% and 25%, sequentially. Here we utilized the SGD optimizer, which was set to a learning rate of 0.05, with a decay in order to slowly reduce the learning rate over time and converge to the global solution more efficiently. Decaying the learning rate is beneficial in reducing overfitting and obtaining a higher classification accuracy. The smaller the learning rates are, the smaller the weight update will be enabling us to converge. The gradient descent method is an iterative optimization algorithm that operates over an optimization surface. It is a simple modification to the standard algorithm of gradient descent. The main purpose of SGD is to calculate the gradient and adjust the weights of the training data (but not on the whole dataset, but rather on a mini batch). All the images were resized to 32×32 aspect ration, the batch size we used was again 24 and the loss fucntion was the binary_crossentropy function. The training of the CNN was done in 30 epochs.

The second neural network resembling the VGG had a similar architecture as depicted in Fig. 1, with the addition of batch normalization layers. Here our training data and the validation data were split 80% and 20%. Here we trained our network once with the SGD optimizer and once more with the Adam optimizer. The learning rate was set to 0.01, with decay and adjustment after each epoch. Both times data scaling, as well as data augmentation was used. All the images were resized to 32×32 aspect ration, the batch size we used was 32 and the loss function was the binary_crossentropy function. The training of the CNN was done in 30 epochs (utilizing the Adam optimizer) and in 100 epochs (using the SGD optimizer).

After the training we implemented a method that takes the weights and the state of the optimizer and serializes them to the disc in a hdf5 format, in order to load them and test the labeling.

D. *Implementation using transfer learning*

The first step in this process is to extract features from VGG16, in doing so we are forward propagating the images until a given layer, and then taking those activations and treating them as feature vectors. Here the main two differences are that we used the standard a batch size of 32 and the training and test split is done at the same time as training, we again split it into 75% training data and 25% test data. Once the extraction of the features was done, we trained the classifier on those features. We also implement the GridSearchCV class to assists us to turn the parameters to the LogisticRegression classifier.

The final results are presented in the following chapter, comparisons are made and a visual representation of the graphs is shown using Matplotlib in order to estimate if there is overfitting.

IV. Results and Comparisons

The results of the CNN resembling LeNet are presented in Table 1. We clearly see that our neural network has classification accuracy of 68%.

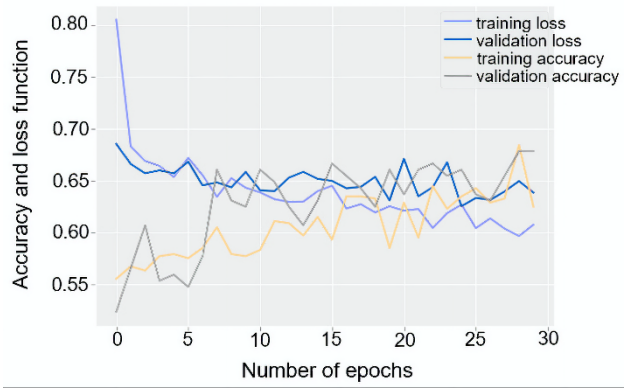

Fig. 2 A graph depicting a convolutional neural network that resembles the LeNet – training and validation loss and accuracy curves

In the following table we use the term precision which represents true positive divided by a sum of true positive

and false positive, recall which represents true positive divided by a sum of true positive and false negative. Therefore, precision is good to determine when the cost of false positives is high, on the other hand recall tells us the number of correctly labeled data. Ultimately, we have the f1-score used to find the weighted average of recall and precision. In analyzing the curves shown in Fig. 2 we see that our network learned until the 30 epoch, beyond that overfitting would occur, as we can clearly see a generalization gap forming in both loss and accuracy curves. Fig. 3 depicts the results when using the network resembling the VGG without data augmentation, here we can observe that the training and validation curves show a wide generalization gap at the 30 epoch resulting in overfitting. The classification accuracy is 76% (Table 1.), this is no good if we have overfitting, that is why the next approach uses data augmentation in order to combat this problem.

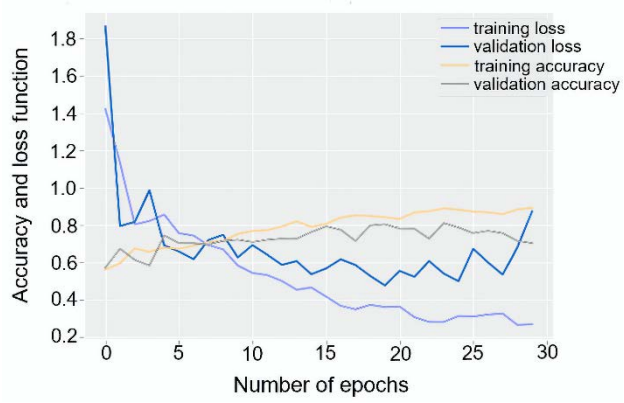

Fig. 3 A graph depicting a convolutional neural network without data augmentation and with batch normalization, that resembles the VGGNet – training and validation loss and accuracy curves

Fig. 4 represents the neural network resembling the VGG, with data augmentation and the SGD optimizer. The classification accuracy obtained after 30 epochs is 72%, and the training and loss curves show slight deviations.

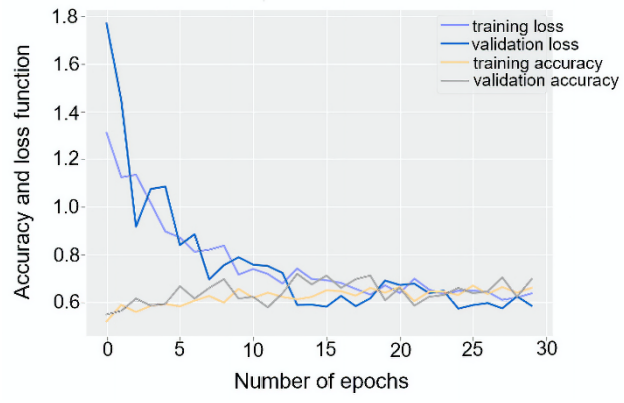

Fig. 4 A graph depicting a convolutional neural network with data augmentation and batch normalization (optimizer SGD), that resembles the VGGNet – training and validation loss and accuracy curves – 30 epochs

The same classification accuracy is acquired when utilizing the Adam optimizer, only then we need 100 epochs to achieve so. Fig. 6 depicts the same neural network explained beforehand when using the SGD optimizer over the course of 100 epochs resulting in a classification accuracy of 75%. We can conclude that data augmentation does indeed help in reducing the generalization gap, however this particular dataset was quite faulty to begin with.

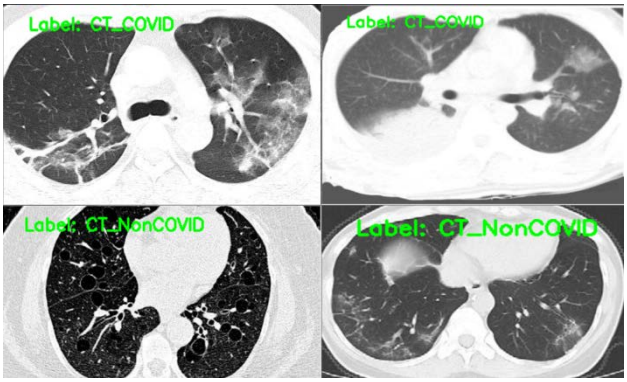

Fig. 5 The pre-trained CNN weights are loaded from the disk and make predictions for 30 randomly selected images. In the upper left and right corner we have an example of CT_COVID scans, and in the lower left and right corner an example of CT Non_COVID scans.

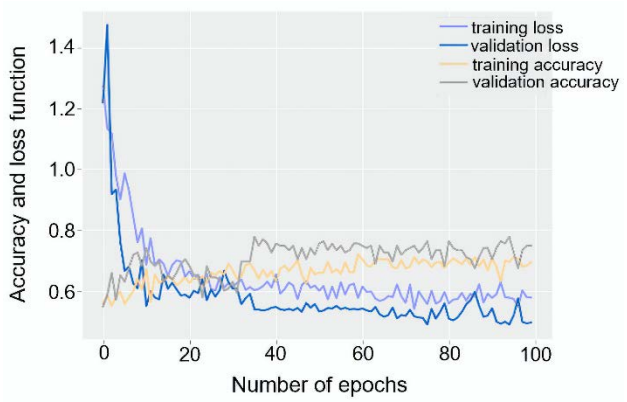

Fig. 6 A graph depicting a convolutional neural network with data augmentation and batch normalization (optimizer SGD), that resembles the VGGNet – training and validation loss and accuracy curves – 100 epochs

TABLE I
EXPERIMENTAL RESULTS

	precision	recall	f1-score
CNN resembling LeNet (Adam optimizer, with data augmentation)			
macro avg	0.68	0.68	0.67
CNN resembling VGG (SGD optimizer, without data augmentation, without batch normalization)			
macro avg	0.76	0.71	0.69
CNN resembling VGG (Adam optimizer, with data augmentation, 100 epochs, with batch normalization)			
macro avg	0.72	0.72	0.72
CNN resembling VGG (SGD optimizer, with data augmentation, 30 epochs, with batch normalization)			
macro avg	0.72	0.70	0.69
CNN resembling VGG (SGD optimizer, with data augmentation, 100 epochs, with batch normalization)			
macro avg	0.75	0.75	0.75
Transfer learning using VGG16			
macro avg	0.90	0.91	0.90

In Table 1 we can see the results obtained by using transfer learning have a classification accuracy of 90%,

which is by far the best. Furthermore, we observe that the CNN in Fig. 4 is the best one if we opted to use a method that does not include transfer learning. Nevertheless, it is clear then when taking into account all four approaches we shall choose transfer learning, because not only does it yield a higher classification accuracy, but it also wasted less computational time. Compared with the original paper that combated this classification problem [12] we were able to achieve only slightly better classification accuracy, with an increase being 1%.

V. Conclusions

In this paper we described four different approaches of using convolutional neural networks to classify a dataset consisting of CT_COVID and CT_NonCOVID images. We used CNNs that we constructed based on the VGGNet and LeNet5 and implemented them with and without data augmentation. Furthermore, we used a transfer learning technique by extracting features of the neural network VGG16 trained on the ImageNet dataset. The main idea of this paper was to see if a different approach can have better results on this particular dataset, as well as see if a smaller neural network could have almost as good classification as transfer learning. The final results, when compared showed a clear advantage when using transfer learning, however it also showed us the importance of data augmentation when approaching a rather small dataset.

Further research will focus on implementing different types of optimizers, including metaheuristic algorithms as optimizers. Also, we will focus on battling larger datasets consisting of Covid19 CT scans, once they become available, as well as obtaining a higher classification accuracy utilizing different methods.

Acknowledgment

This research was supported by the Science Fund of the Republic of Serbia, grant No. 6523109, AI-MISSION4.0, 2020-2022.

This paper was conceived within the research on the project: "*Integrated research in the field of macro, micro and nano mechanical engineering - Deep machine learning of intelligent technological systems in production engineering*", The Ministry of Education, Science and Technological Development of the Republic of Serbia (contract no. 451-03 -68 / 2020-14 / 200105), 2020.

This work was financially supported by the Ministry of Education, Science and Technological Development of the Serbian Government, Grant TR-35029 (2018-2020).

References

[1] L. Laban, R. Jovanović, M. Vesović, V. Zarić, "Classification of Chest X-Ray Images Using Deep Convolutional Neural Networks", International Conference on Electrical, Electronic, and Computing Engineering, Belgrade, Serbia, September 28-30, 2020, to be published.

[2] K. Fukushima, S. Miyake, "Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position," Pattern Recognition, vol. 15, no. 6, pp. 455-469, 1982.

[3] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, L. D. Jackel, "Handwritten digit recognition with a back-propagation network," Advances in Neural Information Processing Systems 2, pp. 396-404, June, 1990.

[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Haung, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, "ImageNet large scale visual recognition challenge," International Journal of Computer Vision, vol. 115, no. 3, pp. 211-252, April, 2015.

[5] A. Krizhevsky, I. Sutskever, G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Advances in Neural Information Processing Systems, vol. 25, no. 2, pp. 1097-1105, 2012.

[6] M. D. Zeiler, R. Fergus, "Visualizing and understanding convolutional networks," 13th European Conference, Zurich, Switzerland, pp. 818-833, September 6-12, 2014.

[7] K. Simonyan, A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[8] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, n. 11, pp. 2278-2324, November, 1998.

[9] Z. Li, W. Yang, S. Peng, F. Liu, "A survey of convolutional neural networks: Analysis, applications and prospects," arXiv preprint arXiv:2004.02806, 2020.

[10] L. Shao, F. Zhu, X. Li, "Transfer learning for visual Categorization: A survey," IEEE Transactions on Neural Networks and Learning Systems, vol. 26, no. 5, pp. 1019-1034, May, 2015.

[11] https://www.mayoclinic.org/diseases-conditions/coronavirus/symptoms-causes/syc-20479963, (last accessed 13/10/2020).

[12] X. Yang, X. He, J. Zhao, Y. Zhang, S. Zhang, P. Xie, "COVID-CT-Dataset: A CT Scan Dataset about COVID-19", https://arxiv.org/abs/2003.13865, 2020, to be published.

[13] Dataset provided on GitHub, https://github.com/UCSD-AI4H/COVID-CT, (last accessed 23/07/2020).

[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", arXiv preprint arXiv:1207.0580, July, 2012.

[15] S. Ioffe, C.Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", Proceedings of the 32nd International Conference on Machine Learning, vol. 37, pp. 448-456, July, 2015.

[16] A. Rosebrock, Deep Learning for computer vision with Pyhton: Starter Bundle, 1st ed. PyImageSearch, 2017.

[17] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, C. Liu, "A survey on deep transfer learning," 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018.

[18] Keras; Python Deep Learning Library https://keras.io, (last accessed 09/03/2020).

[19] D. P. Kingma, J. Ba, "Adam: A Method for Stochastic Optimization", 3rd International Conference for Learning Representations, San Diego, 2015.