Handwritten Digits Classification Using Convolutional Neural Networks

Sumaiya Deen Muhammad . Naga Jyothirmayee Dodda. Roisul Islam Rumi

School of Computer Science,

University of Windsor, Windsor, Ontario, Canada
{deenmuh, doddan, rumir}@uwindsor.ca

Abstract: Convolutional Neural Networks (CNNs) are well-known approaches in the field of Computer Vision. Image classification is one of the most important and high-demand areas of Computer Vision systems. Numerous image classification models have been developed to improve recognition accuracy. In this paper, we have proposed custom-built CNN architecture to classify handwritten digits. We have experimented using four activation functions (ReLU, LeakyReLU, ELU, PReLU) on the renowned MNIST dataset. From this we have picked ReLU and LeakyReLU for further tuning as they were the best performers. After multiple passes on different setups, we found our best result with the CNN architecture with two convolutional Blocks, ReLU activation function, Stochastic Gradient descent with a Learning rate of 0.1, Momentum 0.05, and L2 regularization that gave us the highest accuracy of 98.73% and a loss of 0.05.

Keyword Convolutional neural network, MNIST, image classification, deep learning, handwritten digit recognition.

1 Introduction

Computer vision consists of different techniques, for instance, pattern recognition, image classification, activity recognition, object detection, etc. In the last couple of decades, enormous developments have been achieved in the area of computer vision systems. Specifically remarkable results have been procured in image classification. Many techniques and algorithms have been designed to classify images accurately. Recently, numerous powerful deep learning techniques and convolutional neural network (CNN) architectures have been developed which have proved outstanding accomplishments to solve computer vision-based problems, particularly in image classification (Sultana, Sufian & Dutta, 2018).

At present digit recognition in handwritten format is an important domain of research in computer vision systems. Countless applications have been developed in recent years which require detecting handwritten digits and letters accurately These are certainly complex systems as the main challenge of these applications are to read hand writings of different patterns because handwriting styles varies immensely (Ahlawat & Choudhary, 2020). In the field of handwriting recognition, plenty of algorithms are being used to distinguish handwritten digits, including Decision Tree, CNN, Random Forest, Gaussian Naïve Bayes, SVM, etc. (Javed, 2020).

The report presents different techniques for handwritten digits recognition and the experiments are tested on a well-known database called MNIST database (Modified National Institute of Standards and Technology database).

2 Literature Review

Many researchers already contributed a lot in this area. We discuss some remarkable research works which relate to our paper.

In (Larsen, Noever & MacVittie, 2021) Larsen et al. have experimented on the Overhead-MNIST dataset, which contains satellite images with ten labels to predict. They experimented with twenty-three machine learning algorithms and a few deep learning models to identify baseline comparison metrics to embed in a mission-critical system that can be deployed on edge computers. The dataset they worked on is a balanced dataset, and as a pre-processing step, they only conducted normalization. The training set is divided into training data and 20% validation data. They picked categorical accuracy as the evaluation metric with 10-fold stratified cross-validation while testing unseen data. Among the ML algorithms, the CatBoost Classifier achieved the best accuracy of 83%, and the worst performer was the Logistic Regression, with an accuracy of 45%. However, their hyperparameters tuned CNN achieved the best accuracy of 97% after 500 training epochs, making them opt for this network.

This research work (Ali et al., 2020) proposed a new architecture by combining CNN architecture with a Java-based framework called Deeplearning4j (DL4J), for recognition and classification of the MNIST dataset which is publicly available. In terms of hyperparameter tuning, every layer in this CNN architecture is trained only with Stochastic Gradient Descent (SGD) Algorithm as an optimizer and other parameters such as weights to achieve the optimal results. Herein, they have proposed a state-of-the-art CNN framework that can significantly perform in the image classification field with 99.37 % classification accuracy.

Cheng et al. (Cheng, Tahir, Eric & Li, 2020) demonstrates a comprehensive analysis using GAN (Generative Adversarial Networks (GANs) and its various models which are most promising generative frameworks in the area of image synthesis. To do so, they implemented various GAN variants on MNIST dataset. The authors in (Sultana, Sufian & Dutta, 2018) investigated different CNN model on ImageNet dataset and found that combination of inception model and residual blocks with conventional CNN model, Googlenet and ResNet received better accuracy. A hybrid model has been proposed in (Ahlawat & Choudhary, 2020) that consists of CNN and SVM in order to recognize digits from MNIST dataset and their robust model perceived 99.28% accuracy.

3 Methodology

Our experiments started with constructing a neural network with a couple of fully connected layers wherein there are no convolutional layers involved and have developed three custom convolutional networks from which we have identified the one with highest performance (in terms of highest accuracy and lowest loss) when tuned its parameters and compared the results with VGG16, a predefined architecture. Below sections provide a detailed description of each custom network built.

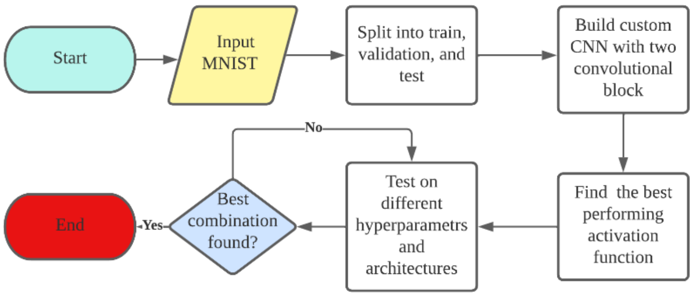

Start
Input MNIST
Split into train, validation, and test
Build custom CNN with two convolutional block
Find the best performing activation function
Test on different hyperparametrs and architectures
Best combination found?
No
Yes
End

Figure 1: Flowchart of proposed system

3.1 Dataset

LeCun et al. (LeCun, Cortes & Burges, 2012) published the MNIST (Mixed National Institute of Standards and Technology) database in 1998. Since then, this dataset has been frequently utilized as a testbed for many machine learning and pattern recognition projects. The MNIST database has 70,000 instances, 60,000 of which are for training and the remaining are for testing. There are 10 classes of handwritten numbers from 0-9. The images are grayscale, and their size is 28 X 28.

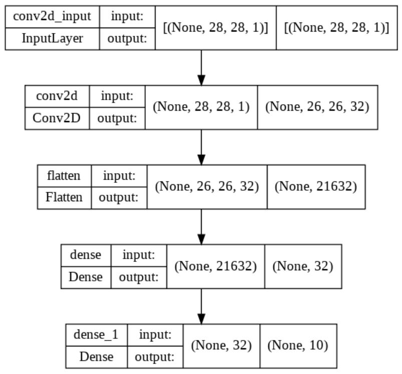
Figure 2: Samples from the MNIST dataset.

3.2 CNN

Consisting of a cluster of kernels, Convolutional Neural Network with its ground-breaking layer has become well-known for image categorization applications. It is a black box classifier which consists of five basic components: structure, kernel, receptive, and field, layer count, and feature maps (Chang, & Sha, 2016). CNN extract features from a picture on a hierarchical basis. This is determined by the layers, such as lower-level layers information such as edges, corners, and so on. On the other hand, the higher-level layers extract more complex features like object shape, color information, etc. CNN works by extracting the features from the images based on the kernel sizes, then these get passed on the higher layers. After that, the features are routed through the fully connected layers and flattened into a single-dimensional array. that is then fed input into the classifier. The classifier outputs the result as a probability of classes which sums to one. The higher probability score defines the class a particular image belongs to.

1. ***Simple Convolutional Neural Network:***

   In the Simple Convolutional Neural Network architecture, we have considered only the fully connected layers to train and validate the data for image classification in the system. There is only one Convolutional 2D layer in this architecture. Below Fig. 3 is the schematic summary of the architecture used.

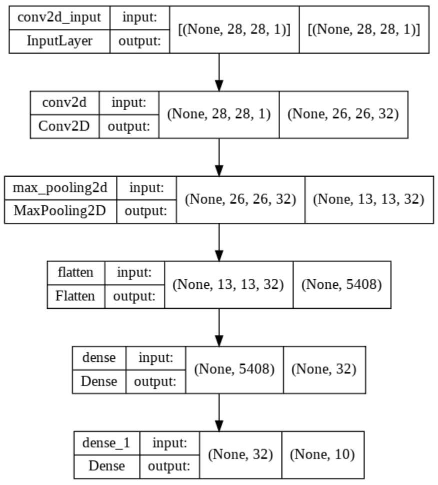


conv2d_input | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 26, 26, 32)
flatten | input: | (None, 26, 26, 32)
Flatten | output: | (None, 21632)
dense | input: | (None, 21632)
Dense | output: | (None, 32)
dense_1 | input: | (None, 32)
Dense | output: | (None, 10)

Figure 3. Simple Convolutional Neural Network

2. ***CNN with one convolutional Block:***

   In the Simple Convolutional Neural Network architecture, we have considered only the fully connected layers to train and validate the system's image classification. There is only one Convolutional Block in this architecture. The term one convolutional block in this context is defined as (Conv 2d Layer + activation function + Maxpooling 2D Layer). Below Fig. 4 is the schematic summary of the architecture used.



conv2d_input | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 26, 26, 32)
max_pooling2d | input: | (None, 26, 26, 32)
MaxPooling2D | output: | (None, 13, 13, 32)
flatten | input: | (None, 13, 13, 32)
Flatten | output: | (None, 5408)
dense | input: | (None, 5408)
Dense | output: | (None, 32)
dense_1 | input: | (None, 32)
Dense | output: | (None, 10)

Figure 4. CNN with one convolutional Block

3. ***CNN with two convolutional blocks:***

   In the Simple Convolutional Neural Network architecture, we have considered only the fully connected layers to train and validate the data for image classification in the system. There are only two Convolutional blocks in this architecture. Below Fig. 5 is the schematic summary of the architecture used.

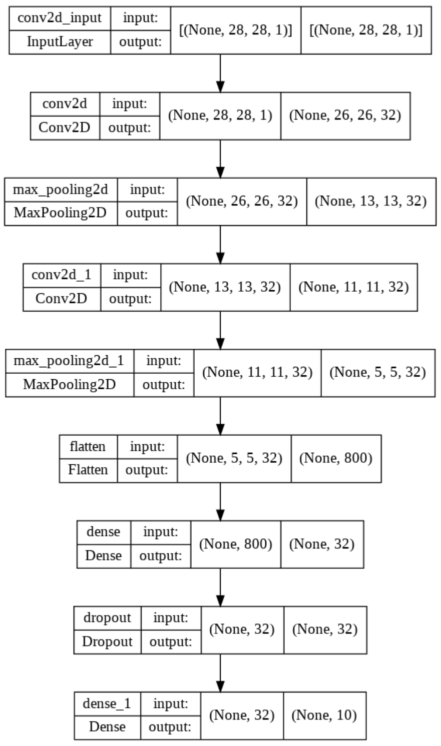

conv2d_input | InputLayer | input: [(None, 28, 28, 1)] | output: [(None, 28, 28, 1)]
conv2d | Conv2D | input: (None, 28, 28, 1) | output: (None, 26, 26, 32)
max_pooling2d | MaxPooling2D | input: (None, 26, 26, 32) | output: (None, 13, 13, 32)
conv2d_1 | Conv2D | input: (None, 13, 13, 32) | output: (None, 11, 11, 32)
max_pooling2d_1 | MaxPooling2D | input: (None, 11, 11, 32) | output: (None, 5, 5, 32)
flatten | Flatten | input: (None, 5, 5, 32) | output: (None, 800)
dense | Dense | input: (None, 800) | output: (None, 32)
dropout | Dropout | input: (None, 32) | output: (None, 32)
dense_1 | Dense | input: (None, 32) | output: (None, 10)

Figure 5 CNN with two convolutional blocks

Layer-1: First convolutional layer incorporates an activation function. The convolutional layer gets the input image of the size N × N=28 × 28 and the convolution filter of size F×F=3 × 3. This layer has a padding P is 0, the stride is one, and uses 32 filters. After completing the convolution operation, a feature maps size of 32@26×26 is obtained in which 32 total feature maps equal to the number of filters are used. 26 is derived from this mathematical operation
O=(N−F+2P/S) +1 = 28−3+2(0)/1+1=26. Afterward activation function is applied on each feature map.

Layer-2: is the max-pooling layer. This layer receives the input from the previously stacked layer of size 32@26×26. After this max-pooling operation, we obtain feature maps of size 32@13×13. We get the same number of feature maps as the previous layer because pooling is done in each feature map, and 13 comes from a similar mathematical operation. The pooling operation is depicted in

Layer-3: First convolutional layer incorporates an activation function. The convolutional layer gets the input image of the size N × N=13 × 13 and the convolution filter of size F×F=3 × 3. This layer has a padding P is 0, the stride is 1, and uses 32 filters. After the completion of the convolution operation, a feature maps size of 32@11×11 is obtained in which 32 total feature maps which are equal to the number of filters are used. 11 is derived from this mathematical operation mentioned earlier. Afterward activation function is applied on each feature map.

Layer-4: is the max-pooling layer. This layer receives the input from the previously stacked layer of size 32@11×11. After this max-pooling operation, we obtain feature maps of size 32@5×5. We get the same number of feature maps as the previous layer because pooling is done in each feature map, and five comes from a similar mathematical operation.

Layer-5: Flatten Layer receives input image of the size N × N=5 × 5. After the completion of the convolution operation, a feature maps size of 32@5×5 is obtained in which 32 total feature maps which are equal to the number of filters are used. Received input from Conv 2D layer is converted into output in 1D with size 800. Afterward activation function is applied on each feature map.

Layer-6: This fully connected layer receives input from the flattened layer. It receives an input of 1D vector with size 800, and it gives output in 1D with size 32. To send data to the final output layer. It uses an activation function on each feature map.

Layer-7: This dropout layer receives input from the flattening layer. It receives an input of 1D vector with size 32, and it gives output in 1D with size 32 with a dropout value of 0.5.

Layer-8: The last layer of the proposed model is also called an output layer. It receives an input of 1D vector with size

32, and it gives output in 1D with size 10. To receive the final output by utilizing SoftMax. Above schematic diagram shows the whole description of the proposed three-layer CNN architecture.

In this study, we have worked on activation functions (Relu, LeakyReLU, PReLU, ELU), optimizers (Adam optimizer and Stochastic Gradient Descent optimizer), Loss functions (Categorical cross-entropy, Mean Squared Error), regularizers (L1 and L2) with a trial-and-error approach to find out the best possible outcomes.

VGG16:

VGG-16 is considered as one of the most powerful CNN architecture in the area of computer vision. It is a light weight deep neural network that made records in ImageNet classification in 2014, reaching more than 90% accuracy and it was considered as a state of the art at that time. It is also known as an extended version of AlexNet. This model consists of 3 X 3 convolutional layers with a stride 1. It always uses maxpooling layer of 2 X 2 filter of stride 2. Also, it has 16 layers that have weight ("Step by step VGG16 implementation in Keras for beginners", 2019). Due to its high performance and ease of implementation it is very popular architecture for image classification.

For our experiments we used a 2D CNN as our input images are two-dimensional to identify the best performed activation function first, with the approach mentioned below:

To select the activation function, we have used a custom CNN architecture as our baseline model. Our architecture has two convolutional blocks (Conv-->Relu--> Maxpooling) and a fully connected layer. For the first convolutional layer the parameters are:
input channels = 1, output channels=16, kernel size=5, stride=1, padding=2.
For the second convolutional layer the parameters are:
input channels = 16, output channels=32, kernel size=5, stride=1, padding=2.

For our final FC layer, the parameters are : Input features=1568 (32*7*7), output features=10 (number of classes). Here, 7*7 has been calculated using the output formula for convolutional and max pooling which are as follows. Formula for convolution is as follows.

$$O = \frac{(W - K + 2P)}{S} + 1$$

Here, O is output height/length, W: input height/length, K: filter size (kernel size) = 5, and same padding (non-zero) has been used which can be defined as below.

$$P = \frac{K - 1}{2}$$

Formula for pooling:

$$O = \frac{W - K}{S} + 1$$

Here, K: filter size = 2, S: stride size = filter size, it is PyTorch default.

In stage two, we have tuned all the hyperparameters with many possible combinations such that the classification can be achieved with the highest possible accuracy compared with the existing model, i.e., *VGG16*

While experimenting on the activation functions, we kept a few of the hyperparameters fixed to have a clear understanding of the comparison. We kept the learning rate = 0.01, batch size = 60, iterations = 1000, epochs = 5, loss function = Cross-Entropy loss, optimizer = Stochastic Gradient Descent. For activation functions we tested with ReLU, ELU, Leaky ReLU, PReLU. All of them had a similar result and among them Relu is the best performer with an accuracy of 97.72%.

4. Experimental Setup

Our experiments were performed using the MNIST dataset obtained through the TensorFlow repository. The columns, which each represent a pixel value ranging from 0 to 255.0, serve as the features (independent variables) for our model. This dataset's last column labeled 'Y' was used as the target/dependent variable.

MNIST data set which consists of 70000 images of size 28 x 28 where we have considered the train, test and validation split of 60:20:20. To construct the algorithms, we use Python3 IDE in Anaconda – Jupyter Notebook. This experiment is performed on the 11th Gen Intel® Core Processor with 2.80 GHz CPU and 16GB RAM running on Windows Platform.

5. Results and Discussions

After running the CNN for 5 epochs with a learning rate of 0.01 the achieved best result of 97.53% accuracy and loss of 0.05 among all the four activation functions which are mentioned below. Among them we have picked LeakyReLU and ReLU for further experiments.

LeakyReLU loss 0.05, 97.53% accuracy, ReLU loss 0.07, accuracy 97.72%, ELU loss 0.19, accuracy 97.37%, PReLU loss 0.18, accuracy 97.23%. Below Fig 6 is the diagram for the ReLU.

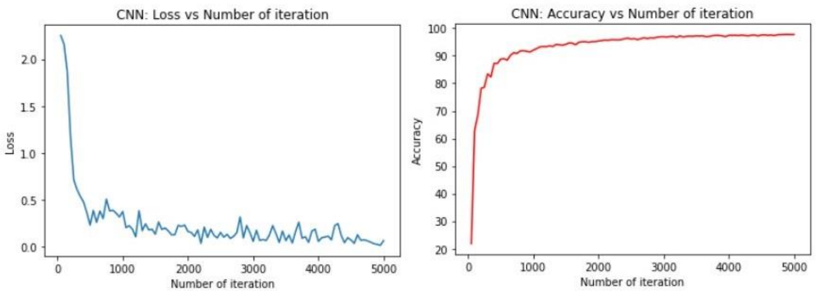


Figure 6: ReLU with two convolutional blocks.

From the first trial, we have identified “Relu and Leaky Relu” as the top two activation functions that performed well in two-layered networks. Now, we considered the Optimization functions “Adam Optimizer with a learning rate of “0.1 & 0.9” and the Stochastic Gradient descent with the Learning rate of “0.1 & 0.0005” and Momentum “ 0.9 & 0.5”, Loss functions “categorical cross-entropy and Mean Squared Error,” Regularization functions “L1 and L2”. With all the possible combinations mentioned below in Table 1, both Relu and Leaky Relu activation functions are used, and experiments were done. We have tuned the parameters of the custom-built networks - Simple Neural Network, Simple Convolutional Neural Network, CNN with one convolutional block, CNN with two convolutional blocks and ran the experiments for 10 epochs.

Table 1: Testcases considered while testing

		Without/With Regularization (L1 & L2)
Loss Functions	**Optimizers**	**Optimizers Learning rate (LR)/ Momentum(M)**
Categorical Cross entropy	Adam optimizer	LR = 0.1
		LR = 0.9
	SGD Optimizer	LR = 0.1 & M = 0.9
		LR = 0.0005 & M = 0.9
		LR = 0.1 & M = 0.05
		LR = 0.0005 & M=0.05
Mean Squared Error	Adam optimizer	LR = 0.1
		LR = 0.9
	SGD Optimizer	LR = 0.1 & M = 0.9
		LR = 0.0005 & M = 0.9
		LR = 0.1 & M = 0.05
		LR = 0.0005 & M=0.05

Below are the Graphs plotted (Fig 7 - 9) with results from each network and the mapping is with training loss to the networks corresponding validation loss and training accuracy Vs. validation loss and validation accuracy, respectively that has the highest accuracy for each network.

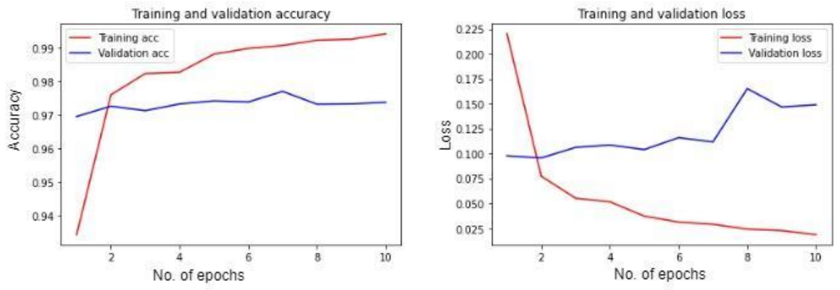


Figure 7 Simple Convolutional Neural Network

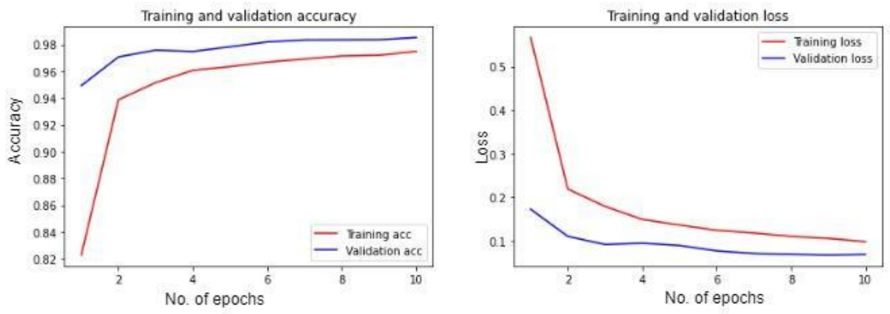


Figure 8 CNN with one Convolutional Block

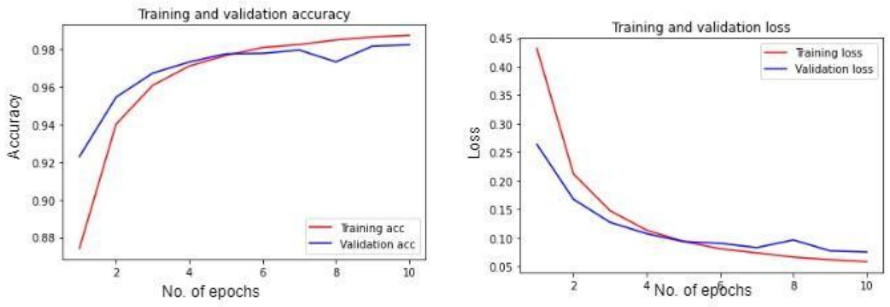


Figure 9 CNN with two Convolutional Blocks

In all the custom-built networks, Optimizer Stochastic Gradient descent with a Learning rate of “0.1” and Momentum “0.05” has outperformed Adam optimizer. Categorical cross-entropy is the best Loss function used in this classification. L2 regularization has given better accuracy along with activation function “ReLU.” Out of every possible combination mentioned in Table 1, the mentioned configuration has resulted in the highest accuracy of 98.73% and with the least loss of 0.05 in the custom-built network of “CNN with two convolutional Blocks.” The below Table 2 depicts the Testing accuracy, Testing Loss, Validation accuracy, and Validation Loss using Relu and Leaky Relu as activation functions in four custom-built networks.

Table 2: Loss and accuracy metrics highest accuracy per network.

	Neural Network type	**Testing Loss**	**Testing Accuracy**	**Vaidation Loss**	**Validation Accuracy**
Relu Activation	Simple Convolutional Neural Network	0.0603	0.9804	0.066	0.9804
	Convolutional Neural Network with one block	0.0682	0.982	0.0769	0.9812
	Convolutional Neural Network with two blocks	0.0567	0.9873	0.0719	0.9839
Leaky Relu Activation	Simple Convolutional Neural Network	0.1332	0.9762	0.149	0.9738
	Convolutional Neural Network with one block	0.684	0.9812	0.0755	0.9822
	Convolutional Neural Network with two blocks	0.0586	0.9848	0.0689	0.9852

During these experiments, along with the best-performed Custom network identification, we have identified a couple of trends.

- We have implemented a neural network without any convolutional layers, but we see that accuracy after training is around 96.44% . Fig 10 shows the accuracy and loss metrics for Training and Validation dataset in this network.
- If the learning rate is high, the algorithm learns fast, But the accuracy will be low, and vice versa gives a better performance result.
- Mean squared Error does not perform well with the Classification task as the data for this kind of data.

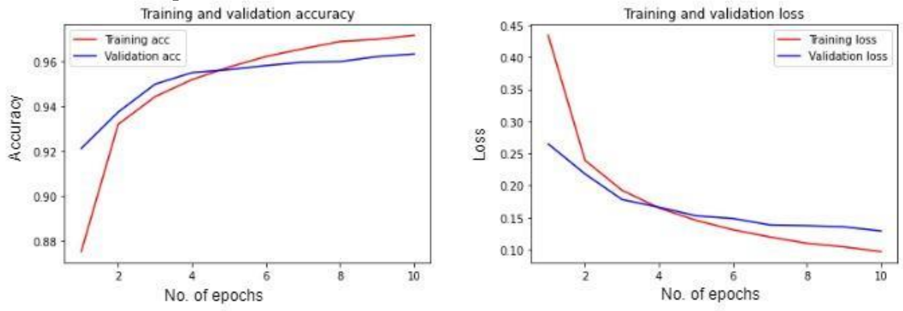


Figure 10 Simple Neural Network

In the VGG 16 model, we have set the learning rate as 0.01 and momentum is 0.9. We have used activation function RELU for the dense layer of 4096. At the end, we have used 10-unit dense layer 'softmax' activation function as we have 10 classes for classification from digit 0 to 9.

Stochastic gradient descent (SGD) optimizer has been used in this model and also, we have applied loss function 'categorical_crossentropy'. To evaluate our model, we have set metrics = accuracy.

The number of epochs we have set is 10. Our received accuracy level is 98.742% which is considered as outstanding performance. Fig 11 shows the accuracy and loss metrics for Training and Validation dataset in VGG16.

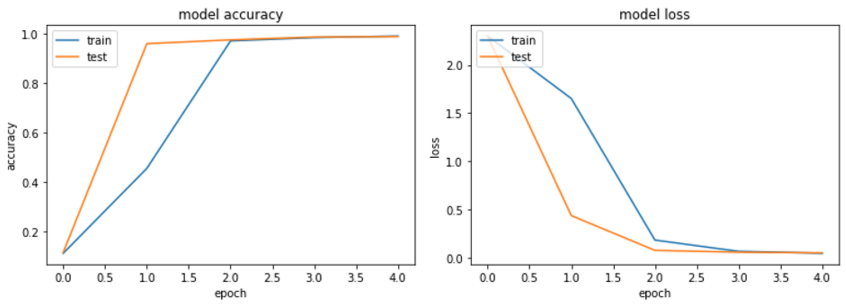


Figure 11: VGG 16 performance

6 Conclusion

In this paper, we have experimented with different setups of CNN architectures, tuned hyperparameters to obtain the best accuracy and lowest loss. After that, we compared our results with VGG16. From the results we found our best result with the CNN architecture with two convolutional blocks, ReLU activation function, Stochastic Gradient Descent with a Learning rate of 0.1, Momentum 0.05, and L2 regularization, which gave us the highest accuracy of 98.73 percent and a loss of 0.05.

References

Ahlawat, S., & Choudhary, A. (2020). Hybrid CNN-SVM Classifier for Handwritten Digit Recognition. Procedia Computer Science, 167, 2554-2560. doi: 10.1016/j.procs.2020.03.309

Ali, S., Sakhawat, Z., Mahmood, T., Aslam, M., Shaukat, Z., & Sahiba, S. (2020). A robust CNN model for handwritten digits recognition and classification. 2020 IEEE International Conference On Advances In Electrical Engineering And Computer Applications(AEECA). doi: 10.1109/aeeca49918.2020.9213530

Cheng, K., Tahir, R., Eric, L. K., & Li, M. (2020). An analysis of generative adversarial networks and variants for image synthesis on MNIST dataset. Multimedia Tools and Applications, 79(19), 13725-13752.

Chang, J., & Sha, J. (2016). An efficient implementation of 2D convolution in CNN. IEICE Electronics Express, 13-20161134.

Javed, M. (2020). The Best Machine Learning Algorithm for Handwritten Digits Recognition. Retrieved 2 March 2022, from https://towardsdatascience.com/the-best-machine-learning-algorithm-for-handwritten-digits-recognition-2c6089ad8f09

Larsen, E., Noever, D., MacVittie, K., & Lilly, J. (2021). Overhead-MNIST: Machine Learning Baselines for Image Classification. arXiv preprint arXiv:2107.00436.

LeCun, Y.; Cortes, C.; Burges, C.J.C. The MNIST Database of Handwritten Digits. 2012. Available online: http://yann.lecun.com/exdb/mnist/ (accessed on 1 Mar 2022).

Step by step VGG16 implementation in Keras for beginners. (2019). Retrieved 2 March 2022, from https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c

Sultana, F., Sufian, A., & Dutta, P. (2018). Advancements in Image Classification using Convolutional Neural Network. 2018 Fourth International Conference On Research In Computational Intelligence And Communication Networks (ICRCICN). doi: 10.1109/icrcicn.2018.8718718