CNN, MLP, and VGG16 on MNSIT Data

Tejeswar Reddy Alle, Pavansai Kuramsetti, Abhinav Mandava, Jay Gajjar
Faculty of Computer Science, Dalhousie University
6050 University Ave, Halifax, Nova Scotia, Canada, B3H 4R2

Abstract

Artificial Neural Networks are a powerful technology used for computer vision. In this paper, we used the MNIST dataset to describe the training, test accuracy of MLP, Convolutional Neural Networks, and VGG16. The MNIST dataset is the subset of the NSIT Special Database. The training set of the MNIST dataset consists of 60,000 patterns where the testing set contains 10,000 patterns. To reduce the processing time and load on GPU, we considered 1024 patterns of the training set and 100 patterns of the testing set. This dataset contains digital images of size 28*28. VGG16 model requires a larger size so digital images ate enlarged using interpolation. The result shows that VGG16 has more accuracy than MLP and CNN. CNN has more accuracy than MLP. Increase in training data from 1024 to 60000 results in an increase of accuracy in both MLP and CNN.

1. Introduction

The main aim of this paper is to give a quick start on working with Neural Networks and Keras. Neural Network is a learning system with the combination of artificial neurons to understand the input and predict the desired output. The concept of neural networks is inspired by neurons in humans. We started working with a different type of neural network algorithms like Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN) using the well known MNIST database to find the difference in accuracies of the algorithms. MNIST database is a collection of handwritten digits with 60,000 training images and 10,000 testing images. We can use MLP and CNN for classifying images but MLP takes vectors as input and CNN takes tensor as input. We have started working parallelly on MLP and CNN then identified the different tuning parameters in both the algorithms. Here we show the change in accuracy for both the algorithms by taking different optimizers and epoch values.

2. Implementation of MLP and CNN

2.1 Multilayer Perceptron

Multilayer Perceptron is a class of feed-forward artificial neural networks. It mainly comprises of three steps when training the MLP model with a dataset. The first step is a forward pass in which we send the input to the layers by multiplying with weights add adding the bias. In the second step after getting a predicted output, we will be comparing the output with the actual output and then calculate the loss. In the final step, we will backpropagate the loss and update the weights. In MLP, the output of the node is decided by the activation function, we have different activation functions like Sigmoid, Tanh, and Rectified Linear Unit.

2.1.1 Findings

Under Multilayer Perceptron, we find that the dense layers and number of neurons in each layer are the hyperparameters. To retrieve the optimal accuracy, we can observe the changes by changing the number of dense layers and the number of neurons in each layer. We can implement the semantic search across the nodes and layers by using a grid. We can also find the changes in the accuracy by changing the optimizer for the model which is used to identify the difference in results and true values then we can adjust weights on the nodes.

By applying the code given in the textbook, we have acquired 88% accuracy represented in figure 1. We have observed almost the same accuracy by running the data multiple times. Every time, the assumed weights in the first layer changes, so there would be some changes in the accuracy. In this case, we have taken five dense layers with 128 neurons each for all the layers. We have used the Nadam optimizer for optimization and softmax activation for output We also observed the change in accuracy by changing the number of dense layers. At first, we have provided five dense layers, in this model by increasing the dense layers the accuracy got decreased. We also find that it gives an accuracy of 69% for the sgd optimizer. As far as our observation is concerned, we have acquired the least accuracy of 11% for the adadelta optimizer.

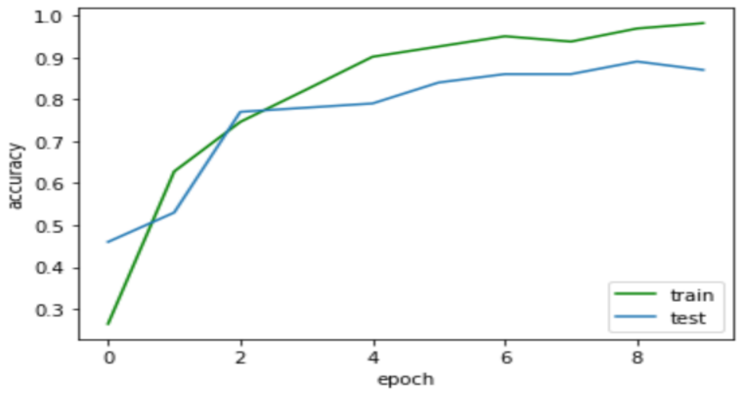

Figure 1 Plot of accuracy vs epoch for training and test accuracy

In our case, we found the better accuracy would be 93.9% when compared to other models. In this case, we have taken 5 dense layers with 512, 256, 128, 64, 32 neurons respectively, and used the Nadam optimizer and the history of accuracy for train and test data is observed using a graph which is represented in figure 2.

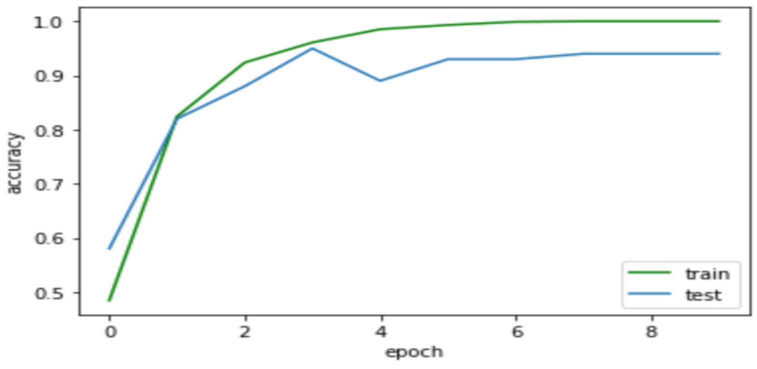

Figure 2 Plot of accuracy vs epoch for best accuracy that we have gained

2.2 Convolution Neural Networks

Convolutional Neural Networks are a class of artificial neural networks that are derived from standard Multilayer Perceptron. CNN adds convolution layers and pooling layers along with the dense layers used in MLP. Convolution and pooling layers are used for feature extraction and dense layers are used yo map the extracted features to the output. The convolution layer performs a mathematical operation called convolution. In the MNSIT dataset, digital images are stored in 2-Dimensional. A 2-Dimensional grid called 'kernel' is used for convolution at each neuron in the convolution layer. Convolution is nothing but a dot product between the kernel and the input tensor. Zero paddings are used to maintain the output kernel size the same as the input kernel size. Similar to MLP, CNN also uses a backpropagation algorithm to learn spatial hierarchies of features and adjust weights. CNN requires graphical processing units for training because of its learnable parameters. Training helps in identifying the best kernels used in the convolution process and also the weights. We will see below how the accuracy of the model changes by tuning the parameters like the size of the kernel, strides, padding, optimizer. At first, we have built a convolutional neural network using two convolution layers with kernel size 3, 1 max-pooling with pool size 2, and 2 Dense layers. We will first see the difference between two optimizers “nadam” and “adadelta” for the same 12 epochs and 32 batch size.

```
Text(0, 0.5, 'accuracy')
```

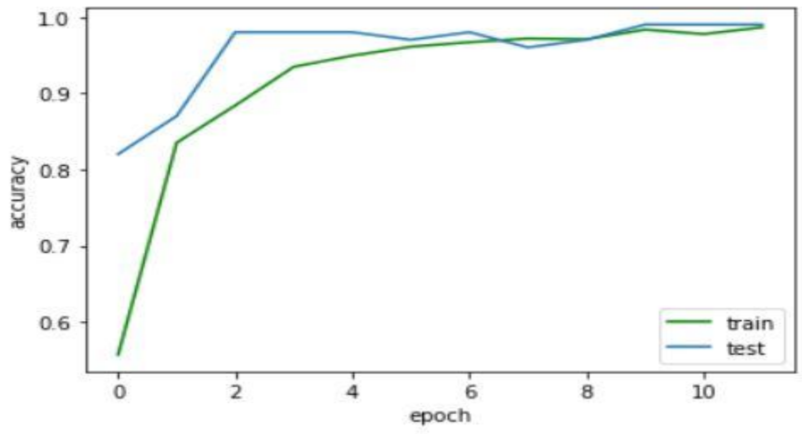

Figure 3 Train and test accuracy for nadam optimizer

```
Text(0, 0.5, 'accuracy')
```

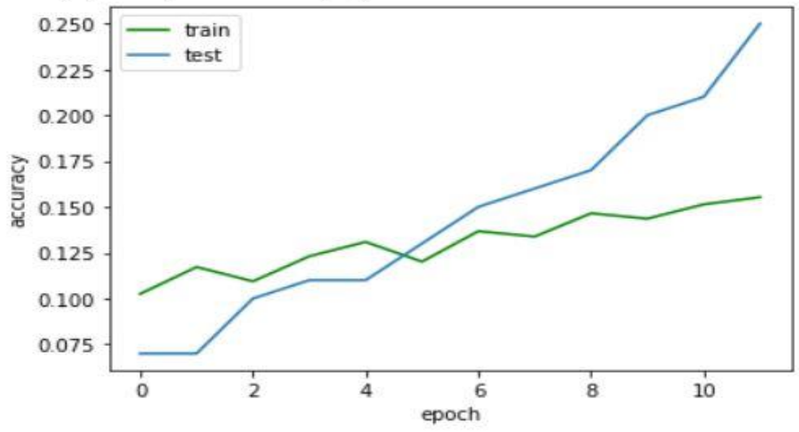

Figure 4 Train and test accuracy for adadelta optimizer

You can see the difference in the above figure 3 and figure 4 when using the same data, same hidden layers but different optimizers we get different accuracies. In the first picture when trained with adam optimizer accuracy of the train starts from 50% and for 12 epochs it ends at almost 100%. Even the accuracy of test data starts at 80% and after 12 epochs it ends at almost 100%. In the second picture when trained with adadelta optimizer, the accuracy of both the train data start at around 1% and ends at 25% for 12 epochs. It is far less than the adam optimizer. The accuracy of the test data ends at around 15%. This does not mean the adadelta is not a good optimizer. When we increase the epochs we get an accuracy which is around 100%. We can also increase the size of the training data set to reach around 100% accuracy. When we compare both the optimizers we can see there is more scope of overfitting and underfitting while using the adam optimizer when compared to the adadelta optimizer. So every optimizer has its advantages and disadvantages.

Tuning hyperparameters is one of the important aspects of training a model. The hyperparameters which can be considered are epochs, batches, adding the convolution layers, max-pooling layers, dense layers.

We have seen in the previous comparison by increase the number of epochs we can increase the accuracy of the adadelta optimizer. We have tried adding a dense layer with 256 neurons with RELU activation along with 0.5 dropouts. We found there is a drop in the accuracy while we add in these two layers. We have also tested an approach which starts with more neurons(256) in the first convolution layer and then decrease it to half(128) in the second layer of convolution. The code goes below like this

```
CNN = Conv2D(256 , kernel_size=(3,3) , activation='relu',padding='valid',iinput_shape=(28,28,1))(inputs)
CNN = Conv2D(128 , kernel_size=(3,3) , activation='relu')(CNN)
CNN = MaxPooling2D(pool_size=(2,2))(CNN)
CNN = Dropout(0.25)(CNN)
CNN = Flatten()(CNN)
CNN = Dense(64,activation='relu')(CNN)
CNN = Dropout(0.5)(CNN)
CNN = Dense(32,activation='relu')(CNN)
```

We took this approach considering an idea where each layer mixes the features from the previous layer and reduces to half. But the accuracy has reduced when compared to the previous version where we doubled the neurons at each layer. This may be due to the input is low when compared to the first layer.

2.3 Comparing accuracies between MLP and CNN

We have identified differences in the accuracies provided by MLP and CNN for the MNIST dataset which is trained on 1024 records and tested on 100 records. As far as our observation, we find that CNN provides more accuracy with an accuracy of 97% by figure 3 compared to 94% for MLP by figure 2.

3. VGG16

VGG16 is a convolutional neural network architecture named after the Oxford group that created it – Visual Geometry Group. It is a very deep CNN, relevant even today, even though recent advances such as Inception and ResNet have overtaken it. We obtained VGG16 and fine-tuned it on the MNIST dataset of handwritten digits. The data we have taken is grayscale with only one color channel, whereas VGG16 requires images with 3 color channels. Therefore, we have gone ahead and converted our images such that they have 3 channels, using np.dstack –

```
x_train=np.dstack([x_train] * 3)
x_test=np.dstack([x_test] * 3)
```

We have used a sequential model for training our data. Initially, we removed the prediction layer (with size = 1000), and froze all the layers, except for the dense layers for the training purposes. We then added a last "dense" layer with a size – 10 i.e. for 10 predictions (10 digits). We then compiled our model with "adam" optimizer and "categorical_crossentropy" loss function.

3.1 Preprocessing & Training the model

For our training data set, we have taken 1024 values, and 256 for our testing data set. We then resized the images into the required size of 224x224, reshaped them to have 3 channels, and preprocessed them as required by VGG16. Once the images have been preprocessed, we trained our model with these data. We are using epochs (20) to measure our iterations i.e. the number of times the training vectors are used. We noticed that with each epoch step – our accuracy increased, becoming closer to 1 as the steps neared completion.

Once we're able to evaluate our VGG16 score using the test data i.e. measuring the test loss, and test accuracy, we plotted the graph for its learning curve.

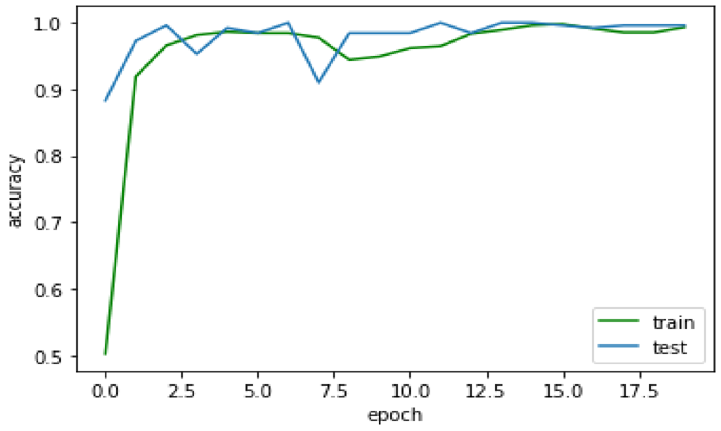

Figure 5 Plot of accuracy vs epoch for best accuracy that we have gained

4. Conclusion

It can be concluded that the fine-tuned VGG16 has better accuracy and a better learning curve when compared to CNN and MLP which were built from scratch. This is mainly due to the training of only the dense layers of VGG16. Thus, transfer learning is a very useful phenomenon where we can reuse pre-trained models and fine-tune them to work with new datasets without having to create a new model or retrain all the layers.

References

Pal, S., & Mitra, S. (1992). Multilayer perceptron, fuzzy sets, and classification. IEEE Transactions On Neural Networks, 3(5), 683-697. https://doi.org/10.1109/72.159058

Wikipedia Contributors. (2020, June 8). Multilayer perceptron. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Multilayer_perceptron

residentmario. (2018, December 2). Keras optimizers. Kaggle.Com; Kaggle. https://www.kaggle.com/residentmario/keras-optimizers

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. Insights into Imaging, 9(4), 611–629. https://doi.org/10.1007/s13244-018-0639-9