Image Recognition Analysis using CNN

Nikhil Dhanda
Department of Computer Science
Georgia Institute of Technology
nnn3@gatech.edu

Angel Sanchez
Department of Electrical Engineering
Georgia Institute of Technology
angeldsanchez@gatech.edu

Abstract

The main objective of this project is to solve the problem of object recognition in images using the machine learning algorithm - Convolutional Neural Networks. We aim to apply an implementation of this algorithm on multiple publicly available datasets to understand how the model works with different configuration and how can the model's performance be improved on these datasets so as to be able to solve this problem efficiently.

1 Introduction

We used two different datasets – MNIST and CIFAR10 for the analysis. Both of these datasets are publicly available and are used extensively in image recognition research. We opted to go with Convolutional Neural Network implementation since CNNs have proven to be quite effective in Computer Vision research and have been used in a lot of work related to our project. We found implementations of this algorithm online and used code from two different sources to train the model on these datasets. Both the sources are publicly available and are cited below [1][2]. They both are written using Tensorflow and have the same logic and implementation for the CNN, they mainly differ in the structure of the code.

We had initially planned on using only the CIFAR10 dataset but ended up using the MNIST dataset as well. The main reason for using two different datasets was to try to be more versatile with the approach since the two datasets, while serving similar purposes, are different from each other. CIFAR10 dataset consists of colored images with 10 different class labels while the MNIST dataset consists of black and white images illustrating handwritten digits ranging from 0 to 9. We wanted to see if due to this aspect one dataset could outperform the other with the CNN model. Another main reason for having two datasets was also to be able to analyze the models more in-depth. The implementation for the CIFAR10 dataset only had two convolutional layers and two fully connected layers (Multi-Layer Perceptron) and instead of trying to understand the code and write our own version to modify number of layers we decided to use that time for the analysis of another dataset since that is the focus of the project.

During our research, we trained the models with varying number of layers, batches, different batch sizes and learning rates to understand the pattern and find out which configuration is ideal and try to figure out the reasons behind them.

2 Convolutional Neural Networks

CNN or Convolutional Neural Networks are a category of artificial neural networks. They have proven to be very effective in image recognition and classification tasks such as identifying objects, patterns, faces etc. This also makes them very effective in tackling Natural Language Processing problems. The main components of a CNN are namely, Convolution, Nonlinearity(ReLU), Pooling, Classification (Fully Connected Layer).

2.1 Convolution

The primary purpose of this step is to extract information from the image. An image is simply a 2D matrix of pixel values. First the computer tries to learn smaller, less complex things like lines and curves by having multiple filters or kernels of different lines and curves. Then takes these filters and convolve them with smaller subsections of the picture until the whole picture is transformed into activation maps or convolved features for each filter. These feature maps tare then stacked together to create the first convolution layer.

2.2 Non-Linearity

In this next step, non-linear operations such as ReLU or sigmoid are applied to the feature map to add nonlinearity to our CNN since most of the real-world data is nonlinear. ReLU is used often because it has been found to perform better in most situations [3]. It stands for Rectified Linear Unit and simply replaces all negative values in the map to zero.

2.3 Pooling

Pooling is basically down sampling or reducing the spatial size of the image by using only the max, sum, mean etc. of a subsection. It is done to reduce the dimensionality of the matrix and to prevent overfitting of the model by reducing the number of parameters or computations in the network. It makes the algorithm more time efficient since the model is now working with a smaller image. Pooling also allows the model to capture a line or curve even if it is slightly moved or in different location in original picture.

2.4 Classification

The last step is the fully connected layer which is simply a multilayer perceptron that uses a SoftMax activation function in the output layer. The function of this step is to use the high-level features of the input image provided by the first three steps to classify the image into various classes based on the training set.

2.5 Training Process

The training process is like the process for the general Artificial Neural Network, it starts with completing all the above-mentioned steps to initialize the weights, values of filers, parameters and get the initial output probabilities for each class. And then use backpropagation algorithm to calculate the gradients of the error with respect to each weight and readjust the filter values, weights and parameters using gradient descent. It then repeats the entire processes to train the model on all the images in the training set.

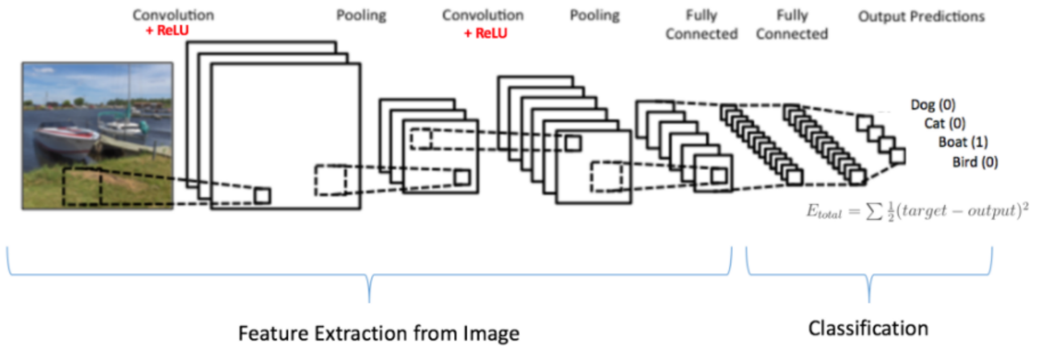


Figure 1: Training process of a Convolutional Neural Network [3].

3 Experiments

We ran the experiments with the CNN models on the respective dataset while varying different parameters to find and analyze patterns or observations in the models trained on the datasets. Below are the examples of output images of the Convolution Layers which act as input to the Fully Connected Layers. The images, of both datasets, have been through various filters, pooling and non-linearity algorithms to reduce the dimensions of the data while emphasizing important features for the Fully Connected Layers to classify them into one of the many classes. From a human perspective, the images are not very clear, especially the one from CIFAR10 datasets, but they contain all the pertinent information needed by the Fully Connected Layers to label them into various classes.

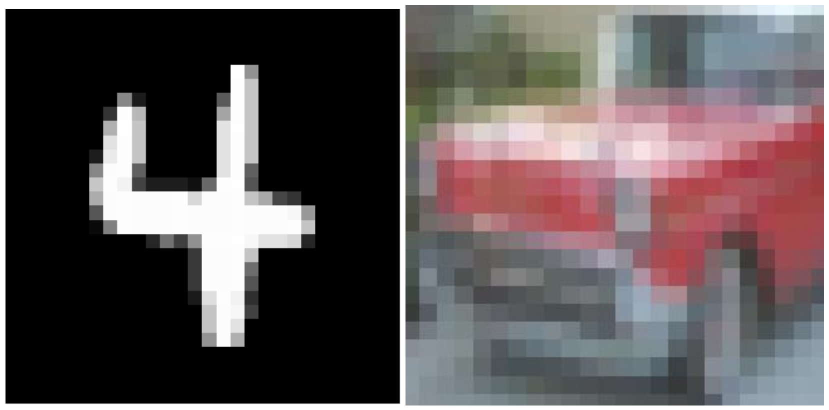


Figure 2:Convolution Layers' output images from MNIST (left) and CIFAR10 (right) datasets

Below is the structure of the Convolutional Neural Network with 2 Convolutional Layers and 2 Fully Connected Layers used in the MNIST experiments. This is similar to the flow chart on CNN shown above but in more details of implementation and different elements. The arrows in the chart show the flow of tensors from one element to another.

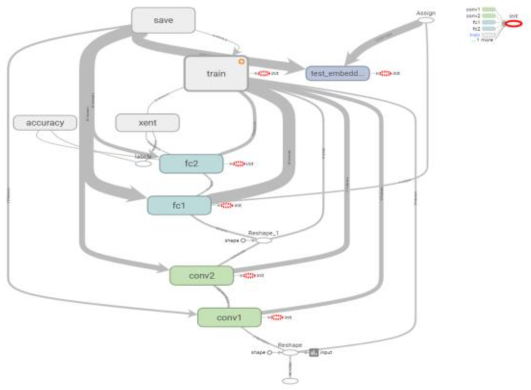


Figure 3: Convolutional Neural Network Structure

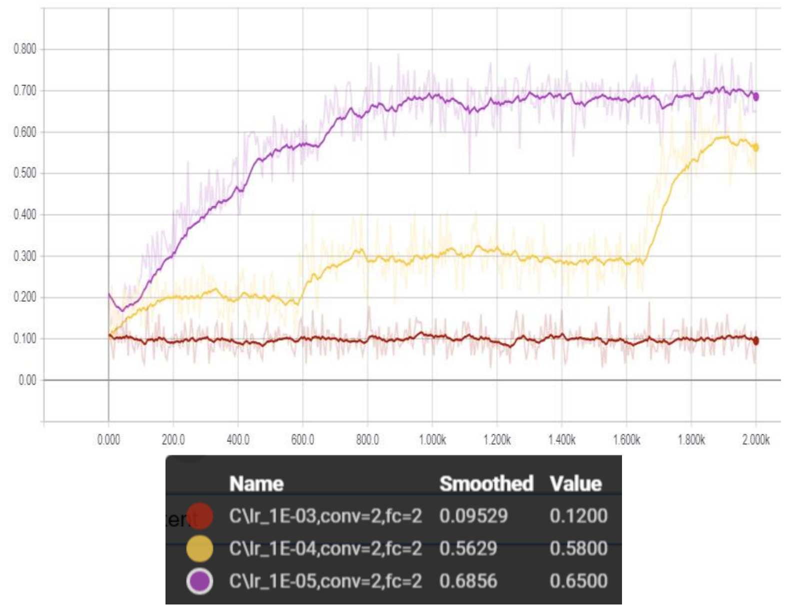
3.1 MNIST Dataset

On MNIST dataset, we ran the CNN model with varying learning rates, number of convolutional layers and number of fully connected layers. We mainly considered two criteria for judging the performance of the model- Cross Entropy Loss and Testing Accuracy. All the graphs for this section are smoothed to make them more legible and remove noise.

3.1.1 Varying Learning Rate

We first trained the model using three different learning rates, 1e-3, 1e-4, 1e-5, for the initial configuration of two Convolutional Layers and two Fully Connected Layers. The graph below represents the Accuracy of the model so represent the varying learning rates.
As can be seen, decrease in learning rates causes an increase in the accuracy. The learning rate of 1e-5 seems to provide the model the highest testing accuracy while the learning rate of 1e-3 fails

to make the model converge and produce any meaningful deductions. This is because a larger learning rate will prevent the model from converging at the global minimum and will make it overstep and converge at a local minimum instead. This could be loosely related to the example of a person walking towards a point in regularly sized steps, too small a step and it will take a very long time to reach there while too large a step will make the person overstep the exact target.

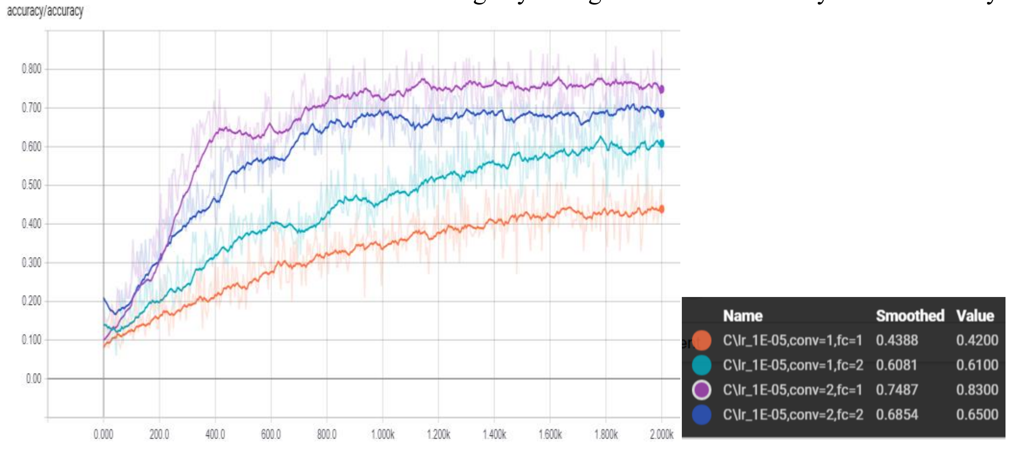


	Name	Smoothed	Value
●	C\lr_1E-03,conv=2,fc=2	0.09529	0.1200
●	C\lr_1E-04,conv=2,fc=2	0.5629	0.5800
●	C\lr_1E-05,conv=2,fc=2	0.6856	0.6500

Figure 4: Accuracy vs. Batches with different learning rates

3.1.2 Varying the Convolutional and Fully Connected Layers

After the first experiment, we trained the model on different combination of Convolutional Layers and Fully Connected Layers with the learning rate set to 1e-5 to see if we could achieve a higher accuracy than the one we currently had. The graph below represents the accuracy of these tests plotted with number of batches the model was trained on. The models trained with only one convolutional layer performed worse since the models missed important information or features provided by the second convolutional layer. An interesting finding was that the model with 2 Convolutional Layers but only 1 Fully Connected Layer outperformed our initial model. This is due to an increase in Classification Layers results in a much more complicated model which usually demands an increase in complexity of the features and more data points. The dataset used might not have been complicated enough for the model and thus caused it to overfit slightly at higher number of Fully Connected layers.



	Name	Smoothed	Value
●	C\lr_1E-05,conv=1,fc=1	0.4388	0.4200
●	C\lr_1E-05,conv=1,fc=2	0.6081	0.6100
●	C\lr_1E-05,conv=2,fc=1	0.7487	0.8300
●	C\lr_1E-05,conv=2,fc=2	0.6854	0.6500

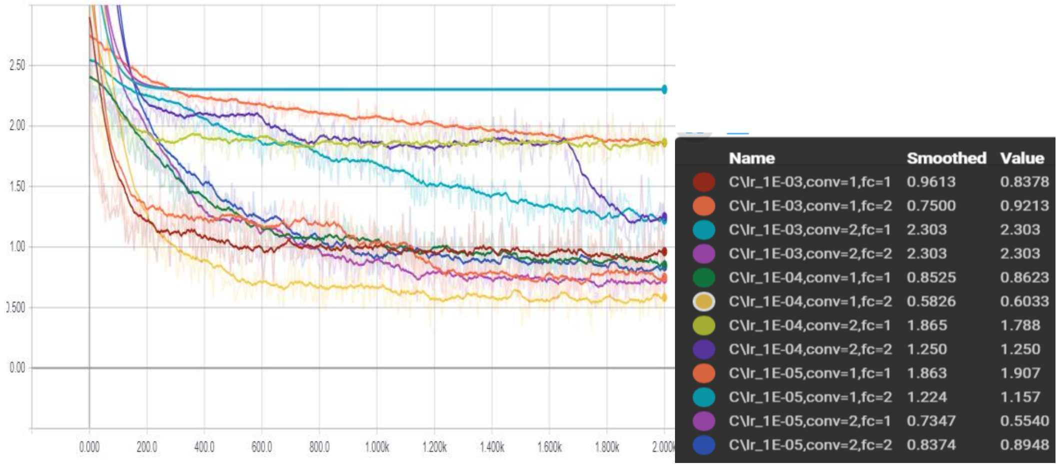
Figure 5: Accuracy vs. Number of Batches at Learning Rate 1e-5

After this observation, we ran the experiment with different learning rates and number of layers. We did not increase the number of batches any further for this dataset as it can be observed from the graphs above that most of the models reached their maximum accuracy after around 800 batches and increasing batches after that did not result in any significant changes in the values which implies that the accuracies reached saturation as we increased the number of batches.

Table 1: All Models at 2.000k batches

Learning Rate	# Convolutional Layers	# Fully Connected Layers	Cross Entropy Loss	Accuracy
1e-3	1	1	0.96	59.3
1e-3	1	2	0.75	69.5
1e-3	2	1	2.3	9.9
1e-3	2	2	2.3	9.2
1e-4	1	1	0.85	68.5
1e-4	1	2	0.58	77.9
1e-4	2	1	1.86	28.0
1e-4	2	2	1.25	56.1
1e-5	1	1	1.86	44.1
1e-5	1	2	1.24	60.1
1e-5	2	1	0.73	73.9
1e-5	2	2	0.83	68.9

From running experiments on all the above different combinations, we observed that, our previous highest accuracy at the model with 1e-5 learning rate and 2 Convolutional Layers and 1 Fully Connected Layer was beaten by the model trained at a higher learning rate of 1e-4. For higher learning rates, 1 Convolutional Layer and 2 Fully Connected Layers configuration has by far outperformed others. We think this is because the higher learning rate makes the model less prone to converge at a global minimum and introducing more layers might have helped counter that aspect which lead to a better performance. Increasing the learning rate further to 1e-3 reduced the performance, thus our best configuration for CNN model on MNIST Dataset has 1 Convolutional Layer, 2 Fully Connected Layers and has a learning rate of 1e-4.



	Name	Smoothed	Value
●	C\lr_1E-03,conv=1,fc=1	0.9613	0.8378
●	C\lr_1E-03,conv=1,fc=2	0.7500	0.9213
●	C\lr_1E-03,conv=2,fc=1	2.303	2.303
●	C\lr_1E-03,conv=2,fc=2	2.303	2.303
●	C\lr_1E-04,conv=1,fc=1	0.8525	0.8623
●	C\lr_1E-04,conv=1,fc=2	0.5826	0.6033
●	C\lr_1E-04,conv=2,fc=1	1.865	1.788
●	C\lr_1E-04,conv=2,fc=2	1.250	1.250
●	C\lr_1E-05,conv=1,fc=1	1.863	1.907
●	C\lr_1E-05,conv=1,fc=2	1.224	1.157
●	C\lr_1E-05,conv=2,fc=1	0.7347	0.5540
●	C\lr_1E-05,conv=2,fc=2	0.8374	0.8948

Figure 6: Cross Entropy Loss vs Number of Batches for all the experiment runs

3.2 CIFAR10 Dataset

On the CIFAR10 dataset, we ran the CNN model with varying number of batches and batch sizes with 2 Convolutional Layers and 2 Fully Connected Layers. We did not change the number of layers for this model since the implementation had only one configuration hardcoded and instead of trying to add more versatility to the code, we decided to spend that time doing more experiments, analysis and thus included the MNIST dataset in our project. Similar to the MNIST experiments, we mainly considered two criteria for judging the performance of the model- Cross Entropy and Testing Accuracy. All the graphs for this section are smoothed to make them more legible and remove noise

3.2.1 Varying the Learning Rate Decay

For the first experiment on Cifar10, we varied the learning rate decay. Learning Rate decay in our CNN implementation is used to exponentially decrease the learning rate instead of just starting with a lower value because this allows for larger changes to the weights at the initial stages of the training process when larger learning rate values are used, and a smaller rate at later stages for smaller training updates to weights. This results in quickly learning good weights and fine tuning them later in the process[4]. We varied the learning rate decay for the initial configuration of the CNN but we were not able to observer anything conclusive, as can be seen in the graph below. All the models seemed to get the very similar results even though the learning rate decay was changed drastically. This is because the initial learning rate of 0.1 was sufficient for the model to quickly find good weights and thus decreasing the learning rate with varying decay only resulted in very minor fine tuning of the weights.

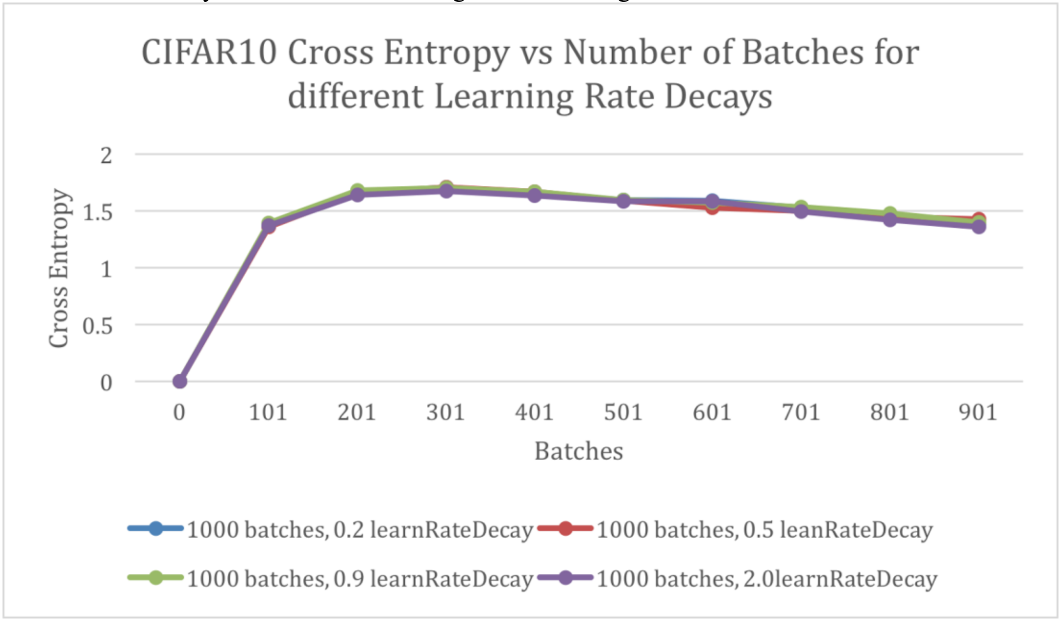


Figure7: Cross Entropy vs Number of Batches for different Learning Rate Decays

3.2.2 Varying the number of Batches

Since the Learning Rate Decay Experiments didn't provide us with any significant pattern, we varied the number of batches the model was trained on the dataset. The graphs below represent the Cross-Entropy plotted with respect to the number of batches for this CNN implementation.

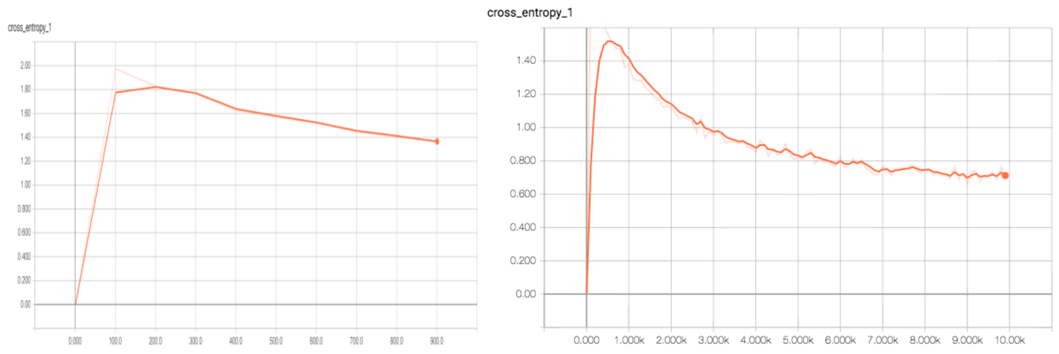


Figure8: Cross-Entropy vs Number of Batches:1000 batches (Left), 10000 batches (Right)

We increased the batches from 500 to 10,000 and observed that as we increased the batches there was an initial jump in the Cross-Entropy at around 200 batches but then it decreased gradually and became very stable near the 6000 batches. We think this is because increasing the batches or iterations caused more backpropagations throughout the network which then resulted in the weights being updated repeatedly. This increased the model's accuracy initially but then stabilized.

3.2.3 Varying the Batch Sizes

From the previous experiment, we knew that increasing batch sizes could reduce the Cross-Entropy to a limit. We wanted to see if changing the batch size will have any effect. So, for this experiment, we trained the CNN on different batch sizes, to see if that would give us any different results. We ran the same configuration of 2 Convolutional Layers and 2 Fully Connected Layers on 1000 Batches with 128, 256 and 512 batch sizes.

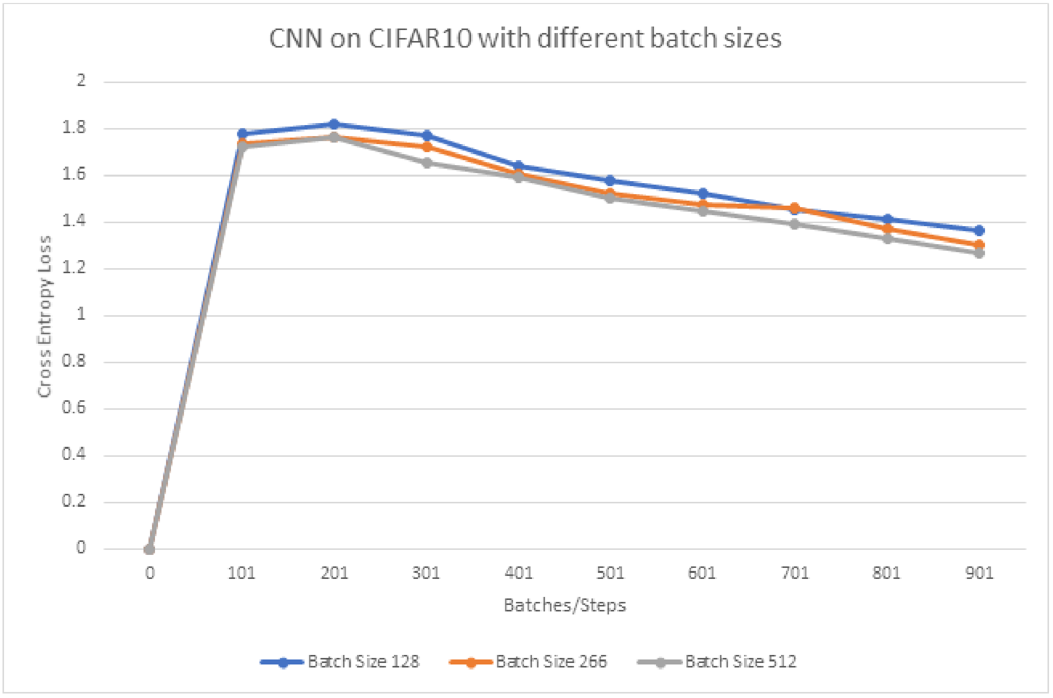


Figure 9:Cross Entropy Loss vs. Batches with different batch sizes

We observed that an increase in the batch size resulted in a minor increase in accuracy and decrease in Cross-Entropy Loss. This is due to the affect the batch size has on the training process. Batch size represents the number of data points that are going to be propagated in the neural network. Typically, in Neural Network training, weights are updated after each propagation by calculating the gradient. So, the smaller the batch size, the more steps it will need to train the net and thus the more the weights will be updated. But there is a catch in this, the gradient is calculated using the error values for each data point. And less number of data points or in this case, lesser batch size will lead to less accurate estimation of the gradient, due to which the weights will not be updated as accurately and hence there will be a decrease, however small, in the overall accuracy of the model, which can ben seen in the graph above. Thus, our highest accuracy model for Cifar10 dataset was the CNN with 2 Convolutional Layers and 2 Fully Connected layers with a learning rate decay of 0.2 and batch size of 512 images.

4 Future Work

We tried to apply Feature Transformation techniques like T-SNE on the two datasets. While we applied T-SNE on MNIST dataset, we were not able to implement it on the CIFAR10 dataset. Below are the result plots for the transformed features. The plot for CIFAR10 is taken from a related work on this topic [5] but it clearly shows that T-SNE is very effective transformation technique on such datasets. We faced a lot of problems while training the model on the feature transformed MNIST dataset and were not able to analyze the change in performance ourselves but both these plots show the potential in performance increase in our model if we train it using these transformed features.

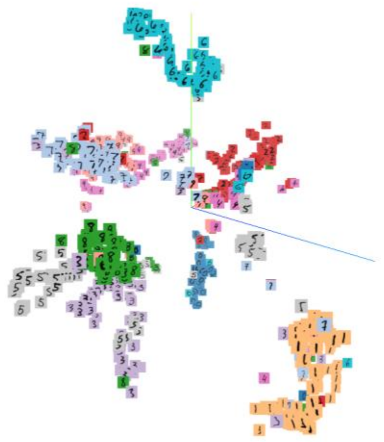


Figure 10: T-SNE on MNIST

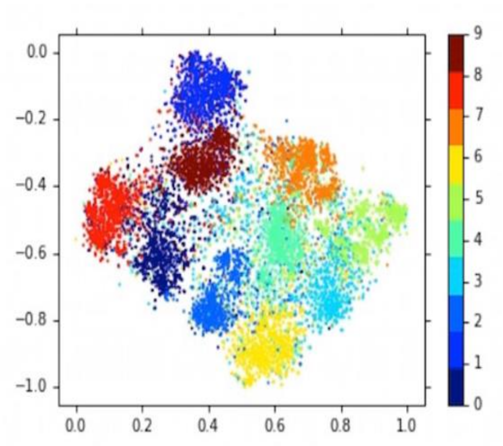


Figure 11: T-SNE on CIFAR10 [5]

5 Conclusion

In this project, we wanted to train Convolutional Neural Networks on the datasets to see if we can find the configurations that would outperform others and see if we can understand the reasons behind such behavior. We discussed how our model worked and performed many experiments on the two datasets. For MNIST, we received the highest accuracy with the configuration of 1 Convolutional Layer and 2 Fully Connected layers at a learning rate of 1e-4. While the best configuration of the Cifar10 dataset was less clear because the rigid structure of the code did not allow us to fully exploit the dataset. Another reason was also because of how the two datasets differ from each other. The MNIST only had black and white images which allowed for better filtering and pooling while the CIFAR10's more complicated RGB images could not. Further experiments could be conducted by training the model on the feature transformed data.

6 References

[1] Siraj Raval. *How to use tensorboard,* https://github.com/llSourcell/how_to_use_tensorboard_live/tree/master

[2] Andrew Selle. *Deep CNN: Launching and Training the Model,* https://github.com/tensorflow/models/tree/master/tutorials/image/cifar10/

[3] Ujjwal Karn. *An Intuitive Explanation of Convolutional Neural Networks,* https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[4] Jason Brownlee. *Using Learning Rate Schedules for Deep Learning Models* http://machinelearningmastery.com/using-learning-rate-schedules-deep-learning-models-python-keras/

[5] Oliver Durr. *Deep learning for lazybones,* http://oduerr.github.io/blog/2016/04/06/Deep-Learning_for_lazybones