# Introduction

A convolutional neural network is a type of neural network which is generally used in applications where the size of the dataset is very large. Due to this, it is computationally very intensive to have all hidden layers fully connected. As a result, in a convolutional neural network, each neuron is only connected to a locally contiguous subset of nodes from the previous layer. It consists of two types of layers, convolutional layer and pooling layer. Each convolutional layer has multiple feature maps, where each feature map is a 2D arrangement of neurons used to calculate a single feature, whose value is a 2D matrix. Each convolutional layer is connected to a pooling layer, which is used to reduce the size of the feature maps in the previous layer by dividing it into disjoint regions and pooling each region to a single scalar value. This kind of network is widely used in computer vision for image and video recognition.

# Strategies

In order to have a fair comparison, we make sure that we use the same training and test set for all the models. We plot the training accuracy against the number of epochs for each model and also the test accuracy against different values of each parameter. We choose the parameter values which give the highest test accuracy. Note that we do not use the training accuracy to choose the parameter values, since the training accuracy will be high although the test accuracy will be low if the model over-fits the training data. Due to this, such a model will not generalise well to new examples.

# Assumptions

In order to remove the fluctuations of the training accuracy of our models across all iterations, we plot the training accuracy against the number of epochs instead of the number of iterations, so that we are able to better study our plots of convergence. However, since MATLAB only provides the mini-batch accuracy after each iteration and not the accuracy of the entire dataset after each epoch, we plot the mini-batch accuracy of the last iteration for each epoch against the number of epochs. Our assumption is that over a large number of epochs, this plot should approximate the plot obtained by plotting the accuracy of the entire training data against the number of epochs. However, this should not have an impact on the results we obtain, since we finally decide all the parameters of our model based on the Test Accuracy and not the Training Accuracy, which is still calculated for the entire test set.

# Methodologies

## Model Architectures

### 1 Convolutional and 1 Pooling Layer

In this architecture, we experiment with the best values for the learning rate, mini-batch size, momentum and decay term.

### 2 Convolutional and 2 Pooling Layers

In this architecture, we use the learning parameters obtained from Part 1 and experiment with the best values for the number of filters for both the layers.

## Learning Algorithms

We use stochastic gradient descent with momentum for training the models. In this algorithm, the training set is divided into multiple batches of equal size and the weights are updated for one batch at a time. Backpropagation is used to implement gradient descent. Momentum is used for speeding up the convergence and ensuring that the algorithm does not oscillate for a long time near the optimum. Decay term is used to ensure that the model does not over-fit the training data.

# Implementation

The file “cnnPreprocess.m” is used to randomly divide our dataset into training set and test set. The same division of data is used to experiment for all the models and architectures.

The file “cnnTrain1” is used to train the architecture for Part 1 for different learning parameters. The file “cnnTrain2.m” is used to train the architecture for Part 2 for different number of filters. We also have separate files for varying the parameters to make it more modular –

1. vary\_lr.m: This file is used to experiment with different learning rates.
2. vary\_batchsize.m: This file is used to experiment with different batch sizes.
3. vary\_momentum\_decay.m: This file is used to experiment with different combinations of momentum and decay terms.
4. vary\_num\_filters.m: This file is used to experiment with the number of filters for both the convolutional layers in Part 2

Once we have found the optimum parameters for both the architectures, we find their accuracy on the test set to evaluate how well the algorithm performs for those parameters.

Note that although we obtain plots of convergence for all the models we train our data on, we only include in this report, the plot for the model that has the highest Test Accuracy and the models whose parameter values are near the parameter values for that model.

# Results and Analysis

## Model with 1 Convolutional and Pooling layer

### Learning Rate

The other learning parameters are set to the following values –

1. *Maximum Epochs = 25*
2. *Batch Size = 128*
3. *Momentum = 0.1*
4. *Decay = 0.0001*

We experiment for the following values of the learning rate:

*[0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3]*

Figure 2: Training Accuracy VS Epochs (Learning Rate = 0.003)

Figure 1: Training Accuracy VS Epochs (Learning Rate = 0.001)



Figure 3: Training Accuracy VS Epochs (Learning Rate = 0.01)

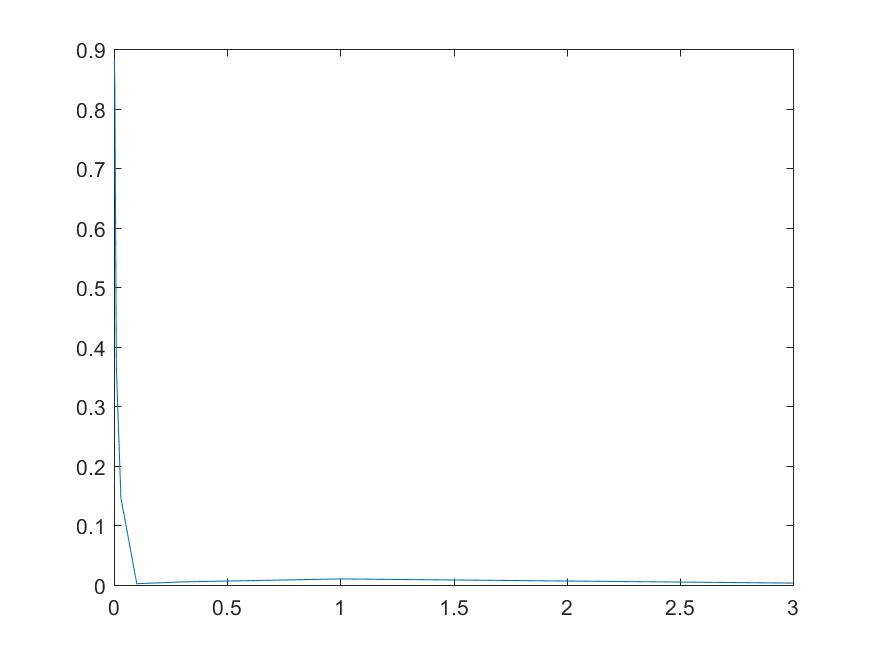


Figure 4: Test Accuracy VS Learning Rate

It can be seen from the above graphs that the Training Accuracy as well as the Test Accuracy decreases after a learning rate of 0.001. This is because if the learning rate is too high, it might oscillate and take a longer time to reach the optimum, or even diverge from the optimum. Hence, we choose a learning rate of 0.001 for our algorithm.

Generally, the learning rate for stochastic gradient descent needs to be lower than for batch gradient descent, since the gradient for stochastic gradient descent is noisier. However, this noise might also help escape local minima in certain situations and converge to a local minima which is closer to or same as the global minima.

### Batch Sizes

The other learning parameters are set to the following values –

1. *Maximum Epochs = 25*
2. *Learning Rate = 0.001*
3. *Momentum = 0.1*
4. *Decay = 0.0001*

We experiment for the following values of the batch size:

*[16, 32, 64, 128]*

Figure 5: Training Accuracy VS Epochs (Batch Size = 32) Figure 6: Training Accuracy VS Epochs (Batch Size = 64)



Figure 7: Training Accuracy VS Epochs (Batch Size = 128)

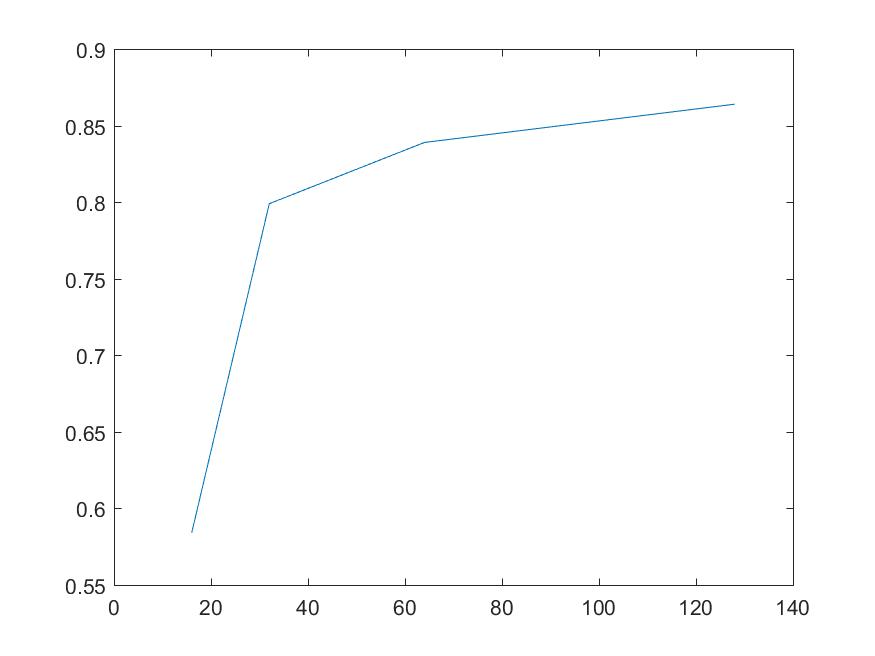


Figure 8: Test Accuracy VS Batch Size

If the size of the training dataset is very large, the computation of the gradient for the entire dataset is very computationally intensive and slow. At the same time, this can also lead to over-fitting. Hence, we divide the data into mini-batches in stochastic gradient descent and the weights are updated over mini-batches of training patterns.

As the size of the mini-batch decreases, the error function for stochastic gradient descent becomes more and more noisy. Hence, we can see lot of fluctuation in the Training Accuracy in the above figures as the batch size decreases.

As we can see from the figures, the Training Accuracy for batch sizes of 64 and 128 was higher compared to the other batch sizes for our training dataset. Although the Training Accuracy for batch sizes of 64 and 128 was almost the same, it can be seen from Figure 8 that the Test Accuracy was increasing with increasing batch sizes and the Test Accuracy for a batch size of 128 was higher than for a batch size of 64. This means that our model generalises better to new data samples with a batch size of 128 compared to a batch size of 64. Hence, we choose a batch size of 128 for our algorithm.

### Momentum and Decay

The other learning parameters are set to the following values –

1. *Maximum Epochs = 25*
2. *Learning Rate = 0.001*
3. *Batch Size = 128*

We experiment for all combinations of the following values of Momentum and Decay:

*Momentum = [0.1, 0.3, 0.9]*

*Decay = [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5]*

|  |  |
| --- | --- |
| Figure 9: Training Accuracy VS Epochs (Momentum = 0.1, Decay = 0.0005) | Figure 10: Training Accuracy VS Epochs (Momentum = 0.1, Decay = 0.001) |
| Figure 11: Training Accuracy VS Epochs (Momentum = 0.1, Decay = 0.005) | Figure 12: Training Accuracy VS Epochs (Momentum = 0.3, Decay = 0.001) |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Decay** | | | | | | | |
|  |  | 0.0001 | 0.0005 | 0.001 | 0.005 | 0.01 | 0.05 | 0.1 | 0.5 |
| **Momentum** | 0.1 | 0.8760 | 0.8690 | 0.8850 | 0.8740 | 0.8660 | 0.8790 | 0.8620 | 0.8810 |
| 0.3 | 0.8630 | 0.8460 | 0.8700 | 0.8400 | 0.8520 | 0.8610 | 0.8790 | 0.8770 |
| 0.9 | 0.5940 | 0.5950 | 0.5730 | 0.6190 | 0.6600 | 0.6940 | 0.6420 | 0.6880 |

Figure 13: Test Accuracy for different combinations of Momentum and Decay

Although the Training Accuracy for a momentum of 0.1 and decay of 0.005 was slightly better than for a momentum of 0.1 and a decay of 0.001, the Test Accuracy for a momentum of 0.1 and decay of 0.001 was the highest which means that it generalises very well for new data samples. Hence, we choose a momentum of 0.1 and decay of 0.001 for our learning algorithm.

The error function we use in stochastic gradient descent might often have many shallow ravines, where we might get trapped instead of converging to the optimum. The addition of the momentum term helps us in taking slightly larger steps so that we can escape these ravines. At the same time, the decay term ensures that our weights are closer to zero so that the model we obtain after training has less variance and does not over-fit the training data. However, the decay term can also not be too high else it will lead to under-fitting of the training data, which will lead to low Training Accuracy as well as low Test Accuracy.

### Recommended Model

Based on the above analysis, we would recommend to use the following learning parameters –

1. *Learning Parameter = 0.001*
2. *Batch Size = 128*
3. *Momentum = 0.1*
4. *Decay = 0.001*



Figure 14: Training Accuracy VS Epochs for the Recommended Model

*Figure 14* shows the plot of convergence for training obtained for this model. We got a Test Accuracy of 88.50% after training with this model.

## Model with 2 Convolutional and Pooling layers

### Number of filters

The learning parameters are set to the following values –

1. *Maximum Epochs = 20*
2. *Learning Rate = 0.001*
3. *Batch Size = 128*
4. *Momentum = 0.1*
5. *Decay = 0.001*

We experiment for all combinations of the following values of number of filters in both the Convolutional Layers:

*Convolutional Layer 1 = [20, 30, 40, 50, 60, 70]*

*Convolutional Layer 2 = [20, 30, 40, 50, 60, 70]*

|  |  |
| --- | --- |
| Figure 15: Training Accuracy VS Epochs (Number of filters = [50, 50]) | Figure 16: Training Accuracy VS Epochs (Number of filters = [50, 60]) |
| Figure 17: Training Accuracy VS Epochs (Number of filters = [50, 70]) | Figure 18: Training Accuracy VS Epochs (Number of filters = [40, 60]) |
|  |  |
|  |  |



Figure 19: Training Accuracy VS Epochs (Number of filters = [60, 60])

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **Number of filters in Convolution Layer 2** | | | | | |
|  |  | 20 | 30 | 40 | 50 | 60 | 70 |
| **Number of filters in Convolution Layer 1** | 20 | 0.8940 | 0.8990 | 0.8840 | 0.8960 | 0.8960 | 0.8860 |
| 30 | 0.8940 | 0.8980 | 0.8950 | 0.8900 | 0.8830 | 0.8990 |
| 40 | 0.8930 | 0.8860 | 0.8900 | 0.8930 | 0.8900 | 0.8910 |
| 50 | 0.8910 | 0.8960 | 0.8900 | 0.8970 | 0.9030 | 0.8820 |
| 60 | 0.8930 | 0.8930 | 0.8920 | 0.8880 | 0.8920 | 0.8910 |
| 70 | 0.8990 | 0.8910 | 0.8970 | 0.9010 | 0.8990 | 0.8950 |

Figure 20: Test Accuracy for different number of filters in Convolutional Layer 1 and Convolutional Layer 2

Although the Training Accuracy with 50 filters in the first Convolutional Layer and 60 filters in the second Convolutional Layer is lower compared to some of the other models, the Test Accuracy for that model is the highest which means that it generalises well to new data samples while the other models slightly over-fit the training data. Hence, we choose a model with 50 filters in the first Convolutional Layer and 60 filters in the second Convolutional Layer.

### Recommended Model

Based on the above analysis, we would recommend to use the following number of filters –

1. *Number of filters in first Convolutional Layer = 50*
2. *Number of filters in second Convolutional Layer = 60*



Figure 21: Training Accuracy VS Epochs for the Recommended Model

On training with this architecture and the learning parameters obtained from the first model, we get a Test Accuracy of 90.30%, which is higher than the Test Accuracy of 88.50% obtained in Part 1. Hence, it is better to train with this architecture instead of the architecture obtained in Part 1, while still using the learning parameters from Part1.

However, it is not always true that the Test Accuracy of the model will increase if we increase the number of convolutional layers. In general, although the Test Accuracy of the model might increase initially on increasing the number of convolutional layers, the Test Accuracy might actually start decreasing if we keep increasing the number of convolutional layers. This is because addition of each convolution layer to the model reduces the number of input features to the fully connected layers.

# Discussions and Challenges

Ideally, we should try all combinations of all the learning parameters and the number of filters while trying to determine the best model to use for training on the training data. However, due to the high computational power required for such a task and the time constraints for this project, we first experimented with individual or sets of two parameters while keeping the other parameters constant, and tried to determine the best parameter values from each experiment. We then again experimented with these parameters in the same way, but we set the constant parameters to the values obtained from the first time. The plots shown in this report are the ones we obtained on the second time, using which we again try to find the optimum values of the parameters. Although we might not be able to get the best model, we hope that this process would help us to get closer to the optimum model. It is much more feasible to try all possible models with all possible combinations of parameters in a distributed environment, where each model can be run separately on different machines, which will help to reduce the time taken to train all the models.

We can also experiment with more parameters like the ratio in which the dataset is divided into train set and test set, the dimensions of the kernel, dimensions of the pooling regions, function used for pooling and so on. However, we do not experiment with these parameters for this project as they are outside the scope of this project.