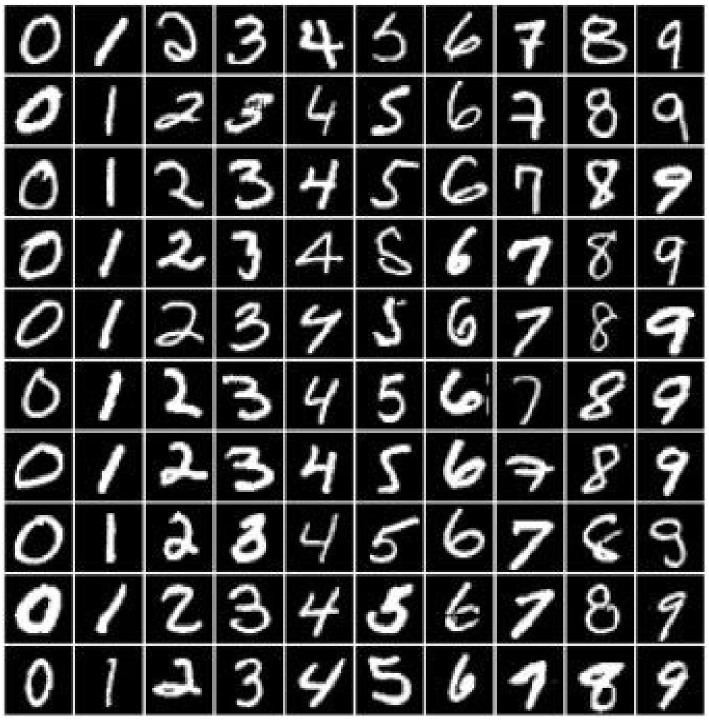
Implementing CNN using Keras on MNIST Dataset

Analysis of results upon tweaking hyperparameters



Rahul Gyawali

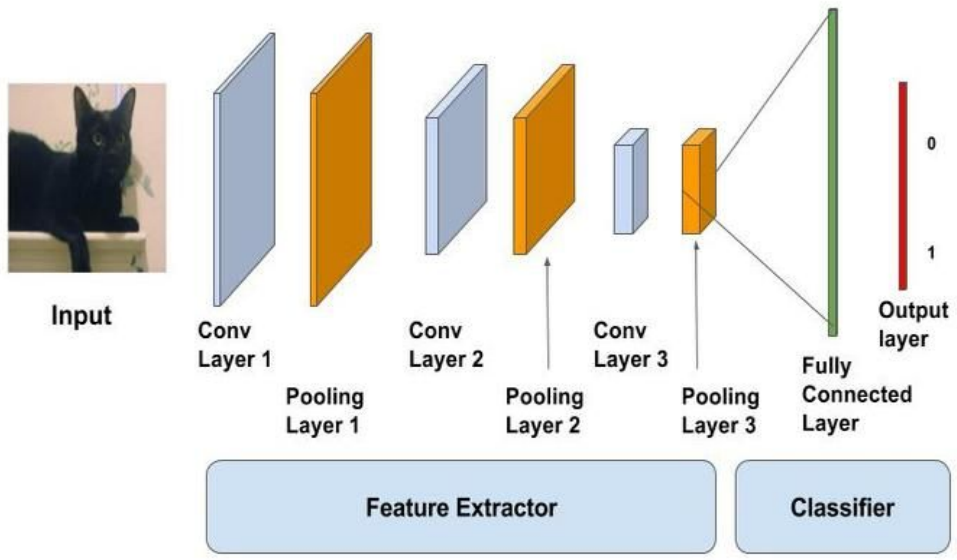
IIT2015005

INTRODUCTION

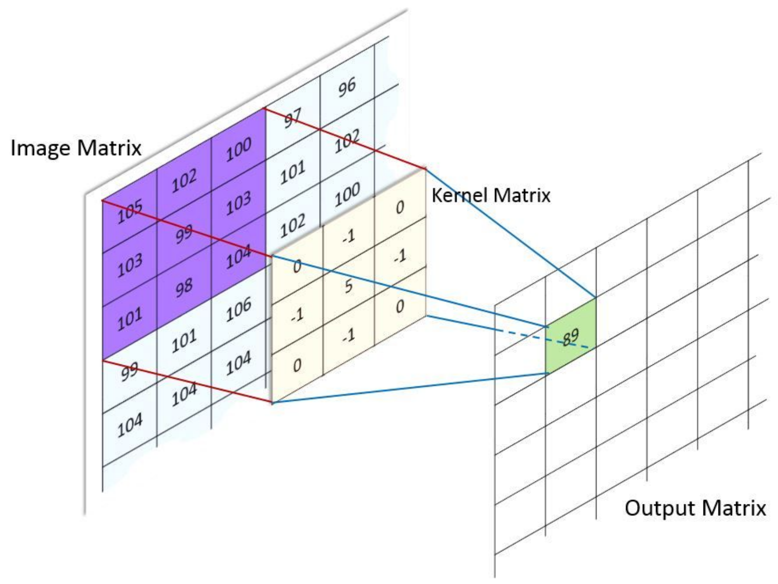
Convolutional Neural Network also popularly known as CNN , is a class of feed-forward neural network mostly used in image processing and computer vision. It is different from regular feed-forward neural network because it extracts relevant features from the input itself rather than providing them separately. It contains two parts i.e. Feature extraction and Classification(Fully Connected Layer).

Feature extraction part is performed by using different layers like convolution, pooling etc. Classification is done after extracting relevant features, it consists of fully connected layers. During both feature extraction as well as classification different activation functions are used in between the layers to capture non-linearity.

Below is the architecture of CNN if it has to classify whether an image is of cat or not.



(*Figure 1 : Different parts of CNN)*



(*Figure 2 : A single step of convolution operation on image matrix with 3x3 kernel)*

To make model better we can adjust hyperparameters accordingly. Hyperparameters for convolutional layer include stride, padding, kernel size etc.

Suppose an image of size ‘n’ is convolved with kernel matrix of size ‘f’ using padding ‘p’ and stride ‘s’ , then output image will be of given size :

$$Output\ size = \left(\frac{n + 2p - f}{s} + 1\right) x \left(\frac{n + 2p - f}{s} + 1\right)$$

(*Figure 3 : size of output image after convolution)*

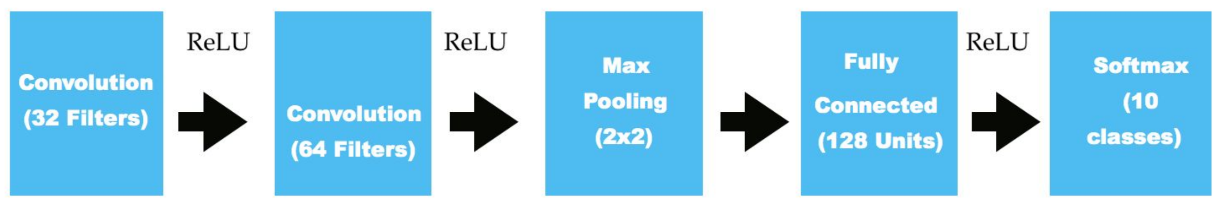
Architecture

A CNN architecture was implemented on MNIST dataset which identifies handwritten digits. It was implemented on Keras, an open source neural network library written in python.

The network consists different layers, which are as follows:

- Convolutional Layer with 32 Filters followed by ReLU activation function.
- Convolutional Layer with 64 Filters followed by ReLU activation function.
- Max Pooling Layer of 2x2.
- Fully Connected Layer with 128 units followed by ReLU activation function.
- Softmax Layer with 10 classes.

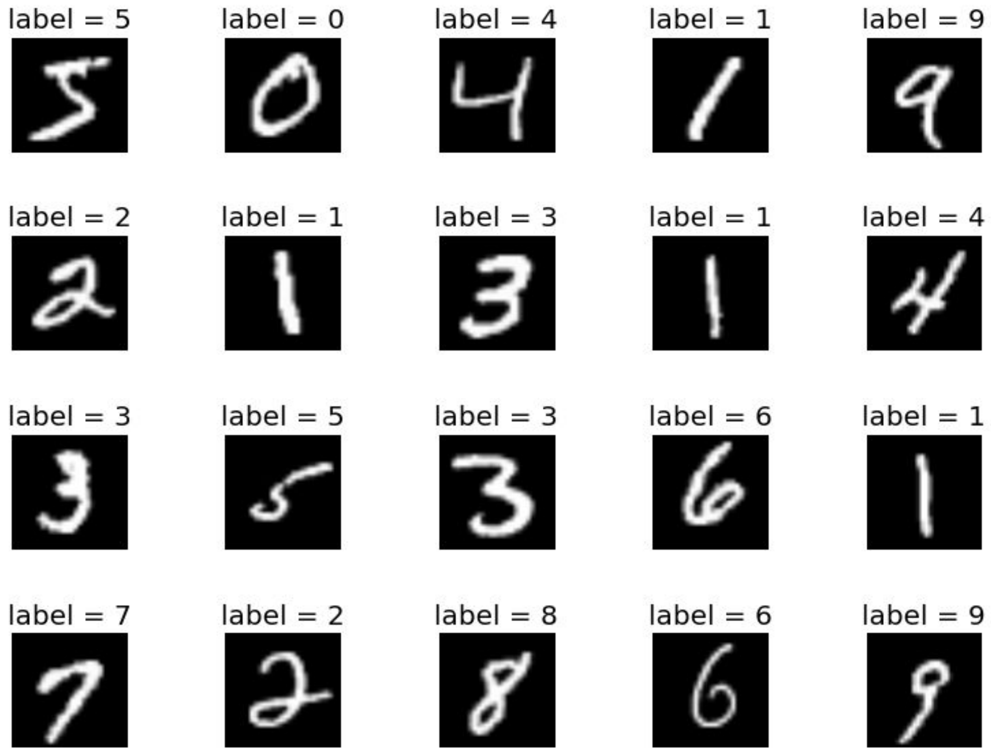
The graphical representation is given below :



(*Figure 4 : Used CNN architecture on MNIST Dataset)*

Dataset

The MNIST Dataset consists of images of handwritten digits containing 60,000 training example and 10,000 test example. Digits have been normalized and centred with only one channel each of 28x28 size. A sample of dataset is given below :



(*Figure 5 : A labeled sample from MNIST Dataset)*

Experiments

After implementing model on keras, it was trained on MNIST dataset. Different hyperparameters we tweaked to get different results. Some of the results are tabulated below using different optimizers :

Stochastic Gradient Descent:

- Batch Size = 128

Epochs	Learning rate	Stride	Kernel	Padding	Dropout	Accuracy
5	0.01	1	3 x 3	None	0.5	94
5	0.01	1	5 x 5	None	0.5	96
5	0.01	3	5 x 5	None	0.5	89
5	0.01	3	3 x 3	None	0.5	76

Adagrad Optimiser :

- Batch Size = 128
- Epsilon = None
- Decay = 0.0

Epochs	Learning rate	Stride	Kernel	Padding	Dropout	Accuracy
5	0.01	1	3 x 3	None	0.5	98
5	0.01	1	5 x 5	None	0.5	98
5	0.01	3	5 x 5	None	0.5	96
5	0.01	3	3 x 3	None	0.5	89

Adagrad Optimiser (Padding) :

- Batch Size = 128
- Epsilon = None
- Decay = 0.0

Epochs	Learning rate	Stride	Kernel	Padding	Dropout	Accuracy
5	0.01	1	3 x 3	Equal	0.5	98
5	0.01	1	5 x 5	Equal	0.5	99
5	0.01	3	5 x 5	Equal	0.5	97
5	0.01	3	3 x 3	Equal	0.5	96

Adam Optimiser :

- β 1 = 0.9
- β 2 = 0.999
- Epsilon = None
- Decay = 0.0
- Amsgrad = None

Epochs	Learning rate	Stride	Kernel	Padding	Dropout	Accuracy
5	0.001	1	3 x 3	None	0.5	99.13
5	0.001	1	5 x 5	None	0.5	99.23
5	0.001	3	5 x 5	None	0.5	97.71
5	0.001	3	3 x 3	None	0.5	91

Adamax Optimiser :

- Batch Size = 128
- β 1 = 0.9
- β 2 = 0.999
- Epsilon = None
- Decay = 0.0
- Amsgrad = None

Epochs	Learning rate	Stride	Kernel	Padding	Dropout	Accuracy
5	0.002	1	3 x 3	None	0.5	99.02
5	0.002	1	5 x 5	None	0.5	99.10
5	0.002	3	5 x 5	None	0.5	96.84
5	0.002	3	3 x 3	None	0.5	90.24

Adamax Optimiser (Varying Dropout) :

- Batch Size = 128
- β 1 = 0.9
- β 2 = 0.999
- Epsilon = None
- Decay = 0.0
- Amsgrad = None

Epochs	Learning rate	Stride	Kernel	Padding	Dropout	Accuracy
5	0.002	1	3 x 3	None	0.25	99.1
5	0.002	1	5 x 5	None	0.25	99.19
5	0.002	3	5 x 5	None	0.25	97.61
5	0.002	3	3 x 3	None	0.25	92.34

Adagrad Optimiser (On Different Epochs) :

- Batch Size = 128
- Epsilon = None
- Decay = 0.0

Epochs	Learning rate	Stride	Kernel	Padding	Dropout	Accuracy
5	0.01	3	5 x 5	Equal	0.25	98
12	0.01	3	5 x 5	Equal	0.25	98.71
20	0.01	3	5 x 5	Equal	0.25	98.83
30	0.01	3	5 x 5	Equal	0.25	98.88

Analysis

After doing different experiments based on various hyperparameters many observation was made.

- On this dataset, It is found that using kernel size of ‘5x5’ produced more accuracy than kernel size of ‘3x3’ keeping other things constant.
- On this dataset, Training time when using stride of 3 and kernel size of ‘5x5’ was the least.
- On this dataset, The least accuracy was observed when stride of 3 and kernel size of ‘3x3’ was used.
- On this dataset, Padding seems to increase the accuracy.
- On this dataset, Adam optimiser produced better results than others.
- On this dataset, As No. of epochs is increased the accuracy increased.
- On this dataset, Varying dropout from 0.50 to 0.25 increased the accuracy.

Conclusion

After implementing CNN and changing different hyperparameters , The effect of changing particular hyperparameter was observed. It is concluded that there is always a trade-off between fast result(training time) and accuracy , So we should adjust our hyperparameters based on our dataset or need .