CS7015 Deep Learning
Programming Assignment 3
Convolutional Neural Network

CS17S008 Nitesh Methani
CS17S013 Pritha Ganguly

March 30, 2018

Contents

List of Figures

1 Know your Data

1.1 Why not MNIST?

A few reasons can be enumerated for choosing Fashion MNIST over MNIST [4].

- **MNIST is too easy** : Convolutional nets can achieve 99.7% on MNIST easily, and similarly, even classic ML algorithms can achieve 97%. 11
- **MNIST is overused** : Almost everyone who has experience with deep learning has come across MNIST at least once.
- **MNIST cannot represent modern CV tasks** : This statement was noted by Franois Chollet, the author of the Keras library.

1.2 Glimpse of Fashion MNIST

Researchers at Zalando(the e-commerce company) had developed a Fashion-MNIST dataset to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

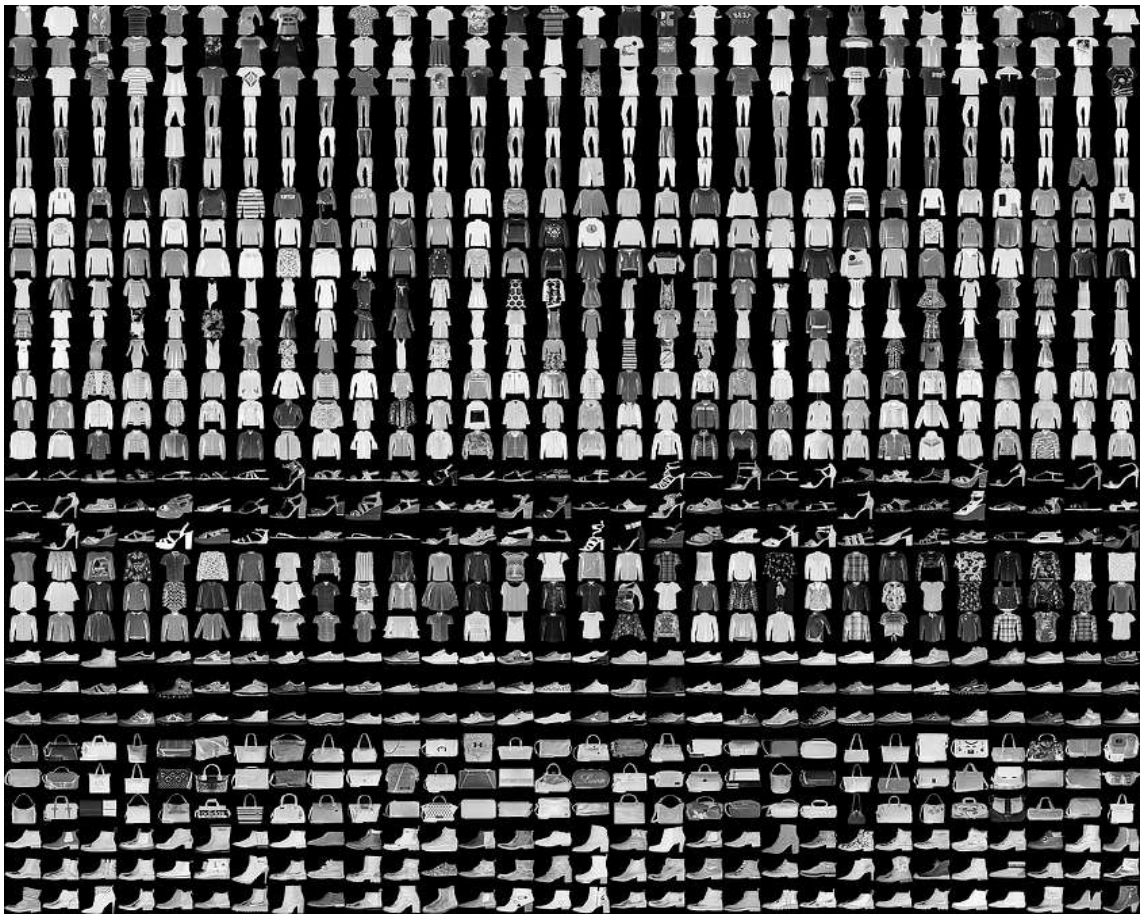


Figure 1: A glimpse of FASHION-MNIST dataset

This new dataset contains images of various types of clothing and accessories such as shirts, bags, shoes and other fashion items.

2 Data Augmentation

We have used Tensorflow to do data augmentation. Here are the few examples of different techniques that we have used in our assignment. Note that we have used the images from the article [5] just for illustration purposes:

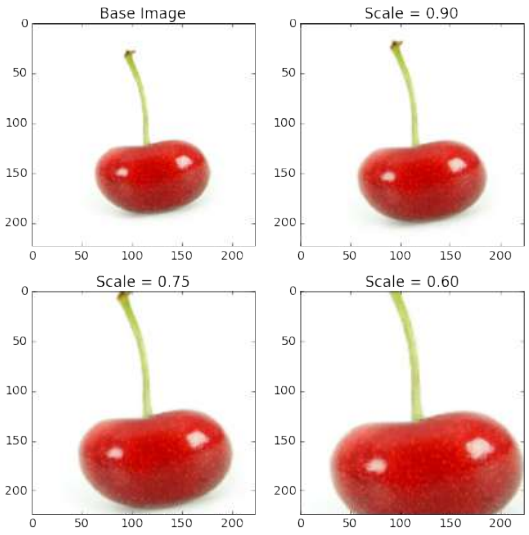


Figure 2: **Scaling**

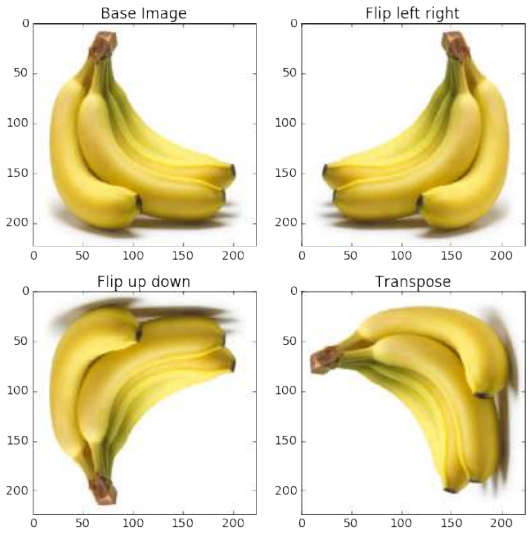


Figure 3: **Flipping**

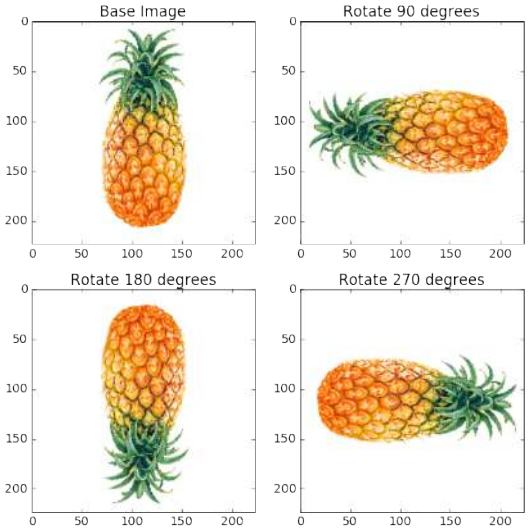


Figure 4: **Rotation (at 90 degree)**

Figure 5: **Lighting Condition**

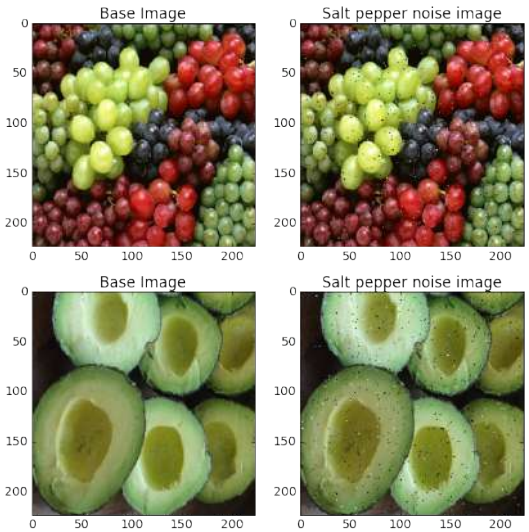


Figure 6: **Adding Salt and Pepper Noise**

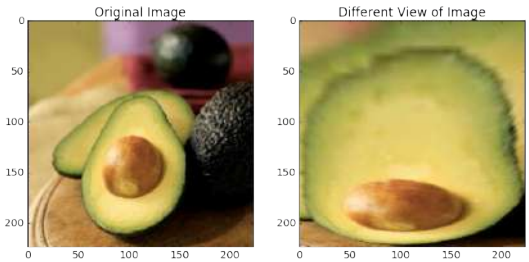


Figure 7: **Perspective Transform**

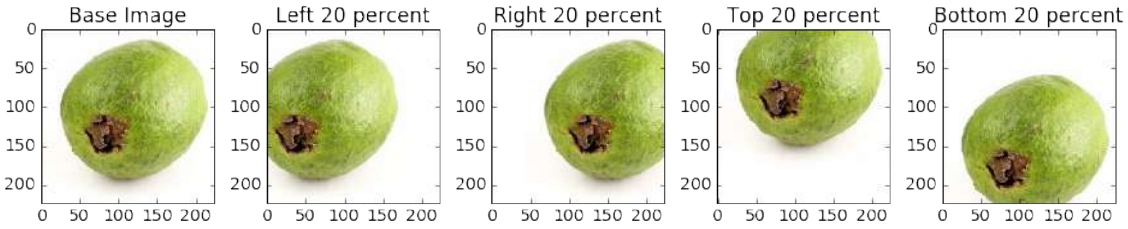


Figure 8: **Translation**

3 Steps to build network

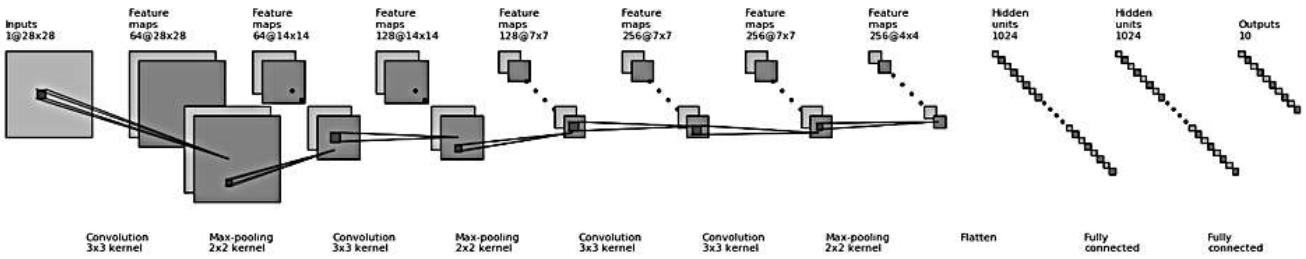
Here are the steps that we carried out to build the CNN model:

1. Set up the network parameters.
2. Create tensorflow placeholders.
3. Initialize the parameters.
4. Forward propagation.
5. Compute cost.
6. Backpropagation

4 Our ConvNet

1. Best Performing Model:
   Our best performing model is the same as the one suggested in the assignment.

Figure 9: **Our ConvNet**

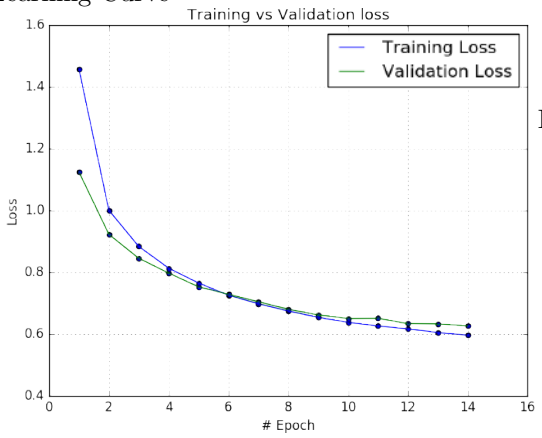


2. Number of parameters in Our ConvNet model are:

Layer	Input Size	Output Size	# Parameters
Conv 1	(28,28,1)	(28,28,64)	$3 \times 3 \times 64 = 576$
Max Pool 1	(28,28,64)	(14,14,64)	0
Conv 2	(14,14,64)	(14,14,128)	$3 \times 3 \times 128 \times 64 = 73728$
Max Pool 2	(14,14,128)	(7,7,128)	0
Conv 3	(7,7,128)	(7,7,256)	$3 \times 3 \times 128 \times 256 = 294912$
Conv 4	(7,7,256)	(7,7,256)	$3 \times 3 \times 256 \times 256 = 589824$
Max Pool 3	(7,7,256)	(4,4,256)	0
FC 1	(4096,1)	(1024,1)	$4069 \times 1024 = 4194304$
FC 2	(1024,1)	(1024,1)	$1024 \times 1024 = 1048576$
Output	(1024,1)	(1024,10)	$1024 \times 10 = 10240$

Table 1: Parameters of Our ConvNet

3. Learning Curve



Default Architecture:

- Batch-size = 1000
- Initialization = Xavier Initialization
- lr = 0.0001
- Optimizer = ADAM
- Keep Prob. of Dropout = 0.8

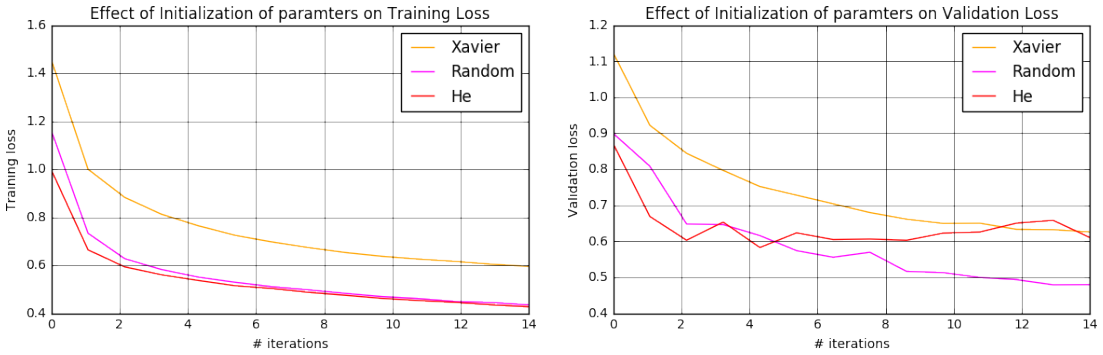
Figure 10: **Learning Curve**

4. Best Validation Accuracy that we achieved was 92.86%. The same model when used on test data gave a Kaggle Accuracy of 93.7%.

5 Weights Initialization

Initializing the parameters of the neural network, namely the weight matrices and biases is an important step in training the network. The neurons in the hidden layer become symmetric if the weights are initialized to zero. On the other hand, if the weights are large, the algorithm might end up in a plateau which implies that the learning will become very slow. Therefore, we have used Xavier and/or He initialization for our weights. The biases, on the other hand has been initialized with zeros. This initialization technique also ensures that the initial weights are small, by multiplying them with some constant which is inversely proportional to the number of neurons in the network.

As it can be observed from the graphs below that the method in which the weights are initialized (Xavier/He/random) have a significant impact on the rate of learning of the network. As it happens in this experiment, the random weights were better initializers but we always cannot depend on it as it is non deterministic. Overall we can see He initializer performs better on this network.

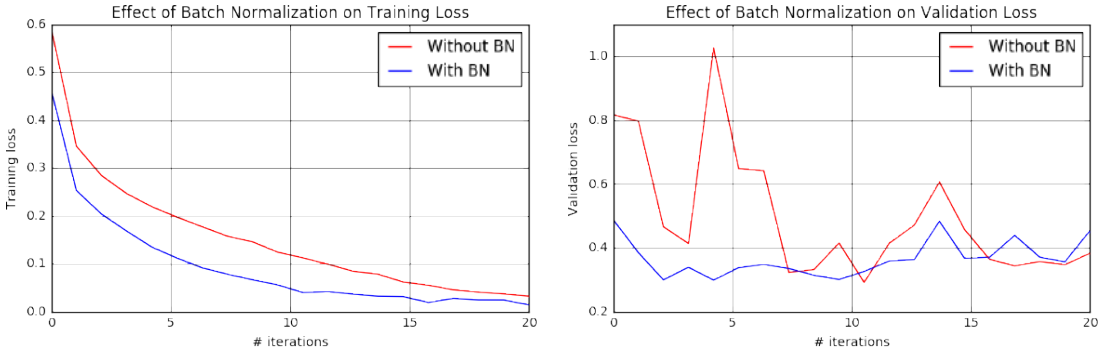


(a) Training Loss (b) Validation Loss

Figure 11: Comparison between different Weight Initialization Techniques

6 Batch Normalization

As normalizing the inputs before passing it onto a model helps in effective training, similarly, for every hidden layer, we can normalize the activations so as to make the training of weights and biases more efficient. In deep networks, small changes to the network gets resonated down the network which causes the input distribution to the hidden layers to shift. Batch normalization reduces the amount by which values of the hidden layers shift around. This is also known as internal covariate shift. It also has a slight regularization effect as it adds some noise to the hidden layer's activations.



(a) Training Loss (b) Validation Loss

Figure 12: Effect of Batch Normalization on our ConvNet

We can observe from the above graphs that a network with batch normalization learns faster as compared to the network without it(Figure 12a). It's effect on validation loss is significant as we see that without batch normalization, the loss constantly oscillates with huge margins especially in the initial iterations.

7 Dropout

Dropout is a regularization technique where in each training phase, individual neurons are either ignored(or dropped) with a probability p or considered(or kept) with probability p, such that a reduced network remains. Dropout effectively prevents over-fitting as a fully connected layer occupies most of the parameters. The neurons, in this case, develop co-dependency amongst each other during training. Introducing dropouts, enable each neuron to learn some feature of the data independently of the other neurons in that layer.

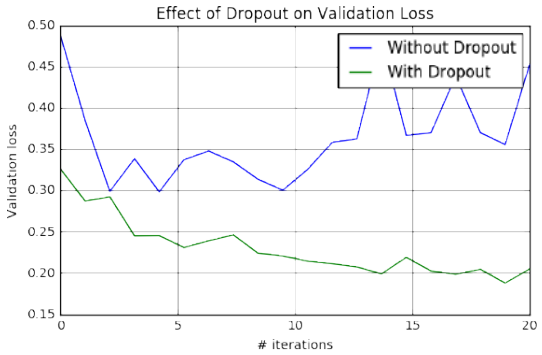


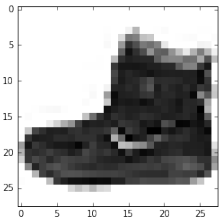
Figure 13: Effect of Dropout(probability 0.8) on our ConvNet

The effect of dropout on the network can be seen as validation loss on a network with dropouts follows a constant decreasing trend but without dropouts, it behaves randomly.

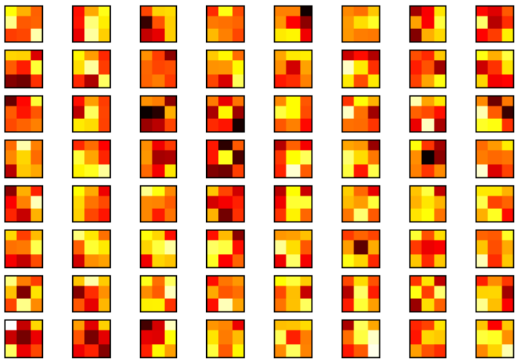
8 Visualizations

8.1 Visualizing Convolutional Layers

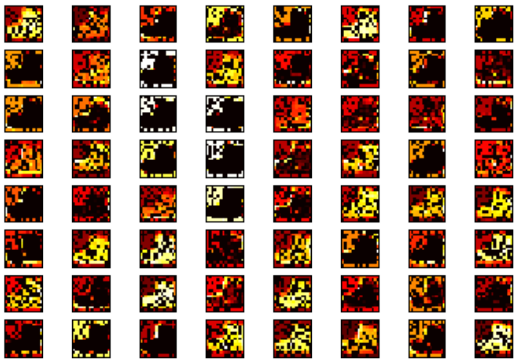
In this sub-section, we have shown the activations of the network during the forward pass. We have plotted the 64 layer1 filters and the effect of the convolutional operator of these filters on the Ankle Boot image are also plotted below: We ca observe that, the activations usually start out



(a) Ankle Boot



(b) Layer 1 Filter weights



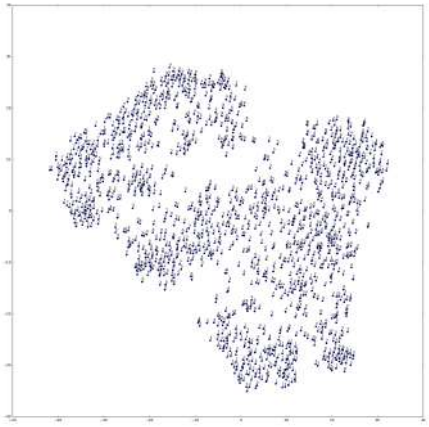
(c) Conv Layer 1 output

Figure 14: Layer Activations and Conv Output Visualizations

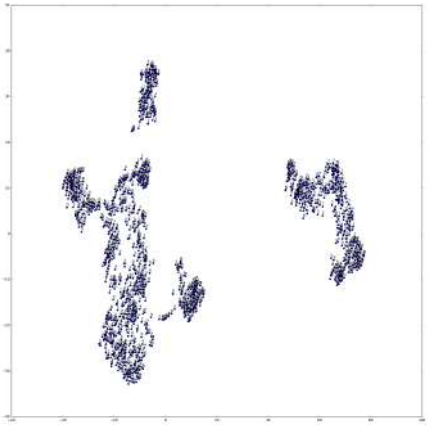
looking relatively blobby and dense, but as the training progresses the activations usually become more sparse and localized. One shortcoming of this technique can be easily noticed that some activation maps may be all zero for many different inputs, which can indicate dead filters, and can be a symptom of high learning rates. [13]
The conv layer outputs are useful to visualize as we can see nice and smooth patterns as in Figure 14(c). This ensures that the network has been properly trained. If the patterns are noisy, that could be an indicator of a poorly trained network.

8.2 Embedding the codes with t-SNE

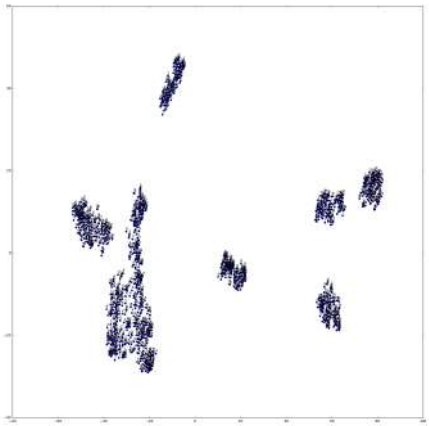
t-SNE stands for t-distributed stochastic neighbor embedding. It is a non-linear dimensionality reduction technique and is often used in machine learning for visualizing higher dimensional data . We have applied this technique on the Fashion-MNIST dataset and the visuals for few training epochs are plotted below.



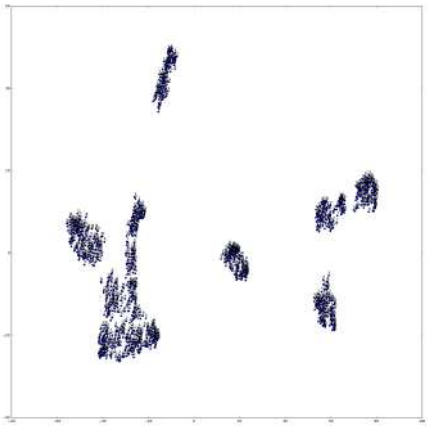
(a) Before training



(b) After 300 epochs



(c) After 7300 epochs

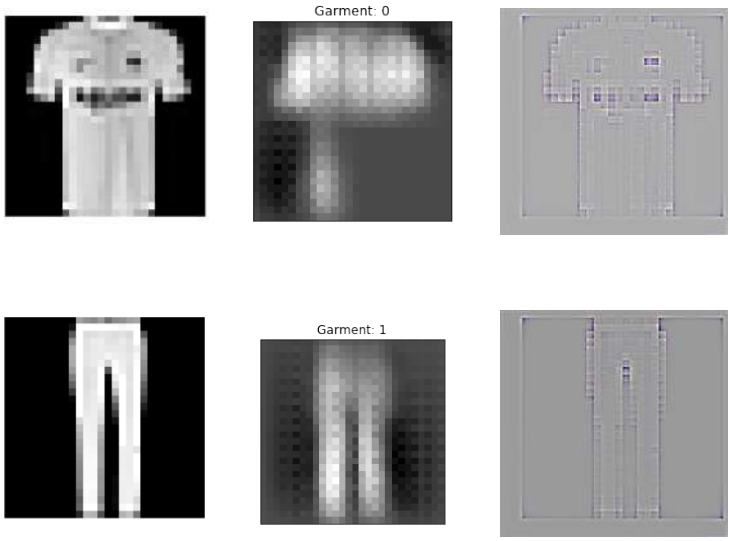


(d) After 9900 epochs

We can observe that as we increase the number of epochs the data is getting clustered which can be interpreted as CNNs gradually transforming the images into a representation in which the classes are separable by a linear classifier. We can get a rough idea about the topology of this space by embedding images into two dimensions.[13]
Later we have also used this technique to support our claim made in section 1.1

8.3 Guided Backpropagation

The motivation behind guided backpropagation is that neurons act like detectors of particular image features. In general, we are only interested in what image features the neuron detects, not in what kind it doesn't detects. So when propagating the gradient, we use a modified ReLu known as Guided ReLu which sets all the negative gradients to 0.



(a) Original Image (b) Our ConvNet (c) VGG

Figure 15: Visualizing Guided Back-propagation

We have randomly selected 10 channels of the conv4 layer output and have applied guided back-propagation on it. We can see interesting patterns that excites the neuron.
For sanity check, we have referred the repository [11] which uses VGG model for guided backpropagation.

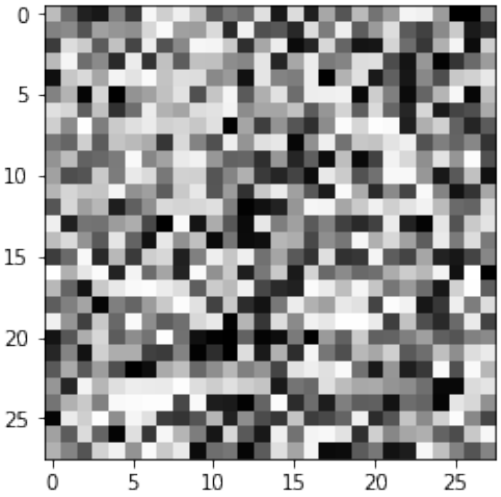
9 Fooling

Convolutional neural networks are powerful models that consider the entire image when classifying it. They can recognize complex shapes and patterns no matter where they appear in the image.
But if you know exactly which pixels to change and exactly how much to change them, you can intentionally force the neural network to predict the wrong output for a given picture without changing the appearance of the picture very much.
That means we can intentionally craft a picture that is clearly a prohibited item but which completely fools our neural network.
There are many ways in which we can fool the CNN. [8]

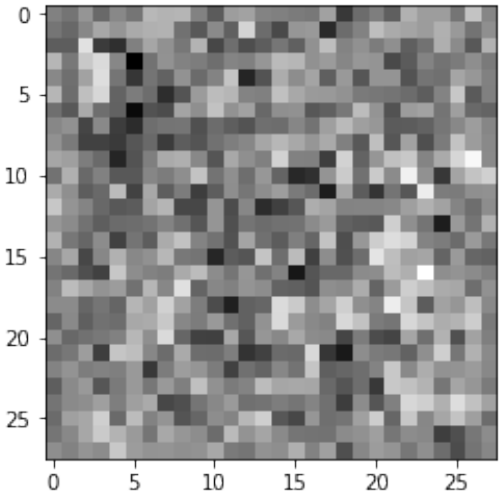
- One way is to start with a noisy image and execute the following steps:

  1. Feed in the noisy image, Figure (a) below.
  2. Check the neural networks prediction (the network was predicting the noisy image as Bag (class 8)).
  3. Decide the class in which you want the noisy image to belong to (say class 0 (T-shirt/Top)).
  4. Tweak the photo using back-propagation to make the final prediction slightly closer to the answer you want to get.
  5. Repeat steps $1-3$ a few thousand times with the same photo until the network gives the answer you want. (Figure (b) below).

     At the end of this, well have an image that fools the neural network without changing anything inside the neural network itself.



(a) Noisy Image



(b) Predicted as T-shirt/Top

- The other way is to take a pre-trained model. Now an image from the dataset and keep on adding some noise to it till it gets misclassified. We tried this using pre-trained VGG model on CIFAR-10 dataset. The original image (a) and the hacked image (b) that we got are shown below.

(a) Persian Cat

(b) Hacked Cat (Goldfish)

Even though both the images look identical to us, it still fools the neural network!
It predicts the Persian cat (a) as goldfish (b) with very high confidence.

10 Experiments

10.1 List of Models

The architecture of the different models that we have used for experimenting are listed below (for details refer to the Appendix section 13).
Note : In section 13, we have only mentioned the basic layer architecture of the models. During experimentation, we have used dropout and batch normalization.

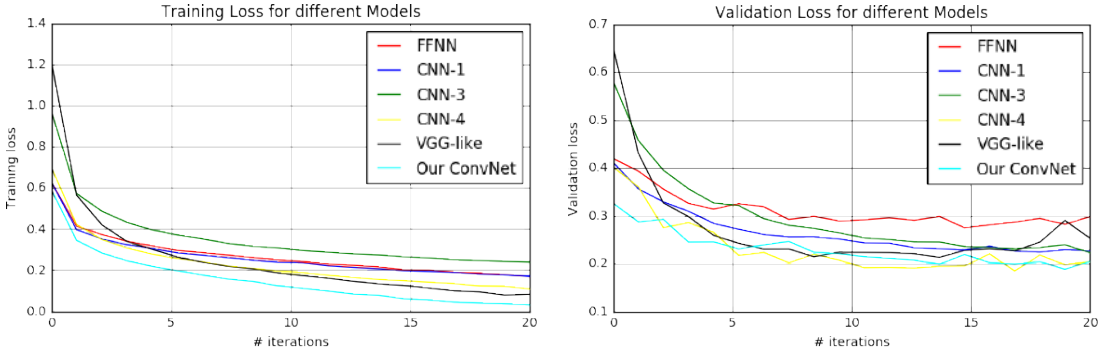
- Feed Forward Neural Network
- CNN with 1 Convolutional Layer

- CNN with 3 Convolutional Layers
- CNN with 4 Convolutional Layers
- VGG Like Model
- Our ConvNet (default architecture as suggested)

We have used a subset of these models to experiment with the hyperparameter space.

10.2 Hyperparameters

10.2.1 Number Of Layers

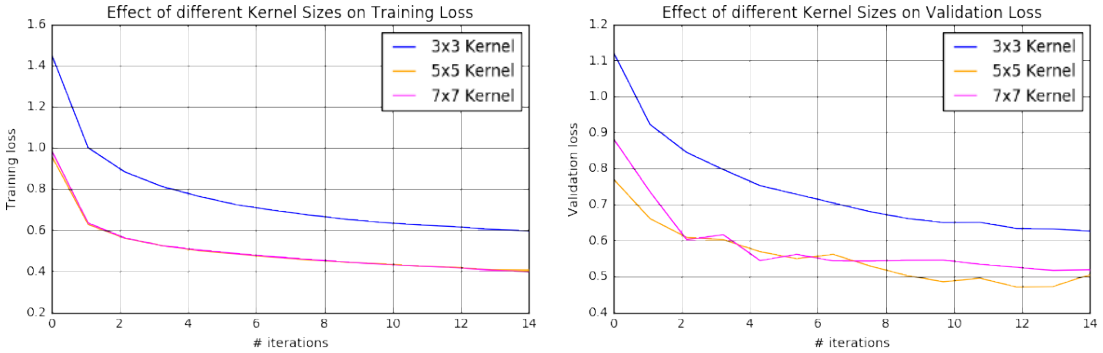


(a) Training Loss (b) Validation Loss

Figure 16: Comparison of loss for different models

It's a common notion as increasing the number of layers in CNN always increases the performance of the network. The trend of decreasing training losses also supports that claim. But this may not hold always as we can see that though VGG-16 has more number of layers than our ConvNet, both the training and validation losses of the latter decreases more rapidly than VGG.

10.2.2 Filter Size

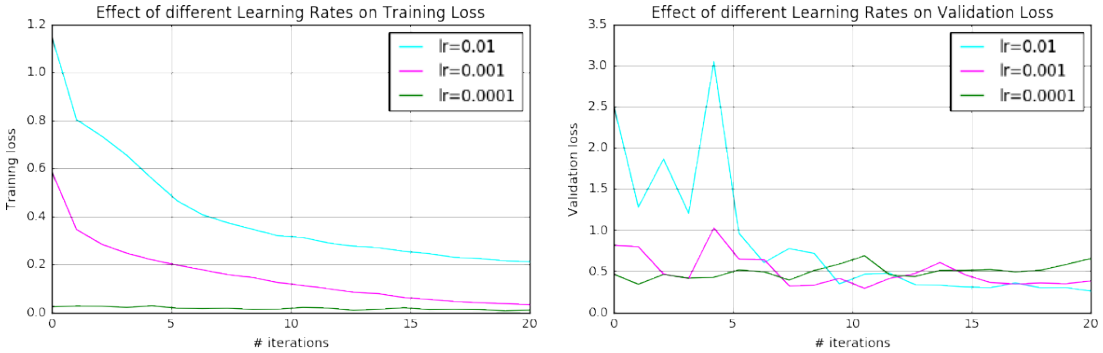


(a) Training Loss (b) Validation Loss

Figure 17: Effect of different Kernel Sizes on our ConvNet

Usually increasing the filter size of the kernel makes the network learn better representations of the image as a wider patch of the image can be captured by a 5x5 image as compared to a 3x3 kernel. But, a 7x7 filter may not capture the subtle details of the image due to it's wider range. Therefore, it's essential to determine the optimal filter size which would help the CNN to learn better abstract representations. For our ConvNet, 5x5 kernel gives better results(as per above graph). In our experiments, we have also observed that a 3x3 kernel too gives good results.

10.2.3 Learning Rate

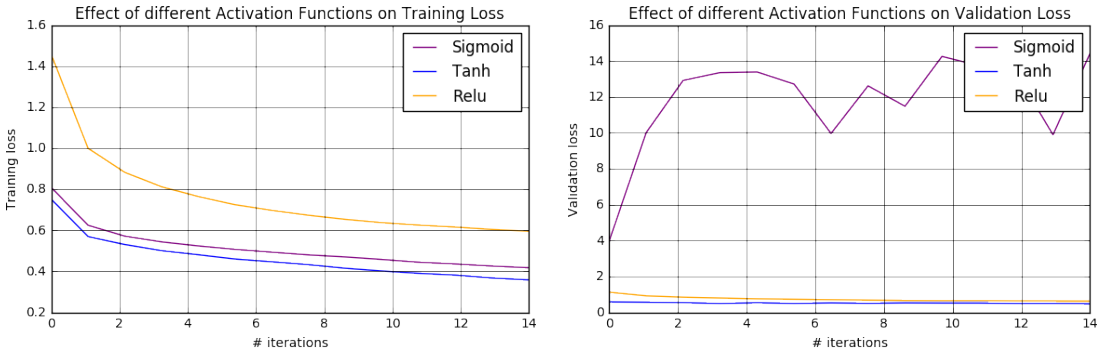


(a) Training Loss (b) Validation Loss

Figure 18: Effect of different learning rates on our ConvNet

The effect of learning rate on the training of any neural network is significant as the rate determines how fast/slow the cost function reaches the minima. Keeping it high, might result in the network oscillating near the optimal point, while keeping it low might result in the network getting stuck in a plateau for a long time. As per the above graphs, the learning rate of 0.0001 works best for our network.

10.2.4 Activation Functions

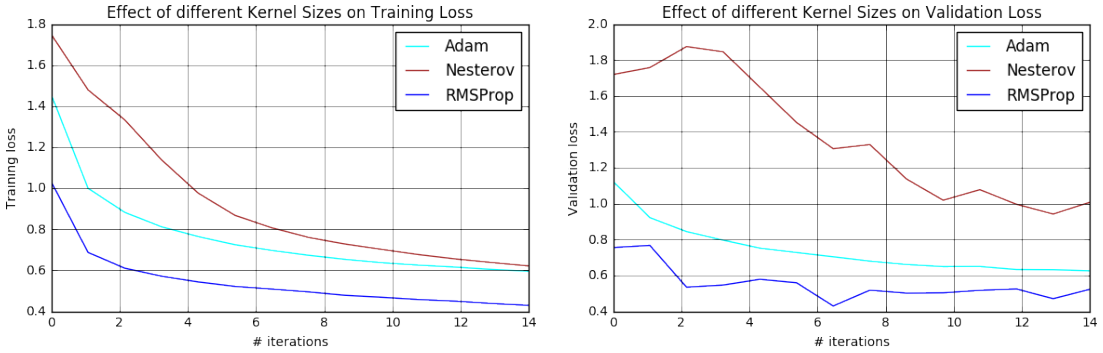


(a) Training Loss (b) Validation Loss

Figure 19: Comparison of loss for different activation functions on our ConvNet

As it can be observed from the graphs, ReLu activation function works best with CNN models as opposed to sigmoid or tanh.

10.2.5 Optimization Algorithms

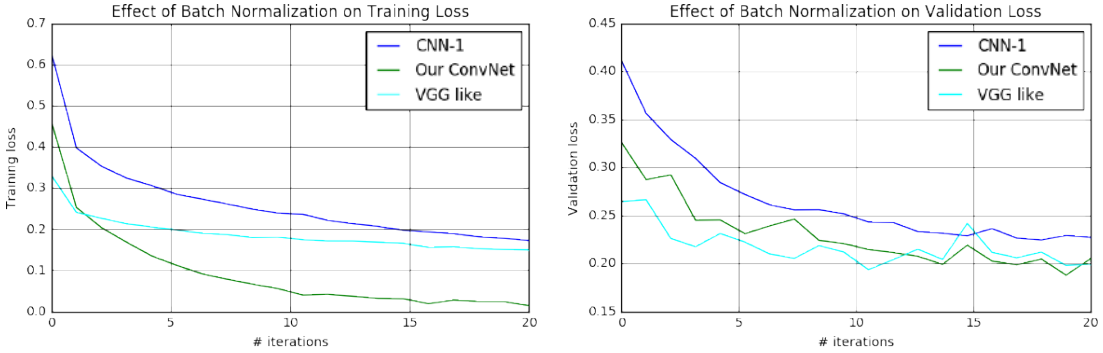


(a) Training Loss (b) Validation Loss

Figure 20: Effect of different learning algorithms on our ConvNet

As expected both ADAM and RMSProp works equally well as optimization algorithm for training the CNN network while minimizing the cross entropy loss.

10.2.6 Batch Normalization

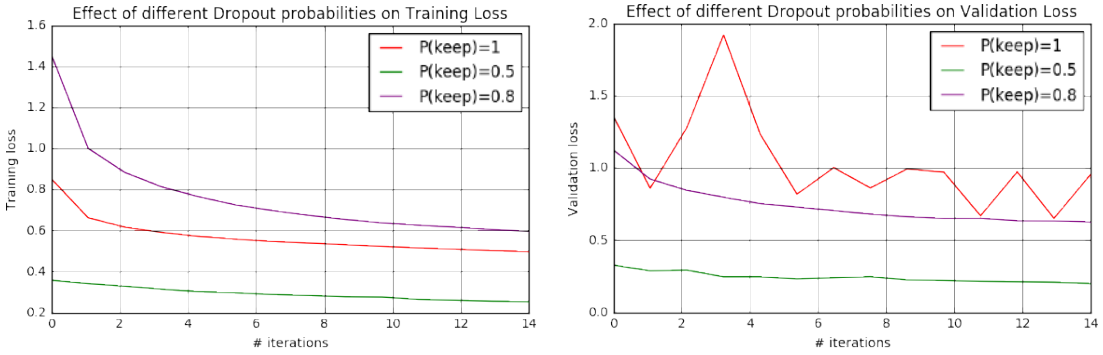


(a) Training Loss (b) Validation Loss

Figure 21: Effect of Batch Normalization on different models

As discussed in 6, batch normalization always makes training faster.

10.2.7 Dropout

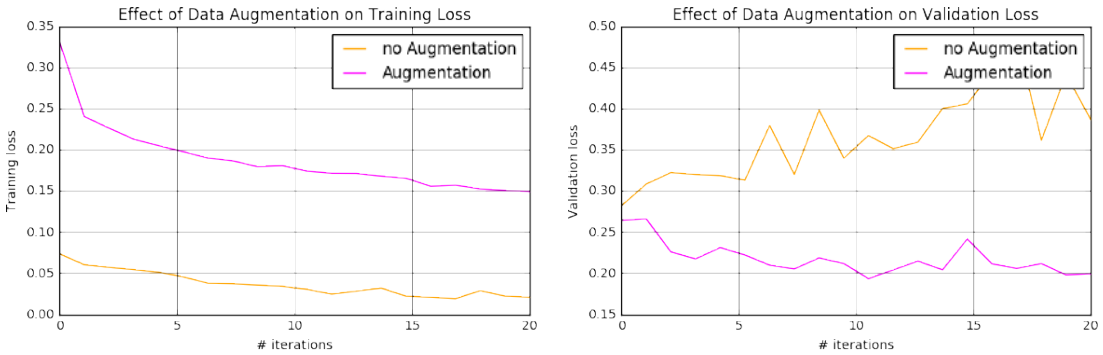


(a) Training Loss (b) Validation Loss

Figure 22: Effect of Dropout with different probabilities on our ConvNet

As discussed in 7, introducing dropouts makes the network learn faster as effective number of parameters decrease.

10.2.8 Data Augmentation



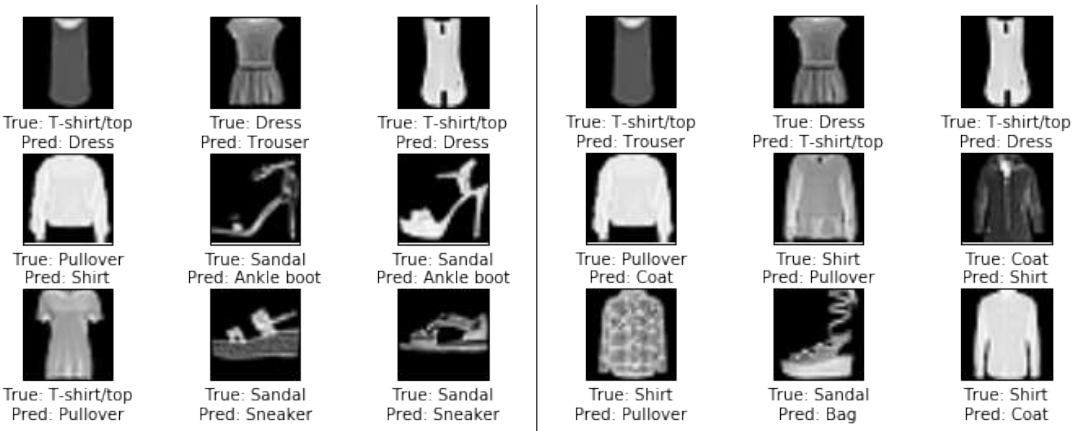
(a) Training Loss (b) Validation Loss

Figure 23: Effect of Data Augmentation on VGG like network

Data augmentation prepares the network to perform better when it sees unseen examples i.e it provides better generalization of the data. It's a kind of regularization technique.

11 Observations

Few images that were misclassifed, after 100^{th} and 1000^{th} epochs, are presented below:



(a) After 100 training steps (b) After 1000 training steps

Figure 24: Misclassified Images

The confusion matrix for our ConvNet is plotted on the left. We can observe that class 6 (Shirt) is hard to classify and class 1 (Trouser) is very easily classified by the network.

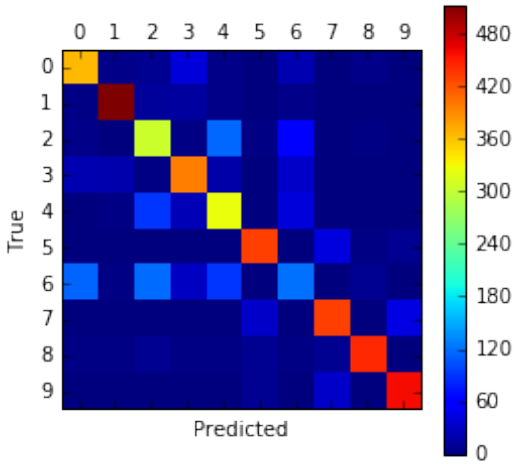
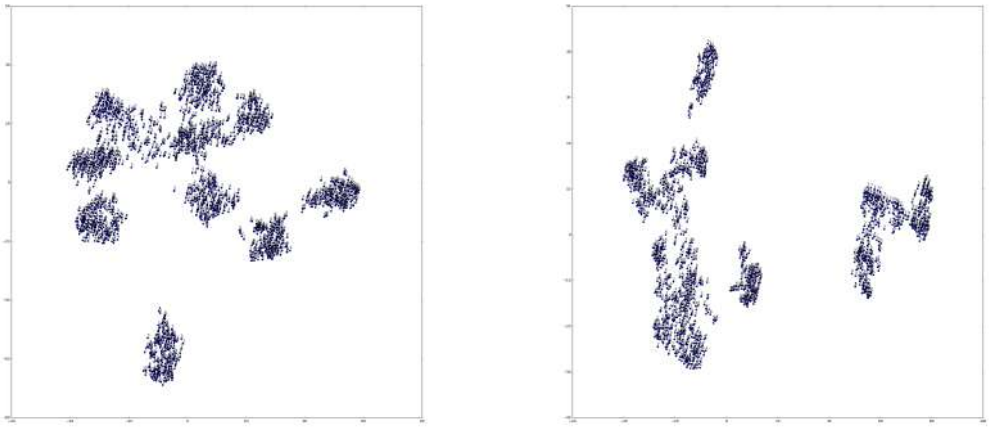


Figure 25: Confusion Matrix

The following plot shows the t-SNE representation of MNIST and Fashion-MNIST dataset after 400 epochs.



(a) MNIST (b) Fashion-MNIST

Figure 26: Comparison between t-SNE representations

We can see that MNIST data is well clustered in just 400 epochs unlike Fashion-MNIST dataset. This observation justifies the claim made in Section 1.1.

12 Conclusion

- All the models achieved a higher accuracy after using data augmentation. Almost always use data augmentation!!
- Regularization techniques like batch-normalization, dropout, etc. achieves better accuracy.
- VGG Like Model With Batchnorm performed the best and achieved a accuracy of 94% using data augmentation.
- We can see that Fashion MNIST could be a potential replacement to MNIST after all, firstly because its a more challenging dataset, and secondly because it gives us a lot more room for improving our modelusing hyperparameter tuning, regularisation, etc.

13 Appendix

The configuration details of the models with which we experimented can be tabulated as follows:

Models	No. of Layers	Sizes
FeedForward Network	2 hidden layers	Layer 1 : 512 Layer 2 : 128 Output : 10
CNN	1 Conv Layer	Conv1 : 32 3 x 3 filters MaxPool1 : 2 x 2 filter FC 1 = 128 neurons Output : 10
CNN	3 Conv Layer	Conv1 : 32 3 x 3 filters MaxPool1 : 2 x 2 filter Conv2 : 64 3 x 3 filters MaxPool2 : 2 x 2 filter Conv3 : 128 3 x 3 filters MaxPool3 : 2 x 2 filter FC 1 = 128 neurons Output : 10
CNN	4 Conv Layer	Conv1 : 32 3 x 3 filters Conv2 : 32 3 x 3 filters Conv3 : 64 3 x 3 filters MaxPool1 : 2 x 2 filter Conv4 : 128 3 x 3 filters FC 1 = 512 neurons FC 2 = 128 neurons Output : 10
CNN	VGG like	Conv1 : 32 3 x 3 filters Conv2 : 32 3 x 3 filters MaxPool1 : 2 x 2 filter Conv3 : 64 3 x 3 filters Conv4 : 64 3 x 3 filters MaxPool2 : 2 x 2 filter Conv5 : 128 3 x 3 filters Conv6 : 128 3 x 3 filters MaxPool3 : 2 x 2 filter Conv7 : 256 3 x 3 filters Conv8 : 256 3 x 3 filters Conv9 : 256 3 x 3 filters MaxPool4 : 2 x 2 filter FC 1 = 512 neurons FC 2 = 512 neurons Output : 10
CNN	Our ConvNet	Conv1 : 64 3 x 3 filters MaxPool1 : 2 x 2 filter Conv2 : 128 3 x 3 filters MaxPool2 : 2 x 2 filter Conv3 : 256 3 x 3 filters Conv4 : 256 3 x 3 filters MaxPool3 : 2 x 2 filter FC 1 = 1024 neurons FC 2 = 1024 neurons Output : 10

References

[1] Mitesh Sir's slides, *course slides*

[2] Dropout *https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5*

[3] Image Convolution *http://setosa.io/ev/image-kernels/*

[4] Fashion MNIST dataset *https://medium.com/tensorist/classifying-fashion-articles-using-tensorflow-fashion-mnist-f22e8a04728a*

[5] Data Augmentation *https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9*

[6] Fooling1 *https://github.com/Evolving-AI-Lab/fooling*

[7] Fooling2 *https://github.com/bethgelab/foolbox*

[8] Fooling3 *https://medium.com/@ageitgey/machine-learning-is-fun-part-8-how-to-intentionally-trick-neural-networks-b55da32b7196*

[9] ROC *https://blogs.sas.com/content/iml/2011/07/29/computing-an-roc-curve-from-basic-principles.html*

[10] Guided Backpropagation-1 *https://github.com/1202kbs/Understanding-NN*

[11] Guided Backpropagation-2 *https://github.com/conan7882/CNN-Visualization*

[12] Guided Backpropagation-3 *https://github.com/Lasagne/Recipes/blob/master/examples/Saliency%20Maps%20and%20Guided%20Backpropagation.ipynb*

[13] CS231 *http://cs231n.github.io/understanding-cnn/*