HOMEWORK 4 - CSCI 5922 - Neural Networks and Deep Learning

Name : Ashwin Sankaralingam
SID : **108593584**

1.a) Using K folds algorithm. Setting to k=5, so creating 5 different models for the given architecture and finding the mean accuracy of each model to find which is the better model.

Training :40000
Validation : 10000
Test : 10000

So after each k, the training set(50000) is shuffled and the first 40000 goes to training and rest for validation.

1.b) Variant 1: One Convolutional Layer -> FC

This model was the most basic convolutional layer model with just one convolutional layer and then without any pooling layer is squashed into a fully connected layer.

Data Augmentation	**Pooling Method**	**Normalisation**	**Dropout**
Flipping random pictures up and down, adjusting brightness and contrast by random	Max Pooling	Local response normalisation (http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf) Hinton et all	No dropouts
Architecture	Input-> Convolutional Layer[features=64,kernel_size = 3*3]-> MaxPooling[stride=2*2]-> outputFC[reshaped,10]		
Accuracy	Average Train Accuracy of 5 models(K-Fold)	54.23%	
	Average Validation Accuracy of 5 models (K-Fold)	50.98%	
	Average Test Accuracy of 5 models (K-Fold)	42.21%	

For all rest of the variations I am using Local Response normalization which can be defined by normalising the activation unit.

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

Where ‘a’ is the output of the activation unit.
For all the variants I am using a decaying learning rate which falls exponentially every 250 epochs.
It starts with 0.1 and every 250 iterations, it decays exponentially by factor of 0.1.
Number of epochs was varied around 2000-3500 based on the convergence.
Activation unit selected in RELU after each Convolutional Layer.
For all variants the batch size was set to 512.

Variant 2: Convolutional Layer -> Max Pooling -> Convolutional Layer -> Max Pooling -> FC

Data Augmentation	**Pooling Method**	**Normalisation**	**Dropout**
Flipping random pictures up and down, adjusting brightness and contrast by random	Max Pooling	Local response normalisation	No dropouts
Architecture	Input-> Convolutional Layer[features=64,kernel_size = 3*3]-> MaxPooling[stride=2*2]-> Convolutional Layer[features=64*64 ,kernel_size = 5*5]-> MaxPooling[stride=2*2]-> outputFC[reshaped,10]		
Accuracy	Average Train Accuracy of 5 models(K fold)		72.12%
	Average Validation Accuracy of 5 models (Kfold)		76.78%
	Average Accuracy of Test of 5 models(Kfold)		67.34%

Variant 3: Three Convolutional Layer -> Max Pooling ->2 FC (with batch normalisation and dropouts)

Data Augmentation	**Pooling Method**	**Normalisation**	**Dropout**
Flipping random pictures up and down, adjusting brightness and contrast by random	Max Pooling	Batch Normalisation	Probability= 0.4
Architecture	Input->Convolutional Layer[features=64,kernel_size = 3*3]->Dropout[0.4]->MaxPooling[stride=2*2]->Convolutional Layer[features=64*64 ,kernel_size = 5*5]->Dropout[0.4]->MaxPooling[stride=2*2]->outputFC[reshaped,10]		
Accuracy	Average Train Accuracy of 5 models(K fold)	80.34%	
	Average Validation Accuracy of 5 models (Kfold)	76.12%	
	Average Accuracy of Test of 5 models(Kfold)	73.21%	

Variant 4: 3 Convolutional Layer -> Max Pooling -> 3FC (with dropout 0.4)

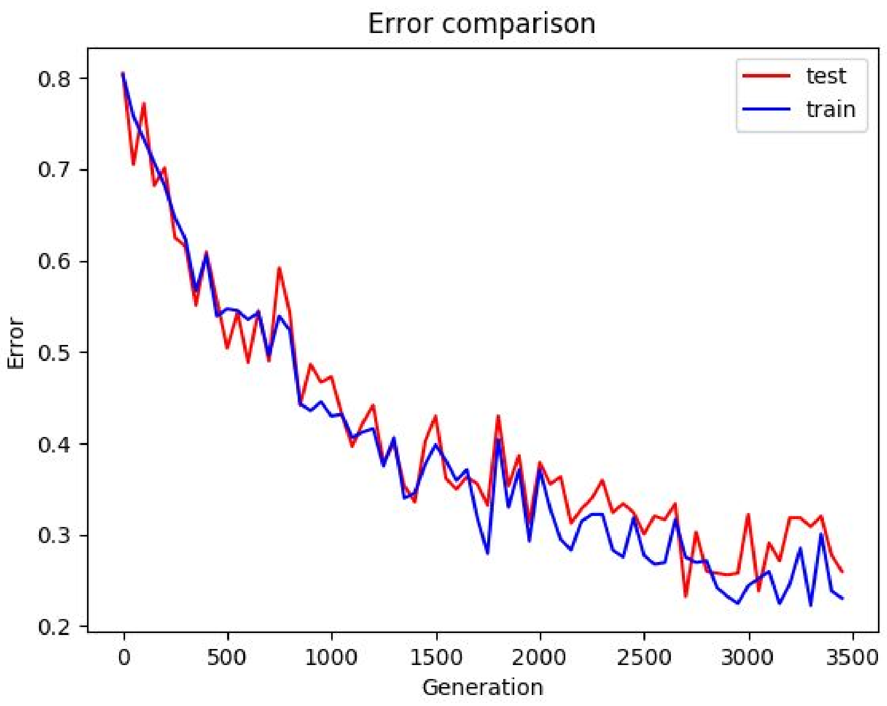
Data Augmentation	**Pooling Method**	**Normalisation**	**Dropout**
Flipping random pictures up and down, adjusting brightness and contrast by random	Max Pooling	Local response normalisation	Probability 0.4
Architecture	Input-> Convolutional Layer[features=64,kernel_size = 3*3]->		

	Dropout[0.4]->MaxPooling[stride=2*2]-> Convolutional Layer[features=64*64 ,kernel_size = 5*5]->Dropout[0.4]->MaxPooling[stride=2*2]-> Convolutional Layer[features=64*64 ,kernel_size = 6*6]->Dropout[0.4]-> MaxPooling[stride=3*3]-> FC[reshaped,384]-> FC[384,192]-> outputFC[192,10]	
Accuracy	Average Train Accuracy of 5 models(K fold)	85.23%
	Average Validation Accuracy of 5 models (Kfold)	83.12%
	Average Accuracy of Test of 5 models(Kfold)	73.89%

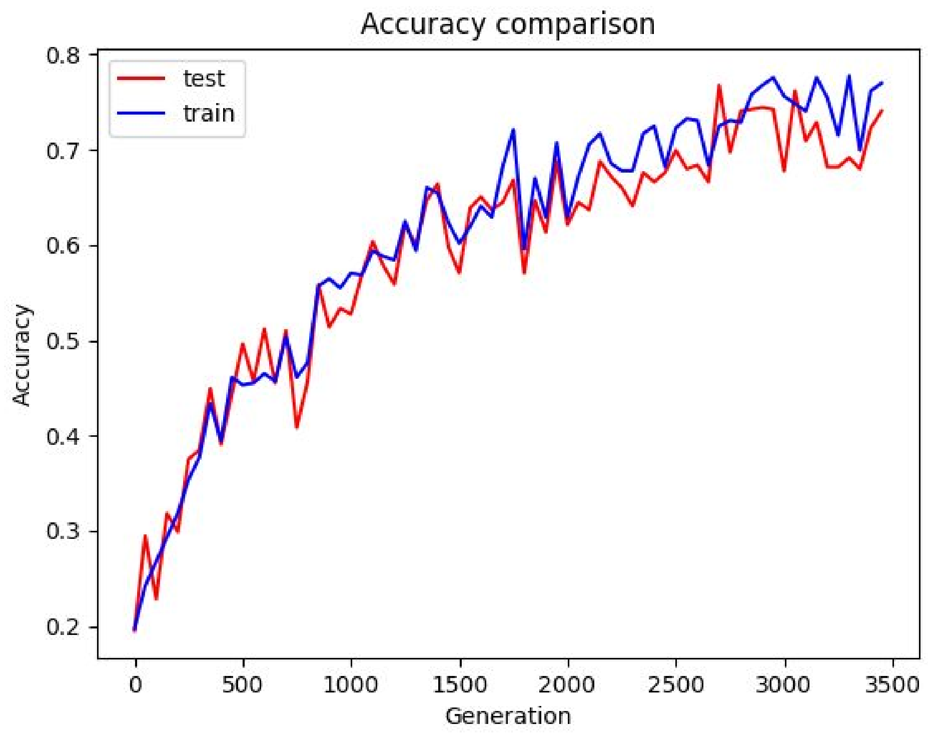
Intuition for selecting the kernel size is that lower layers will have lower kernel size maintaining the associations between nearer pixels and useful for finding low level features like lines,edges. And kernel size for higher level layers increases to detect higher level features like head, legs etc. [based on the talk with Prof. Mozer]

2.a) According the models I tried, Variant 4 turns out to produce a better average accuracy for both train as well as test cases. It reaches a better accuracy in a comparatively a lesser epochs.

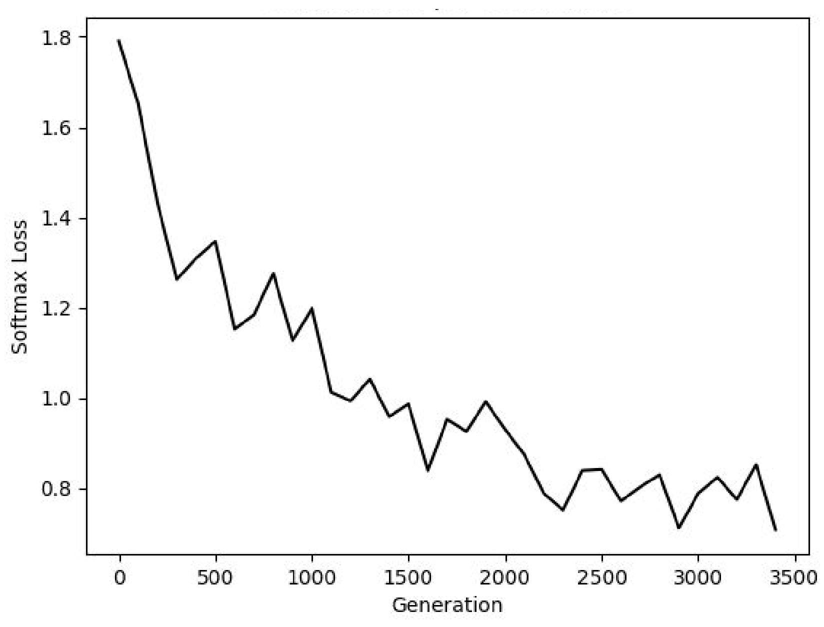
Drawing the error graph :



Accuracy graph is the inversion of error:



Loss function used - Softmax cross entropy (For the training examples in one of the folds)


Softmax Loss
Generation