Machine Learning Assignment 4

Ankur Sharma
2015CS50278

Fixed Algorithms:

(A) **Classification using K-means** :

(a) K-means implemented for 20 clusters with n_init set to 10. All the points have been loaded as float32 np array with the values normalised in the preprocessing step. For each cluster identified, the majority label of the points assigned to that cluster has been chosen as the label for that centroid. In each run of the algorithm, different centroid points have been chosen in the initialisation step (default random state), and the algorithm has been made to run couple of times(10 times), and the majority voting of the different runs is used to assign the final label to each test point.

(b) Accuracies:

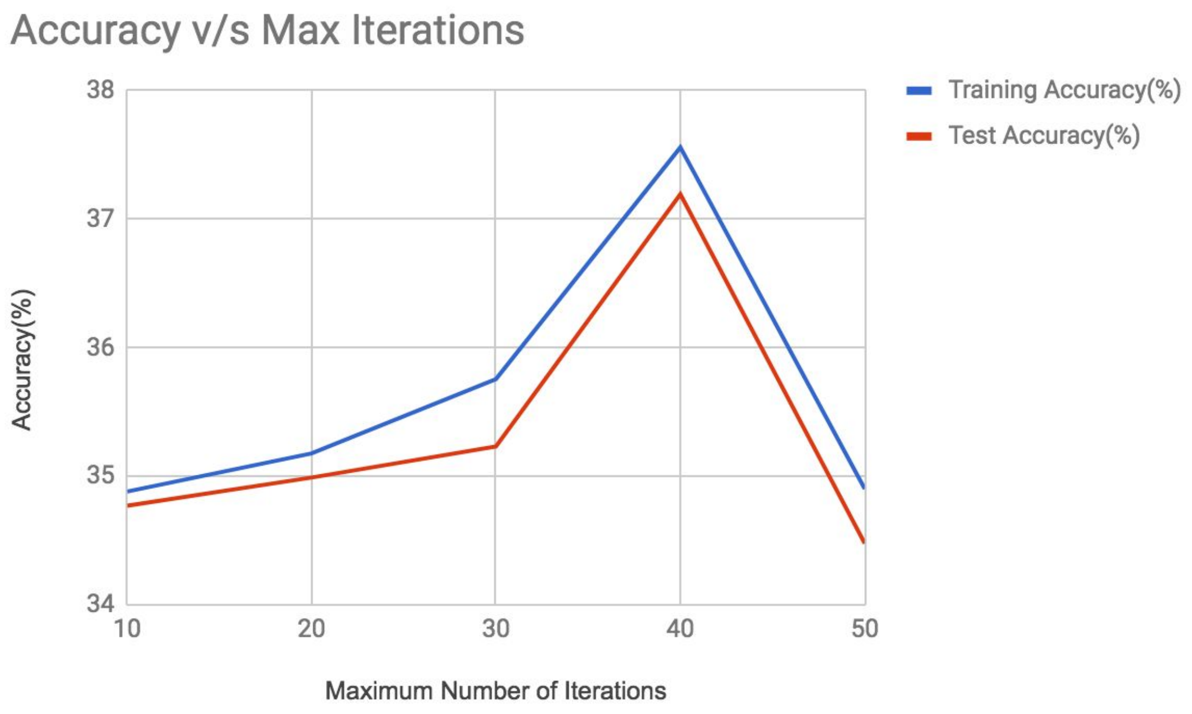
(i) Training Set Accuracy : 34.548%

(ii) Test Set Accuracy : 34.222%

(iii) Observations :

(c) Table showing the variations in accuracies with the bound on max_iter:

max_iter	**Training Accuracy(%)**	**Test Accuracy(%)**
10	34.881	34.77
20	35.178	34.99
30	35.756	35.232
40	37.559	37.195
50	34.901	34.477

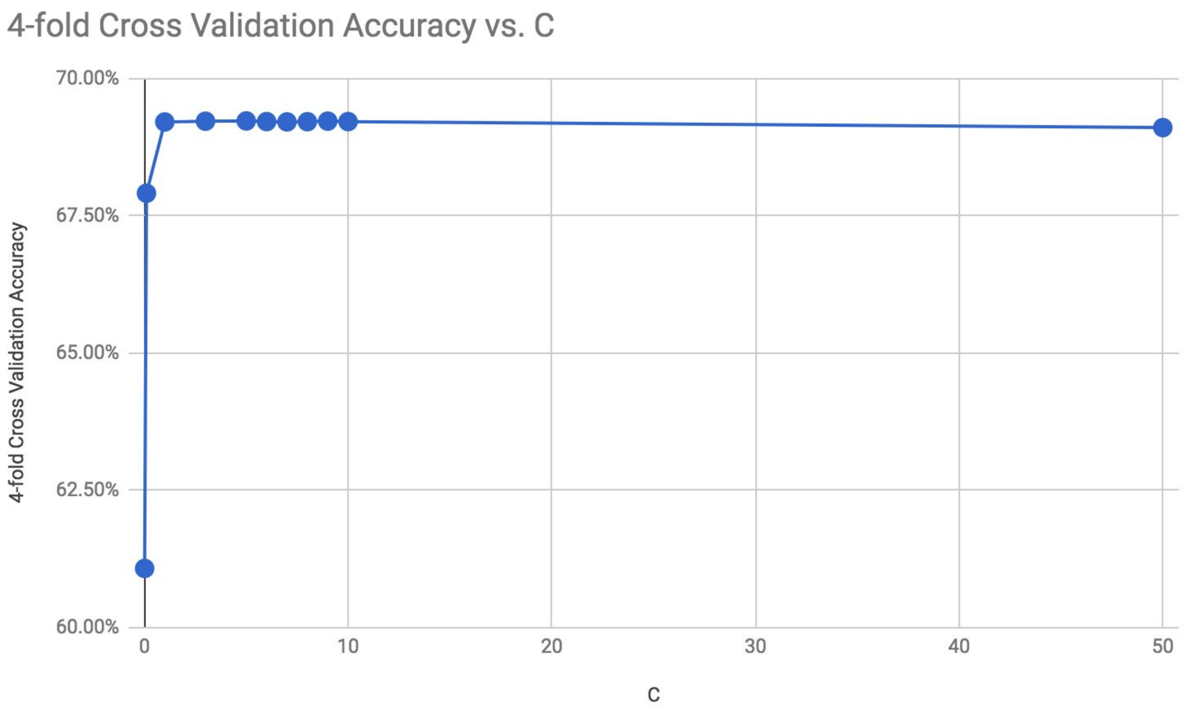


Observations : As the number of iteration increases, the training as well as the test accuracy increases upto a certain point (~40 iters), and after that it starts decreasing. This means that K Means converges in much less number of iterations, and after this it only slightly decreases till maximum number of iterations (i.e. 300 iters). It can also be observed that there is no 1-1 mapping between the clusters and the labels since the data is not separable into 20 different clusters in linear space. The accuracies compared are pretty low since the data points are presumably from a higher dimensional space, and by applying k-means we also lose spatial information about the correlation between the image pixels, which substantially lowers down the performance of this classifier.

(B) **PCA + SVM**

In this part, I have used PCA to lower the number of dimensions down to 50. And then used LIBSVM for multiclass classification. To determine the best value of the hyperparameter C, I have done a 4-fold cross validation on the data set. After trying a number of values, the best cross validation accuracy is achieved for C = 5.

C	**4-fold Cross Validation Accuracy**
0.01	61.073%
0.1	67.916%
1	69.216%
3	69.23%
5	**69.235%**
6	69.223%
7	69.217%
8	69.221%
9	69.231%
10	69.223%
50	69.113%



<u>Model : Linear SVM Classifier with C = 5</u>

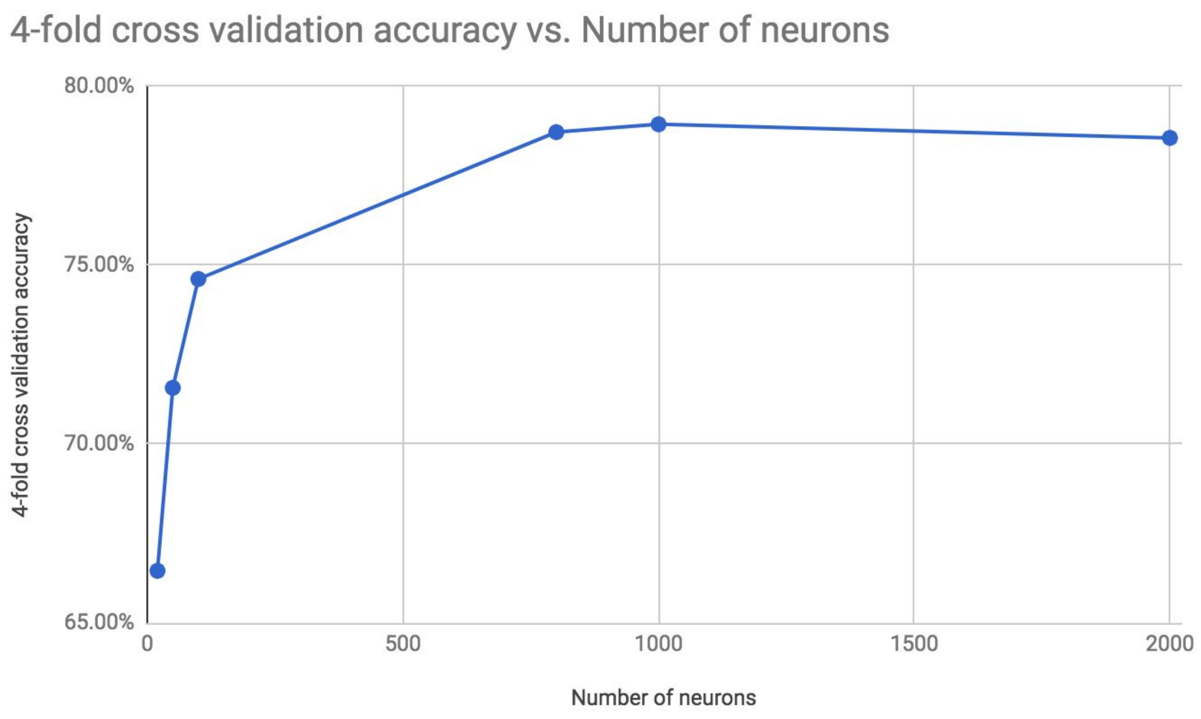
Training Set Accuracy : 65.675%

Testing Set Accuracy : 61.185%

Observations : Using the concept of PCA, we were able to reduce the number of dimensions from 784 to 50. These 50 dimensions selected from PCA reduction captures most of the variance in the data, and help us to compress the data into features that can be easily modeled. The above graph has been computed with 4-fold cross validation performed on the training set for different values of C. As we can see, the best cross-validation accuracy has been achieved for C ~ 5. For very low values of C, the validation accuracy is very poor, and not very useful for prediction. Values above 5 are nearly same, and the value of the validation accuracy decreases at a very slow rate further as we increase the value. As we increase the value of C, we allow slackness for these points, and at the same time penalising them according to the value of C, which clearly shows that there is a tradeoff.

(C) **Fully Connected Single Layer Neural Network**

Number of neurons in the hidden layer	4-fold cross validation accuracy
20	66.46% (0.41%)
50	71.57% (0.30%)
100	74.61% (0.18%)
800	78.71% (0.31%)
1000	**78.93% (0.26%)**
2000	78.55% (0.58%)



So, the optimal number of neurons in the hidden layer is around 1000, as achieved by 4-fold cross validation.

<u>NN with a single hidden layer of 1000 neurons</u>
Training Set Accuracy : 92.50%
Testing Set Accuracy : 81.27%

Observations: As we increase the number of neurons in this fully connected architecture, we observe that the validation accuracy increase upto a certain point (~1000) neurons after which it plateaus, and increasing the number of neurons only overfits the training set. 4-fold cross validation was performed using KFold module present in sklearn.model_selection, and we also observed that the validation accuracies were quite comparable to the testing set accuracies which shows that cross validation is a good measure to check the performance of the model. The training and validations splits were selected by randomly shuffling the data set with a split of 3 : 1.

(D) **Convolutional Neural Network (CNN)**

FInding the best set of parameters: Using the **GridsearchCV** Module of **scikit-learn**, all the possible set of combinations of all the different parameters were tried and the model was then evaluated on the best set of params. GridSearch performs an internal cross validation for a fixed number of epochs (15 epochs in this case) for every possible combination of parameters. For each possible parameter combination, we have a separate cross validation accuracy.

Parameter Grid:
num_of_kernels = [16, **32**, 128, 512]
kernel_size = [2, 3, **5**]
hidden_size = [100, **500**, 1000]

Best parameters:
Number of kernels = 32
Kernel size = 5 x 5
Number of hidden units = 500

CNN Architecture (32, 5, 500)
Training Set Accuracy : 93.70%
Testing Set Accuracy : 86.27%

Observations :

1. Small size of the kernel (like 2 x 2) will capture every minute details in the image, and won't be able to capture big raw strokes in the image. And very large kernel size like 7 x 7 won't be able to capture finer details in the image. Hence, it makes sense to conclude that the optimal kernel size came out to be 3 x 3 by internal cross validation.
2. Generally, it has been observed that smaller kernel sizes with increasing number of kernels in successive layers works better. Hence, it makes sense to observe that in a simple architecture like this, we can achieve good results with a smaller number of kernels. Using too small a number of kernels might not be able to capture all the patterns

in the original data, hence we can only determine the optimal number of kernels by such experimentations.

3. Increasing the number of hidden neurons increases the validation accuracy initially, but as we increase the hidden number of neurons, the network overfits the training set and accuracy starts falling. Hence, 500 seems like an apt choice for the number of neurons in the hidden layer.

Overall Comparison

Model (Classifier)	Training Accuracy(%)	Testing Accuracy(%)
K-Means	34.548	34.222
PCA + SVM	65.675	61.185
Single Layer Fully Connected NN	92.50	81.27
CNN	93.70	86.27

Performance : CNN > Single Layer NN > PCA + SVM > K-Means

K-Means: As discussed earlier, this classifier wasn't able to capture the spatial correlations among the features in the linear space, hence it wasn't able to classify the images well.

PCA-SVM: A better approach than K-Means but it reduces the number of dimensions from 784 to 50. This possibly will cause a loss in the spatial information and correlations between the pixels, which could be very relevant while learning image components. Also, we are using a linear kernel which may not be able to capture the nonlinear trends between the data points.

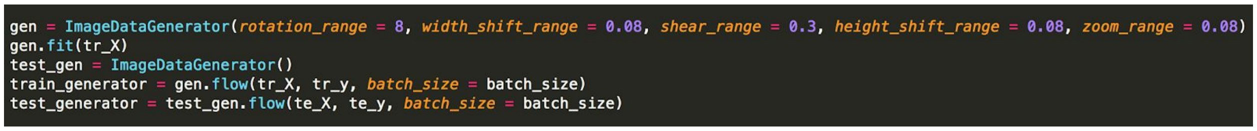
Neural Network : It gives a much better accuracy due to increased number of parameters, and is more adaptive. Hence, it makes sense to conclude that the accuracy of improved thereby.

Convolutional Neural Networks : CNN performs the best when it comes to an image classification task because the way it works is that CNN will learn to recognize components of an image (e.g., lines, curves, etc.) by convolution operations and then learn to combine these components to recognize larger structures (e.g., faces, objects, etc.) by pooling and adding dense layers. This means they learn more easily, and they learn the things we want - things that generalize well from the images it's trained on to unseen images.

COMPETITION PART

Key Points in all the models:

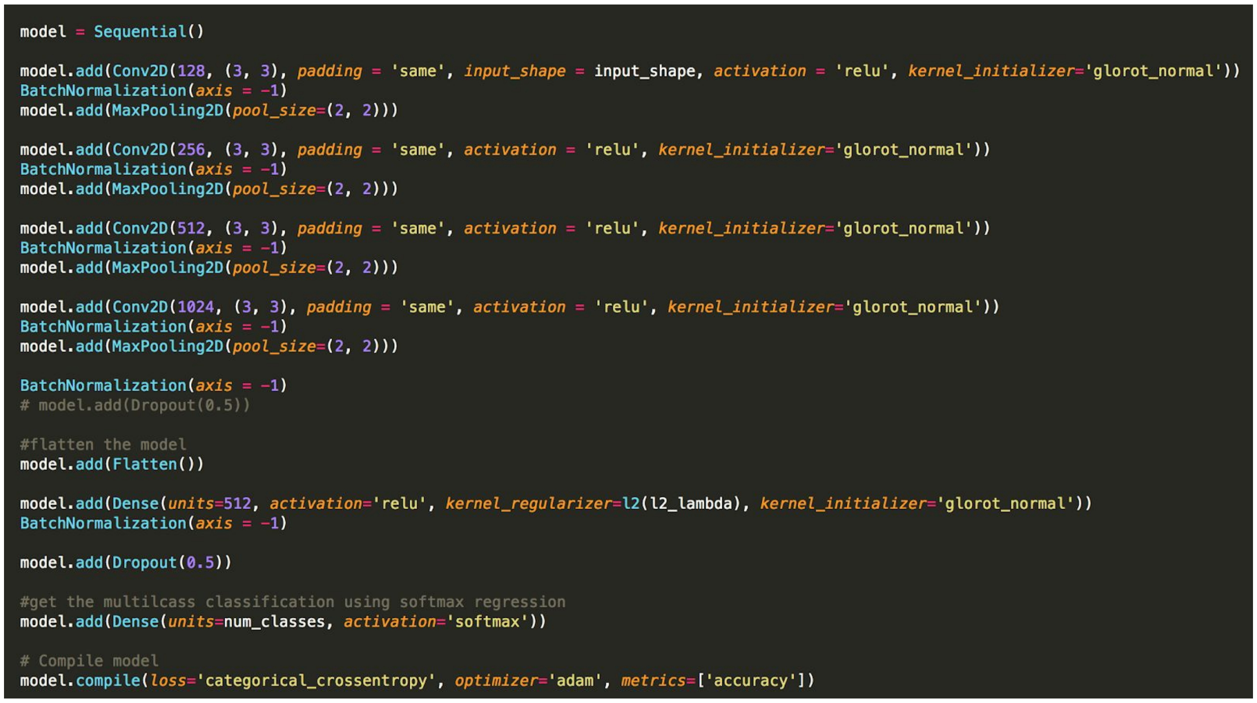
1. **Batch Normalisation :** This has been used to improve the accuracy of the model and for faster convergence by reducing the output of the layer to zero mean and unit variance, thus keeping the gradients in check.
2. **L2 Regularisation (with Dropouts)**: Dropouts constraints network adaptation to the data at training time, to avoid it becoming "too smart" in learning the input data; it thus helps to avoid overfitting. At each training step in a mini-batch, the dropout procedure creates a different network (by randomly removing some units), which is trained using backpropagation as usual.
   A regulariser has also been used which is used to regularise all the weights so as to decrease the complexity of the model by adding a L2 regularisation term in the loss function so as to penalise the weights appropriately. Here, lambda is used to specify the relative importance of minimising the norm to minimising the loss on the training set.
3. **Network Initialisation** : Initialisation of the kernel weights is also an important factor, and here we have used *glorot_normal* initialisation.
4. **Data Augmentation** : Augmented features were added to the data points by creating an image generator to cover rotation, height shift, width shift and other image operations. This has been seen to improve the accuracy by upto 1% on the model 1, 2 and 4. (In model 3, augmented features were in fact decreasing the accuracy, so I omitted them from that part)



```
gen = ImageDataGenerator(rotation_range = 8, width_shift_range = 0.08, shear_range = 0.3, height_shift_range = 0.08, zoom_range = 0.08)
gen.fit(tr_X)
test_gen = ImageDataGenerator()
train_generator = gen.flow(tr_X, tr_y, batch_size = batch_size)
test_generator = test_gen.flow(te_X, te_y, batch_size = batch_size)
```

5. **Ensembles:** Since each classifier has been trained separately, it has learned different 'aspects' of the data and their mistakes are different. Combining them helps to produce an stronger classifier, which is less prone to overfitting. Hence, to get the best results, majority prediction of these 4 classifiers has been taken which gives an accuracy rise of upto 1%.
6. **Early Stopping:** This has been used for tuning the number of epochs by declaring convergence when the accuracy on the validation set doesn't increase for a fixed number of epochs (also called patience = 5).
7. **Activation Function:** ReLU and Leaky ReLU have been tried, and it is seen that they converge much faster than the original sigmoid activation function, and also gives much simpler models than sigmoid.

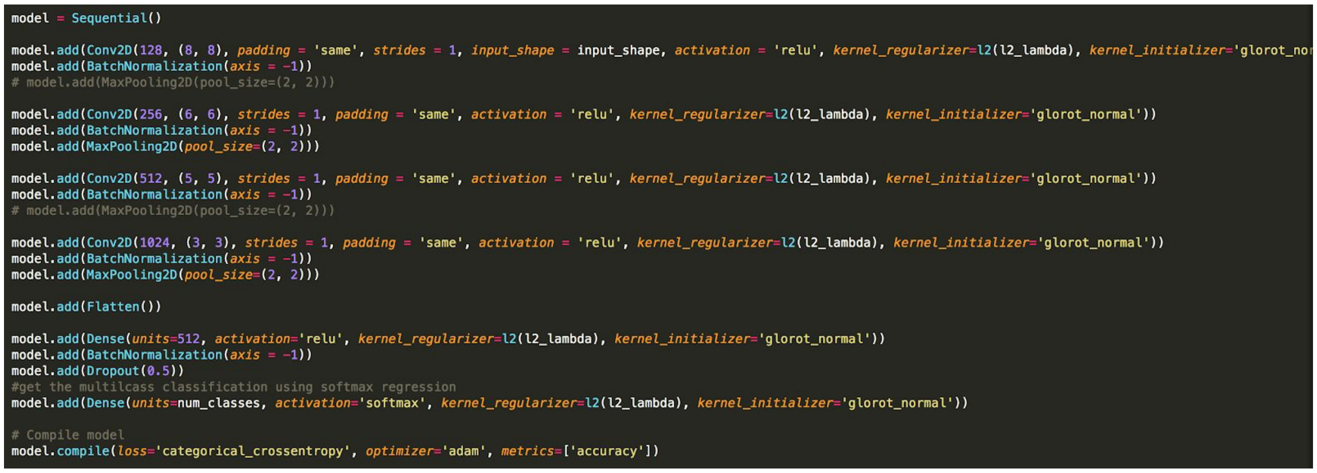
Model 1



```
model = Sequential()

model.add(Conv2D(128, (3, 3), padding = 'same', input_shape = input_shape, activation = 'relu', kernel_initializer='glorot_normal'))
BatchNormalization(axis = -1)
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3), padding = 'same', activation = 'relu', kernel_initializer='glorot_normal'))
BatchNormalization(axis = -1)
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(512, (3, 3), padding = 'same', activation = 'relu', kernel_initializer='glorot_normal'))
BatchNormalization(axis = -1)
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(1024, (3, 3), padding = 'same', activation = 'relu', kernel_initializer='glorot_normal'))
BatchNormalization(axis = -1)
model.add(MaxPooling2D(pool_size=(2, 2)))

BatchNormalization(axis = -1)
# model.add(Dropout(0.5))

#flatten the model
model.add(Flatten())

model.add(Dense(units=512, activation='relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))
BatchNormalization(axis = -1)

model.add(Dropout(0.5))

#get the multilcass classification using softmax regression
model.add(Dense(units=num_classes, activation='softmax'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In this model, the each convolution layer is followed by a maxpool layer. The number of kernels have been increased keeping the size of kernels as the same. Hidden dense layer has been added after flattening, and dropouts have been added to avoid overfitting.

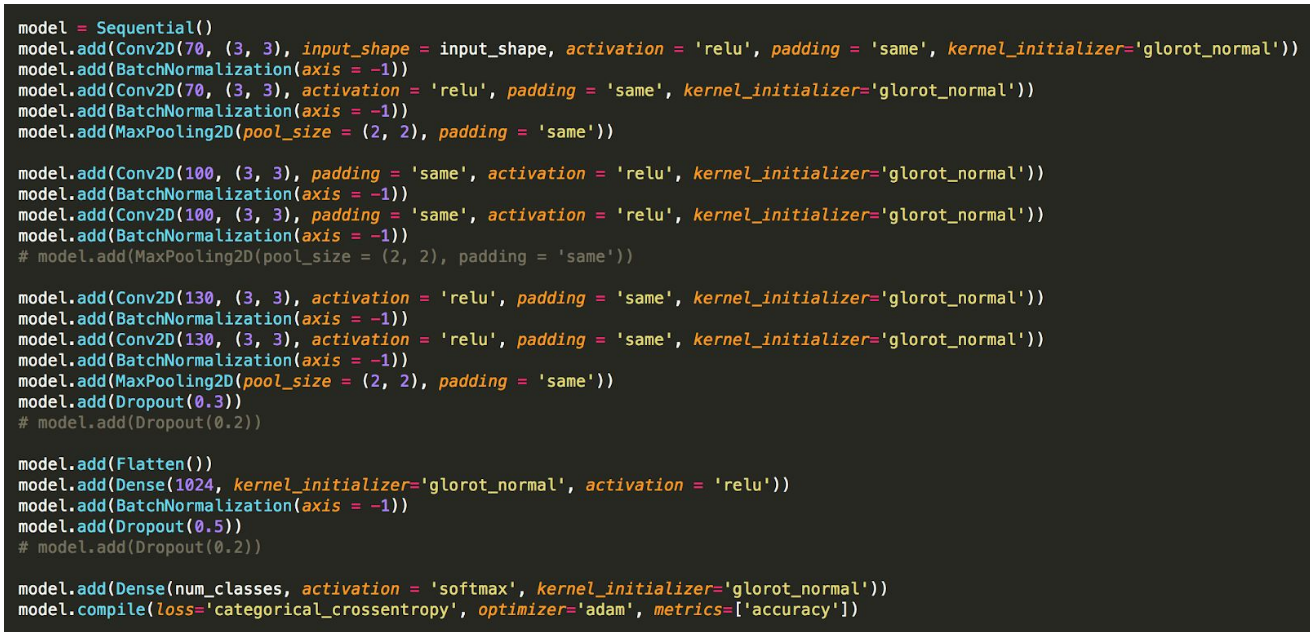
Model 2



```
model = Sequential()

model.add(Conv2D(128, (8, 8), padding = 'same', strides = 1, input_shape = input_shape, activation = 'relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_nor
model.add(BatchNormalization(axis = -1))
# model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (6, 6), strides = 1, padding = 'same', activation = 'relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(512, (5, 5), strides = 1, padding = 'same', activation = 'relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))
# model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(1024, (3, 3), strides = 1, padding = 'same', activation = 'relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())

model.add(Dense(units=512, activation='relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))
model.add(Dropout(0.5))
#get the multilcass classification using softmax regression
model.add(Dense(units=num_classes, activation='softmax', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

This is the same as the previous model, just that the size of kernels isn’t the same, and decreased with the number of layers.

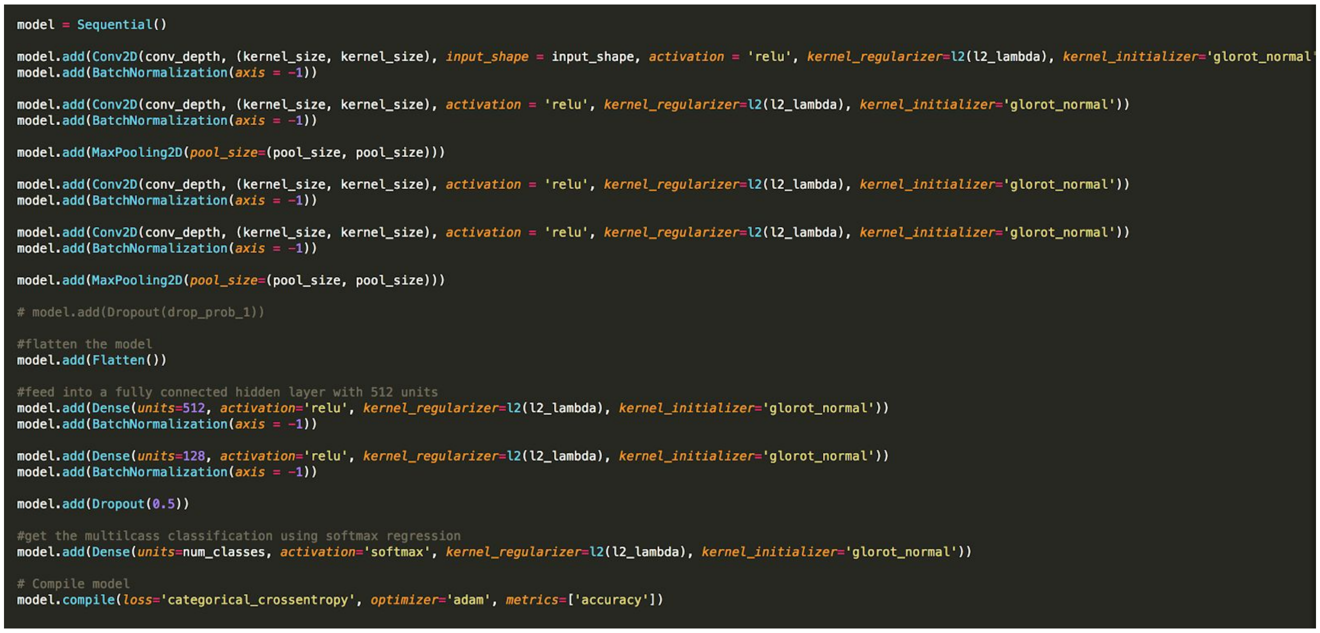
Model 3



```
model = Sequential()
model.add(Conv2D(70, (3, 3), input_shape = input_shape, activation = 'relu', padding = 'same', kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))
model.add(Conv2D(70, (3, 3), activation = 'relu', padding = 'same', kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))
model.add(MaxPooling2D(pool_size = (2, 2), padding = 'same'))

model.add(Conv2D(100, (3, 3), padding = 'same', activation = 'relu', kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))
model.add(Conv2D(100, (3, 3), padding = 'same', activation = 'relu', kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))
# model.add(MaxPooling2D(pool_size = (2, 2), padding = 'same'))

model.add(Conv2D(130, (3, 3), activation = 'relu', padding = 'same', kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))
model.add(Conv2D(130, (3, 3), activation = 'relu', padding = 'same', kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))
model.add(MaxPooling2D(pool_size = (2, 2), padding = 'same'))
model.add(Dropout(0.3))
# model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(1024, kernel_initializer='glorot_normal', activation = 'relu'))
model.add(BatchNormalization(axis = -1))
model.add(Dropout(0.5))
# model.add(Dropout(0.2))

model.add(Dense(num_classes, activation = 'softmax', kernel_initializer='glorot_normal'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In this architecture, there were 6 convolutional layers, and one dense layer with Batch Normalisation after each step. Here, the number of kernels increases with the increase in the number of layers, with the same kernel size. 2 versions of this model with similar results were kept while ensembling.

Model 4

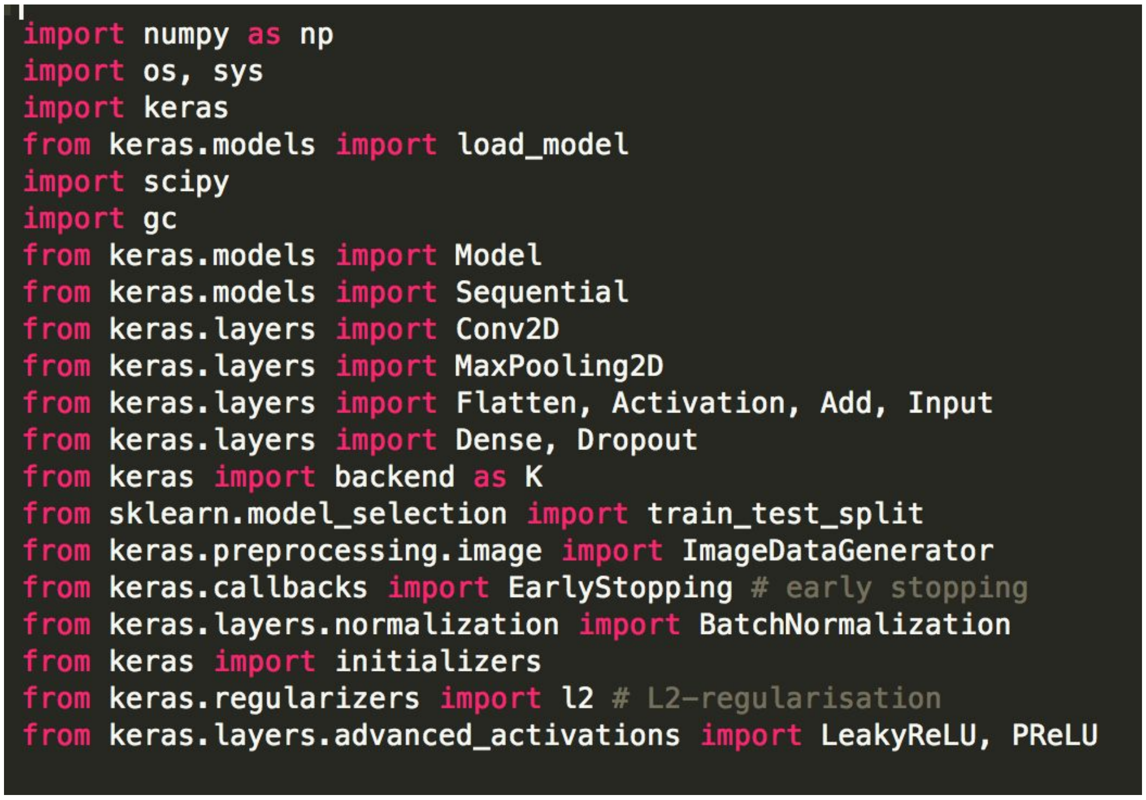


```
model = Sequential()

model.add(Conv2D(conv_depth, (kernel_size, kernel_size), input_shape = input_shape, activation = 'relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'
model.add(BatchNormalization(axis = -1))

model.add(Conv2D(conv_depth, (kernel_size, kernel_size), activation = 'relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))

model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

model.add(Conv2D(conv_depth, (kernel_size, kernel_size), activation = 'relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))

model.add(Conv2D(conv_depth, (kernel_size, kernel_size), activation = 'relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))

model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

# model.add(Dropout(drop_prob_1))

#flatten the model
model.add(Flatten())

#feed into a fully connected hidden layer with 512 units
model.add(Dense(units=512, activation='relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))

model.add(Dense(units=128, activation='relu', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))
model.add(BatchNormalization(axis = -1))

model.add(Dropout(0.5))

#get the multilcass classification using softmax regression
model.add(Dense(units=num_classes, activation='softmax', kernel_regularizer=l2(l2_lambda), kernel_initializer='glorot_normal'))

# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In this architecture, we have 2 sets of (conv, conv maxpool) layers followed by 2 maxpool layers with appropriate batch regularization, and dropouts so as to ensure better results.

Final prediction: Predictions of all these models were ensembled and the majority prediction was then taken to get the final label. This gives an accuracy of **93.910%** on the private leaderboard, and **94.097%** on the public leaderboard.

Libraries : The given picture shows all the libraries that have been used in Keras with Tensorflow backend. ImageDataGenerator was used to add augmented data features onto the input images. From sklearn.model_selection, I imported train_test_split, so as to get a validation test on which the validation accuracies can be evaluated and submitted with confidence.



```
import numpy as np
import os, sys
import keras
from keras.models import load_model
import scipy
import gc
from keras.models import Model
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten, Activation, Add, Input
from keras.layers import Dense, Dropout
from keras import backend as K
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping # early stopping
from keras.layers.normalization import BatchNormalization
from keras import initializers
from keras.regularizers import l2 # L2-regularisation
from keras.layers.advanced_activations import LeakyReLU, PReLU
```

Final Models: All of the above 4 models can be accessed via a drive link mentioned here https://drive.google.com/open?id=16lVGc97E3iNatj2DGrhBppsV6zvugqCU
my_best_model_1.h5 -> Model 1
my_best_model_2.h5 -> Model 2
my_best_model_3.h5 -> Model 3
my_best_model_4.h5 -> Model 4

To use the model simply uncomment the line *model = load_model (<model_name.h5>)*, and make the predictions. The code to ensemble the predictions has been submitted on moodle.