1. Structure of the code.

largeMarginSoftmaxLoss :
Function compute the large margin softmax loss for the CNN. We have global weight vectors corresponding to each class and the Xi components. Find the angle thetha between the weight vector and Xi. Then using the following formula compute the loss Li.

$$L_i = -\log\left(\frac{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\cos(\theta_{y_i})}}{\sum_j e^{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_j)}}\right)$$

CNN layer structure:-

```
model.add(ZeroPadding2D((1,1),input_shape=x_train.shape[1:]))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

We have used the pretrained model. We have applied two 64 filters of 3*3 and two 128 filters of 3*3. For the fully connected neural layers, We have used the two Dense layers of 4096, and the final one of 10(i.e no of classes). We have used 1*1 padding and 2*2 pooling for the CNN.

Stochastic gradient descent

We have used stochastic gradient as an optimizer for our model simulation.
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)

DATASETS used
We have trained our model on MNIST and Ciphar10 datasets.

2. Instructions for setting up and running the code.

Use python2 to run the code.
Install keras and all its dependencies.

3. list of methods

We have written the following LARGE MARGIN SOFTMAX loss function.

```
def largeMarginSoftmaxLoss(y_true, y_pred):
        wt=wt.reshape(10,4096);
        totalLoss=0;
        predictedClass = y_pred
        normw=normCalculation(w[predictedClass])
        normx=normCalculation(x)
        angleXW=angle(x,w[predictedClass])
        angleXW = m*angleXW
        xwi=normx*normw*math.cos(angleXW)
        xwi=math.exp(xwi)
        totalSum=0.0
        for each in w:
        normw=normCalculation(w[each])
        angleXW=angle(x,w[each])*m
        xwt=normx*normw*math.cos(angleXW)
        totalSum=totalSum + math.exp(xwt)
        totalLoss = totalLoss - math.log(xwi/totalSum)

        return totalLoss
```

4. RESULTS

For CIPHAR10

Value of m	Error rate
1	9.10
2	7.75
3	7.67
4	7.60