CSE574: Introduction to Machine Learning 50351114

Project Report 2 Image Classification

mlara@buffalo.edu

Introduction

The task of this project is to classify an image into one of ten classes using CIFAR-10 dataset. There are two approaches to the image classification task that is being implemented.

Dataset: The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. ”Automobile” includes sedans, SUVs, things of that sort. ”Truck” includes only big trucks. Neither includes pickup trucks.

CIFAR-10 Examples - Random images for each of the 10 class

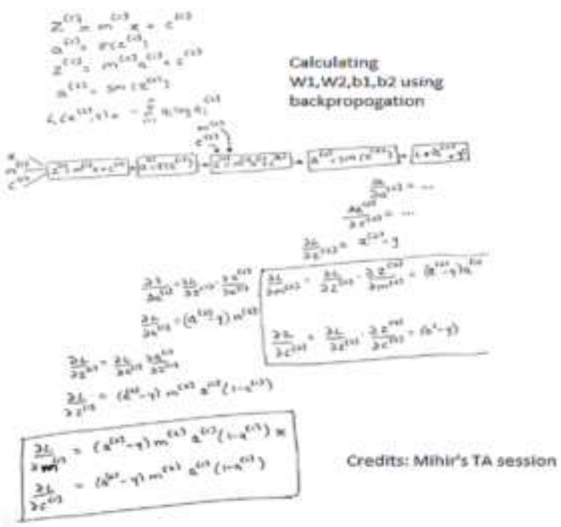
The two approaches that are used to the image classification tasks are:

1. Supervised Learning Approach (SLA): Build a Neural Network Classifier (NN) with one hidden layer to be trained and tested on CIFAR-10 dataset.
2. Unsupervised Learning Approach (USLA): Extract image features using a Convolutional Auto Encoder (Conv-AE) for CIFAR10 dataset and then Classify Auto-Encoded image features using K-Means clustering algorithm

Approach 1: Supervised Learning Approach (SLA)

Experimental Setup:

1. Import all the libraries required and load the data using keras.datasets
2. Scaling Image Pixel values from 0 to 1 by dividing the training and testing dataset by 255, since the maximum pixel value of RGB is 255 which represent white colour.
3. One hot encoding of target variable to represent categorical variables as binary vectors using OneHotEncoder object from sklearn.preprocessing.
4. Then initializing all the hyper parameters and variables necessary for implementing gradient descent algorithm to train a neural network with 1 hidden layer.
5. The genesis equation used is ˆy = Softmax(W2.Sigmoid(W1X +b1)+b2) where W1 & W2 are the weight arrays, X is the input features and ˆy is the SoftMax (SM) class probability.
   The softmax activation function is a function that turns a vector of K real values into a vector of K real values that sum to 1. If one of the inputs is small or negative, the softmax turns it into a small probability, and if an input is large, then it turns it into a large probability, but it will always remain between 0 and 1. And the inner activation function used is sigmoid for 1 hidden layer.
6. Then after predicting the target variables, cost is being calculated using categorical cross entropy loss i.e $H(p) = -\sum (p(x)\log p(x))$
7. After loss is calculated, W1, b1, W2, b2 is updated using the technique of **backpropogation**.
   $W1 = W1 - \eta * \Delta W1$, $b1 = b1 - \eta * \Delta b1$, $W2 = W2 - \eta * \Delta W2$, $b2 = b2 - \eta * \Delta b2$ where η is learning rate.



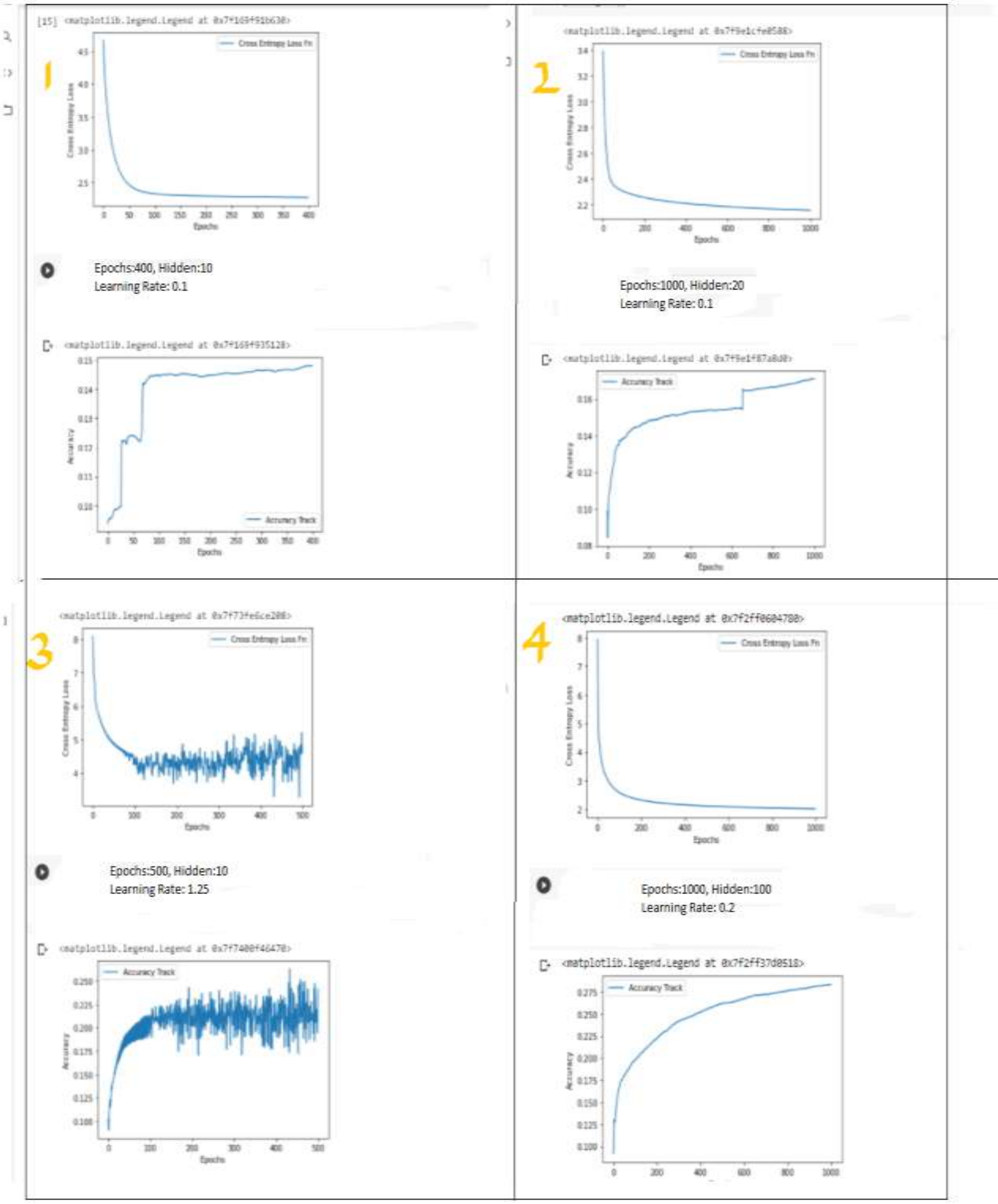
8. Finally, accuracy is measured using mean of np.argmax(ypred,axis=1) == np.argmax(ytrain,axis=1).

Results:

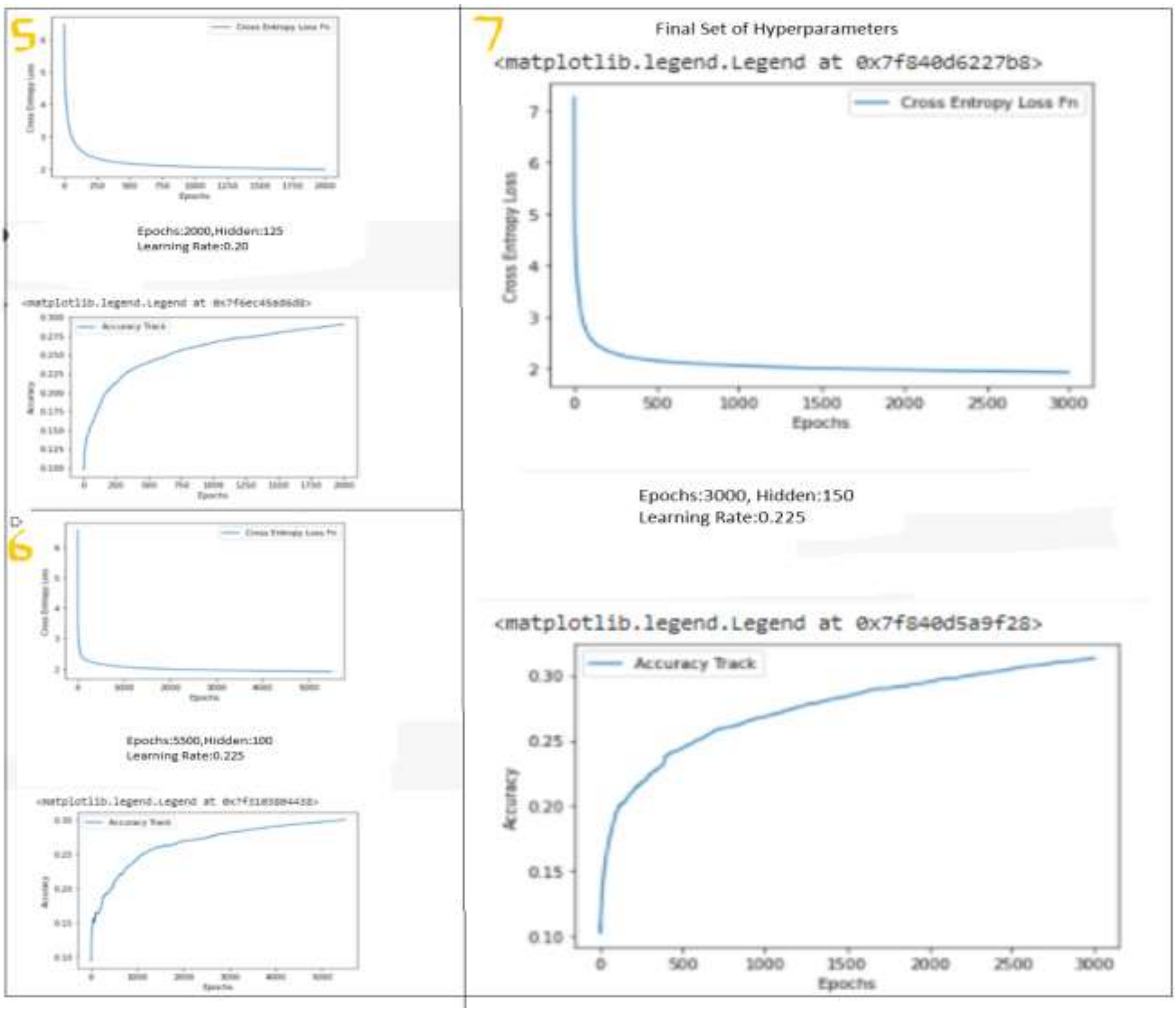
The accuracy of training and testing data with 1 hidden layer NN is close to **31.01%** and **29.90%** respectively. After careful observation, my final set of hyperparameters are epochs:**3000**, learning rate:**0.225** and number of hidden neurons:**150** (fig7 below)

Comparision of Results:

Below are the plots of loss and accuracy for multiple set of hyperparameters used.



Observations: In Figure3, As the learning rate/number of hidden neurons increases, the graph of loss and accuracy gets **scattered** in nature.
The running time for the model is directly proportional to the number of epochs, hidden neurons and learning rate.
Average running time for my final set of hyperparameters is close to **4 hours**. And the accuracy gets stalled at around 32% even after increasing the epochs to 6000 which ran for around 6 hours**.**
If the learning rate is less than 0.225, the accuracy is bit less and close to **28.5%** which can be seen below in the figure and if the learning rate/No. of neurons is high, the graph gets scattered randomly.
Considering the number of hidden neurons, initially 10 hidden neurons were taken and the accuracy was close to 15% and gradually increasing the number of hidden neurons to 50 then 100 and finally 150 increased my accuracy which later on stalled even after increasing the hidden neurons.



So, these are the comparisions of the results obtained and observations noted during running the neural network classifier with 1 hidden layer to be trained.

Confusion Matrix: Also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).

Confusion matrix for test dataset using approach 1 SLA is :

```
print("Confusion Matrix b/w Ytest and Ypred test\n",matrix)
```

```
[[418  44  65  15  48  15  29  61 234  71]
 [ 78 257  27  35  31  45  65  46 160 256]
 [142  35 174  41 209  79 166  60  54  40]
 [ 62  61  82 104 115 178 153 117  51  77]
 [ 66  31 115  26 322  72 196  93  44  35]
 [ 46  47 107  73 116 271 144 105  60  31]
 [ 23  37  82  49 171  86 393  86  24  49]
 [ 76  49 104  53 129  71 110 245  57 106]
 [136  67  51  24  15  46  14  15 524 108]
 [ 52 140  19  23  18  21  66  63 170 428]]
```

Approach 2: Unsupervised Learning Approach (USLA)

Experimental Setup and Comparison of Results:

1. Importing all the required libraries, downloading data from keras.datasets and scaling each image by dividing them with 255.
2. Using keras library, different convolution auto encoder's are built which are summarised as below and the model from fig1. is considered since batch normalization is used to make the model faster and stable by re-scaling and reducing overfitting. Conv2d_transpose layer is used to upsample with learned weights/kernel and is a convolution operation.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 32, 32, 16)        448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 8)         1160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 8)           0
_________________________________________________________________
Encode_Layer (Conv2D)        (None, 8, 8, 3)           219
_________________________________________________________________
Flatten_Layer (Flatten)      (None, 192)               0
_________________________________________________________________
Dense_Layer (Dense)          (None, 64)                12352
_________________________________________________________________
reshape (Reshape)            (None, 8, 8, 1)           0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 16, 16, 3)         30
_________________________________________________________________
batch_normalization (BatchNo (None, 16, 16, 3)         12
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 32, 32, 8)         224
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 8)         32
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 3)         219
=================================================================
Total params: 14,696
Trainable params: 14,674
Non-trainable params: 22
```

1

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 32, 32, 16)        448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 8)         1160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 8)           0
_________________________________________________________________
Encode_Layer (Conv2D)        (None, 8, 8, 3)           219
_________________________________________________________________
Flatten_Layer (Flatten)      (None, 192)               0
_________________________________________________________________
Dense_Layer (Dense)          (None, 64)                12352
_________________________________________________________________
reshape (Reshape)            (None, 8, 8, 1)           0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 8)           80
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 16)        1168
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 3)         435
=================================================================
Total params: 15,862
Trainable params: 15,862
Non-trainable params: 0
_________________________________________________________________
```

2

Different Models Used for Conv. Auto Encoder

3. Next the model is compiled using "SGD" (Stochastic Gradient Descent) optimizer with learning rate of **0.225** and loss used is "binary cross entropy".
4. The model is fitted using parameters x_train as input and output, with batch size of 64 and 10 epochs. The accuracy using convolution auto encoder model is close to **50%** and loss is around **0.69**. 'Relu' activation function is used in convolution layers and 'softmax' activation function is used in dense layer of the model.

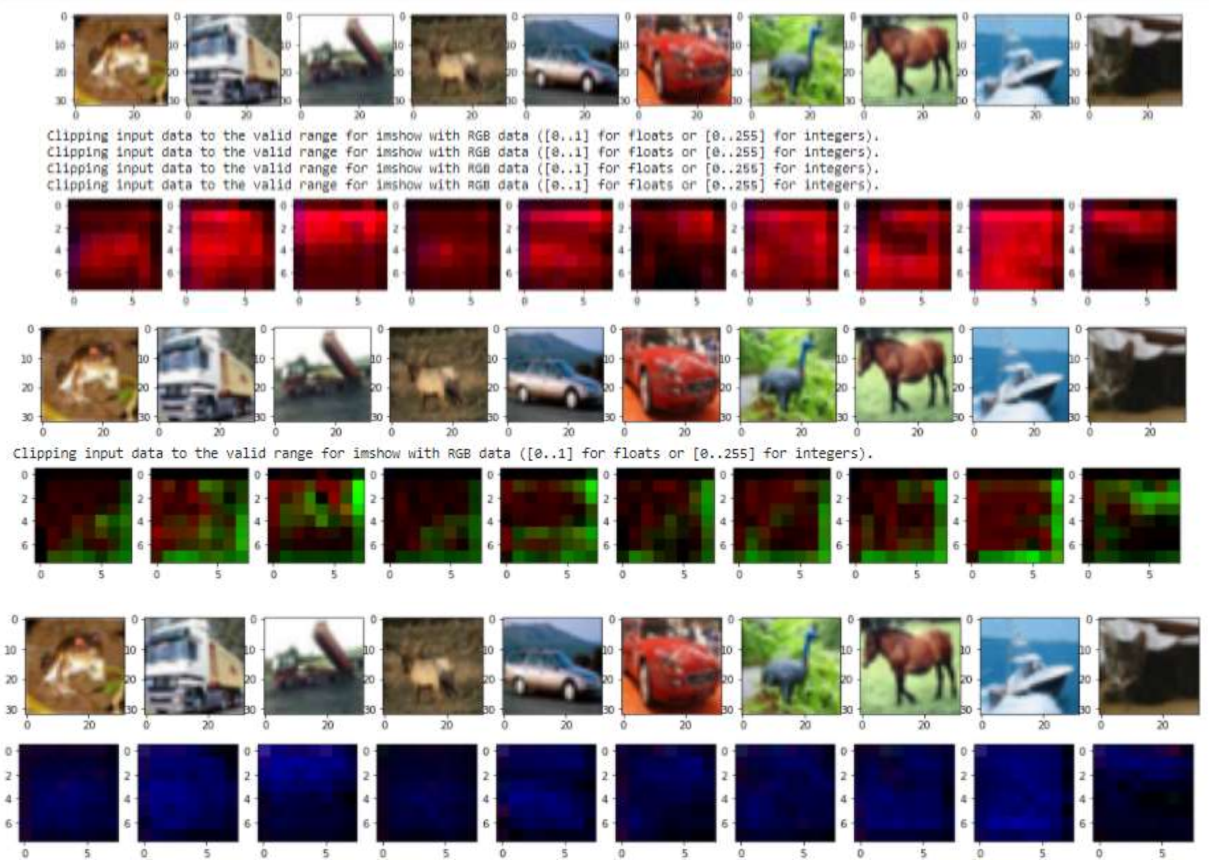
```
Epoch 1/10
782/782 [==============================] - 90s 115ms/step - loss: 0.6910 - accuracy: 0.5011
Epoch 2/10
782/782 [==============================] - 89s 113ms/step - loss: 0.6908 - accuracy: 0.4983
Epoch 3/10
782/782 [==============================] - 92s 118ms/step - loss: 0.6908 - accuracy: 0.4982
Epoch 4/10
782/782 [==============================] - 89s 114ms/step - loss: 0.6908 - accuracy: 0.4980
Epoch 5/10
782/782 [==============================] - 88s 113ms/step - loss: 0.6908 - accuracy: 0.4959
Epoch 6/10
782/782 [==============================] - 89s 114ms/step - loss: 0.6908 - accuracy: 0.4936
Epoch 7/10
782/782 [==============================] - 90s 115ms/step - loss: 0.6908 - accuracy: 0.4926
Epoch 8/10
782/782 [==============================] - 90s 116ms/step - loss: 0.6907 - accuracy: 0.4924
Epoch 9/10
782/782 [==============================] - 91s 116ms/step - loss: 0.6907 - accuracy: 0.4924
Epoch 10/10
782/782 [==============================] - 91s 117ms/step - loss: 0.6907 - accuracy: 0.4924
```

```
optim = tf.keras.optimizers.SGD(learning_rate=0.1, name='SGD')                        #defining c
model.compile(optimizer=optim, loss='binary_crossentropy',metrics=['accuracy'])         #Compilir
model.fit(x_train, x_train, batch_size=64, epochs =10)                                  #Fitting
```

```
Epoch 1/10
782/782 [==============================] - 95s 121ms/step - loss: 0.6910 - accuracy: 0.5089
Epoch 2/10
782/782 [==============================] - 93s 119ms/step - loss: 0.6909 - accuracy: 0.5065
Epoch 3/10
782/782 [==============================] - 92s 118ms/step - loss: 0.6909 - accuracy: 0.5033
Epoch 4/10
782/782 [==============================] - 92s 117ms/step - loss: 0.6909 - accuracy: 0.5033
Epoch 5/10
782/782 [==============================] - 93s 118ms/step - loss: 0.6909 - accuracy: 0.5031
Epoch 6/10
782/782 [==============================] - 91s 116ms/step - loss: 0.6908 - accuracy: 0.5022
Epoch 7/10
782/782 [==============================] - 96s 122ms/step - loss: 0.6908 - accuracy: 0.4997
Epoch 8/10
782/782 [==============================] - 91s 117ms/step - loss: 0.6908 - accuracy: 0.4976
Epoch 9/10
782/782 [==============================] - 91s 117ms/step - loss: 0.6908 - accuracy: 0.4971
Epoch 10/10
782/782 [==============================] - 90s 115ms/step - loss: 0.6908 - accuracy: 0.4971
<tensorflow.python.keras.callbacks.History at 0x7fb679823f60>
```

It is observed from the above figures that the accuracy and loss remains same for various learning rates such as 0.1, 0.225 and there is no major change in accuracy as well as loss due to learning rate change. Additionally the increase in number of epochs from 10 to 20-50 doesn't change the accuracy and loss of the model.

Below is the comparison b/w 3 sets of input images and encoded images from Convolution Auto encoder. It is observed that the encoded images are different for each set of iteration since the colours generated are completely different.



5. Kmeans clustering model is fitted with the encoded images from the latent layer and then cluster ids for auto-encoded image features is fetched using kmeans.labels_
6. Confusion matrix is created b/w output Y and labels, and initial accuracy is calculated by adding maximum values from each column and dividing it by the total sum of the elements of the matrix.
7. The cluster ids are obtained through np.argmax of each column of the confusion matrix formed and thus getting maximum value of each column as a list.

```
Confusion Matrix b/w Ytrain and label
 [[ 561  817  300  132  170  704  358  372 1144  442]
 [1009  403  605  201  440  528  486  326  388  614]
 [ 280  592 1000  645  516  139  085  239  347  248]
 [ 367  548  634  525  830  131  788  655  285  237]
 [ 251  299 1019  810  953  164  774  340  124  266]
 [ 651  525  640  331  808  138  760  749  220  178]
 [ 270  415 1119 1034  778   30  837  230  164  123]
 [ 498  556  629  269  570  287  816  465  151  759]
 [ 856  474  278   65  217 1323  290  423  312  762]
 [ 485  656  417   67  160  749  547  138  319 1462]]
 Confusion Matrix b/w YTest and labelTest
 [[124 188  53  16  31 140  71  68 245  64]
 [216 104 111  52  76 106  93  54  76 112]
 [ 56 126 209 119 101  32 190  51  78  38]
 [ 91 109 135  87 176  23 156 111  58  54]
 [ 57  55 211 179 179  19 173  49  21  57]
 [134 102 112  56 148  35 154 166  49  44]
 [ 65  75 212 207 146   8 187  37  40  23]
 [ 90 110 130  36 106  70 148  94  34 182]
 [179  78  34  11  50 287  64  85  52 160]
 [ 89 128  81  12  34 163 118  29  71 275]]
Column wise max values for both the matrices
 [1 0 6 6 4 8 2 5 0 9]
 [1 0 6 6 4 8 2 5 0 9]
```

8. Then the cluster ids are assigned to true labels using a for loop iterating over all the images.
9. Accuracy is then calculated after assigning cluster ids to true labels using sklearns.accuracy_score.
10. There are multiple functions created to handle special case where multiple cluster ids have same maximum value of Y (most probable). In such cases, the duplicates are removed and replaced by the numbers that were not assigned as cluster ids following ascending order from 1 to 10.
11. Again the accuracy is calculated after removing duplicate cluser ids and then assigning back remaining cluster ids to true labels.
12. Finally the **maximum** value from (step9, step11) is taken as **final accuracy**.

Results: The final accuracy of training and testing data is **21.19**% and **21.65**% respectively. It is observed that the accuracy gets decreased after replacing duplicate cluster ids having same Y (max probable) as the probability of picking the wrong cluster id increases more and its noted that the accuracy after removing duplicate cluster ids is around 13.91% and 18.34% respectively.

The confusion matrix for testing dataset using approach2 is:

```
Confusion Matrix for Testing Dataset is:
 [[433 124  71   0  31  68  69   0 140  64]
 [180 216  93   0  76  54 163   0 106 112]
 [204  56 190   0 101  51 328   0  32  38]
 [167  91 156   0 176 111 222   0  23  54]
 [ 76  57 173   0 179  49 390   0  19  57]
 [151 134 154   0 148 166 168   0  35  44]
 [115  65 187   0 146  37 419   0   8  23]
 [144  90 148   0 106  94 166   0  70 182]
 [130 179  64   0  50  85  45   0 287 160]
 [199  89 118   0  34  29  93   0 163 275]]
```

Hence, these are the two approaches that are used for image classification task of cifar-10 dataset and its found that the accuracy in approch1 SLA (NN with 1 hidden layer) is better than accuracy in approach2 USLA (Kmeans clustering of Convolution Auto Encoder).

--THANK YOU--