Multimedia Retrieval Basic

Project Report

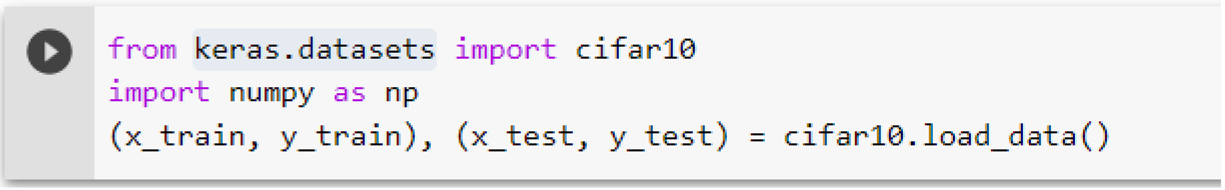
Ali Asghar Marvi

Piyush Talele

Nethra Srinivasan

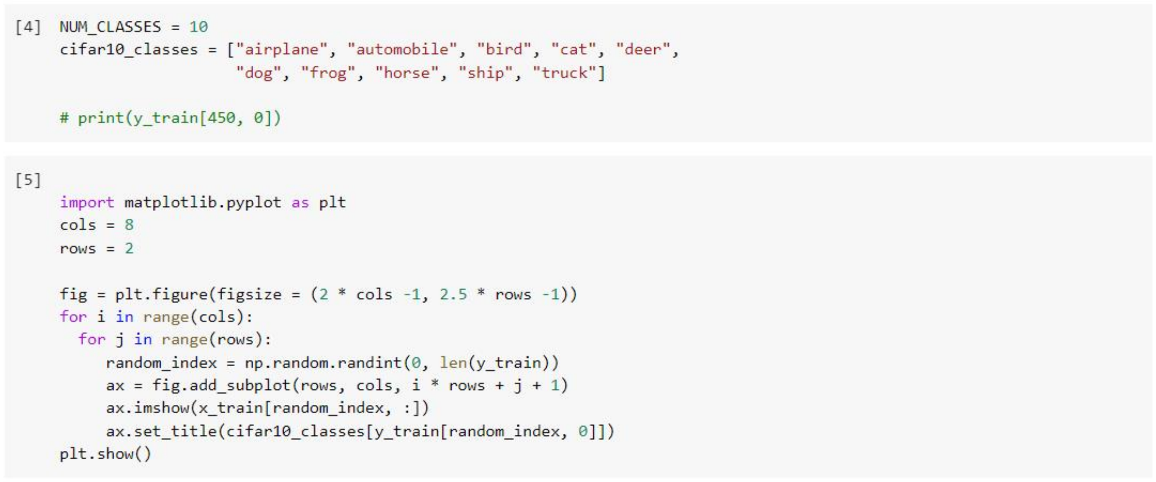
Feature Encoding

For this project we first loaded dataset in our notebook using keras.dataset. This API call does the splitting of test and train set for us.



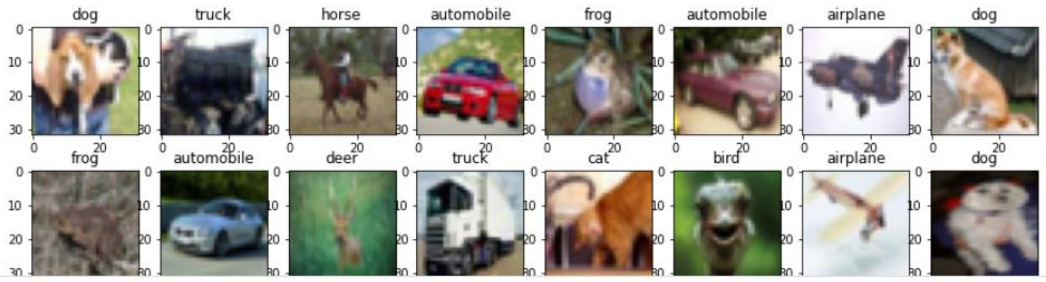
```
from keras.datasets import cifar10
import numpy as np
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

This dataset contains of 60000 rows of images for 10 specific classes. These classes were named in order for better visualization along the way.

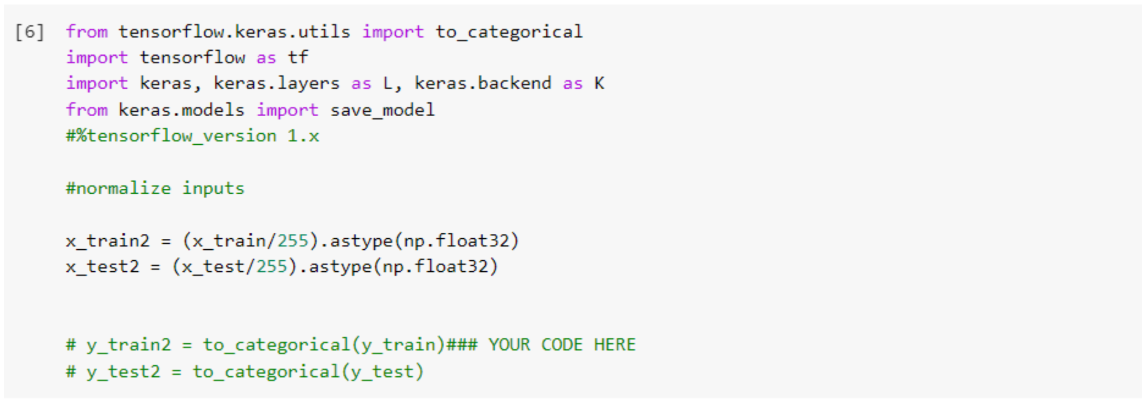


```
[4] NUM_CLASSES = 10
    cifar10_classes = ["airplane", "automobile", "bird", "cat", "deer",
                       "dog", "frog", "horse", "ship", "truck"]

    # print(y_train[450, 0])
```

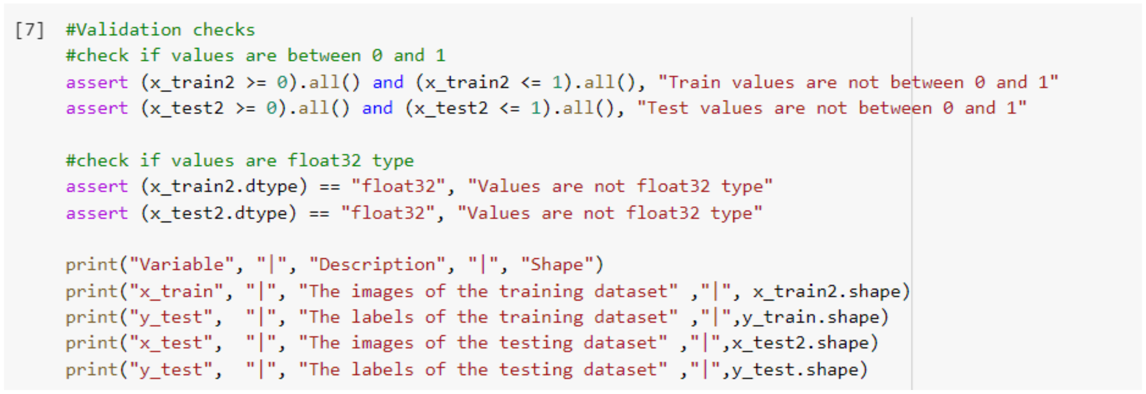
```
[5]
    import matplotlib.pyplot as plt
    cols = 8
    rows = 2

    fig = plt.figure(figsize = (2 * cols -1, 2.5 * rows -1))
    for i in range(cols):
      for j in range(rows):
         random_index = np.random.randint(0, len(y_train))
         ax = fig.add_subplot(rows, cols, i * rows + j + 1)
         ax.imshow(x_train[random_index, :])
         ax.set_title(cifar10_classes[y_train[random_index, 0]])
    plt.show()
```



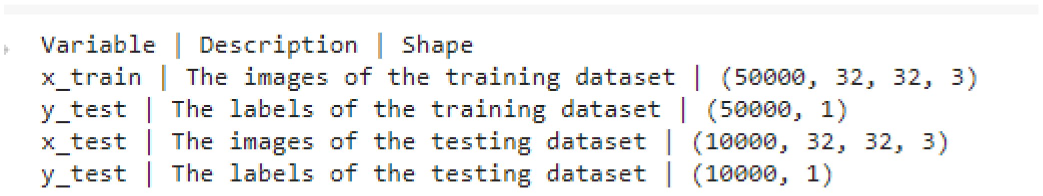
Next we did some sanity checks as instructed in the assignment document. We normalized our dataset and did several assertions to get desired output.



```
[6] from tensorflow.keras.utils import to_categorical
    import tensorflow as tf
    import keras, keras.layers as L, keras.backend as K
    from keras.models import save_model
    #%tensorflow_version 1.x

    #normalize inputs

    x_train2 = (x_train/255).astype(np.float32)
    x_test2 = (x_test/255).astype(np.float32)


    # y_train2 = to_categorical(y_train)### YOUR CODE HERE
    # y_test2 = to_categorical(y_test)
```



```
[7] #Validation checks
    #check if values are between 0 and 1
    assert (x_train2 >= 0).all() and (x_train2 <= 1).all(), "Train values are not between 0 and 1"
    assert (x_test2 >= 0).all() and (x_test2 <= 1).all(), "Test values are not between 0 and 1"

    #check if values are float32 type
    assert (x_train2.dtype) == "float32", "Values are not float32 type"
    assert (x_test2.dtype) == "float32", "Values are not float32 type"

    print("Variable", "|", "Description", "|", "Shape")
    print("x_train", "|", "The images of the training dataset" ,"|", x_train2.shape)
    print("y_test",  "|", "The labels of the training dataset" ,"|",y_train.shape)
    print("x_test",  "|", "The images of the testing dataset" ,"|",x_test2.shape)
    print("y_test",  "|", "The labels of the testing dataset" ,"|",y_test.shape)
```



```
Variable | Description | Shape
x_train | The images of the training dataset | (50000, 32, 32, 3)
y_test | The labels of the training dataset | (50000, 1)
x_test | The images of the testing dataset | (10000, 32, 32, 3)
y_test | The labels of the testing dataset | (10000, 1)
```

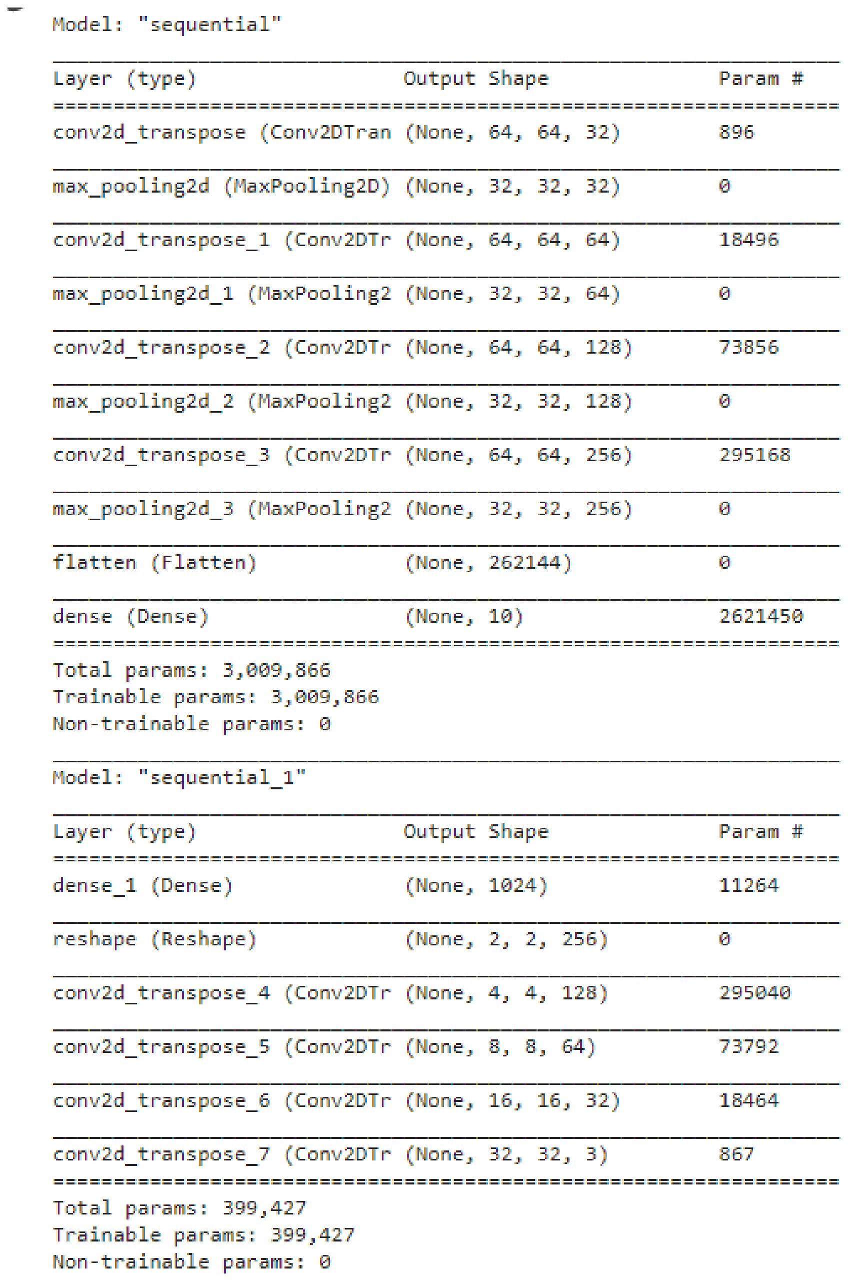
Model building and Training

Next we designed our Autoencoder model using Keras. Initially we started with two layers in encoding and decoding architecture and later we expanded it to 4 layers excluding the flattening, input and dense layers. We used Conv2DTranspose layer for encoding followed by Max Pool layer until flattening out in Latent space. This approach ensured that both height and weight of images stays same along with adding in more filters. It certainly increased our set of trainable parameters but it helped learning features in entirety.

For decoder architecture, we simply used Conv2DTranspose layers to upscale dimension of image until we get 32 by 32 by 3, which is our reproduced output. Since the final output is just an image hence no activations are needed at final output.

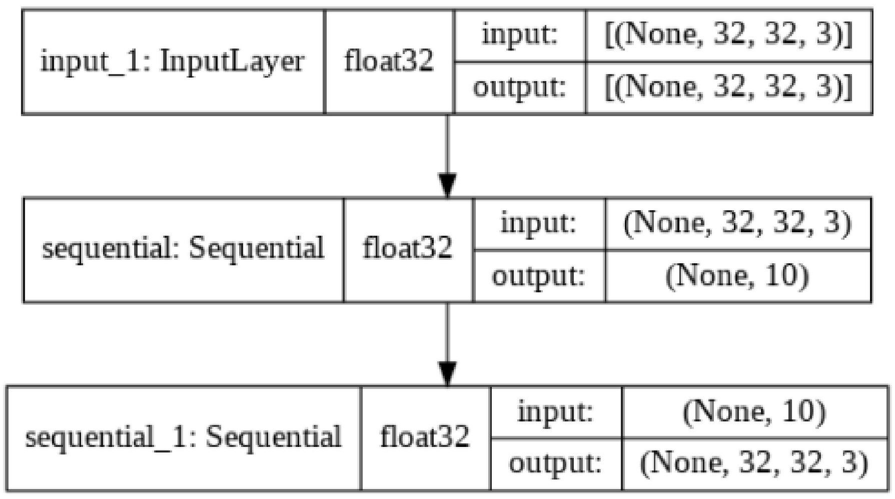
```
def build_deep_autoencoder(img_shape, latent_size):
    H,W,C = img_shape

    # encoder
    encoder = keras.models.Sequential()
    encoder.add(L.InputLayer(img_shape, name="Input Layer"))

    ### YOUR CODE HERE: define encoder as per instructions above ###
    encoder.add(L.Conv2DTranspose(filters=32, kernel_size=3, strides=2, activation='elu', padding='same'))
    encoder.add(L.MaxPooling2D(pool_size = 2))
    encoder.add(L.Conv2DTranspose(filters=64, kernel_size=3, strides=2, activation='elu', padding='same'))
    encoder.add(L.MaxPooling2D(pool_size = 2))
    encoder.add(L.Conv2DTranspose(filters=128, kernel_size=3, strides=2, activation='elu', padding='same'))
    encoder.add(L.MaxPooling2D(pool_size = 2))
    encoder.add(L.Conv2DTranspose(filters=256, kernel_size=3, strides=2, activation='elu', padding='same'))
    encoder.add(L.MaxPooling2D(pool_size = 2))
    encoder.add(L.Flatten())
    encoder.add(L.Dense(latent_size))

    # decoder
    decoder = keras.models.Sequential()
    decoder.add(L.InputLayer((latent_size), name="Latent Layer"))

    ### YOUR CODE HERE: define decoder as per instructions above ###
    decoder.add(L.Dense(2*2*256))
    decoder.add(L.Reshape((2,2,256)))
    decoder.add(L.Conv2DTranspose(filters=128, kernel_size=3, strides=2, activation='elu', padding='same'))
    decoder.add(L.Conv2DTranspose(filters=64, kernel_size=3, strides=2, activation='elu', padding='same'))
    decoder.add(L.Conv2DTranspose(filters=32, kernel_size=3, strides=2, activation='elu', padding='same'))
    decoder.add(L.Conv2DTranspose(filters=3, kernel_size=3, strides=2, activation=None, padding='same'))
    return encoder, decoder
```

We also tried basic Conv2D initially but results were more satisfactory using Conv2DTranspose approach. We tried to use model_to_dot but keras repeatedly gave an assertion error due to which the model_to_dot output was not available. Online forums were of no help either, however as a remedy we will be adding in model summary to better explain sequence of two models:



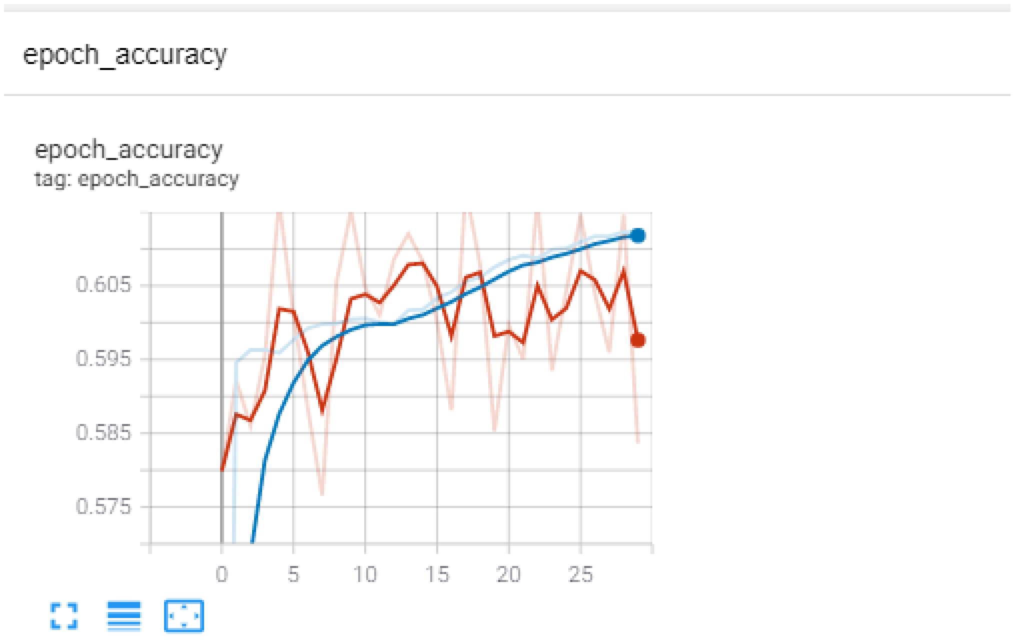
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_transpose (Conv2DTran (None, 64, 64, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 64, 64, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 32, 32, 64)        0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 64, 64, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 128)       0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 64, 64, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 32, 32, 256)       0
_________________________________________________________________
flatten (Flatten)            (None, 262144)            0
_________________________________________________________________
dense (Dense)                (None, 10)                2621450
=================================================================
Total params: 3,009,866
Trainable params: 3,009,866
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1024)              11264
_________________________________________________________________
reshape (Reshape)            (None, 2, 2, 256)         0
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 4, 4, 128)         295040
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 8, 8, 64)          73792
_________________________________________________________________
conv2d_transpose_6 (Conv2DTr (None, 16, 16, 32)        18464
_________________________________________________________________
conv2d_transpose_7 (Conv2DTr (None, 32, 32, 3)         867
=================================================================
Total params: 399,427
Trainable params: 399,427
Non-trainable params: 0
```

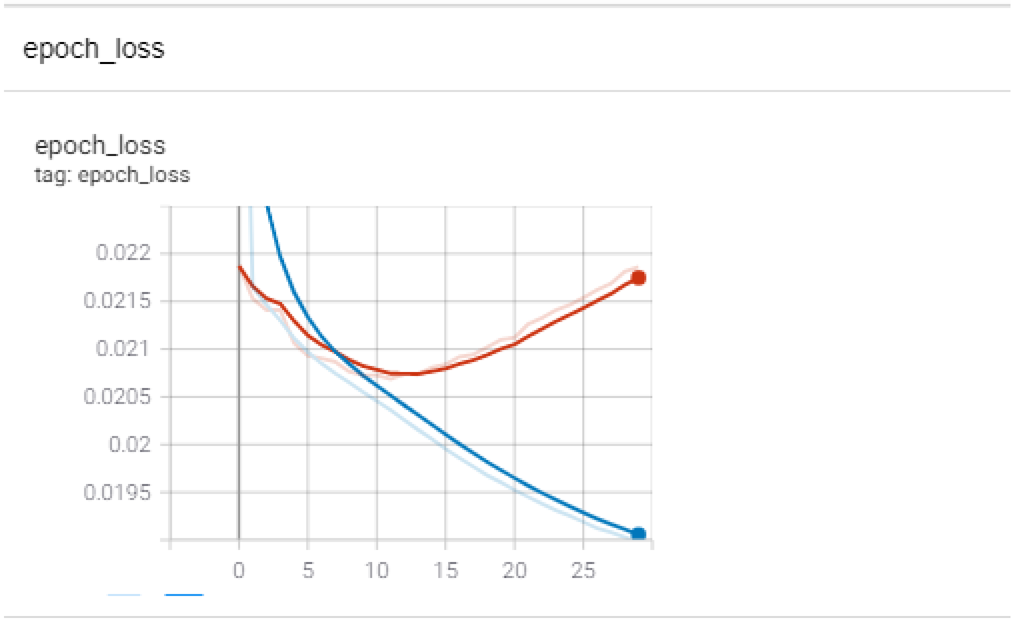
As you can see we have approximately 3 million trainable parameters in encoder and 400, 000 parameters in decoder model. In nutshell our Autoencoder model (this worked fine):



To compile and train the model, we used adamax optimizer along with “mse” as error metric. Model was trained for 30 epochs, where accuracy remained stable at 60%. Custom Callback layers were used to monitor model progress on each epoch. To plot our training output we used Tensorboard extension to plot graph.

**Red line shows validation metric and blue line shows training metric





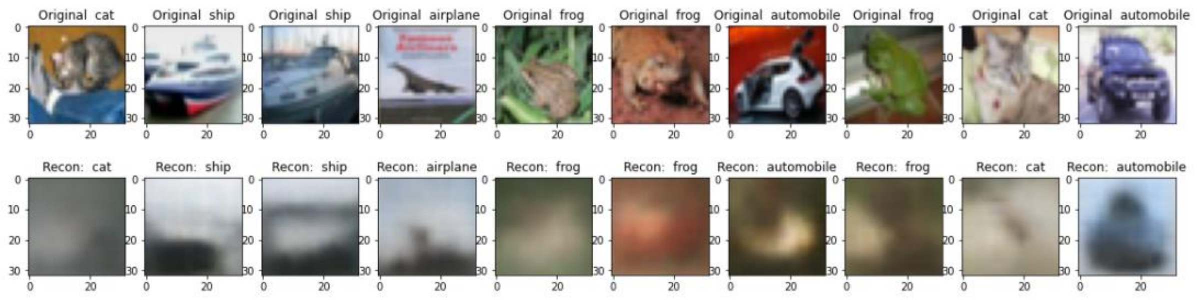
Training process was very smooth, infact accuracy curve flattened after 6th epoch and stayed roughly around 60% accuracy.

Sanity Check

As part of our sanity check we first visualized our results with both original and reconstructed images side by side. As it can be seen that certain output images depict the same reconstructed shades of original image, however these are more blurry than usual. Blurriness can be improved using much higher latent space for visualization.

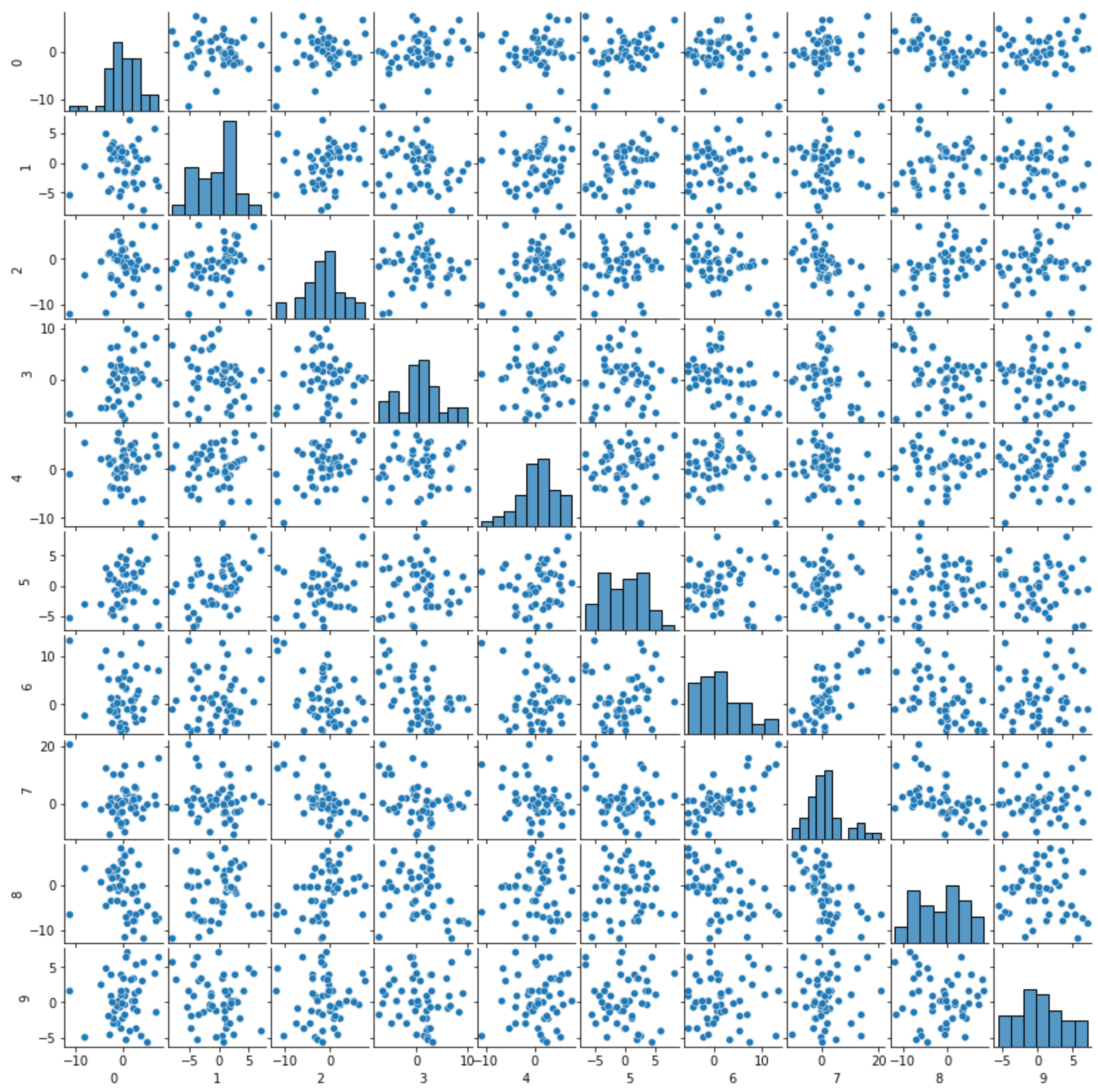
```
def plot_images(original_list):
  plt.figure(figsize=(20,5))
  for i in range(len(original_list)):
    code = encoder.predict(original_list[i][None])[0]
    reco = decoder.predict(code[None])[0]
    ax = plt.subplot(2,10, i+1)
    plt.title("Original" +"  " + cifar10_classes[y_test[i,0]])
    plt.imshow(np.clip(original_list[i], 0,1))

    ax = plt.subplot(2,10, i+1+10)
    plt.title("Recon:" +"  " + cifar10_classes[y_test[i,0]])
    plt.imshow(np.clip(reco, 0,1))
```

```
# score = autoencoder.evaluate(x_test2,x_test2,verbose=0)
# print("PCA MSE:", score)

original_list = []
for i in range(10):
    img = x_test2[i]
    original_list.append(img)

plot_images(original_list)
```

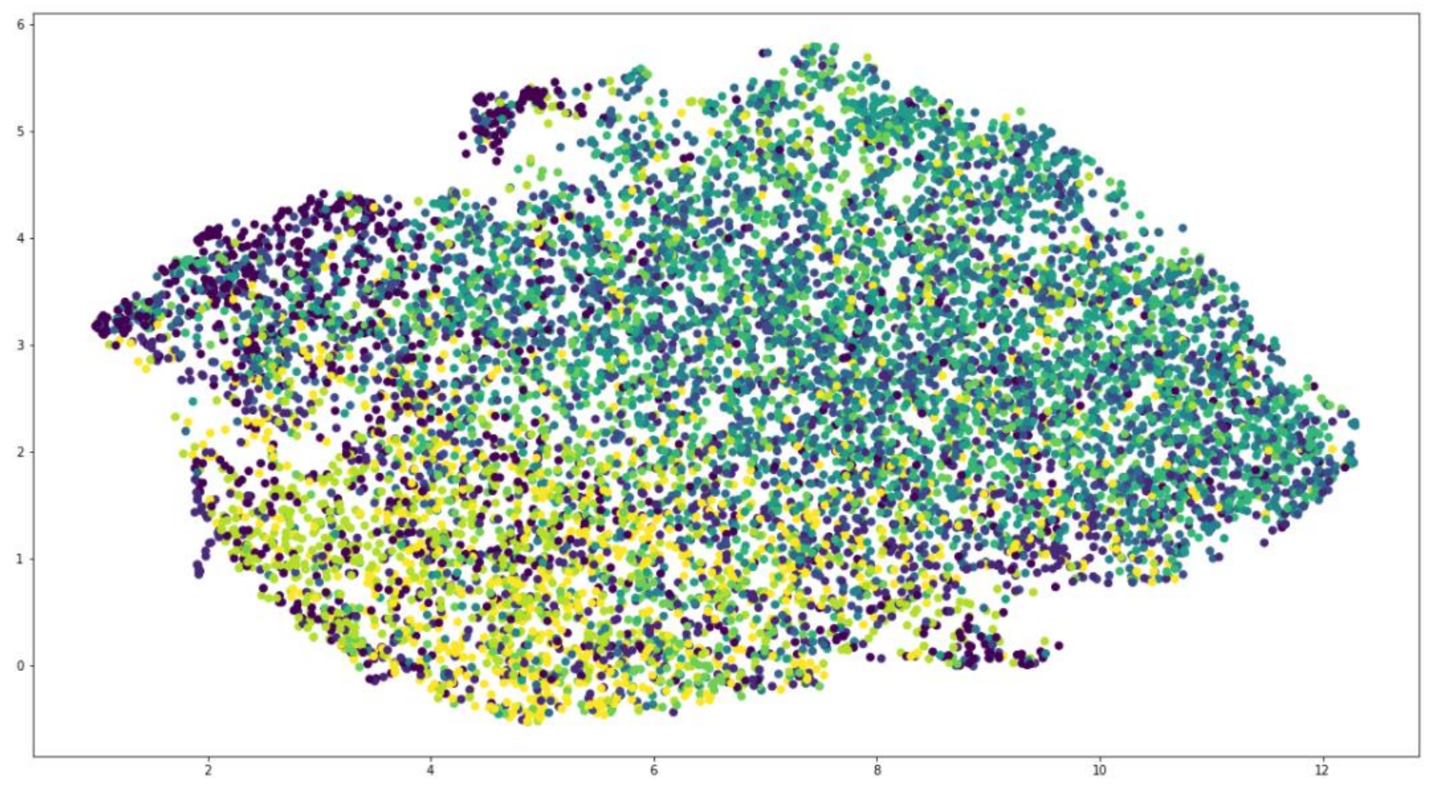


The “recon” name in above image means reconstructed image.

Distribution analysis was also carried out for 50 random images and its output in latent space. This analysis was done on a scatter plot matrix and as per the plot it looks as if there are no weird grouping in each of the pairs. This indicates that all 50 of these random images have some sort of similarity in them regardless of the classes.



To further visualize separation of our classes, we also plot a UMAP plot to find rough estimations about class difference. This turns out to be the class difference all within one cluster but certain classes have specific clusters in certain regions:

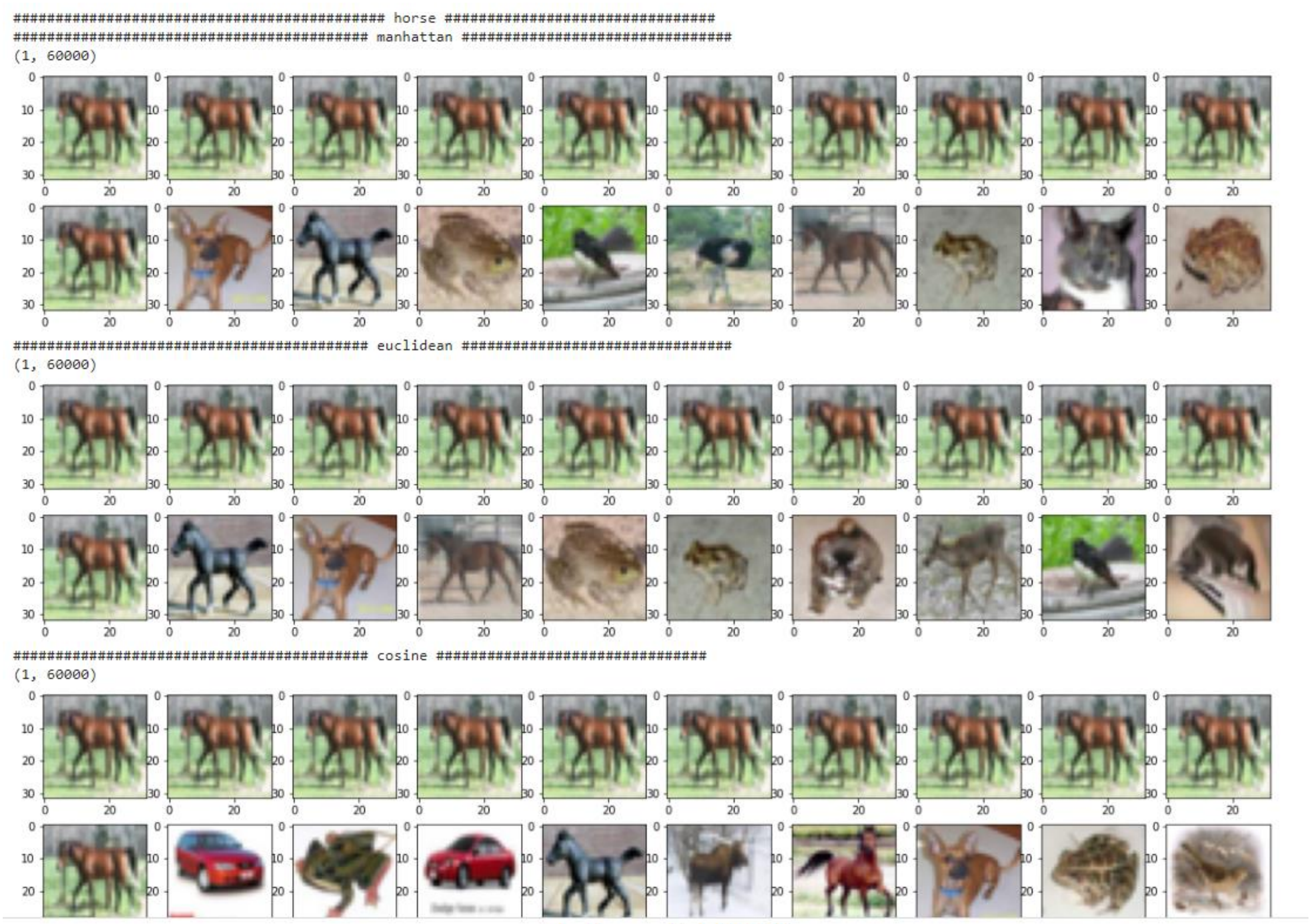


This visualization technique helped to project 10 dimensions in 2 dimensional space for a plot to analyse results.

Data Querying

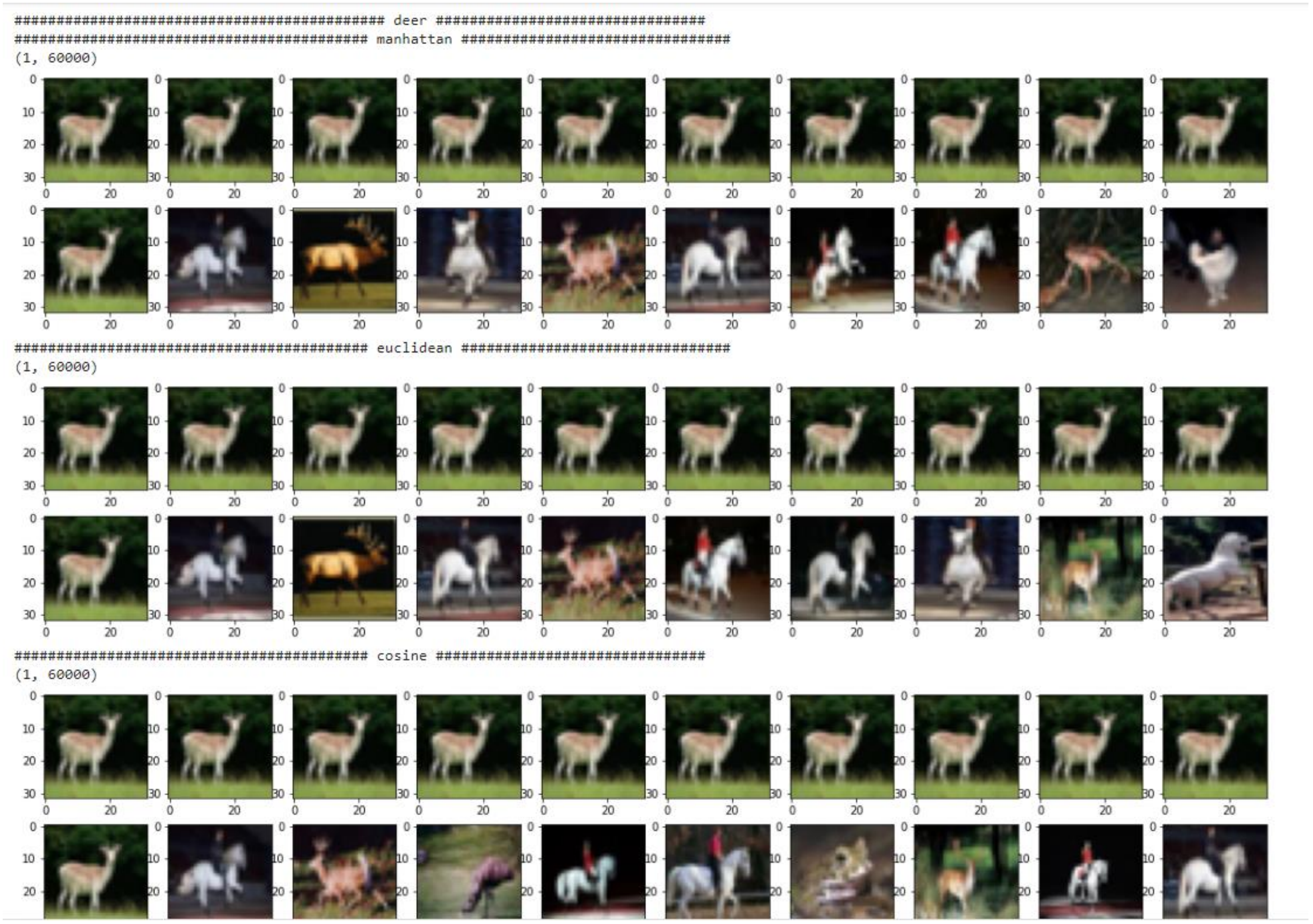
In this section we used our trained model to find similar images using pairwise distance. Then we sorted top 10 images to find most similar images. This process was repeated for each distance metric, mainly cosine, Euclidean and Manhattan distance. For our output we had best results in querying using Euclidean distance only.

For instance we observed that for all horse images we had the most optimal output when using Euclidean output:



```
############################################ horse ################################
########################################## manhattan ################################
(1, 60000)
########################################## euclidean ################################
(1, 60000)
########################################## cosine ################################
(1, 60000)
```

Similar observations were found for deer as well. One possible reason of horse appearing in similarity results is because of resemblance in background, skin color and the number of legs too. It was the case when we queried for horse too.:



```
############################################ deer ################################
########################################## manhattan ################################
(1, 60000)
```

```
########################################## euclidean ################################
(1, 60000)
```

```
########################################## cosine ################################
(1, 60000)
```

This is the main function for querying pairwise distances. First we combined train and test set, encoded all the images and then we picked 20 random images, computed pairwise distance against entire dataset and then for each metric used top 10 images in ascending order were displayed.:

```python
from sklearn.metrics import pairwise_distances
def pair_distances(number_of_images = 20):
        metrics = ["manhattan", "euclidean", "cosine"]

        similar_imgs = list()

        combined_data = np.vstack((x_train2,x_test2))
        labels_combined = np.vstack((y_train,y_test))
        # print(combined_data.shape)
        full_encoded = encoder.predict(combined_data)
        # print(full_encoded.shape)
        indexes = list(range(0, full_encoded.shape[0]))

        for j in range(number_of_images):

          num = np.random.randint(1, high =len(combined_data))
          label = cifar10_classes[labels_combined[num,0]]
          org_img = combined_data[num]
          # print(org_img.shape)
          val = encoder.predict(org_img[None])[0]
          val = val.reshape(1,10)
          # print(val.shape)
          # val = np.reshape(val, (1, 10))
          print("############################################", label,"################################")
          for metric in metrics:
            print("##########################################",metric, "################################")
            distance = pairwise_distances(val, full_encoded, metric=metric)
            print(distance.shape)
            result = zip(indexes, distance[0][indexes])
            # print(result[0][0])
            # [(i, distance[0][i]) for i in range(len(indexes))]
            sort_tup = sorted(result, key = lambda x: x[1])

            for i in range(10):
              var = combined_data[sort_tup[i][0]]
              similar_imgs.append(var)
            similar = np.asarray(similar_imgs)

            plot_images(10, org_img, similar)
            similar_imgs.clear()
```