hw4

May 21, 2021

[1]:
```
import os
#import cv2
import numpy as np
#from tqdm import tqdm
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from tensorflow.keras import layers
from sklearn.decomposition import PCA
from tensorflow.keras import models
from mpl_toolkits.axes_grid1 import ImageGrid
from sklearn.metrics import confusion_matrix ,plot_confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator

epocs= 50
```

0.1 Let's Get some data!

So we started by just getting the fashion mnist dataset. Tensorflow had this build it, so it was very conveniet. Because the autoencoder takes in the training images as the label and the input the training labels were only needed for the final 2-D mapping.

[2]:
```
(train_images, train_labels), (test_images, test_labels) = keras.datasets.
↪fashion_mnist.load_data()
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

#train_labels = np.asarray(train_labels).astype('float32').reshape((-1,1))
#test_labels = np.asarray(test_labels).astype('float32').reshape((-1,1))
```

0.2 Make Noisy images now

- https://keras.io/examples/vision/autoencoder/

These functions I took directly from the Keras Site. It just displays the images and it also adds noice to the image. I did change the type of noise however, and I was sure to give it 0 mean gaussian noise.

[3]:
```
def noise(array):
    """
    Adds random noise to each image in the supplied array.
    """

    mean = 0.0   # some constant
    std = 1.0    # some constant (standard deviation)
    noisy_array = array + np.random.normal(mean, std, array.shape)

    return np.clip(noisy_array, 0.0, 1.0)

def display(array1, array2):
    """
    Displays ten random images from each one of the supplied arrays.
    """

    n = 10

    indices = np.random.randint(len(array1), size=n)
    images1 = array1[indices, :]
    images2 = array2[indices, :]

    plt.figure(figsize=(20, 4))
    for i, (image1, image2) in enumerate(zip(images1, images2)):
        ax = plt.subplot(2, n, i + 1)
        plt.imshow(image1.reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

        ax = plt.subplot(2, n, i + 1 + n)
        plt.imshow(image2.reshape(28, 28))
        plt.gray()
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

    plt.show()
```

[4]:
```
train_images = train_images.astype('float32')/255.
test_images = test_images.astype('float32')/255.

noisy_train_data = noise(train_images)
noisy_test_data = noise(test_images)
```

0.3 Build Model

Thd middle of the autoencoder was definitely the most difficult part. If I didn't get the sizes right, I ended up with an output image that was not (28×28) and my final error check would fail. Thought it's not 100% symmetric, it does line the final shape up very well.

The loss, stayed relatively close. It dropped initally and then stayed around ~0.39. With more training though my results would sometimes be better, other times, I would end up with just a white square.

[5]:
```
input_img = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(4, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional
bottleNeckOut = layers.Flatten(input_shape=x.shape)(encoded)
#bottleNeckOut = layers.Dense(4, activation='relu')(bottleNeckOut)
bottleNeckOut = layers.Dense(2, activation='relu')(bottleNeckOut)



bottleNeckIn = layers.Dense(4, activation='relu')(bottleNeckOut)
bottleNeckIn = layers.Reshape((2,2,1))(bottleNeckIn)

x = layers.Conv2D(4, (3, 3), activation='relu', padding='same')(bottleNeckIn)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(16, (3, 3), activation='relu')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = keras.Model(input_img, decoded)
denoiser = keras.models.clone_model(autoencoder)
encoder = keras.Model(input_img, bottleNeckOut)
autoencoder.compile(optimizer='adam',␣
↪loss='binary_crossentropy',metrics=['accuracy'])
denoiser.compile(optimizer='adam',␣
↪loss='binary_crossentropy',metrics=['accuracy'])

autoencoder.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 16)        160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 8)         1160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 8)           0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 8)           584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 8)           0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 4)           292
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 2, 2, 4)           0
_________________________________________________________________
flatten (Flatten)            (None, 16)                0
_________________________________________________________________
dense (Dense)                (None, 2)                 34
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 12
_________________________________________________________________
reshape (Reshape)            (None, 2, 2, 1)           0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 2, 2, 4)           40
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 4, 4, 4)           0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 8)           296
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 8, 8, 8)           0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 8)           584
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 14, 14, 16)        1168
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 28, 28, 16)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 1)         145
=================================================================
```

```
Total params: 4,475
Trainable params: 4,475
Non-trainable params: 0
_________________________________________________________________
```

[6]:
```python
history = autoencoder.fit(train_images, train_images,
                          batch_size=64,
                          steps_per_epoch = 500,
                          shuffle = True,
                          epochs=epocs)
```

```
Epoch 1/50
500/500 [==============================] - 11s 21ms/step - loss: 0.5247 -
accuracy: 0.4878
Epoch 2/50
500/500 [==============================] - 11s 21ms/step - loss: 0.3830 -
accuracy: 0.4882
Epoch 3/50
500/500 [==============================] - 11s 21ms/step - loss: 0.3682 -
accuracy: 0.4900
Epoch 4/50
500/500 [==============================] - 11s 21ms/step - loss: 0.3577 -
accuracy: 0.4938
Epoch 5/50
500/500 [==============================] - 11s 21ms/step - loss: 0.3506 -
accuracy: 0.4954
Epoch 6/50
500/500 [==============================] - 11s 21ms/step - loss: 0.3497 -
accuracy: 0.4941
Epoch 7/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3486 -
accuracy: 0.4927
Epoch 8/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3447 -
accuracy: 0.4970
Epoch 9/50
500/500 [==============================] - 11s 21ms/step - loss: 0.3438 -
accuracy: 0.4971
Epoch 10/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3442 -
accuracy: 0.4948
Epoch 11/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3429 -
accuracy: 0.4952
Epoch 12/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3434 -
accuracy: 0.4947
Epoch 13/50
```

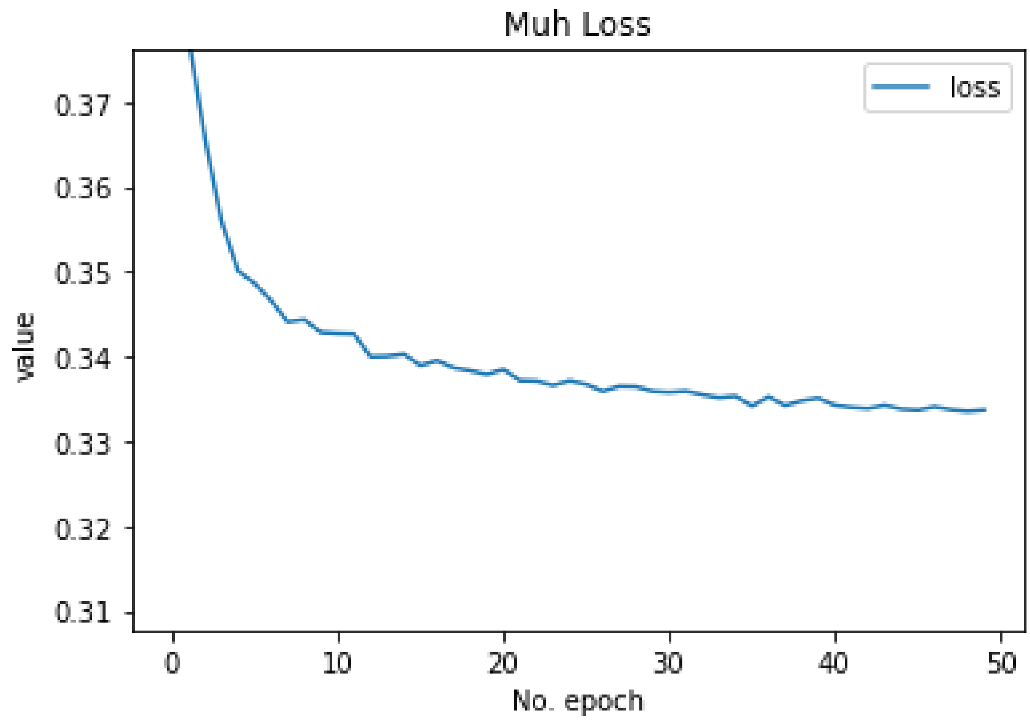
```
500/500 [==============================] - 11s 22ms/step - loss: 0.3396 -
accuracy: 0.4982
Epoch 14/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3404 -
accuracy: 0.4973
Epoch 15/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3415 -
accuracy: 0.4949
Epoch 16/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3395 -
accuracy: 0.4970
Epoch 17/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3398 -
accuracy: 0.4962
Epoch 18/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3385 -
accuracy: 0.4976
Epoch 19/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3381 -
accuracy: 0.4976
Epoch 20/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3380 -
accuracy: 0.4973
Epoch 21/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3380 -
accuracy: 0.4972
Epoch 22/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3372 -
accuracy: 0.4969
Epoch 23/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3369 -
accuracy: 0.4981
Epoch 24/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3359 -
accuracy: 0.4984
Epoch 25/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3376 -
accuracy: 0.4957
Epoch 26/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3374 -
accuracy: 0.4972
Epoch 27/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3368 -
accuracy: 0.4967
Epoch 28/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3367 -
accuracy: 0.4966
Epoch 29/50
```

```
500/500 [==============================] - 11s 22ms/step - loss: 0.3371 -
accuracy: 0.4960
Epoch 30/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3358 -
accuracy: 0.4975
Epoch 31/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3359 -
accuracy: 0.4982
Epoch 32/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3366 -
accuracy: 0.4962
Epoch 33/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3364 -
accuracy: 0.4968
Epoch 34/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3349 -
accuracy: 0.4982
Epoch 35/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3352 -
accuracy: 0.4978
Epoch 36/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3342 -
accuracy: 0.4992
Epoch 37/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3355 -
accuracy: 0.4967
Epoch 38/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3351 -
accuracy: 0.4984
Epoch 39/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3341 -
accuracy: 0.4984
Epoch 40/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3353 -
accuracy: 0.4974
Epoch 41/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3343 -
accuracy: 0.4989
Epoch 42/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3332 -
accuracy: 0.4992
Epoch 43/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3347 -
accuracy: 0.4973
Epoch 44/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3345 -
accuracy: 0.4976
Epoch 45/50
```

```
500/500 [==============================] - 11s 22ms/step - loss: 0.3338 -
accuracy: 0.4978
Epoch 46/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3338 -
accuracy: 0.4978
Epoch 47/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3338 -
accuracy: 0.4979
Epoch 48/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3339 -
accuracy: 0.4986
Epoch 49/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3326 -
accuracy: 0.4993
Epoch 50/50
500/500 [==============================] - 11s 22ms/step - loss: 0.3339 -
accuracy: 0.4971
```

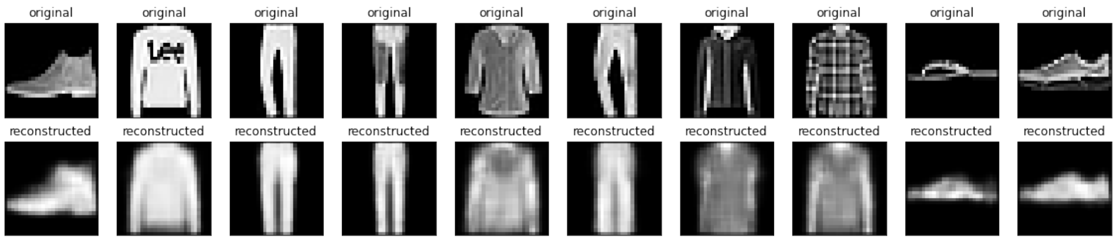
[7]:

```
rebuild = autoencoder.predict(test_images)
avee = sum(history.history['loss'])/len(history.history['loss'])
plt.plot(history.history['loss'],label = "loss")
plt.title('Muh Loss')
plt.ylabel('value')
plt.xlabel('No. epoch')
plt.ylim(avee-(avee*.10),avee+(avee*.10))
plt.legend(loc="upper right")
plt.show()
```



[8]:
```
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(test_images[i])
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(rebuild[i])
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

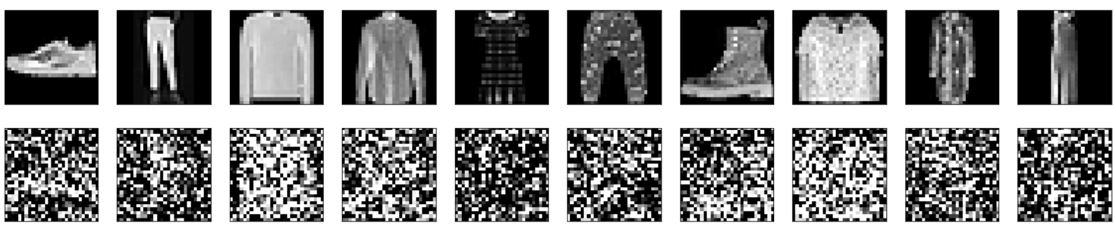


1 De-noise

- https://keras.io/examples/vision/autoencoder/

My denoiser thinks everything looks like a shirt. I assume this is because shirts are rectangular and take up most of the image size. This is similar to what the noise looks like on the picture. If I use a different kind of noise, or less than a standard deviation of 1, my images come out a lot better. But I was trying to follow the instructions as stated.

[9]:
```
display(train_images, noisy_train_data)
```



[10]:
```
history = denoiser.fit(noisy_train_data, train_images,
                          batch_size=64,
                          steps_per_epoch = 500,
                          shuffle = True,
                          epochs=epocs)
```

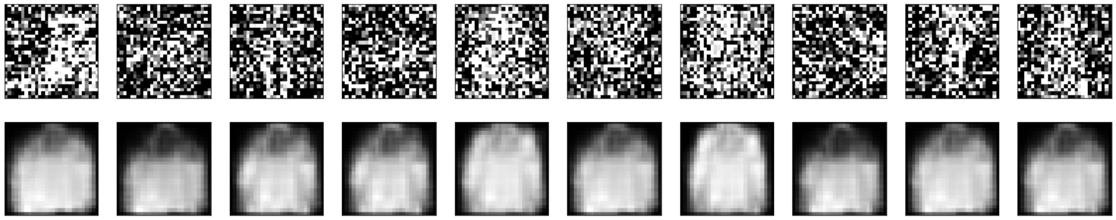
```
Epoch 1/50
500/500 [==============================] - 11s 22ms/step - loss: 0.5842 -
accuracy: 0.4910
Epoch 2/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4960 -
accuracy: 0.4834
Epoch 3/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4942 -
accuracy: 0.4853
Epoch 4/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4930 -
```

```
accuracy: 0.4858
Epoch 5/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4918 -
accuracy: 0.4875
Epoch 6/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4920 -
accuracy: 0.4866
Epoch 7/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4921 -
accuracy: 0.4864
Epoch 8/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4910 -
accuracy: 0.4877
Epoch 9/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4922 -
accuracy: 0.4875
Epoch 10/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4911 -
accuracy: 0.4873
Epoch 11/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4911 -
accuracy: 0.4898
Epoch 12/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4909 -
accuracy: 0.4878
Epoch 13/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4910 -
accuracy: 0.4881
Epoch 14/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4920 -
accuracy: 0.4857
Epoch 15/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4915 -
accuracy: 0.4867
Epoch 16/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4913 -
accuracy: 0.4879
Epoch 17/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4903 -
accuracy: 0.4876
Epoch 18/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4912 -
accuracy: 0.4877
Epoch 19/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4917 -
accuracy: 0.4879
Epoch 20/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4911 -
```

```
accuracy: 0.4872
Epoch 21/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4901 -
accuracy: 0.4887
Epoch 22/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4917 -
accuracy: 0.4878
Epoch 23/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4913 -
accuracy: 0.4873
Epoch 24/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4903 -
accuracy: 0.4882
Epoch 25/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4913 -
accuracy: 0.4865
Epoch 26/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4916 -
accuracy: 0.4865
Epoch 27/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4899 -
accuracy: 0.4893
Epoch 28/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4911 -
accuracy: 0.4882
Epoch 29/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4915 -
accuracy: 0.4867
Epoch 30/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4915 -
accuracy: 0.4872
Epoch 31/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4913 -
accuracy: 0.4864
Epoch 32/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4898 -
accuracy: 0.4887
Epoch 33/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4901 -
accuracy: 0.4882
Epoch 34/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4909 -
accuracy: 0.4878
Epoch 35/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4912 -
accuracy: 0.4864
Epoch 36/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4918 -
```

```
accuracy: 0.4867
Epoch 37/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4908 -
accuracy: 0.4873
Epoch 38/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4913 -
accuracy: 0.4863
Epoch 39/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4903 -
accuracy: 0.4881
Epoch 40/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4917 -
accuracy: 0.4869
Epoch 41/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4915 -
accuracy: 0.4866
Epoch 42/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4914 -
accuracy: 0.4871
Epoch 43/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4914 -
accuracy: 0.4870
Epoch 44/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4904 -
accuracy: 0.4879
Epoch 45/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4906 -
accuracy: 0.4887
Epoch 46/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4914 -
accuracy: 0.4861
Epoch 47/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4905 -
accuracy: 0.4885
Epoch 48/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4910 -
accuracy: 0.4880
Epoch 49/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4897 -
accuracy: 0.4893
Epoch 50/50
500/500 [==============================] - 11s 22ms/step - loss: 0.4905 -
accuracy: 0.4877
```

[11]:
```python
predictions = autoencoder.predict(noisy_test_data)
display(noisy_test_data, predictions)
```

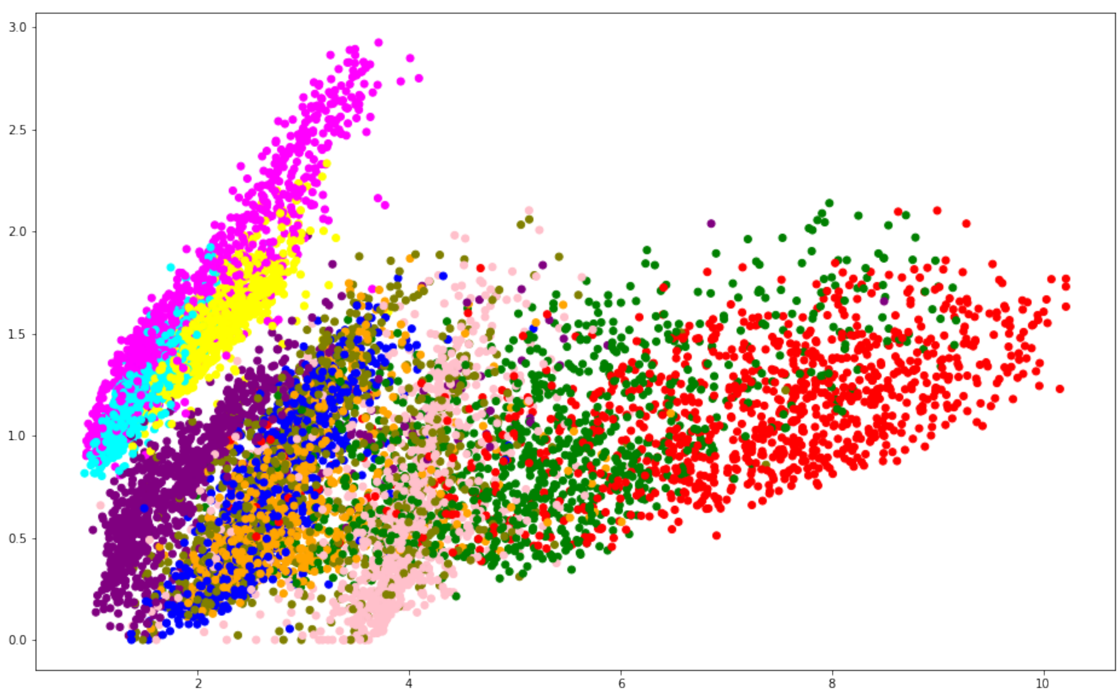


2 2-D Plot

The 2-D plot didn't always come out nice. There's definitely some randomness to the 2-D plot. I realize that sometimes I get a nice shape that is more similar to the PCA, but other times, it will be very far off.

[13]:
```
cdict = {0: 'pink',1: 'red', 2: 'blue', 3: 'green',4: 'orange', 5: 'magenta', 6:
 ↪ 'olive',
         7: 'cyan', 8: 'purple', 9: 'yellow'}
color_list = [cdict[i] for i in test_labels ]
x_test_encoded = encoder.predict(test_images, batch_size=32)
print(x_test_encoded.shape)
plt.figure(figsize=(16, 10))
plt.scatter(x_test_encoded[:, 0], x_test_encoded[:, 1], c = color_list)
plt.show()
```

```
(10000, 2)
```



3 PCA

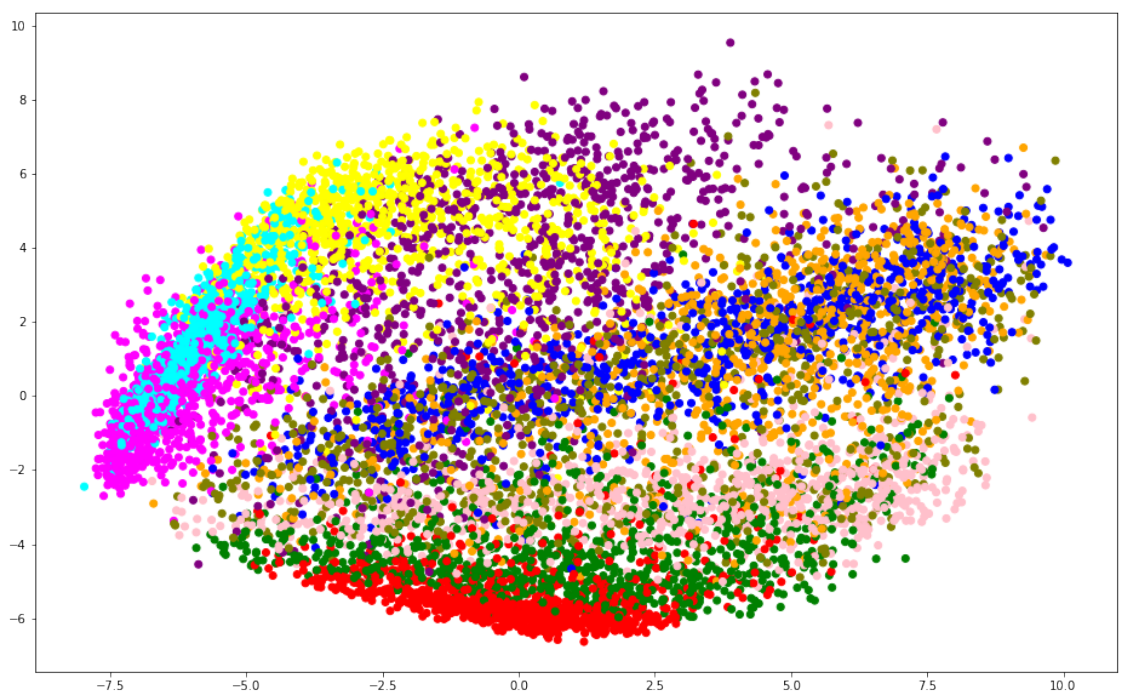
For the PCA portion. I didn't understand what Dr. Rhodes meant by "using a for loop", so I instead just did this in the same way that've used PCA in my last classes. I transformed the data from 3 dimensions to 2 before I fed it into the PCA function. In otherwords, I just vectorized the images using reshape. In this way I could reduce the components down to just two parameters(2 columns) and make the 2-D graph from that.

The shape of the PCA and the autoencoder is not always the same, but it is similar. There's definitely an overall shape that comes up when this is trained. I assume with more training the autoencoder and PCA 2-D maps would look more similar. But I was having an issue with consistentcy in my autoencoder.

[14]:
```
pca = PCA(n_components=2)
ti = test_images.reshape(10000,-1)
principalComponents = pca.fit_transform(ti)
principalComponents.shape
```

[14]: (10000, 2)

[15]:
```
m,n = principalComponents.shape
plt.figure(figsize=(16, 10))
plt.scatter(principalComponents[:, 0], principalComponents[:, 1], c =␣
 ↪color_list[:m])
plt.show()
```



```
[ ]:
```