ML_prog5

August 8, 2020

1 ML_Prog4

1.1 Demonstrating lda and pca

1.2 Importing necessary libraries

[330]:
```
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from matplotlib import image
import numpy as np
%matplotlib inline
import seaborn as sns; sns.set()
from PIL import Image
import os, sys
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from mpl_toolkits.mplot3d import Axes3D
```

1.2.1 Class map. Dict containing values of each class

[331]:
```
class_map = {0:"T-shirt/top",
1:"Trouser/pants",
2:"Pullover shirt",
3:"Dress",
4:"Coat",
5:"Sandal",
6:"Shirt",
7:"Sneaker",
8:"Bag",
9:"Ankle boot"}
```

1.3 Using fashion mnist dataset

1.3.1 Has about 60000 train and 10000 test images rangin through 10 classes

[332]:
```
mnist = tf.keras.datasets.fashion_mnist
```

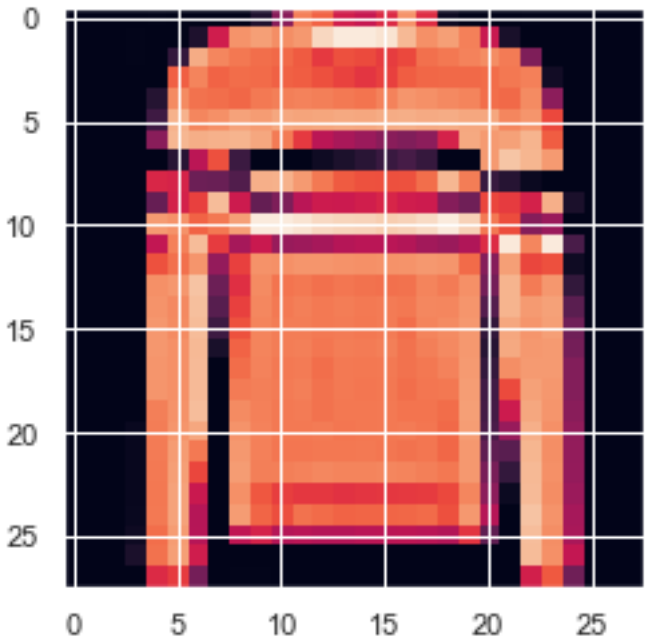
[333]:
```
(training_images, training_labels), (test_images, test_labels) = mnist.
↪load_data()
```

1.3.2 Use index to see through the dataset (Train)

Within indices [0:60000]

[334]:
```
index = 5
np.set_printoptions(linewidth=200)
plt.imshow(training_images[index])
print(class_map[training_labels[index]])
#print(training_images[index])
```

```
Pullover shirt
```



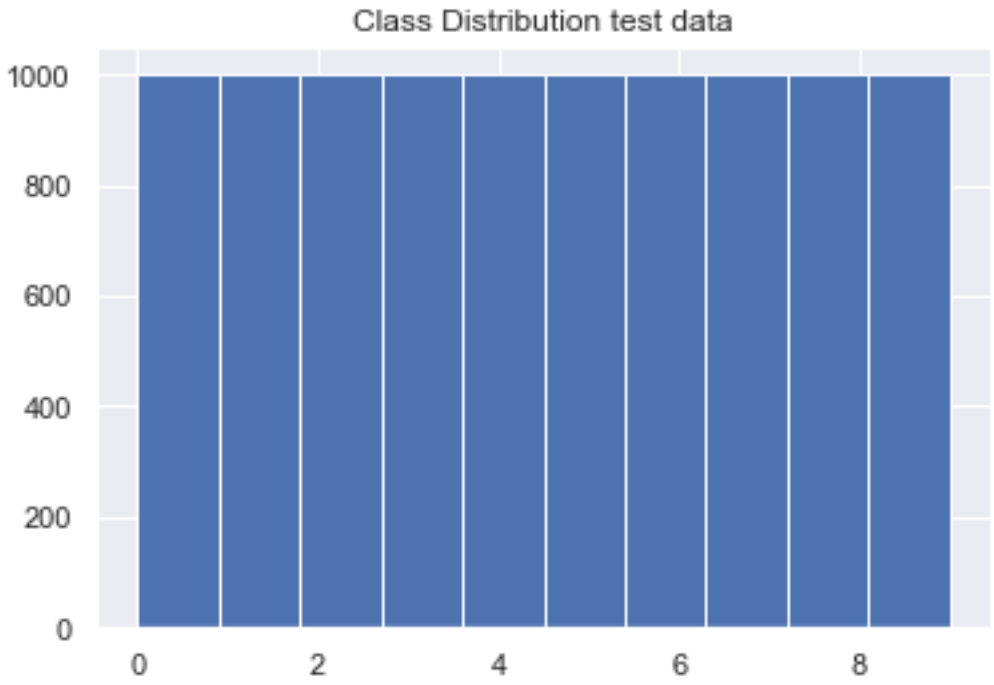
1.4 Class distribution of training set. Uniform

[78]:
```
plt.hist(training_labels,bins = 10)
plt.title("Class Distribution train data")
plt.show()
```

1.5 Class distribution of test set. Also uniform

1.5.1 Class distirbutions help debuggin classifier issues and having correct assumptions about data

[335]:
```
plt.hist(test_labels,bins = 10)
plt.title("Class Distribution test data")
plt.show()
```



1.6 Normalizing data and converting to float

1.6.1 tf needs float values

[83]:
```
training_images = training_images/255.0
test_images = test_images/255.0
```

1.7 Applying PCA() on the dataset

1.7.1 PCA maps the input feature space into another feature space based on eigenvectors. It considers variance of data

1.7.2 It is unsupervised and helps remove correlated attributes. Denoising can also be done.

1.7.3 Has disadvantages. Showcased later

1.7.4 Displaying Correlation between a subset of input feature space

[338]:
```
tot = list()
for i in range(0,3):
    for j in range(0,3):
        if i !=j:
            cor = np.corrcoef(training_images.reshape(training_images.
 ↪shape[0],-1)[:][i],training_images.reshape(training_images.shape[0],-1)[:
 ↪][j])[0,1]
            print("Correlation : ",i,j,"---",cor)
```

```
            tot.append(cor)

print("Average correlation: ",sum(tot)/len(tot))
```

```
Correlation :  0 1 --- 0.13536707659555364
Correlation :  0 2 --- 0.16735204377956048
Correlation :  1 0 --- 0.13536707659555364
Correlation :  1 2 --- 0.5990603230873564
Correlation :  2 0 --- 0.1673520437795605
Correlation :  2 1 --- 0.5990603230873564
Average correlation:  0.3005931478208235
```

1.7.5 Converting to PCA feature space

[337]:
```
train_vector = training_images.reshape(training_images.shape[0],-1)
test_vector = test_images.reshape(test_images.shape[0],-1)
pca = PCA(784)
projected = pca.fit_transform(train_vector)
projected_test = pca.fit_transform(test_vector)
```

1.7.6 Displaying correlation of pca feature space.

Clearly reduced.

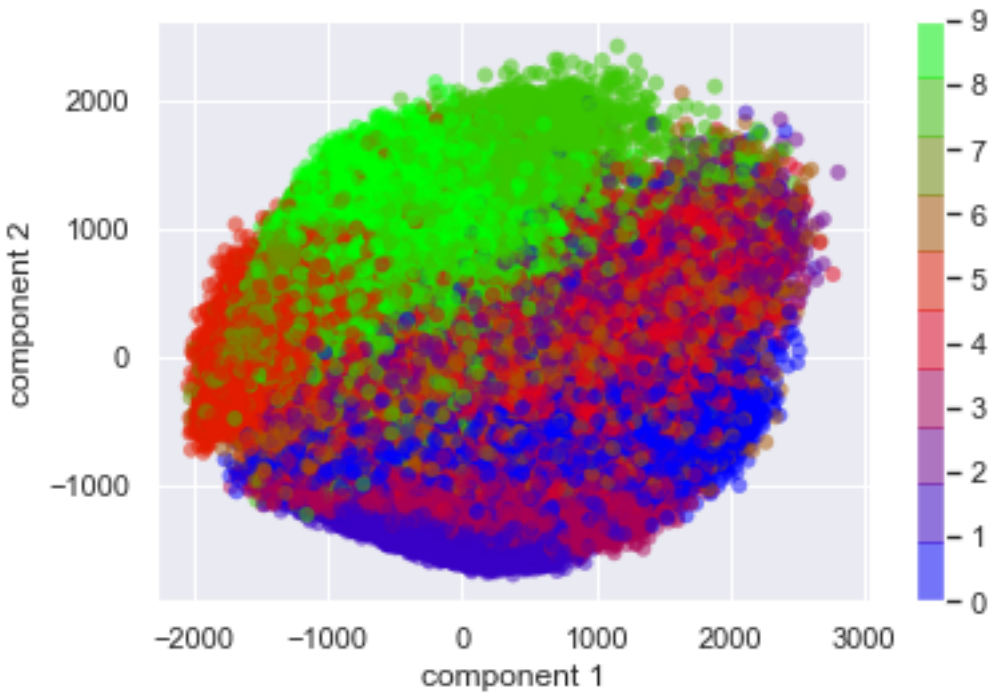
[ ]:
```
tot = list()
for i in range(0,3):
    for j in range(0,3):
        if i !=j:
            cor = np.corrcoef(projected[:][i],projected[:][j])[0,1]
            print("Correlation : ",i,j,"---",cor)
            tot.append(cor)

print("Average correlation: ",sum(tot)/len(tot))
```

1.8 Visualizing PCA vs LDA

1.8.1 PCA feature space in 2D plot

Evident that it maintains maximum variance

[340]:
```
plt.scatter(projected[:, 0], projected[:, 1],
            c=training_labels, edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('brg', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```



1.9 Converting inpout space to LDA feature space

1.9.1 It is important to note that LDA is not only a dimensionality reducing technique, rather it is also a classificaton technique. It essentially converts the feature space to find the axis that

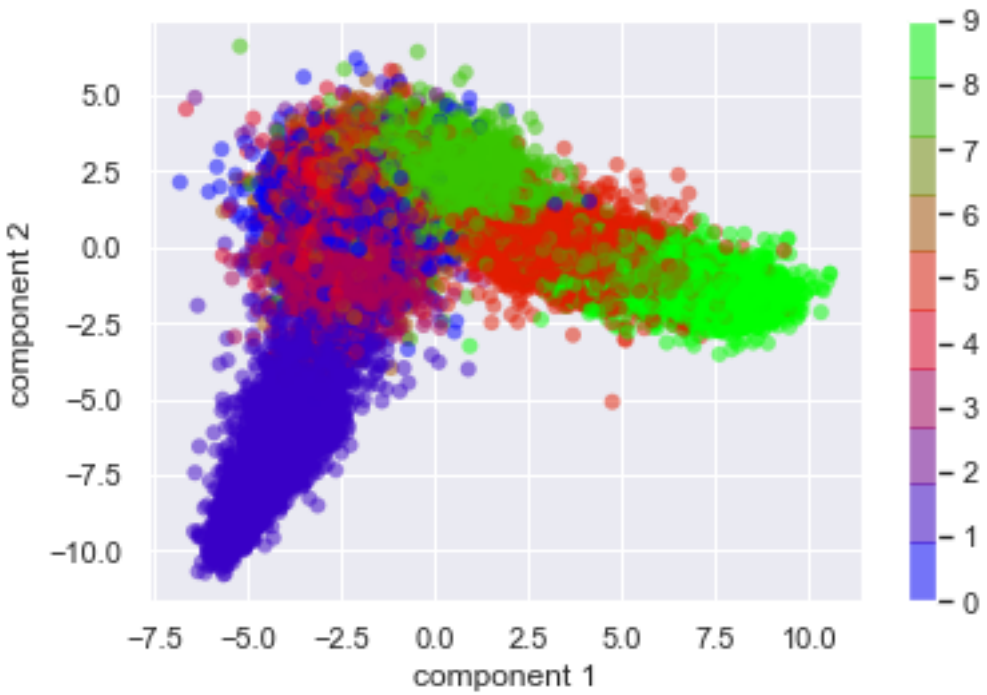
1. Reduces the scatter within the class
2. Increases distance/linear seperability between means

[341]:
```
lda = LinearDiscriminantAnalysis(n_components=9)
projected_lda = lda.fit(train_vector, training_labels).transform(train_vector)
projected_lda_test = lda.fit(test_vector,test_labels).transform(test_vector)
```

1.10 2D plot of LDA feature space.

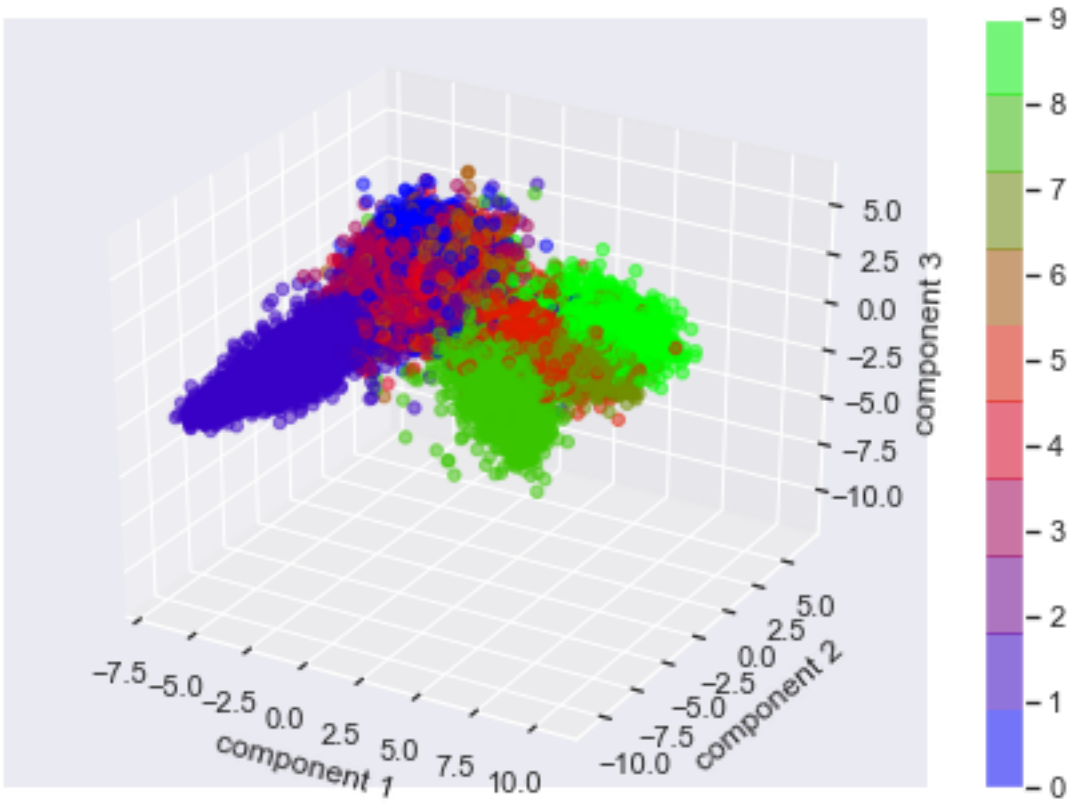
1.10.1 The difference btween PCA and LDA is evident here. LDA plot has more separability

[342]:
```
plt.scatter(projected_lda[:, 0], projected_lda[:, 1],
            c=training_labels, edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('brg', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```



1.11 3D plot of the LDA feature space to inspect separability further

[343]:
```
ax = Axes3D(plt.figure())
p = ax.scatter(projected_lda[:, 0], projected_lda[:, 1],projected_lda[:,2],
            c=training_labels, alpha=0.5,
            cmap=plt.cm.get_cmap('brg', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
ax.set_zlabel('component 3')
plt.colorbar(p);
```



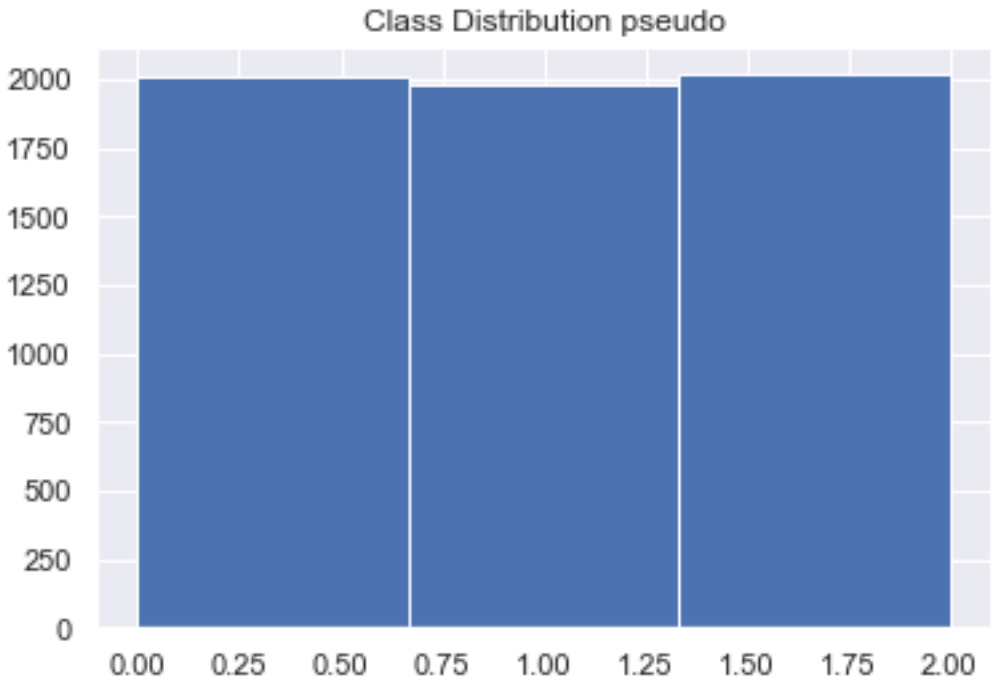
1.12 LDA (As a classifier also) tends to overfit the data as it is a supervised technique.

1.12.1 It aims to maximise the distance between class means using class labels and this behavior tends to overfit the data if not careful

1.12.2 Here we choose class 0 (Shirt/top) and separate that class into 3 classes.

There is no difference (Atleast not that much) between the images in class 0. But we are telling the algorithm that there is.

[344]:
```
pseudo_target = np.random.randint(3,size = (6000,))
plt.hist(pseudo_target,bins = 3)
plt.title("Class Distribution pseudo")
plt.show()
```



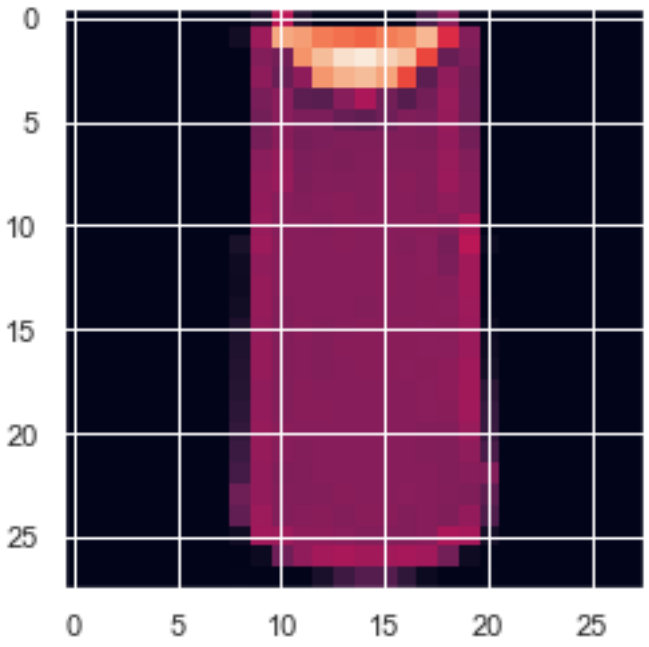
1.13 Visualizing class 0.

1.13.1 It is the same class for us.

[345]:
```
idc = np.where(training_labels == 0)[0]
print(idc)
temp_train = np.array([training_images[i] for i in idc])
plt.imshow(temp_train[1])
```

```
[    1     2     4 ... 59974 59985 59998]
```

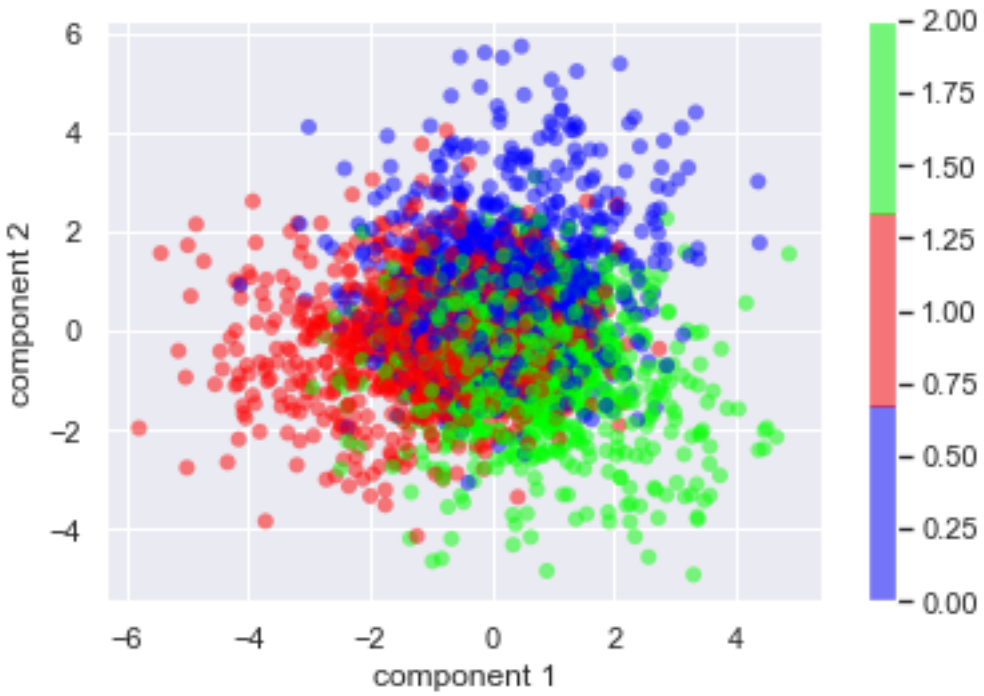
[345]: <matplotlib.image.AxesImage at 0x19c20628348>



1.14 Plotting LDA feature space of constructed pseudo classes.

1.14.1 It is clear to see that the model tries to separate classes even when there is no difference

[346]:
```
lda = LinearDiscriminantAnalysis(n_components=2)
ps_lda = lda.fit(temp_train.reshape(temp_train.shape[0],-1),pseudo_target).
 ↪transform(temp_train.reshape(temp_train.shape[0],-1))
plt.scatter(ps_lda[:, 0], ps_lda[:, 1],
            c=pseudo_target, edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('brg', 3))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```



1.15 Explorng how classification fares on different feature spaces

1.16 Network trained on Straight up original input space (Best performing model)

Can be further optimized. But for now, this is good enough

[84]:
```
init_model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                         tf.keras.layers.Dense(units = 512,␣
 ↪activation = tf.nn.relu),
                                        tf.keras.layers.Dense(units =␣
 ↪10,activation = tf.nn.softmax)])
init_model.compile(optimizer = "adam",loss =␣
 ↪"sparse_categorical_crossentropy",metrics = ["accuracy"])
init_model.fit(training_images,training_labels,batch_size = 512,epochs = 20)
```

```
Epoch 1/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.5994 -
acc: 0.7936
Epoch 2/20
60000/60000 [==============================] - 1s 14us/sample - loss: 0.4130 -
acc: 0.8567
Epoch 3/20
60000/60000 [==============================] - 1s 14us/sample - loss: 0.3727 -
acc: 0.8686
Epoch 4/20
```

```
60000/60000 [==============================] - 1s 12us/sample - loss: 0.3454 -
acc: 0.8769
Epoch 5/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.3263 -
acc: 0.8830
Epoch 6/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.3038 -
acc: 0.8909
Epoch 7/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.2949 -
acc: 0.89330s - loss: 0.3098 -
Epoch 8/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.2841 -
acc: 0.89600s - loss: 0.2786 -
Epoch 9/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.2697 -
acc: 0.9018
Epoch 10/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.2593 -
acc: 0.9057
Epoch 11/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.2539 -
acc: 0.9076
Epoch 12/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.2442 -
acc: 0.91050s - loss: 0.2277 -
Epoch 13/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.2349 -
acc: 0.9145
Epoch 14/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.2278 -
acc: 0.9184
Epoch 15/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.2193 -
acc: 0.9205
Epoch 16/20
60000/60000 [==============================] - 1s 13us/sample - loss: 0.2119 -
acc: 0.9235
Epoch 17/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.2086 -
acc: 0.9243
Epoch 18/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.2004 -
acc: 0.9271
Epoch 19/20
60000/60000 [==============================] - 1s 12us/sample - loss: 0.1963 -
acc: 0.9276
Epoch 20/20
```

```
60000/60000 [==============================] - 1s 12us/sample - loss: 0.1954 -
acc: 0.9298
```

[84]:
```
<tensorflow.python.keras.callbacks.History at 0x19b1ad9e7c8>
```

[87]:
```
print("Train accuracy:",init_model.
 ↪evaluate(training_images,training_labels)[1]*100)
print("Test accuracy:",init_model.evaluate(test_images,test_labels)[1]*100)
```

```
60000/60000 [==============================] - 2s 32us/sample - loss: 0.1791 -
acc: 0.9349
Train accuracy: 93.48833560943604
10000/10000 [==============================] - 0s 31us/sample - loss: 0.3093 -
acc: 0.8917
Test accuracy: 89.1700029373169
```

1.17 Network trained on PCA input space.

The accuracies are really low. Showing that PCA is not a mandatory step in the machine learning pipeline. Here we have lost valuable information(features) by projecting into PCA feature space. And this stands as a disadvantage of PCA

[312]:
```
feat = 128
pca_model = tf.keras.models.Sequential([tf.keras.layers.Input(shape = [feat]),
                                        tf.keras.layers.Dense(units = 512,␣
 ↪activation = tf.nn.relu,
                                                              ␣
 ↪kernel_regularizer = tf.keras.regularizers.l2(0.01)),
                                       tf.keras.layers.Dense(units =␣
 ↪10,activation = tf.nn.softmax)])
pca_model.compile(optimizer = "adam",loss =␣
 ↪"sparse_categorical_crossentropy",metrics = ["accuracy"])
pca_model.fit(projected[:,:feat],training_labels,batch_size = 512,epochs = 40)
```

```
Epoch 1/40
60000/60000 [==============================] - 1s 12us/sample - loss: 1.4349 -
acc: 0.7662
Epoch 2/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.6257 -
acc: 0.8358
Epoch 3/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.5585 -
acc: 0.8444
Epoch 4/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.5262 -
acc: 0.8506
Epoch 5/40
60000/60000 [==============================] - 0s 7us/sample - loss: 0.5081 -
```

```
acc: 0.8534
Epoch 6/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4933 -
acc: 0.8569
Epoch 7/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4816 -
acc: 0.8593
Epoch 8/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4718 -
acc: 0.8615
Epoch 9/40
60000/60000 [==============================] - 0s 7us/sample - loss: 0.4641 -
acc: 0.8636
Epoch 10/40
60000/60000 [==============================] - 0s 7us/sample - loss: 0.4579 -
acc: 0.8653
Epoch 11/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4513 -
acc: 0.8667
Epoch 12/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4447 -
acc: 0.8684
Epoch 13/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4394 -
acc: 0.8687
Epoch 14/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4348 -
acc: 0.8708
Epoch 15/40
60000/60000 [==============================] - 1s 9us/sample - loss: 0.4306 -
acc: 0.8711
Epoch 16/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4255 -
acc: 0.8735
Epoch 17/40
60000/60000 [==============================] - 0s 7us/sample - loss: 0.4230 -
acc: 0.8735
Epoch 18/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4194 -
acc: 0.8744
Epoch 19/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4174 -
acc: 0.8762
Epoch 20/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4134 -
acc: 0.8759
Epoch 21/40
60000/60000 [==============================] - 0s 7us/sample - loss: 0.4084 -
```

```
acc: 0.8784
Epoch 22/40
60000/60000 [==============================] - 0s 7us/sample - loss: 0.4082 -
acc: 0.8781
Epoch 23/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.4044 -
acc: 0.8782
Epoch 24/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3999 -
acc: 0.8807
Epoch 25/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3976 -
acc: 0.8814
Epoch 26/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3976 -
acc: 0.8805
Epoch 27/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3958 -
acc: 0.8805
Epoch 28/40
60000/60000 [==============================] - 0s 7us/sample - loss: 0.3912 -
acc: 0.8834
Epoch 29/40
60000/60000 [==============================] - 0s 7us/sample - loss: 0.3907 -
acc: 0.8841
Epoch 30/40
60000/60000 [==============================] - 0s 7us/sample - loss: 0.3876 -
acc: 0.8837
Epoch 31/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3864 -
acc: 0.8844
Epoch 32/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3834 -
acc: 0.8858
Epoch 33/40
60000/60000 [==============================] - 0s 7us/sample - loss: 0.3819 -
acc: 0.8865
Epoch 34/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3800 -
acc: 0.8865
Epoch 35/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3785 -
acc: 0.8870
Epoch 36/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3769 -
acc: 0.8883
Epoch 37/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3756 -
```

```
acc: 0.8877
Epoch 38/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3756 -
acc: 0.8874
Epoch 39/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3710 -
acc: 0.8897
Epoch 40/40
60000/60000 [==============================] - 0s 8us/sample - loss: 0.3709 -
acc: 0.8896
```

[312]:
```
<tensorflow.python.keras.callbacks.History at 0x19c61fe0d88>
```

[313]:
```
print("Train accuracy:",pca_model.evaluate(projected[:,:
 ↪feat],training_labels)[1]*100)
print("Test accuracy:",pca_model.evaluate(projected_test[:,:
 ↪feat],test_labels)[1]*100)
```

```
60000/60000 [==============================] - 3s 52us/sample - loss: 0.3549 -
acc: 0.8959
Train accuracy: 89.58666920661926
10000/10000 [==============================] - 1s 51us/sample - loss: 2.5582 -
acc: 0.5152
Test accuracy: 51.52000188827515
```

1.18 Network trained on LDA feature space

Similar to PCA, we seem to have lost a good dea of information. And hence the difference between training and test errors keeep increasing (Overfitting)

[310]:
```
feat = 6
lda_model = tf.keras.models.Sequential([tf.keras.layers.Input(shape = [feat]),
                                        tf.keras.layers.Dense(units = 512,␣
 ↪activation = tf.nn.relu,
                                                                          ␣
 ↪kernel_regularizer = tf.keras.regularizers.l2(1)),
                                       tf.keras.layers.Dense(units =␣
 ↪10,activation = tf.nn.softmax)])
lda_model.compile(optimizer = "adam",loss =␣
 ↪"sparse_categorical_crossentropy",metrics = ["accuracy"])
lda_model.fit(projected_lda[:,:feat],training_labels,batch_size = 512,epochs =␣
 ↪50)
```

```
Epoch 1/50
60000/60000 [==============================] - 0s 8us/sample - loss: 4.4935 -
acc: 0.6783
Epoch 2/50
60000/60000 [==============================] - 0s 5us/sample - loss: 1.2046 -
```

```
acc: 0.7578
Epoch 3/50
60000/60000 [==============================] - 0s 5us/sample - loss: 1.0380 -
acc: 0.7661
Epoch 4/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.9558 -
acc: 0.7676
Epoch 5/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.9011 -
acc: 0.7710
Epoch 6/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.8624 -
acc: 0.7716
Epoch 7/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.8324 -
acc: 0.7733
Epoch 8/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.8105 -
acc: 0.7741
Epoch 9/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.7920 -
acc: 0.7732
Epoch 10/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.7780 -
acc: 0.7735
Epoch 11/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.7650 -
acc: 0.7738
Epoch 12/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.7527 -
acc: 0.7752
Epoch 13/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.7440 -
acc: 0.7732
Epoch 14/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.7344 -
acc: 0.7756
Epoch 15/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.7269 -
acc: 0.7745
Epoch 16/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.7202 -
acc: 0.7756
Epoch 17/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.7146 -
acc: 0.7766
Epoch 18/50
60000/60000 [==============================] - 0s 6us/sample - loss: 0.7115 -
```

```
acc: 0.7751
Epoch 19/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.7054 -
acc: 0.7739
Epoch 20/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6994 -
acc: 0.7754
Epoch 21/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6978 -
acc: 0.7758
Epoch 22/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6919 -
acc: 0.7751
Epoch 23/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6880 -
acc: 0.7761
Epoch 24/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6845 -
acc: 0.7764
Epoch 25/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6820 -
acc: 0.7738
Epoch 26/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6820 -
acc: 0.7736
Epoch 27/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6800 -
acc: 0.7745
Epoch 28/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6745 -
acc: 0.7748
Epoch 29/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6726 -
acc: 0.7737
Epoch 30/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6719 -
acc: 0.7764
Epoch 31/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6674 -
acc: 0.7772
Epoch 32/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6660 -
acc: 0.7766
Epoch 33/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6655 -
acc: 0.7756
Epoch 34/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6619 -
```

```
acc: 0.7761
Epoch 35/50
60000/60000 [==============================] - 0s 6us/sample - loss: 0.6590 -
acc: 0.7770
Epoch 36/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6570 -
acc: 0.7776
Epoch 37/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6588 -
acc: 0.7768
Epoch 38/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6559 -
acc: 0.7754
Epoch 39/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6532 -
acc: 0.7768
Epoch 40/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6527 -
acc: 0.7765
Epoch 41/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6520 -
acc: 0.7763
Epoch 42/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6497 -
acc: 0.7780
Epoch 43/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6493 -
acc: 0.7772
Epoch 44/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6485 -
acc: 0.7768
Epoch 45/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6456 -
acc: 0.7785
Epoch 46/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6455 -
acc: 0.7777
Epoch 47/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6433 -
acc: 0.7779
Epoch 48/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6430 -
acc: 0.7772
Epoch 49/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6430 -
acc: 0.7764
Epoch 50/50
60000/60000 [==============================] - 0s 5us/sample - loss: 0.6394 -
```

```
acc: 0.7789
```

[310]:
```
<tensorflow.python.keras.callbacks.History at 0x19c61d79588>
```

[311]:
```
print("Train accuracy:",lda_model.evaluate(projected_lda[:,:
 ↪feat],training_labels)[1]*100)
print("Test accuracy:",lda_model.evaluate(projected_lda_test[:,:
 ↪feat],test_labels)[1]*100)
```

```
60000/60000 [==============================] - 3s 50us/sample - loss: 0.6410 -
acc: 0.7758
Train accuracy: 77.57999897003174
10000/10000 [==============================] - 0s 49us/sample - loss: 0.6113 -
acc: 0.7654
Test accuracy: 76.53999924659729
```

1.19 Using LDA feature space of PCA feature space to train network

1.19.1 Applying PCA to LDA first acts as a regualarizer to LDA and gives better performance than either one. (In this case)

Definitely not arguing that PCA on LDA is a good regularization method. But merely stating observations. Because vanilla regularization (l1,l2,dropout) is much more efficient

[314]:
```
lda = LinearDiscriminantAnalysis(n_components=9)
pca_lda = lda.fit(projected,training_labels).transform(projected)
pca_lda_test = lda.fit(projected_test,test_labels).transform(projected_test)
```

[328]:
```
feat = 6
lda_pca_model = tf.keras.models.Sequential([tf.keras.layers.Input(shape =␣
 ↪[feat]),
                                            tf.keras.layers.Dense(units = 512,␣
 ↪activation = tf.nn.relu,
                                                                  ␣
 ↪kernel_regularizer = tf.keras.regularizers.l2(0.001)),
                                           tf.keras.layers.Dense(units =␣
 ↪10,activation = tf.nn.softmax)])
lda_pca_model.compile(optimizer = "adam",loss =␣
 ↪"sparse_categorical_crossentropy",metrics = ["accuracy"])
lda_pca_model.fit(pca_lda[:,:feat],training_labels,batch_size = 1024,epochs = 50)
```

```
Epoch 1/50
60000/60000 [==============================] - 0s 7us/sample - loss: 0.9974 -
acc: 0.6878
Epoch 2/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.6347 -
acc: 0.7713
Epoch 3/50
```

```
60000/60000 [==============================] - 0s 3us/sample - loss: 0.6064 -
acc: 0.7804
Epoch 4/50
60000/60000 [==============================] - 0s 4us/sample - loss: 0.5894 -
acc: 0.7861
Epoch 5/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5781 -
acc: 0.7904
Epoch 6/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5701 -
acc: 0.7936
Epoch 7/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5645 -
acc: 0.7939
Epoch 8/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5599 -
acc: 0.7952
Epoch 9/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5563 -
acc: 0.7967
Epoch 10/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5535 -
acc: 0.7974
Epoch 11/50
60000/60000 [==============================] - 0s 4us/sample - loss: 0.5513 -
acc: 0.7974
Epoch 12/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5489 -
acc: 0.7987
Epoch 13/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5467 -
acc: 0.7989
Epoch 14/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5463 -
acc: 0.7988
Epoch 15/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5440 -
acc: 0.7997
Epoch 16/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5431 -
acc: 0.7997
Epoch 17/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5415 -
acc: 0.8000
Epoch 18/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5398 -
acc: 0.8001
Epoch 19/50
```

```
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5389 -
acc: 0.8008
Epoch 20/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5382 -
acc: 0.8014
Epoch 21/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5376 -
acc: 0.8018
Epoch 22/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5361 -
acc: 0.8014
Epoch 23/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5348 -
acc: 0.8024
Epoch 24/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5341 -
acc: 0.8020
Epoch 25/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5334 -
acc: 0.8016
Epoch 26/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5324 -
acc: 0.8017
Epoch 27/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5320 -
acc: 0.8030
Epoch 28/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5311 -
acc: 0.8034
Epoch 29/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5308 -
acc: 0.8031
Epoch 30/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5297 -
acc: 0.8037
Epoch 31/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5293 -
acc: 0.8034
Epoch 32/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5283 -
acc: 0.8044
Epoch 33/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5285 -
acc: 0.8027
Epoch 34/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5269 -
acc: 0.8040
Epoch 35/50
```

```
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5271 -
acc: 0.8036
Epoch 36/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5268 -
acc: 0.8036
Epoch 37/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5259 -
acc: 0.8043
Epoch 38/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5258 -
acc: 0.8037
Epoch 39/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5250 -
acc: 0.8037
Epoch 40/50
60000/60000 [==============================] - 0s 4us/sample - loss: 0.5247 -
acc: 0.8044
Epoch 41/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5233 -
acc: 0.8043
Epoch 42/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5236 -
acc: 0.8040
Epoch 43/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5233 -
acc: 0.8046
Epoch 44/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5225 -
acc: 0.8050
Epoch 45/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5219 -
acc: 0.8045
Epoch 46/50
60000/60000 [==============================] - 0s 4us/sample - loss: 0.5212 -
acc: 0.8047
Epoch 47/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5214 -
acc: 0.8048
Epoch 48/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5210 -
acc: 0.8047
Epoch 49/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5210 -
acc: 0.8042
Epoch 50/50
60000/60000 [==============================] - 0s 3us/sample - loss: 0.5206 -
acc: 0.8045
```

[328]: <tensorflow.python.keras.callbacks.History at 0x19c666926c8>

[329]:
```
print("Train accuracy:",lda_pca_model.evaluate(pca_lda[:,:
 ↪feat],training_labels)[1]*100)
print("Test accuracy:",lda_pca_model.evaluate(pca_lda_test[:,:
 ↪feat],test_labels)[1]*100)
```

```
60000/60000 [==============================] - 4s 59us/sample - loss: 0.5184 -
acc: 0.8063
Train accuracy: 80.62833547592163
10000/10000 [==============================] - 1s 53us/sample - loss: 0.5304 -
acc: 0.7869
Test accuracy: 78.6899983882904
```

End of notebook