Fashion-mnist autoencoders

In [ ]:

```
from keras.layers import Dense,Input
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping
from tensorflow.keras.utils import to_categorical
```

In [ ]:

```
from numpy import argmax, array_equal
import matplotlib.pyplot as plt

from keras.models import Model
from imgaug import augmenters
from random import randint

import pandas as pd
import numpy as np
```

Reading Dataset

In [ ]:

```
train = pd.read_csv("/content/sample_data/fashion-mnist_test.csv")
train_x = train[list(train.columns)[1:]].values
train_y = train['label'].values
```

Normalize and reshape the predictors

In [ ]:

```
train_x = train_x / 255
```

Creating train and validation datasets

In [ ]:

```
train_x, val_x, train_y, val_y = train_test_split(train_x, train_y, test_size=0.2)
```

Reshape the inputs

In [ ]:

```
train_x = train_x.reshape(-1, 784)
val_x = val_x.reshape(-1, 784)
```

Autoencoder architecture

Input Layer

In [ ]:

```
input_layer = Input(shape=(784,))
```

Encoding architecture

In [ ]:

```
encode_layer1 = Dense(1500, activation='relu')(input_layer)
encode_layer2 = Dense(1200, activation='relu')(encode_layer1)
encode_layer3 = Dense(900, activation='relu')(encode_layer2)
encode_layer4 = Dense(600, activation='relu')(encode_layer3)
```

Latent view

In [ ]:

```
latent_view   = Dense(10, activation='relu')(encode_layer4)
```

Decoding architecture

In [ ]:

```
decode_layer1 = Dense(600, activation='relu')(latent_view)
decode_layer2 = Dense(900, activation='relu')(decode_layer1)
decode_layer3 = Dense(1200, activation='relu')(decode_layer2)
decode_layer4 = Dense(1500, activation='relu')(decode_layer3)
```

Output Layer

In [ ]:

```
output_layer  = Dense(784)(decode_layer4)
model = Model(input_layer, output_layer)
```

Summary

In [ ]:

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 784)]             0         
                                                                 
 dense (Dense)               (None, 1500)              1177500   
                                                                 
 dense_1 (Dense)             (None, 1200)              1801200   
                                                                 
 dense_2 (Dense)             (None, 900)               1080900   
                                                                 
 dense_3 (Dense)             (None, 600)               540600    
                                                                 
 dense_4 (Dense)             (None, 10)                6010      
                                                                 
 dense_5 (Dense)             (None, 600)               6600      
                                                                 
 dense_6 (Dense)             (None, 900)               540900    
                                                                 
 dense_7 (Dense)             (None, 1200)              1081200   
                                                                 
 dense_8 (Dense)             (None, 1500)              1801500   
                                                                 
 dense_9 (Dense)             (None, 784)               1176784   
                                                                 
=================================================================
Total params: 9,213,194
Trainable params: 9,213,194
Non-trainable params: 0
_________________________________________________________________
```

Compiling and fitting

In [ ]:

```
model.compile(optimizer='adam', loss='mse')
early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=1,
mode='auto')
model.fit(train_x, train_x, epochs=20, batch_size=2048, validation_data=(val_x, val_x),
callbacks=[early_stopping])
```

```
Epoch 1/20
4/4 [==============================] - 8s 2s/step - loss: 0.1778 - val_los
s: 0.1196
Epoch 2/20
4/4 [==============================] - 6s 1s/step - loss: 0.0992 - val_los
s: 0.0688
Epoch 3/20
4/4 [==============================] - 6s 1s/step - loss: 0.0727 - val_los
s: 0.0679
Epoch 4/20
4/4 [==============================] - 6s 1s/step - loss: 0.0705 - val_los
s: 0.0665
Epoch 5/20
4/4 [==============================] - 6s 1s/step - loss: 0.0684 - val_los
s: 0.0659
Epoch 6/20
4/4 [==============================] - 7s 2s/step - loss: 0.0675 - val_los
s: 0.0653
Epoch 7/20
4/4 [==============================] - 6s 1s/step - loss: 0.0669 - val_los
s: 0.0643
Epoch 8/20
4/4 [==============================] - 6s 1s/step - loss: 0.0658 - val_los
s: 0.0626
Epoch 9/20
4/4 [==============================] - 6s 1s/step - loss: 0.0633 - val_los
s: 0.0578
Epoch 10/20
4/4 [==============================] - 6s 1s/step - loss: 0.0574 - val_los
s: 0.0541
Epoch 11/20
4/4 [==============================] - 6s 1s/step - loss: 0.0541 - val_los
s: 0.0515
Epoch 12/20
4/4 [==============================] - 6s 1s/step - loss: 0.0506 - val_los
s: 0.0476
Epoch 13/20
4/4 [==============================] - 6s 1s/step - loss: 0.0472 - val_los
s: 0.0453
Epoch 14/20
4/4 [==============================] - 6s 1s/step - loss: 0.0447 - val_los
s: 0.0424
Epoch 15/20
4/4 [==============================] - 6s 1s/step - loss: 0.0427 - val_los
s: 0.0419
Epoch 16/20
4/4 [==============================] - 6s 1s/step - loss: 0.0420 - val_los
s: 0.0411
Epoch 17/20
4/4 [==============================] - 6s 2s/step - loss: 0.0410 - val_los
s: 0.0397
Epoch 18/20
4/4 [==============================] - 7s 1s/step - loss: 0.0399 - val_los
s: 0.0386
Epoch 19/20
4/4 [==============================] - 6s 1s/step - loss: 0.0389 - val_los
s: 0.0382
Epoch 20/20
4/4 [==============================] - 6s 1s/step - loss: 0.0384 - val_los
s: 0.0378
```

Out[ ]:

```
<keras.callbacks.History at 0x7f054eb43e90>
```

Predictions on validation data

In [ ]:

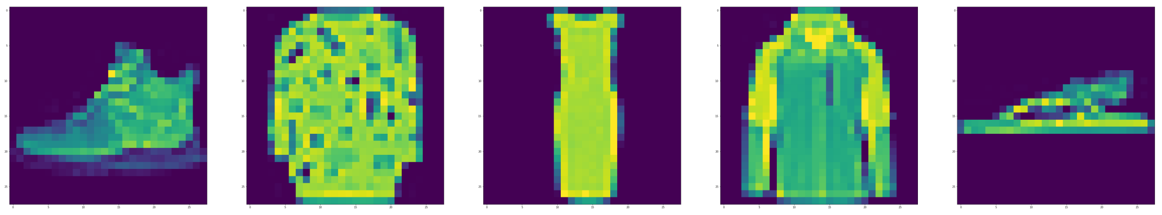
```
preds = model.predict(val_x)
```

Plotting

Input : Actual images

In [ ]:

```
from PIL import Image
f, ax = plt.subplots(1,5)
f.set_size_inches(80, 40)
for i in range(5):
    ax[i].imshow(val_x[i].reshape(28, 28))
plt.show()
```



Predicted : Autoencoder Output

In [ ]:

```
f, ax = plt.subplots(1,5)
f.set_size_inches(80, 40)
for i in range(5):
    ax[i].imshow(preds[i].reshape(28, 28))
plt.show()
```