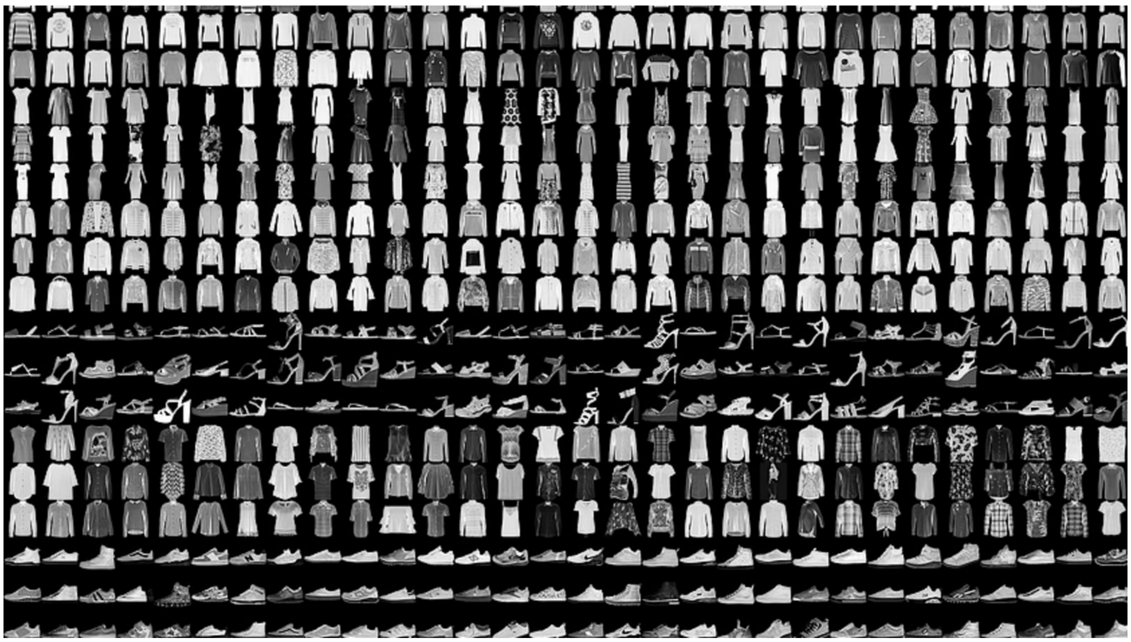
MNIST Fashion

November 23, 2021

[1]:
```python
from IPython.display import Image
Image(filename='Fashion Photo.jpeg')
```

[1]:



[2]:
```python
import pandas as pd
import numpy as np
import tensorflow as tf
from keras.layers import Dense
from tensorflow.keras.layers import Dense
from tensorflow import keras
import matplotlib.pyplot as plt
import os
import time
from scipy.stats import reciprocal
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
```

[3]:
```
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()
```

[4]:
```
X_train_full.shape
```

[4]:
```
(60000, 28, 28)
```

[5]:
```
X_train_full.dtype
```

[5]:
```
dtype('uint8')
```

You can see on line 2 that the training and test sets are already split. The next line of code creates the validation set. Additionally, its ideal to scale down the pixel intensity to a 0-1 range by dividing them by 255 which also converts them to a float type variable.

[6]:
```
X_valid, X_train = X_train_full[:5000] / 255.0, X_train_full[5000:] / 255.0
y_valid, y_train = y_train_full[:5000] / 255.0, y_train_full[5000:]
```

[7]:
```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal",␣
↪"Shirt", "Sneaker", "Bag", "Ankle boot"]
```

The next chunk of code performs the following tasks: 1. Creates a sequential model leveraging keras. Also known as a Sequential API (Single stack layers connected in a sequential manner, hence the naming convention). When you flatten the images (Second line of code), you convert the images into a 1D array (receives x input data and computes a reshape (-1, 1), for example). The dense layer number indicates the number of neurons (300, 100, and 10 respectively).

[8]:
```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

The model summary below demonstrates that the Flatten layer has 235,500 parameters. To calculate this, you would 784X300 which is 235,200 plus the 300 bias terms for a total of 235,500 parameters. This provides flexibility; however, it can run the risk of overfitting as well. If you want the model layers for recall later, input the following code: model.layers to retrieve them.

[9]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 300)               235500

 dense_1 (Dense)             (None, 100)               30100
```

```
 dense_2 (Dense)             (None, 10)                1010

=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

[10]:
```python
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

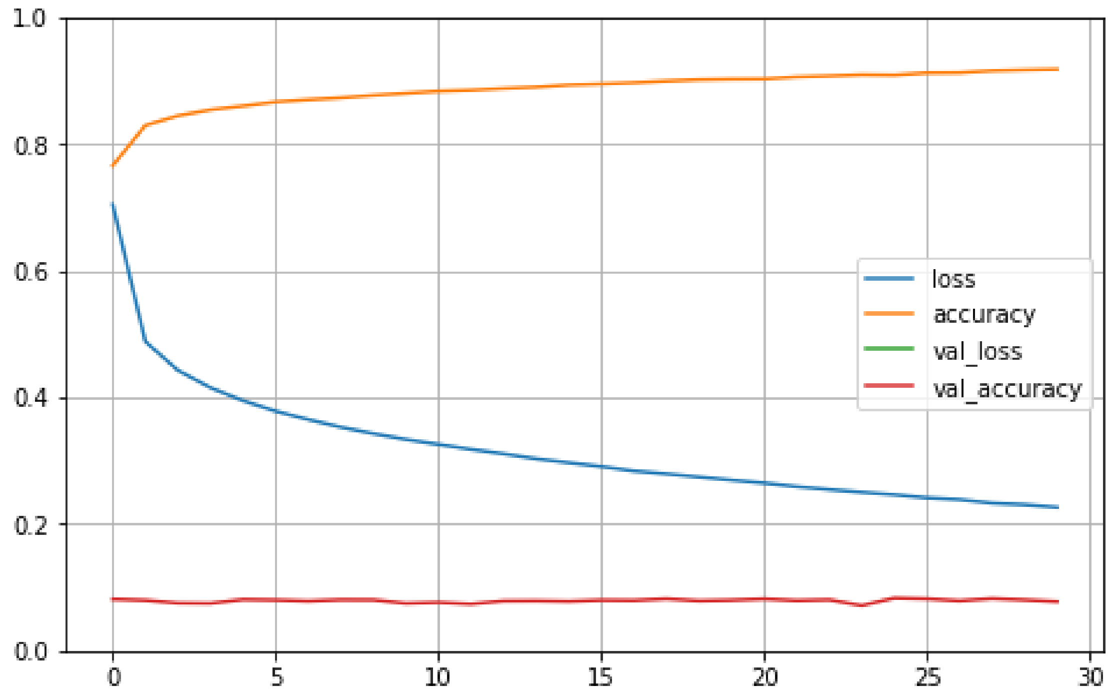
[11]:
```python
history = model.fit(X_train, y_train, epochs=30,
                    validation_data=(X_valid, y_valid))
```

```
Epoch 1/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.7049 -
accuracy: 0.7660 - val_loss: 5.9453 - val_accuracy: 0.0814
Epoch 2/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.4885 -
accuracy: 0.8297 - val_loss: 6.6280 - val_accuracy: 0.0796
Epoch 3/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.4428 -
accuracy: 0.8449 - val_loss: 7.1807 - val_accuracy: 0.0752
Epoch 4/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.4155 -
accuracy: 0.8543 - val_loss: 7.5309 - val_accuracy: 0.0748
Epoch 5/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.3950 -
accuracy: 0.8603 - val_loss: 7.6678 - val_accuracy: 0.0808
Epoch 6/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.3784 -
accuracy: 0.8668 - val_loss: 7.4561 - val_accuracy: 0.0800
Epoch 7/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.3651 -
accuracy: 0.8703 - val_loss: 7.5399 - val_accuracy: 0.0784
Epoch 8/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.3530 -
accuracy: 0.8732 - val_loss: 7.9217 - val_accuracy: 0.0804
Epoch 9/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.3429 -
accuracy: 0.8771 - val_loss: 8.0265 - val_accuracy: 0.0802
Epoch 10/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.3336 -
accuracy: 0.8803 - val_loss: 8.1549 - val_accuracy: 0.0748
Epoch 11/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.3258 -
accuracy: 0.8837 - val_loss: 8.3079 - val_accuracy: 0.0762
```

```
Epoch 12/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.3180 -
accuracy: 0.8853 - val_loss: 8.0543 - val_accuracy: 0.0738
Epoch 13/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.3110 -
accuracy: 0.8879 - val_loss: 8.3507 - val_accuracy: 0.0786
Epoch 14/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.3031 -
accuracy: 0.8901 - val_loss: 8.4408 - val_accuracy: 0.0788
Epoch 15/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2969 -
accuracy: 0.8937 - val_loss: 8.4385 - val_accuracy: 0.0780
Epoch 16/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2909 -
accuracy: 0.8952 - val_loss: 8.2256 - val_accuracy: 0.0798
Epoch 17/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2838 -
accuracy: 0.8971 - val_loss: 8.6416 - val_accuracy: 0.0794
Epoch 18/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2793 -
accuracy: 0.8999 - val_loss: 8.3365 - val_accuracy: 0.0820
Epoch 19/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2744 -
accuracy: 0.9020 - val_loss: 8.6944 - val_accuracy: 0.0788
Epoch 20/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2694 -
accuracy: 0.9029 - val_loss: 9.0355 - val_accuracy: 0.0798
Epoch 21/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2647 -
accuracy: 0.9031 - val_loss: 8.5646 - val_accuracy: 0.0818
Epoch 22/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2591 -
accuracy: 0.9063 - val_loss: 8.6802 - val_accuracy: 0.0792
Epoch 23/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2545 -
accuracy: 0.9078 - val_loss: 9.1139 - val_accuracy: 0.0804
Epoch 24/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2501 -
accuracy: 0.9098 - val_loss: 8.9075 - val_accuracy: 0.0714
Epoch 25/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2463 -
accuracy: 0.9092 - val_loss: 8.8738 - val_accuracy: 0.0832
Epoch 26/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2414 -
accuracy: 0.9127 - val_loss: 9.1589 - val_accuracy: 0.0820
Epoch 27/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2386 -
accuracy: 0.9130 - val_loss: 8.4330 - val_accuracy: 0.0788
```

```
Epoch 28/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2331 -
accuracy: 0.9161 - val_loss: 9.0822 - val_accuracy: 0.0824
Epoch 29/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2306 -
accuracy: 0.9174 - val_loss: 9.4218 - val_accuracy: 0.0802
Epoch 30/30
1719/1719 [==============================] - 2s 1ms/step - loss: 0.2264 -
accuracy: 0.9184 - val_loss: 9.3971 - val_accuracy: 0.0776
```

[12]:
```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```



[ ]: