untitled1

April 29, 2025

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense
```

[2]:
```
# Load dataset
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()

# Normalize pixel values to [0, 1]
X_train = X_train / 255.0
X_test = X_test / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515             0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880             0s
0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
5148/5148             0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4422102/4422102             0s
0us/step
```

[3]:
```
model = Sequential()

# Flatten 28x28 images into 1D array of 784
model.add(Flatten(input_shape=(28,28)))

# Add hidden layer
model.add(Dense(128, activation='relu'))

# Output layer (10 classes)
```

```
model.add(Dense(10, activation='softmax'))
```

```
/usr/local/lib/python3.11/dist-
packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

[4]:
```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[10]:
```
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.1)
```

```
Epoch 1/20
1688/1688             7s 4ms/step -
accuracy: 0.9129 - loss: 0.2348 - val_accuracy: 0.8843 - val_loss: 0.3414
Epoch 2/20
1688/1688             8s 5ms/step -
accuracy: 0.9168 - loss: 0.2239 - val_accuracy: 0.8888 - val_loss: 0.3244
Epoch 3/20
1688/1688             9s 4ms/step -
accuracy: 0.9199 - loss: 0.2194 - val_accuracy: 0.8855 - val_loss: 0.3361
Epoch 4/20
1688/1688             8s 5ms/step -
accuracy: 0.9214 - loss: 0.2113 - val_accuracy: 0.8913 - val_loss: 0.3364
Epoch 5/20
1688/1688             8s 5ms/step -
accuracy: 0.9237 - loss: 0.2048 - val_accuracy: 0.8867 - val_loss: 0.3258
Epoch 6/20
1688/1688             7s 4ms/step -
accuracy: 0.9264 - loss: 0.1953 - val_accuracy: 0.8822 - val_loss: 0.3631
Epoch 7/20
1688/1688             8s 5ms/step -
accuracy: 0.9263 - loss: 0.1949 - val_accuracy: 0.8893 - val_loss: 0.3384
Epoch 8/20
1688/1688             7s 4ms/step -
accuracy: 0.9288 - loss: 0.1908 - val_accuracy: 0.8840 - val_loss: 0.3493
Epoch 9/20
1688/1688             8s 5ms/step -
accuracy: 0.9326 - loss: 0.1831 - val_accuracy: 0.8850 - val_loss: 0.3552
Epoch 10/20
1688/1688             8s 5ms/step -
accuracy: 0.9331 - loss: 0.1777 - val_accuracy: 0.8950 - val_loss: 0.3566
Epoch 11/20
1688/1688             9s 4ms/step -
accuracy: 0.9369 - loss: 0.1700 - val_accuracy: 0.8883 - val_loss: 0.3547
```

```
Epoch 12/20
1688/1688             8s 5ms/step -
accuracy: 0.9371 - loss: 0.1661 - val_accuracy: 0.8892 - val_loss: 0.3564
Epoch 13/20
1688/1688             8s 5ms/step -
accuracy: 0.9386 - loss: 0.1662 - val_accuracy: 0.8823 - val_loss: 0.3880
Epoch 14/20
1688/1688             9s 4ms/step -
accuracy: 0.9416 - loss: 0.1596 - val_accuracy: 0.8915 - val_loss: 0.3609
Epoch 15/20
1688/1688             8s 5ms/step -
accuracy: 0.9425 - loss: 0.1544 - val_accuracy: 0.8907 - val_loss: 0.3831
Epoch 16/20
1688/1688             10s 5ms/step -
accuracy: 0.9440 - loss: 0.1507 - val_accuracy: 0.8897 - val_loss: 0.3671
Epoch 17/20
1688/1688             7s 4ms/step -
accuracy: 0.9435 - loss: 0.1495 - val_accuracy: 0.8918 - val_loss: 0.3744
Epoch 18/20
1688/1688             8s 5ms/step -
accuracy: 0.9457 - loss: 0.1464 - val_accuracy: 0.8908 - val_loss: 0.3990
Epoch 19/20
1688/1688             12s 6ms/step -
accuracy: 0.9470 - loss: 0.1424 - val_accuracy: 0.8908 - val_loss: 0.3981
Epoch 20/20
1688/1688             8s 5ms/step -
accuracy: 0.9476 - loss: 0.1397 - val_accuracy: 0.8885 - val_loss: 0.4010
```

[10]:
```
<keras.src.callbacks.history.History at 0x79adb83f6fd0>
```

[11]:
```
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc*100:.2f}%")
```

```
313/313             1s 3ms/step -
accuracy: 0.8860 - loss: 0.4019
Test Accuracy: 88.64%
```

[15]:
```
# Define class names for easy understanding
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Select an image to predict (for example: the 5th test image)
index = int(input("Enter the index of the image to predict: "))
img = X_test[index]

# Reshape the image because the model expects (1, 28, 28)
img_reshaped = img.reshape(1, 28, 28)
```

```python
# Predict the class
prediction = model.predict(img_reshaped)

# Get the index of the highest probability
predicted_class_index = np.argmax(prediction)

# Get the class name
predicted_class_name = class_names[predicted_class_index]

# Also get the true label for comparison
true_class_name = class_names[y_test[index]]

# Display the image and prediction
import matplotlib.pyplot as plt

plt.imshow(img, cmap='gray')
plt.title(f"Predicted: {predicted_class_name}\nActual: {true_class_name}")
plt.axis('off')
plt.show()
```

```
Enter the index of the image to predict: 15
1/1             0s 44ms/step
```