Multiclass Classification

When you have more than two classes as an option, it is called multi-class classification.

For this multi-class classification problem, we are going to build a neural network to classify different images of clothing.

In [33]:

```
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist

# The data has already been sorted into train and test sets
(train_data, train_label), (test_data, test_label) = fashion_mnist.load_data()
```

In [34]:

```
# Show the first training example
print(f"First training data:\n{train_data[0]}\n")
print(f"First training label:\n{train_label[0]}")
```

```
First training data:
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0
    0   1   4   0   0   0   0   1   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62
   54   0   0   0   1   3   4   0   0   3]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134
  144 123  23   0   0   0   0  12  10   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178
  107 156 161 109  64  23  77 130  72  15]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216
  216 163 127 121 122 146 141  88 172  66]
 [  0   0   0   0   0   0   0   0   0   1   1   1   0 200 232 232 233 229
  223 223 215 213 164 127 123 196 229   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 183 225 216 223 228
  235 227 224 222 224 221 223 245 173   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 193 228 218 213 198
  180 212 210 211 213 223 220 243 202   0]
 [  0   0   0   0   0   0   0   0   0   1   3   0  12 219 220 212 218 192
  169 227 208 218 224 212 226 197 209  52]
 [  0   0   0   0   0   0   0   0   0   0   6   0  99 244 222 220 218 203
  198 221 215 213 222 220 245 119 167  56]
 [  0   0   0   0   0   0   0   0   0   4   0   0  55 236 228 230 228 240
  232 213 218 223 234 217 217 209  92   0]
 [  0   0   1   4   6   7   2   0   0   0   0   0 237 226 217 223 222 219
  222 221 216 223 229 215 218 255  77   0]
 [  0   3   0   0   0   0   0   0   0  62 145 204 228 207 213 221 218 208
  211 218 224 223 219 215 224 244 159   0]
 [  0   0   0   0  18  44  82 107 189 228 220 222 217 226 200 205 211 230
  224 234 176 188 250 248 233 238 215   0]
 [  0  57 187 208 224 221 224 208 204 214 208 209 200 159 245 193 206 223
  255 255 221 234 221 211 220 232 246   0]
 [  3 202 228 224 221 211 211 214 205 205 205 220 240  80 150 255 229 221
  188 154 191 210 204 209 222 228 225   0]
 [ 98 233 198 210 222 229 229 234 249 220 194 215 217 241  65  73 106 117
  168 219 221 215 217 223 223 224 229  29]
 [ 75 204 212 204 193 205 211 225 216 185 197 206 198 213 240 195 227 245
  239 223 218 212 209 222 220 221 230  67]
 [ 48 203 183 194 213 197 185 190 194 192 202 214 219 221 220 236 225 216
  199 206 186 181 177 172 181 205 206 115]
 [  0 122 219 193 179 171 183 196 204 210 213 207 211 210 200 196 194 191
```

```
  195 191 198 192 176 156 167 177 210  92]
 [  0   0  74 189 212 191 175 172 175 181 185 188 189 188 193 198 204 209
  210 210 211 188 188 194 192 216 170   0]
 [  2   0   0   0  66 200 222 237 239 242 246 243 244 221 220 193 191 179
  182 182 181 176 166 168  99  58   0   0]
 [  0   0   0   0   0   0   0  40  61  44  72  41  35   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]

First training label:
9
```

In [35]:

```python
# Check the shapes of our training sample
train_data[0].shape, train_label[0].shape
```

Out[35]:

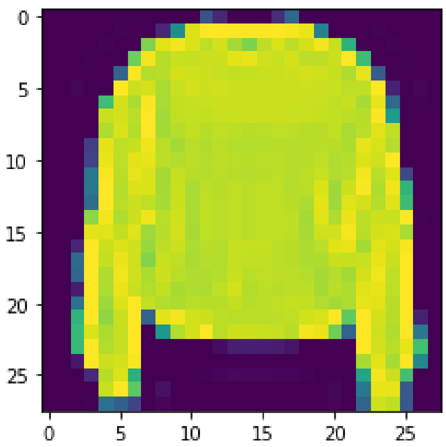
```
((28, 28), ())
```

In [36]:

```python
#Show a single sample
import matplotlib.pyplot as plt
plt.imshow(train_data[7])
```

Out[36]:

```
&lt;matplotlib.image.AxesImage at 0x7ff691bb68d0&gt;
```



In [37]:

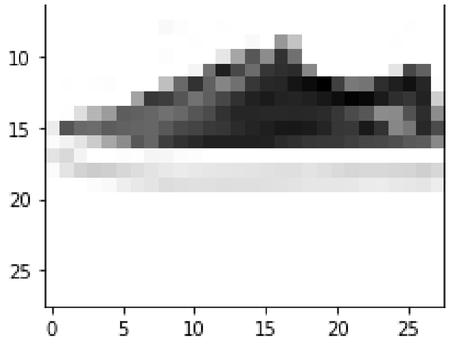
```python
# Create a small list to index onto training labels so they are human readable
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat", "Sandal", "Shirt",
"Sneaker", "Bag", "Ankle boot"]

len(class_names)
```

Out[37]:

```
10
```

In [38]:

```python
# Plot an example image and its label
index_of_choice = 131
plt.imshow(train_data[index_of_choice], cmap=plt.cm.binary)
plt.title(class_names[train_label[index_of_choice]]);
```



In [39]:

```
# Plot multiple random images of fashion MNIST
import random

plt.figure(figsize=(20, 12))
for i in range(14):
  ax = plt.subplot(2, 7, i+1)
  index_val = random.choice(range(len(train_data)))
  plt.imshow(train_data[index_val], cmap=plt.cm.binary)
  plt.title(class_names[train_label[index_val]])
  plt.axis(False)
```

Building a multi-class classification model

- **Input shape - 28 x 28**
- **Output shape = 10**
- **Loss function:**
  - **If the labels are one hot encoded, we can use the** `CategoricalCrossentropy()`
  - **If the labels are in integer form, we need to use the** `SparseCategoricalCrossentropy()`
- **Output layer activation -** `Softmax`

In [40]:

```
# Set random seed
tf.random.set_seed = 42

# 1. Create a model
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(4, activation="relu"),
    tf.keras.layers.Dense(4, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
```

```
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# 3. Fit the model
non_norm_history = model.fit(train_data, tf.one_hot(train_label, depth=10), epochs=10, v
alidation_data=(test_data, tf.one_hot(test_label, depth=10)))
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 2.4100 - accuracy: 0.102
9 - val_loss: 2.2779 - val_accuracy: 0.1144
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 2.2115 - accuracy: 0.143
1 - val_loss: 2.0544 - val_accuracy: 0.2036
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 1.7671 - accuracy: 0.291
2 - val_loss: 1.6049 - val_accuracy: 0.3291
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 1.5798 - accuracy: 0.334
3 - val_loss: 1.5635 - val_accuracy: 0.3398
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 1.5419 - accuracy: 0.343
0 - val_loss: 1.5312 - val_accuracy: 0.3483
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 1.5195 - accuracy: 0.351
6 - val_loss: 1.5113 - val_accuracy: 0.3490
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 1.5140 - accuracy: 0.353
5 - val_loss: 1.5184 - val_accuracy: 0.3567
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 1.5075 - accuracy: 0.355
9 - val_loss: 1.6631 - val_accuracy: 0.3596
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 1.5004 - accuracy: 0.358
9 - val_loss: 1.4997 - val_accuracy: 0.3553
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 1.4957 - accuracy: 0.357
8 - val_loss: 1.4931 - val_accuracy: 0.3631
```

In [41]:

```
# Check the model summary
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_4 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_12 (Dense)             (None, 4)                 3140
_________________________________________________________________
dense_13 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_14 (Dense)             (None, 10)                50
=================================================================
Total params: 3,210
Trainable params: 3,210
Non-trainable params: 0
_________________________________________________________________
```

In [42]:

```
# Check the min and max value of the train data
train_data.min(), train_data.max()
```

Out[42]:

```
(0, 255)
```

Neural network prefers data to be scaled (or nomalized), this means they like to have values between 1 and 0.

In [43]:

```
# We can get our train and test data values between 0 and 1 by dividing them by the max v
alue
train_data_norm = train_data / 255.0
test_data_norm = test_data / 255.0

train_data_norm.min(), train_data_norm.max()
```

Out[43]:

```
(0.0, 1.0)
```

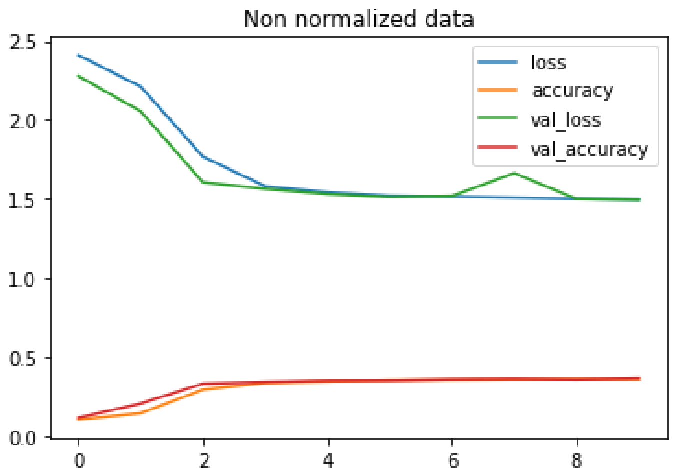
In [44]:

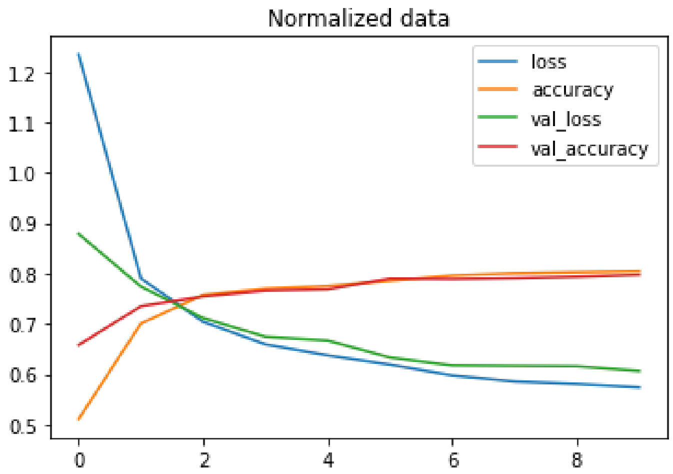
```
# Now that our data is normalized, we will build another model to find patterns in it.
# Set random seed
tf.random.seed = 42

# Create a model
model_2 = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(4, activation="relu"),
    tf.keras.layers.Dense(4, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

# Compile the model
model_2.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])

# Fit the model
norm_history = model_2.fit(train_data_norm, tf.one_hot(train_label, depth=10), epochs=10
,
                           validation_data=(test_data_norm, tf.one_hot(test_label, depth
=10)))
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 1.2346 - accuracy: 0.510
8 - val_loss: 0.8784 - val_accuracy: 0.6579
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7902 - accuracy: 0.700
6 - val_loss: 0.7740 - val_accuracy: 0.7349
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7040 - accuracy: 0.757
4 - val_loss: 0.7115 - val_accuracy: 0.7541
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6593 - accuracy: 0.769
7 - val_loss: 0.6743 - val_accuracy: 0.7660
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6373 - accuracy: 0.775
2 - val_loss: 0.6669 - val_accuracy: 0.7685
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6189 - accuracy: 0.785
4 - val_loss: 0.6331 - val_accuracy: 0.7898
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5975 - accuracy: 0.796
1 - val_loss: 0.6173 - val_accuracy: 0.7890
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5856 - accuracy: 0.800
4 - val_loss: 0.6164 - val_accuracy: 0.7907
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5807 - accuracy: 0.802
6 - val_loss: 0.6156 - val_accuracy: 0.7934
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.5740 - accuracy: 0.804
6 - val_loss: 0.6064 - val_accuracy: 0.7975
```

In [45]:

```
import pandas as pd
# Plot the non normalized loss curve
pd.DataFrame(non_norm_history.history).plot(title="Non normalized data")
# Plot the normalized loss curve
pd.DataFrame(norm_history.history).plot(title="Normalized data");
```





Finding the idea learning rate

In [46]:

```
# Set the random seed
tf.random.set_seed = 42

# Create a model
model_3 = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(4, activation="relu"),
    tf.keras.layers.Dense(4, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

# Compile the model
model_3.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])

# Set a learning rate scheduler
lr_scheduler = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-3*10**(epoch/20
))

# Fit the model
history_3 = model_3.fit(train_data_norm, train_label, epochs=40, callbacks=[lr_scheduler
], validation_data=(test_data_norm, test_label))
```

```
Epoch 1/40
1875/1875 [==============================] - 4s 2ms/step - loss: 1.2434 - accuracy: 0.522
```

```
1875/1875 [==============================] - 4s 2ms/step - loss: 1.2434 - accuracy: 0.522
3 - val_loss: 0.9204 - val_accuracy: 0.6508
Epoch 2/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.8492 - accuracy: 0.675
4 - val_loss: 0.8450 - val_accuracy: 0.6738
Epoch 3/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7733 - accuracy: 0.707
1 - val_loss: 0.7649 - val_accuracy: 0.7165
Epoch 4/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7216 - accuracy: 0.735
8 - val_loss: 0.7204 - val_accuracy: 0.7416
Epoch 5/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6751 - accuracy: 0.755
1 - val_loss: 0.6923 - val_accuracy: 0.7594
Epoch 6/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6465 - accuracy: 0.766
1 - val_loss: 0.6638 - val_accuracy: 0.7694
Epoch 7/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6335 - accuracy: 0.768
7 - val_loss: 0.6953 - val_accuracy: 0.7460
Epoch 8/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6239 - accuracy: 0.771
9 - val_loss: 0.6552 - val_accuracy: 0.7653
Epoch 9/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6217 - accuracy: 0.771
7 - val_loss: 0.6494 - val_accuracy: 0.7625
Epoch 10/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6189 - accuracy: 0.773
5 - val_loss: 0.6678 - val_accuracy: 0.7513
Epoch 11/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6150 - accuracy: 0.772
8 - val_loss: 0.6670 - val_accuracy: 0.7588
Epoch 12/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6147 - accuracy: 0.775
0 - val_loss: 0.6419 - val_accuracy: 0.7677
Epoch 13/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6167 - accuracy: 0.772
8 - val_loss: 0.6395 - val_accuracy: 0.7697
Epoch 14/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6145 - accuracy: 0.773
5 - val_loss: 0.6564 - val_accuracy: 0.7622
Epoch 15/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6198 - accuracy: 0.770
6 - val_loss: 0.6739 - val_accuracy: 0.7575
Epoch 16/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6203 - accuracy: 0.769
6 - val_loss: 0.6401 - val_accuracy: 0.7684
Epoch 17/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6184 - accuracy: 0.771
7 - val_loss: 0.6526 - val_accuracy: 0.7670
Epoch 18/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6216 - accuracy: 0.769
1 - val_loss: 0.6538 - val_accuracy: 0.7630
Epoch 19/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6252 - accuracy: 0.768
2 - val_loss: 0.6845 - val_accuracy: 0.7492
Epoch 20/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6305 - accuracy: 0.764
9 - val_loss: 0.6472 - val_accuracy: 0.7631
Epoch 21/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6320 - accuracy: 0.765
6 - val_loss: 0.6591 - val_accuracy: 0.7645
Epoch 22/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6335 - accuracy: 0.764
9 - val_loss: 0.6370 - val_accuracy: 0.7686
Epoch 23/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6379 - accuracy: 0.766
3 - val_loss: 0.7210 - val_accuracy: 0.7341
Epoch 24/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6428 - accuracy: 0.763
8 - val_loss: 0.7063 - val_accuracy: 0.7439
Epoch 25/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6481 - accuracy: 0.763
```

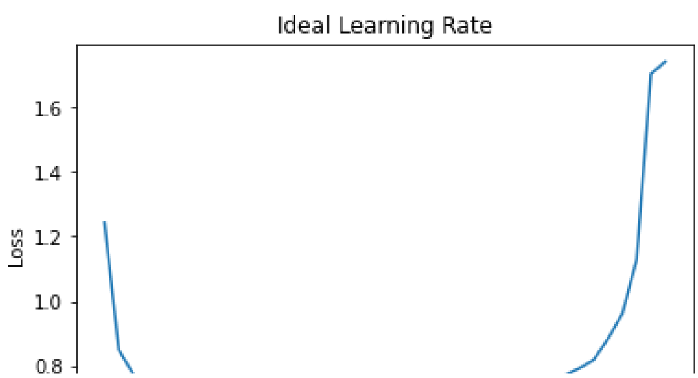
```
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6481 - accuracy: 0.763
5 - val_loss: 0.6535 - val_accuracy: 0.7694
Epoch 26/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6530 - accuracy: 0.758
0 - val_loss: 0.7231 - val_accuracy: 0.7291
Epoch 27/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6646 - accuracy: 0.751
6 - val_loss: 0.6961 - val_accuracy: 0.7441
Epoch 28/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6782 - accuracy: 0.745
2 - val_loss: 0.7203 - val_accuracy: 0.7315
Epoch 29/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6889 - accuracy: 0.736
7 - val_loss: 0.6825 - val_accuracy: 0.7323
Epoch 30/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7152 - accuracy: 0.730
3 - val_loss: 0.6971 - val_accuracy: 0.7482
Epoch 31/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7130 - accuracy: 0.730
4 - val_loss: 0.7185 - val_accuracy: 0.7111
Epoch 32/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7401 - accuracy: 0.721
9 - val_loss: 0.7130 - val_accuracy: 0.7293
Epoch 33/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7697 - accuracy: 0.712
2 - val_loss: 0.7178 - val_accuracy: 0.7260
Epoch 34/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7914 - accuracy: 0.710
0 - val_loss: 0.8834 - val_accuracy: 0.6922
Epoch 35/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.8171 - accuracy: 0.700
5 - val_loss: 0.7531 - val_accuracy: 0.7224
Epoch 36/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.8826 - accuracy: 0.677
4 - val_loss: 0.9390 - val_accuracy: 0.6725
Epoch 37/40
1875/1875 [==============================] - 3s 2ms/step - loss: 0.9598 - accuracy: 0.649
8 - val_loss: 0.9803 - val_accuracy: 0.7015
Epoch 38/40
1875/1875 [==============================] - 3s 2ms/step - loss: 1.1281 - accuracy: 0.579
8 - val_loss: 1.3806 - val_accuracy: 0.4022
Epoch 39/40
1875/1875 [==============================] - 3s 2ms/step - loss: 1.7049 - accuracy: 0.245
0 - val_loss: 1.7351 - val_accuracy: 0.1987
Epoch 40/40
1875/1875 [==============================] - 3s 2ms/step - loss: 1.7420 - accuracy: 0.202
1 - val_loss: 1.7198 - val_accuracy: 0.1996
```

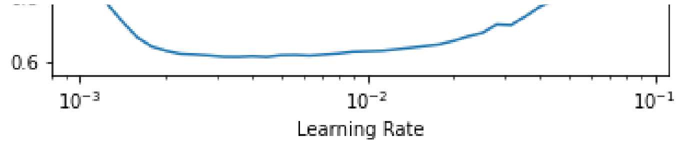
In [47]:

```
# Plot the learning rate decay curve
lrs = 1e-3*(10**(tf.range(40)/20))
plt.semilogx(lrs, history_3.history["loss"])
plt.xlabel("Learning Rate")
plt.ylabel("Loss")
plt.title("Ideal Learning Rate")
```

Out[47]:

```
Text(0.5, 1.0, 'Ideal Learning Rate')
```





In [48]:

```python
# Let's refit a model with the ideal learning rate
# Set random seed
tf.random.set_seed = 42

# Create a model
model_4 = tf.keras.Sequential([
      tf.keras.layers.Flatten(input_shape=(28, 28)),
      tf.keras.layers.Dense(4, activation="relu"),
      tf.keras.layers.Dense(4, activation="relu"),
      tf.keras.layers.Dense(10, activation="softmax")
])

# Compile the model
model_4.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                metrics=["accuracy"])

# Fit the model
history_4 = model_4.fit(train_data_norm, train_label, epochs=20, validation_data=(test_d
ata_norm, test_label))
```

```
Epoch 1/20
1875/1875 [==============================] - 4s 2ms/step - loss: 1.2169 - accuracy: 0.555
6 - val_loss: 0.8433 - val_accuracy: 0.7006
Epoch 2/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7527 - accuracy: 0.730
5 - val_loss: 0.7169 - val_accuracy: 0.7504
Epoch 3/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6748 - accuracy: 0.756
2 - val_loss: 0.6692 - val_accuracy: 0.7585
Epoch 4/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6344 - accuracy: 0.771
4 - val_loss: 0.6357 - val_accuracy: 0.7745
Epoch 5/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.6036 - accuracy: 0.783
5 - val_loss: 0.6225 - val_accuracy: 0.7821
Epoch 6/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5836 - accuracy: 0.791
8 - val_loss: 0.6201 - val_accuracy: 0.7856
Epoch 7/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5685 - accuracy: 0.799
3 - val_loss: 0.5910 - val_accuracy: 0.7969
Epoch 8/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5561 - accuracy: 0.805
6 - val_loss: 0.5834 - val_accuracy: 0.7964
Epoch 9/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5460 - accuracy: 0.808
6 - val_loss: 0.6083 - val_accuracy: 0.7833
Epoch 10/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5394 - accuracy: 0.810
9 - val_loss: 0.5682 - val_accuracy: 0.8053
Epoch 11/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5311 - accuracy: 0.814
4 - val_loss: 0.5713 - val_accuracy: 0.8012
Epoch 12/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5270 - accuracy: 0.816
1 - val_loss: 0.5662 - val_accuracy: 0.8027
Epoch 13/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5225 - accuracy: 0.818
1 - val_loss: 0.5623 - val_accuracy: 0.8051
Epoch 14/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5193 - accuracy: 0.818
4 - val_loss: 0.5701 - val_accuracy: 0.8008
Epoch 15/20
```

```
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5162 - accuracy: 0.821
0 - val_loss: 0.5690 - val_accuracy: 0.8031
Epoch 16/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5111 - accuracy: 0.821
8 - val_loss: 0.5544 - val_accuracy: 0.8088
Epoch 17/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5086 - accuracy: 0.823
3 - val_loss: 0.5524 - val_accuracy: 0.8122
Epoch 18/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5086 - accuracy: 0.824
6 - val_loss: 0.5473 - val_accuracy: 0.8128
Epoch 19/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5052 - accuracy: 0.826
1 - val_loss: 0.5934 - val_accuracy: 0.7957
Epoch 20/20
1875/1875 [==============================] - 3s 2ms/step - loss: 0.5031 - accuracy: 0.826
0 - val_loss: 0.5547 - val_accuracy: 0.8103
```

Evaluating out multi-class classification model

In [49]:

```
# Create a confusion matrix

import itertools
from sklearn.metrics import confusion_matrix
import numpy as np

figsize = (10, 10)

def make_confusion_matrix(y_true, y_pred, classes=None, figsize=(10, 10), text_size=15):

  # Create the confusion matrix
  cm = confusion_matrix(y_true,y_pred)
  cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis] # normalize it
  n_classes = cm.shape[0]

  # Let's prettify it
  fig, ax = plt.subplots(figsize=figsize)
  # Create a matrix plot
  cax = ax.matshow(cm, cmap=plt.cm.Blues) # https://matplotlib.org/3.2.0/api/_as_gen/mat
plotlib.axes.Axes.matshow.html
  fig.colorbar(cax)

  # Set labels to be classes
  if classes:
    labels = classes
  else:
    labels = np.arange(cm.shape[0])

  # Label the axes
  ax.set(title="Confusion Matrix",
         xlabel="Predicted label",
         ylabel="True label",
         xticks=np.arange(n_classes),
         yticks=np.arange(n_classes),
         xticklabels=labels,
         yticklabels=labels)

  # Set x-axis labels to bottom
  ax.xaxis.set_label_position("bottom")
  ax.xaxis.tick_bottom()

  # Adjust label size
  ax.xaxis.label.set_size(text_size)
  ax.yaxis.label.set_size(text_size)
  ax.title.set_size(text_size)

  # Set threshold for different colors
  threshold = (cm.max() + cm.min()) / 2.
```

```
  # Plot the text on each cell
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, f"{cm[i, j]} ({cm_norm[i, j]*100:.1f}%)",
             horizontalalignment="center",
             color="white" if cm[i, j] > threshold else "black",
             size=15)
```

In [50]:

```
# Make some predicts with our model
y_probs = model_4.predict(test_data_norm) # Predicting prediction probabilities

y_probs[:5]
```

Out[50]:

```
array([[4.8882468e-04, 1.4429985e-04, 1.0241910e-04, 1.6993914e-05,
         1.2099149e-06, 6.2050700e-02, 3.4417110e-04, 1.4220557e-01,
         3.2813565e-05, 7.9461300e-01],
        [1.1718890e-03, 1.9092899e-03, 5.6140381e-01, 3.4181033e-03,
         3.7955901e-01, 3.9510212e-25, 5.0771408e-02, 4.0166852e-18,
         1.7665674e-03, 8.6277345e-21],
        [3.9860523e-05, 9.9986935e-01, 8.5994307e-06, 8.1936189e-05,
         2.2922793e-08, 3.5317448e-34, 2.9330545e-07, 7.0361155e-23,
         6.9850825e-10, 1.9067036e-25],
        [6.8685898e-05, 9.9966955e-01, 6.4473529e-06, 2.5496993e-04,
         1.2432023e-08, 4.6383071e-29, 3.3592326e-07, 2.8634728e-20,
         5.8803407e-09, 2.2923197e-23],
        [1.5579122e-01, 2.2018240e-03, 5.1115226e-02, 2.3586694e-02,
         6.5345675e-02, 1.9107099e-12, 7.0152622e-01, 3.8570411e-11,
         4.3322917e-04, 1.9612645e-10]], dtype=float32)
```

In [51]:

```
# Convert all our prediction probabilities into integers
y_preds = y_probs.argmax(axis=1)

# View first ten predictions
y_preds[:10]
```

Out[51]:

```
array([9, 2, 1, 1, 6, 1, 4, 6, 5, 7])
```

In [52]:

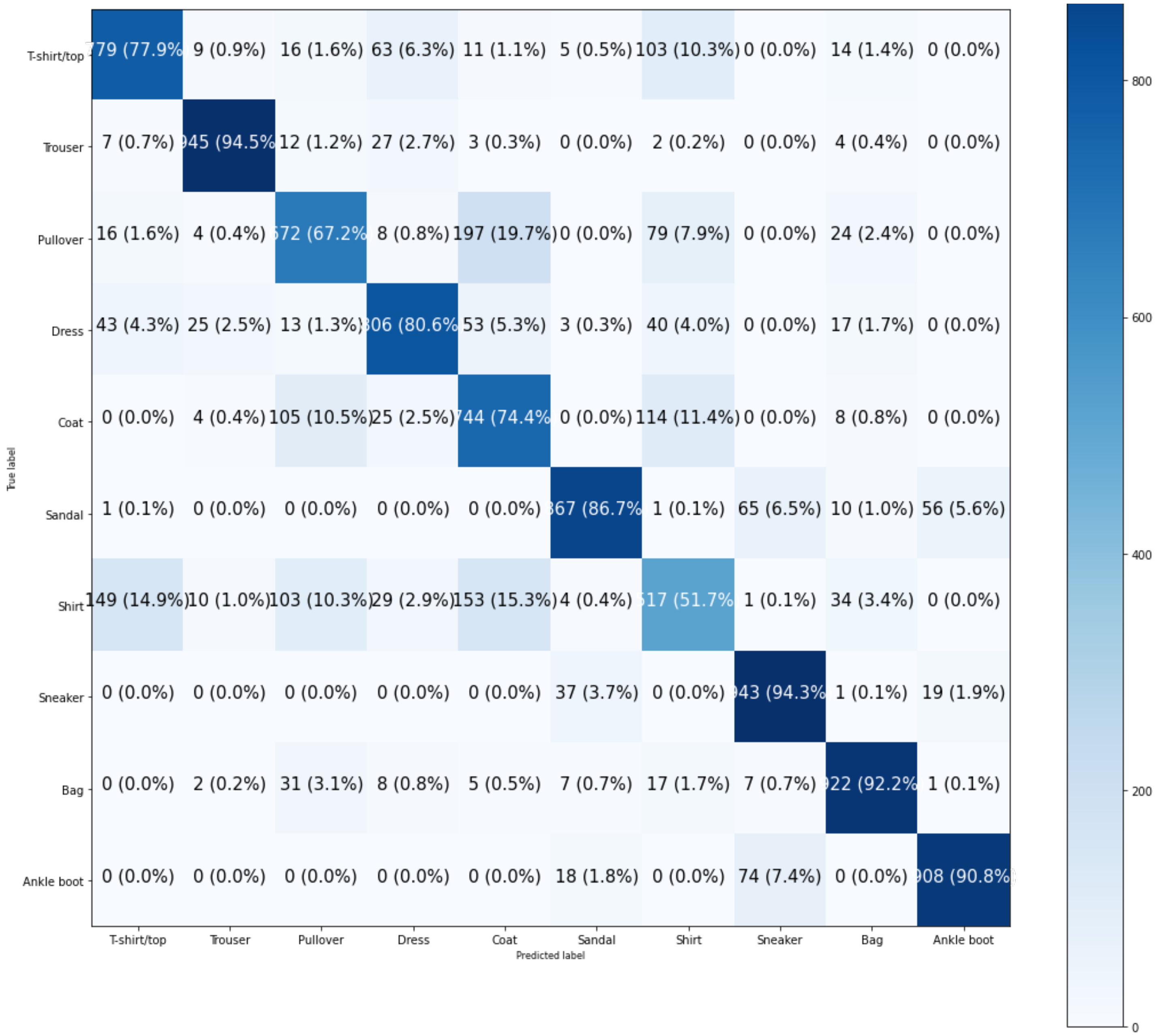
```
# Make a simple confusion_matrix
confusion_matrix(y_true=test_label, y_pred=y_preds)
```

Out[52]:

```
array([[779,   9,  16,  63,  11,   5, 103,   0,  14,   0],
       [  7, 945,  12,  27,   3,   0,   2,   0,   4,   0],
       [ 16,   4, 672,   8, 197,   0,  79,   0,  24,   0],
       [ 43,  25,  13, 806,  53,   3,  40,   0,  17,   0],
       [  0,   4, 105,  25, 744,   0, 114,   0,   8,   0],
       [  1,   0,   0,   0,   0, 867,   1,  65,  10,  56],
       [149,  10, 103,  29, 153,   4, 517,   1,  34,   0],
       [  0,   0,   0,   0,   0,  37,   0, 943,   1,  19],
       [  0,   2,  31,   8,   5,   7,  17,   7, 922,   1],
       [  0,   0,   0,   0,   0,  18,   0,  74,   0, 908]])
```

In [53]:

```
# Make a more visual confusion matrix
make_confusion_matrix(y_true=test_label, y_pred=y_preds, classes=class_names, figsize=(1
8,18), text_size=8)
```



Let's make a function to:

- **Plot an image**
- **Make prediction on the image**
- **Label the plot with truth label and predicted label**

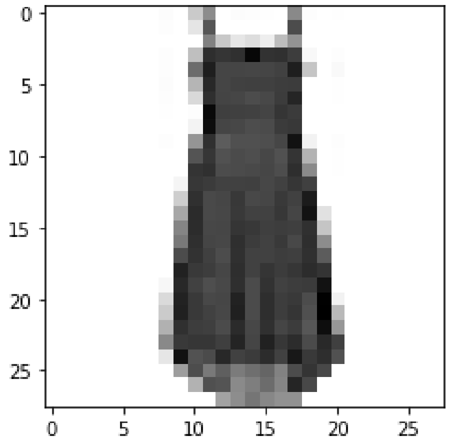
In [54]:

```
import random

def plot_random_image(model, images, true_labels, classes):
  """
  Picks a random image, plots it with truth and predicted label
  """
  # Set a random integer
  i = random.randint(0, len(images))

  # Predict on the model
  target_image = images[i]
  pred_probs = model.predict(target_image.reshape(1, 28, 28) )
  pred_label = classes[pred_probs.argmax()]
  true_label = classes[true_labels[i]]

  # Plot the image
  plt.imshow(target_image, cmap=plt.cm.binary)

  # Change the color of the text according to right or wrong prediction
```

```
  if pred_label == true_label:
    color="green"
  else:
    color="red"

  # Add xlabel information
  plt.xlabel(f"Predicted: {pred_label}, Actual: {true_label}", color=color)
```

In [55]:

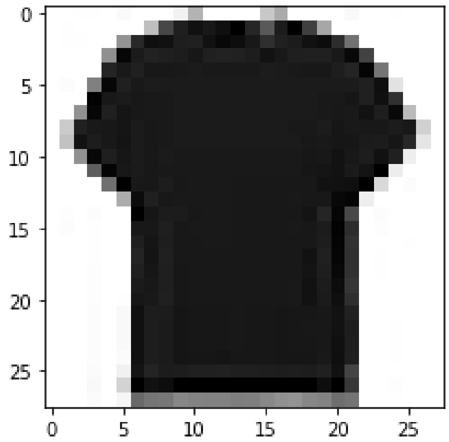
```
# Check a random image and prediction
plot_random_image(model=model_4, images=test_data_norm, true_labels=test_label, classes=
class_names)
```



Predicted: Dress, Actual: Dress

In [56]:

```
# Check again a random image and prediction
plot_random_image(model=model_4, images=test_data_norm, true_labels=test_label, classes=
class_names)
```



Predicted: T-shirt/top, Actual: Shirt

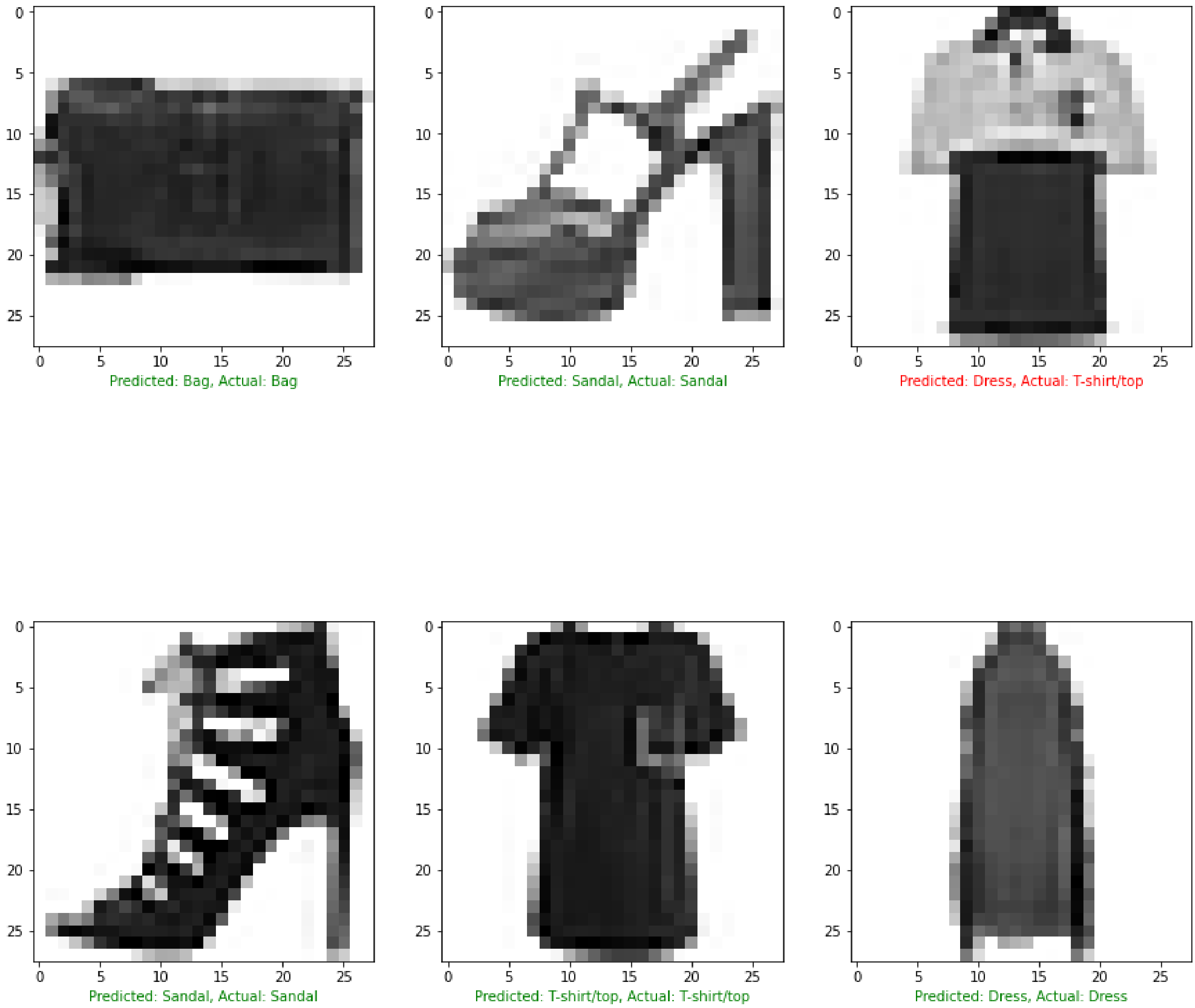
In [60]:

```
# Create a function to view multiple random predictions
def plot_multiple_random_preds(model, images, true_labels, classes):

  plt.figure(figsize=(15,15))
  for i in range(6):
    ax = plt.subplot(2, 3, i+1)
    # Set a random integer
    i = random.randint(0, len(images))

    # Predict on the model
    target_image = images[i]
    pred_probs = model.predict(target_image.reshape(1, 28, 28) )
    pred_label = classes[pred_probs.argmax()]
    true_label = classes[true_labels[i]]

    # Plot the image
```

```python
    plt.imshow(target_image, cmap=plt.cm.binary)

    # Change the color of the text according to right or wrong prediction
    if pred_label == true_label:
      color="green"
    else:
      color="red"

    # Add xlabel information
    plt.xlabel(f"Predicted: {pred_label}, Actual: {true_label}", color=color)
```

In [61]:

```python
# Check random images and prediction
plot_multiple_random_preds(model=model_4, images=test_data_norm, true_labels=test_label,
classes=class_names)
```



What patterns are our model learning

In [63]:

```python
# Check the layers of last model
model_4.layers
```

Out[63]:

```
[<keras.layers.core.Flatten at 0x7ff6a2d9ca10>,
 <keras.layers.core.Dense at 0x7ff6a8a18750>,
 <keras.layers.core.Dense at 0x7ff6a84eee50>,
 <keras.layers.core.Dense at 0x7ff6a84ee990>]
```

In [65]:

```
# Extracting a particular layer
model_4.layers[0]
```

Out[65]:

```
<keras.layers.core.Flatten at 0x7ff6a2d9ca10>
```

In [70]:

```
# Get the patterns of a layer
weights, biases = model_4.layers[1].get_weights()

# Check the values and shape of weights
weights, weights.shape
```

Out[70]:

```
(array([[-0.41568333,  0.8797607 , -0.77914935,  0.04163137],
        [-0.6733711 ,  1.4912477 ,  0.06178872, -0.21465848],
        [ 0.71542835, -0.2587731 , -1.3024038 , -0.5073505 ],
        ...,
        [-0.78605264,  0.58368194, -0.15824306,  0.23683397],
        [-0.53820276,  0.73391545, -0.1619309 , -0.35920298],
        [-0.07231373,  1.3045212 , -0.25310114,  0.30887684]],
       dtype=float32), (784, 4))
```

In [72]:

```
# Now let's check the bias vector
biases, biases.shape
```

Out[72]:

```
(array([1.4785889 , 1.0795792 , 1.3908318 , 0.71767086], dtype=float32), (4,))
```

In [ ]:

```

```