In [10]:
```
# Import Libraries
import keras
import numpy as np
import matplotlib.pyplot as plt
```

In [11]:
```
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_da
print(train_labels.shape)
```

```
(60000,)
```

In [12]:
```
# Class names are not included
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

In [13]:
```
# Training Data
train_images.shape
```

Out[13]:
```
(60000, 28, 28)
```

There are 60,000 images, each images is a 28 x 28 numpy array consisting of pixel values from 0 to 255.

In [14]:
```
# Pixel Values fall between 0 to 255, We need to scale them between 0 to 1
train_images = train_images / 255.0
test_images = test_images / 255.0
```

In [15]:
```
# Verify data is in correct format
plt.figure(figsize = (10, 10))
for i in range(25):
  plt.subplot(5, 5, i + 1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(train_images[i], cmap = plt.cm.binary)
  plt.xlabel(class_names[train_labels[i]])
plt.show
```

Out[15]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

To build the model, we first configure the layers of the model and then compile the model.

In [16]:
```
# Set up Layers
model = keras.Sequential([
    keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
model.summary()
# Flatten transforms two 2D images array into 1D
# Dense layers are fully connected neural layers.
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 32)        0
 g2D)

 flatten_1 (Flatten)         (None, 5408)              0

 dense_2 (Dense)             (None, 128)               692352

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 693962 (2.65 MB)
Trainable params: 693962 (2.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [17]:

```
# Compile the model
model.compile(optimizer = 'adam', loss = keras.losses.SparseCategoricalCrossentr
```

the optimizer parameter determines the optimization algorithm used to update the model's weights, the loss parameter defines the loss function to measure the model's performance during training, and the metrics parameter specifies the evaluation metric(s) to monitor the model's performance during training and testing.

`optimizer='adam'` : This parameter specifies the optimization algorithm used during training. In this case, the Adam optimizer is used. Adam stands for Adaptive Moment Estimation and is a popular optimization algorithm that adjusts the learning rate adaptively based on the gradient.

`loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True)` : This parameter defines the loss function used to measure the difference between the predicted output and the true labels during training. SparseCategoricalCrossentropy is a loss function commonly used for multi-class classification tasks. It computes the cross-entropy loss between the predicted probabilities (after applying a softmax activation function) and the true class labels. The from_logits=True argument indicates that the model's output is not directly normalized probabilities, but logits (raw output) that need to be converted into probabilities using a softmax activation function before computing the loss.

`metrics=['accuracy']` : This parameter specifies the evaluation metric(s) used to monitor the model's performance during training and testing. In this case, the accuracy metric is used. Accuracy measures the percentage of correctly predicted labels compared to the true labels. It is a commonly used metric for classification tasks.

Logits are the raw output of the model before applying any activation function. They represent the unnormalized predictions for each class. The values of the logits can be positive, negative, or zero. The magnitude of the logits doesn't necessarily correspond to the probability or confidence of a particular class. Setting from_logits=True in the loss function indicates that the model's output has not undergone a softmax activation, which would convert the logits into normalized probabilities across all

classes. Instead, the model's logits are treated as raw predictions and are directly used in the loss calculation

In [18]:
```
# Train the model
model.fit(train_images, train_labels, epochs = 5, verbose=1)
```

```
Epoch 1/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.3928 - accurac
y: 0.8606
Epoch 2/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.2674 - accura
cy: 0.9030
Epoch 3/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.2225 - accura
cy: 0.9183
Epoch 4/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1929 - accurac
y: 0.9292
Epoch 5/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.1638 - accurac
y: 0.9388
```

Out[18]:
```
<keras.src.callbacks.History at 0x28fbfd210>
```

In [19]:
```
# Test the accuracy on test data
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose = 2)
print("\n Test accuracy = ", test_acc*100)
print("\n Test loss = ", test_loss)
```

```
313/313 - 1s - loss: 0.2526 - accuracy: 0.9142 - 502ms/epoch - 2ms/step

 Test accuracy =  91.42000079154968

 Test loss =  0.2526157796382904
```

Model is now trained, let's try making prediction on images

In [20]:
```
# Convert logits to probability values using softmax
# probability_model = keras.Sequential([model, keras.layers.Softmax()])
```

In [21]:
```
# predictions = probability_model.predict(test_images)
predictions = model.predict(test_images)
```

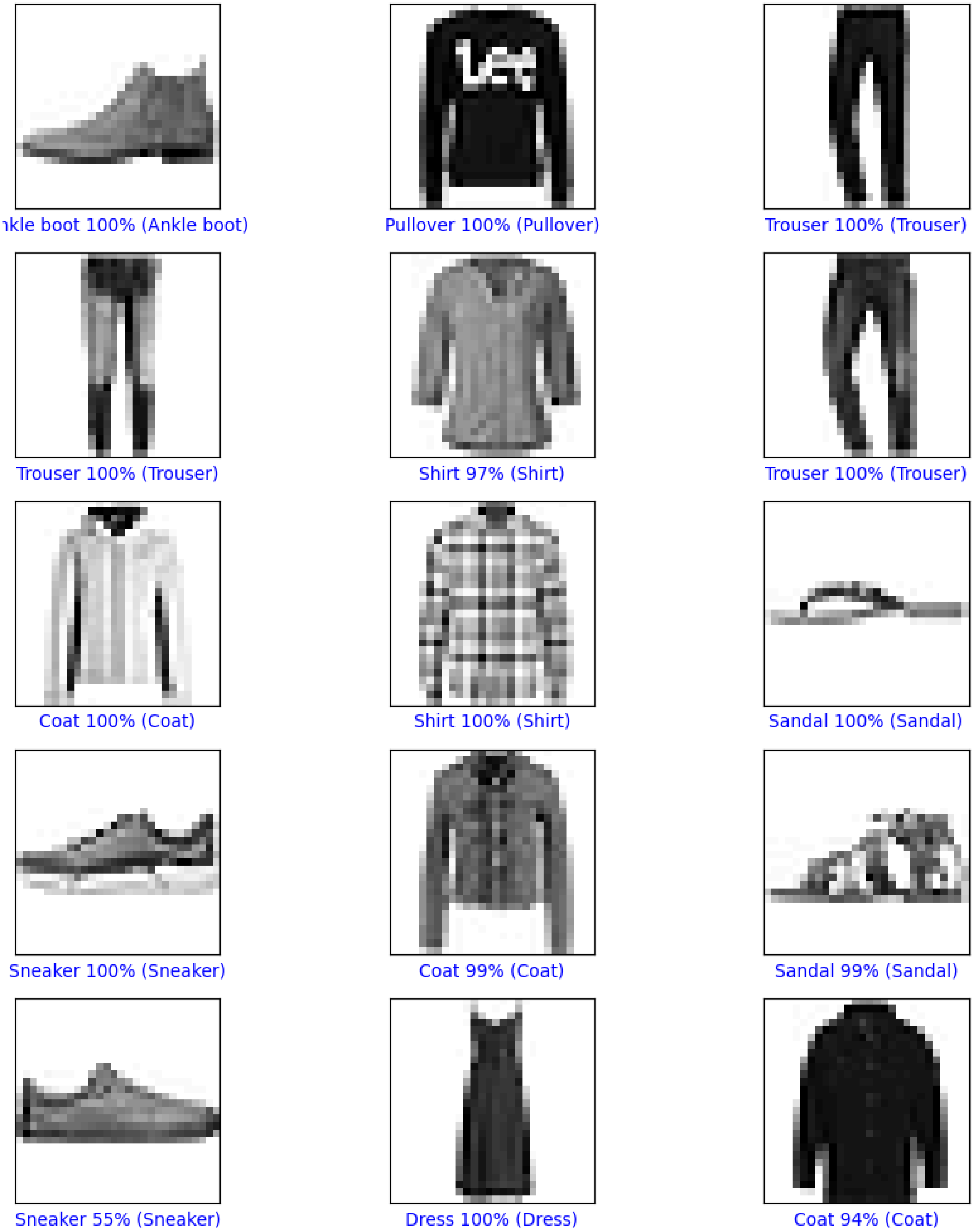
```
313/313 [==============================] - 1s 1ms/step
```

In [22]:
```
def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
```

```
                                                class_names[true_label]),
                                                color=color)
```

Model has made its predictions, let us visualize its predictions.

In [23]:
```
# Let us plot the predictions, red means incorrect while blue means correct pred
rows = 5
cols = 3
total_images = rows * cols
plt.figure(figsize = (10, 10))
for i in range(total_images):
  plt.subplot(rows, cols, i + 1)
  plot_image(i, predictions[i], test_labels, test_images)
plt.tight_layout()
plt.show()
```


nkle boot 100% (Ankle boot)
Pullover 100% (Pullover)
Trouser 100% (Trouser)
Trouser 100% (Trouser)
Shirt 97% (Shirt)
Trouser 100% (Trouser)
Coat 100% (Coat)
Shirt 100% (Shirt)
Sandal 100% (Sandal)
Sneaker 100% (Sneaker)
Coat 99% (Coat)
Sandal 99% (Sandal)
Sneaker 55% (Sneaker)
Dress 100% (Dress)
Coat 94% (Coat)

In [ ]: