Fashion MNIST Dataset comperasion Basemodel Vs AlexNet

In [1]:
```
 # importing all libraries
import itertools
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Dense, Dropout,Conv2D, MaxPooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.metrics import SparseCategoricalAccuracy
```

In [2]:
```
# setup GPU
if True:
    print("GPU device is set:", tf.config.list_physical_devices('GPU'))
else:
    print("No GPU devices found.")
```

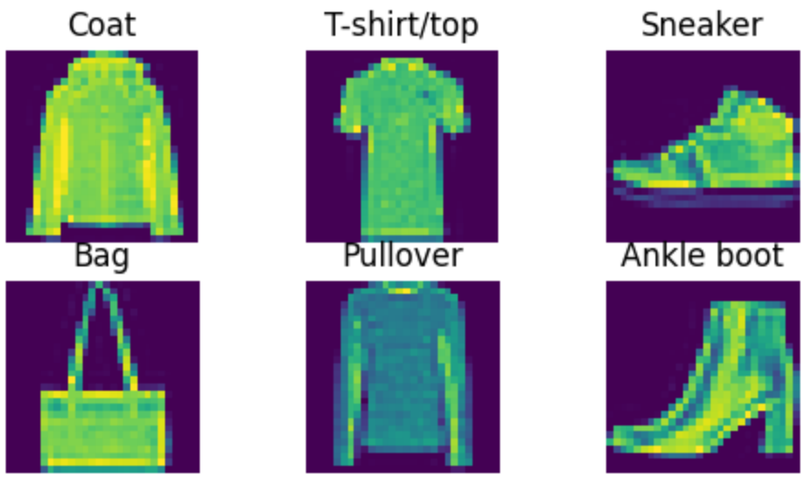
```
GPU device is set: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Load Dataset and Data preprocessing

In [3]:
```
# load the Dataset
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.fashion_mnist.load_data()
train_images = train_images / 255.0
test_images = test_images / 255.0
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
num_classes = 10
```

In [4]:

```
import random
plt.figure(figsize=(6, 3))
for i in range(6):
  ax = plt.subplot(2, 3, i + 1)
  rand_index = random.choice(range(len(train_images)))
  plt.imshow(train_images[rand_index])
  plt.title(class_names[train_labels[rand_index]])
  plt.axis(False)
```



Creating NN (BaseModel and AlexNet)

In [5]:

```
class BaseModelFashionMNIST(tf.keras.Model):
    def __init__(self, num_classes:int):
        super(BaseModelFashionMNIST, self).__init__()
        self.model = Sequential([
                                  Flatten(),
                                  Dense(100, activation='relu'),
                                  Dropout(0.2),
                                  Dense(num_classes, activation='softmax')
                                ])

    def call(self, inputs):
        return self.model(inputs)

model_0 = BaseModelFashionMNIST(num_classes)
```

In [6]:

```
class AlexNetFashionMNIST(tf.keras.Model):
    def __init__(self, num_classes):
        super(AlexNetFashionMNIST, self).__init__()
        self.conv_layers = Sequential([
                                       Conv2D(96, (3, 3), strides=(1, 1), activation='relu', input_shape=(28, 28, 1)),
                                       MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
                                       Conv2D(256, (5, 5), padding='same', activation='relu'),
                                       MaxPooling2D(pool_size=(2, 2), strides=(2, 2)),
                                       Conv2D(384, (3, 3), padding='same', activation='relu'),
                                       Conv2D(384, (3, 3), padding='same', activation='relu'),
                                       Conv2D(256, (3, 3), padding='same', activation='relu'),
                                       MaxPooling2D(pool_size=(2, 2), strides=(2, 2))
                                      ])

        self.fc_layers = Sequential([
                                      Flatten(),
                                      Dense(4096, activation='relu'),
                                      Dropout(0.5),
                                      Dense(4096, activation='relu'),
                                      Dropout(0.5),
                                      Dense(num_classes, activation='softmax')
                                     ])

    def call(self, inputs):
        x = self.conv_layers(inputs)
        x = self.fc_layers(x)
        return x

model_1 = AlexNetFashionMNIST(num_classes)
```

In [7]:

```
def compile_and_fit(model, train_images, train_labels, test_images,
                    test_labels, epochs=10, batch_size=32):
    optimizer = Adam(learning_rate=0.001)
    loss = SparseCategoricalCrossentropy()
    metrics = SparseCategoricalAccuracy()
    model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
    history = model.fit(train_images, train_labels, epochs=epochs, batch_size=batch_size,
                        validation_data=(test_images, test_labels))
    return history

def plot_loss_accuracy(loss_df, common_title):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
    ax1.plot(loss_df['loss'], color='red', label='Train Loss')
    ax1.plot(loss_df['val_loss'], color='blue', label='Validation Loss')

    ax1.set_title('Loss over epochs')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Loss')

    ax2.plot(loss_df['sparse_categorical_accuracy'], color='red', label='Train Accuracy')
    ax2.plot(loss_df['val_sparse_categorical_accuracy'], color='blue', label='Validation Accuracy')

    ax2.set_title('Accuracy over epochs')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Accuracy')
    ax2.legend()
    ax1.legend()
    fig.suptitle(common_title, fontweight='bold', fontsize=14, fontfamily='serif')
```

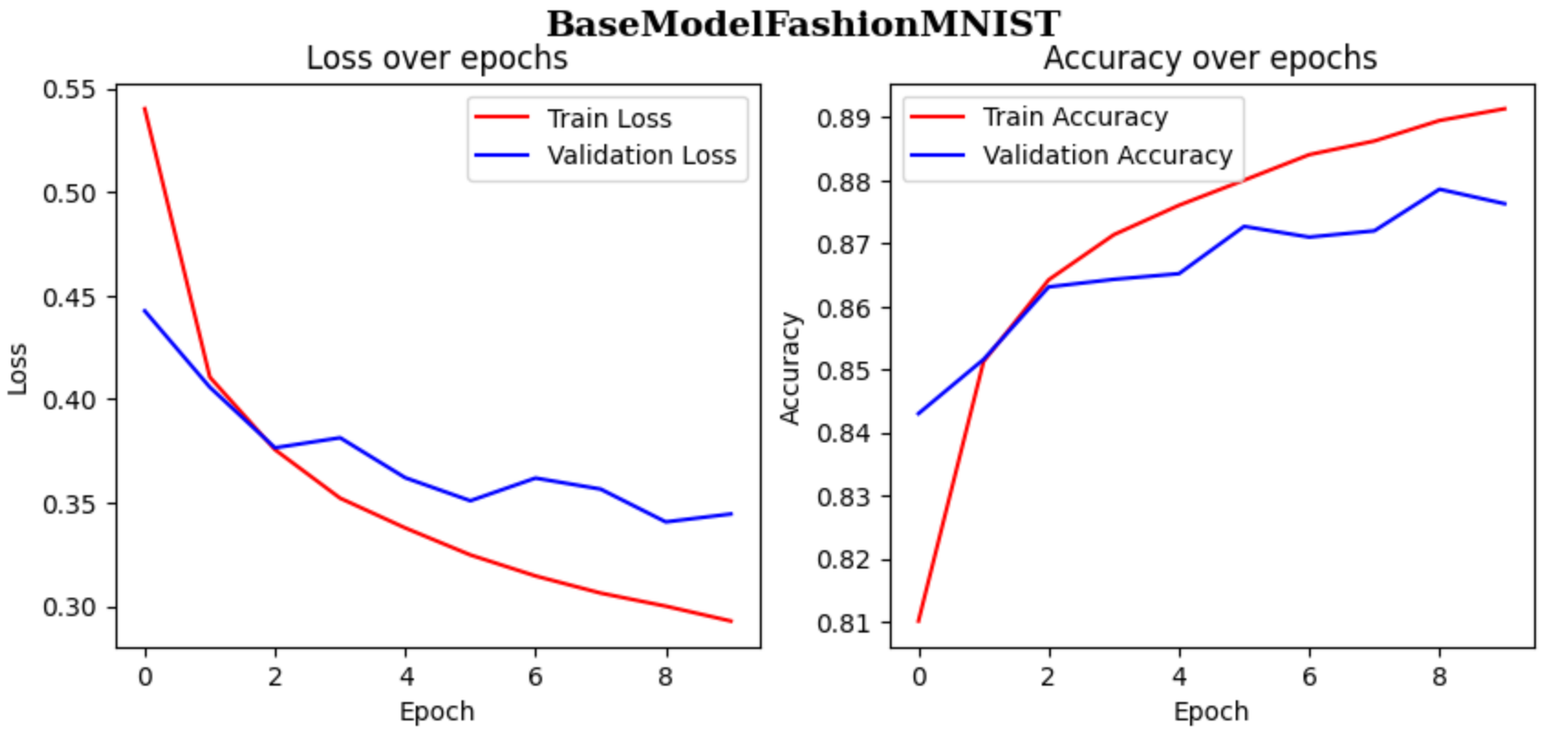
In [8]:

```
basemodel_history = compile_and_fit(model_0, train_images, train_labels, test_images,test_labels, epochs=10, batch_size=32)
basemodel_df = pd.DataFrame(basemodel_history.history)
```

```
Epoch 1/10
1875/1875 [==============================] - 12s 5ms/step - loss: 0.5402 - sparse_categorical_accuracy: 0.8101 - val_loss: 0.4426 - val_sparse_categorical_
accuracy: 0.8430
Epoch 2/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.4105 - sparse_categorical_accuracy: 0.8513 - val_loss: 0.4057 - val_sparse_categorical_
accuracy: 0.8516
Epoch 3/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.3755 - sparse_categorical_accuracy: 0.8642 - val_loss: 0.3764 - val_sparse_categorical_
accuracy: 0.8631
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3521 - sparse_categorical_accuracy: 0.8714 - val_loss: 0.3813 - val_sparse_categorical_a
ccuracy: 0.8643
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3377 - sparse_categorical_accuracy: 0.8760 - val_loss: 0.3620 - val_sparse_categorical_a
ccuracy: 0.8652
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3247 - sparse_categorical_accuracy: 0.8800 - val_loss: 0.3509 - val_sparse_categorical_a
ccuracy: 0.8727
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3145 - sparse_categorical_accuracy: 0.8841 - val_loss: 0.3618 - val_sparse_categorical_a
ccuracy: 0.8710
Epoch 8/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.3062 - sparse_categorical_accuracy: 0.8863 - val_loss: 0.3565 - val_sparse_categorical_a
ccuracy: 0.8720
Epoch 9/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2999 - sparse_categorical_accuracy: 0.8895 - val_loss: 0.3406 - val_sparse_categorical_a
ccuracy: 0.8786
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2927 - sparse_categorical_accuracy: 0.8914 - val_loss: 0.3445 - val_sparse_categorical_a
ccuracy: 0.8763
```

##plot the epochs history

In [9]:
```
1 plot_loss_accuracy(basemodel_df, common_title='BaseModelFashionMNIST')
```

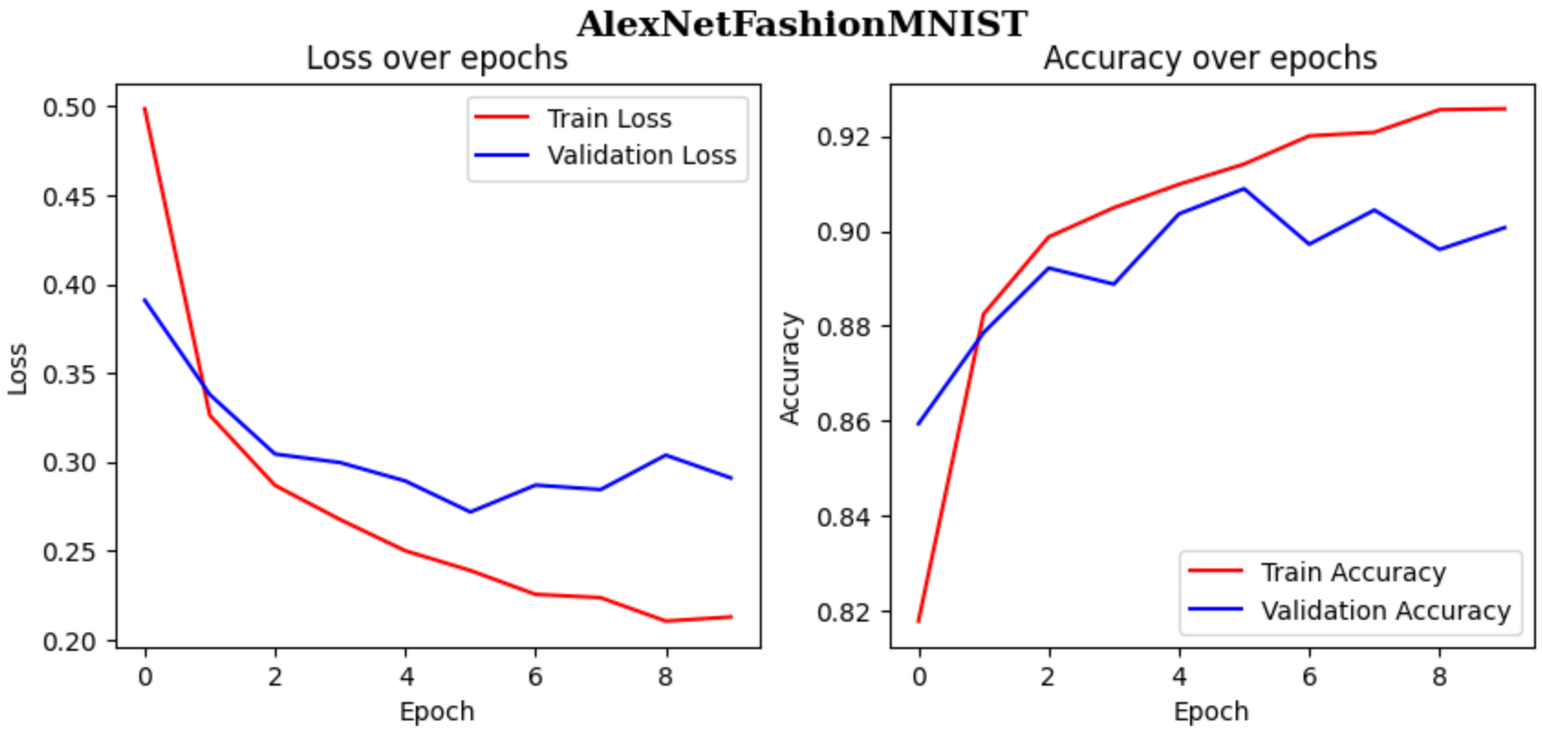


In [10]:

```
alexnet_history = compile_and_fit(model_1, train_images, train_labels, test_images,test_labels, epochs=10, batch_size=32)
alexnet_df = pd.DataFrame(alexnet_history.history)
```

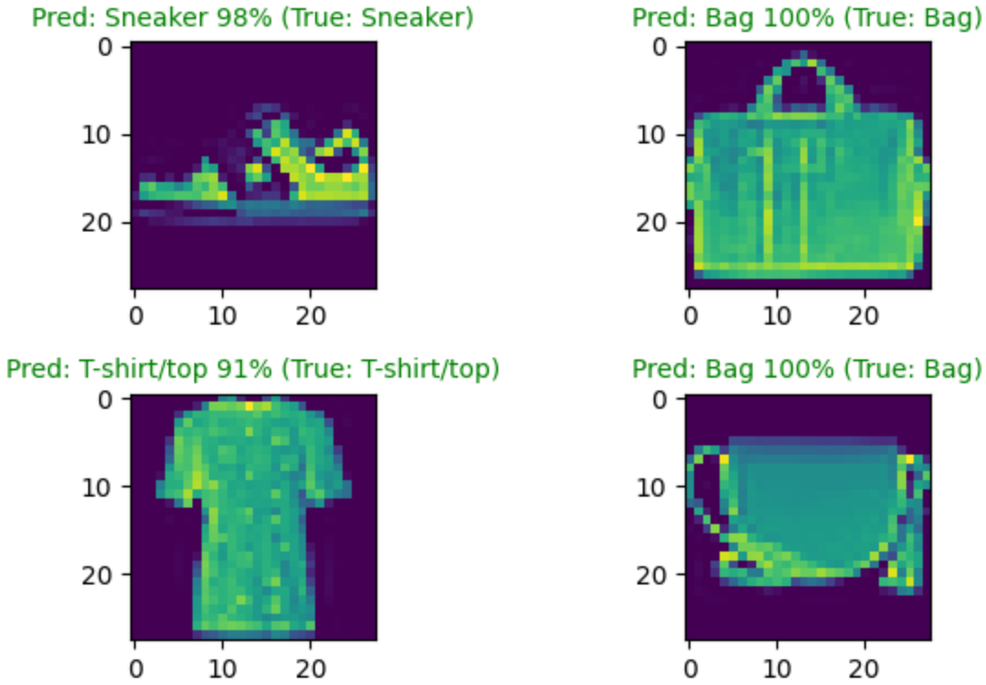
```
Epoch 1/10
1875/1875 [==============================] - 37s 17ms/step - loss: 0.4986 - sparse_categorical_accuracy: 0.8179 - val_loss: 0.3909 - val_sparse_categorical
_accuracy: 0.8594
Epoch 2/10
1875/1875 [==============================] - 33s 17ms/step - loss: 0.3261 - sparse_categorical_accuracy: 0.8825 - val_loss: 0.3377 - val_sparse_categorical
_accuracy: 0.8786
Epoch 3/10
1875/1875 [==============================] - 32s 17ms/step - loss: 0.2867 - sparse_categorical_accuracy: 0.8988 - val_loss: 0.3043 - val_sparse_categorical
_accuracy: 0.8922
Epoch 4/10
1875/1875 [==============================] - 33s 18ms/step - loss: 0.2674 - sparse_categorical_accuracy: 0.9049 - val_loss: 0.2995 - val_sparse_categorical
_accuracy: 0.8888
Epoch 5/10
1875/1875 [==============================] - 31s 17ms/step - loss: 0.2499 - sparse_categorical_accuracy: 0.9098 - val_loss: 0.2892 - val_sparse_categorical
_accuracy: 0.9036
Epoch 6/10
1875/1875 [==============================] - 33s 17ms/step - loss: 0.2388 - sparse_categorical_accuracy: 0.9141 - val_loss: 0.2717 - val_sparse_categorical
_accuracy: 0.9089
Epoch 7/10
1875/1875 [==============================] - 31s 16ms/step - loss: 0.2254 - sparse_categorical_accuracy: 0.9200 - val_loss: 0.2868 - val_sparse_categorical
_accuracy: 0.8972
Epoch 8/10
1875/1875 [==============================] - 32s 17ms/step - loss: 0.2235 - sparse_categorical_accuracy: 0.9208 - val_loss: 0.2843 - val_sparse_categorical
_accuracy: 0.9044
Epoch 9/10
1875/1875 [==============================] - 31s 17ms/step - loss: 0.2103 - sparse_categorical_accuracy: 0.9255 - val_loss: 0.3037 - val_sparse_categorical
_accuracy: 0.8961
Epoch 10/10
1875/1875 [==============================] - 32s 17ms/step - loss: 0.2126 - sparse_categorical_accuracy: 0.9257 - val_loss: 0.2909 - val_sparse_categorical
_accuracy: 0.9007
```

In [11]:
```
1 plot_loss_accuracy(alexnet_df, common_title='AlexNetFashionMNIST')
```



In [12]:

```
def plot_random_image(model, images, true_labels, classes):
    plt.figure(figsize=(7, 4))
    for i in range(4):
        ax = plt.subplot(2, 2, i + 1)
        rand_index = random.choice(range(len(test_images)))
        target_image = images[rand_index]
        pred_probs = model.predict(target_image.reshape(1, 28, 28), verbose=0)
        pred_label = classes[pred_probs.argmax()]
        true_label = classes[true_labels[rand_index]]

        plt.imshow(target_image)

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

        plt.title("Pred: {} {:2.0f}% (True: {})".format(pred_label,
                                                        100 * tf.reduce_max(pred_probs),
                                                        true_label),
                  color=color, fontsize=10)

    plt.tight_layout()
plot_random_image(model_1, test_images, test_labels, class_names)
plt.show()
```



In [13]:
```
1 y_preds = model_1.predict(test_images)
2 y_preds = y_preds.argmax(axis=1)
3 y_preds
```

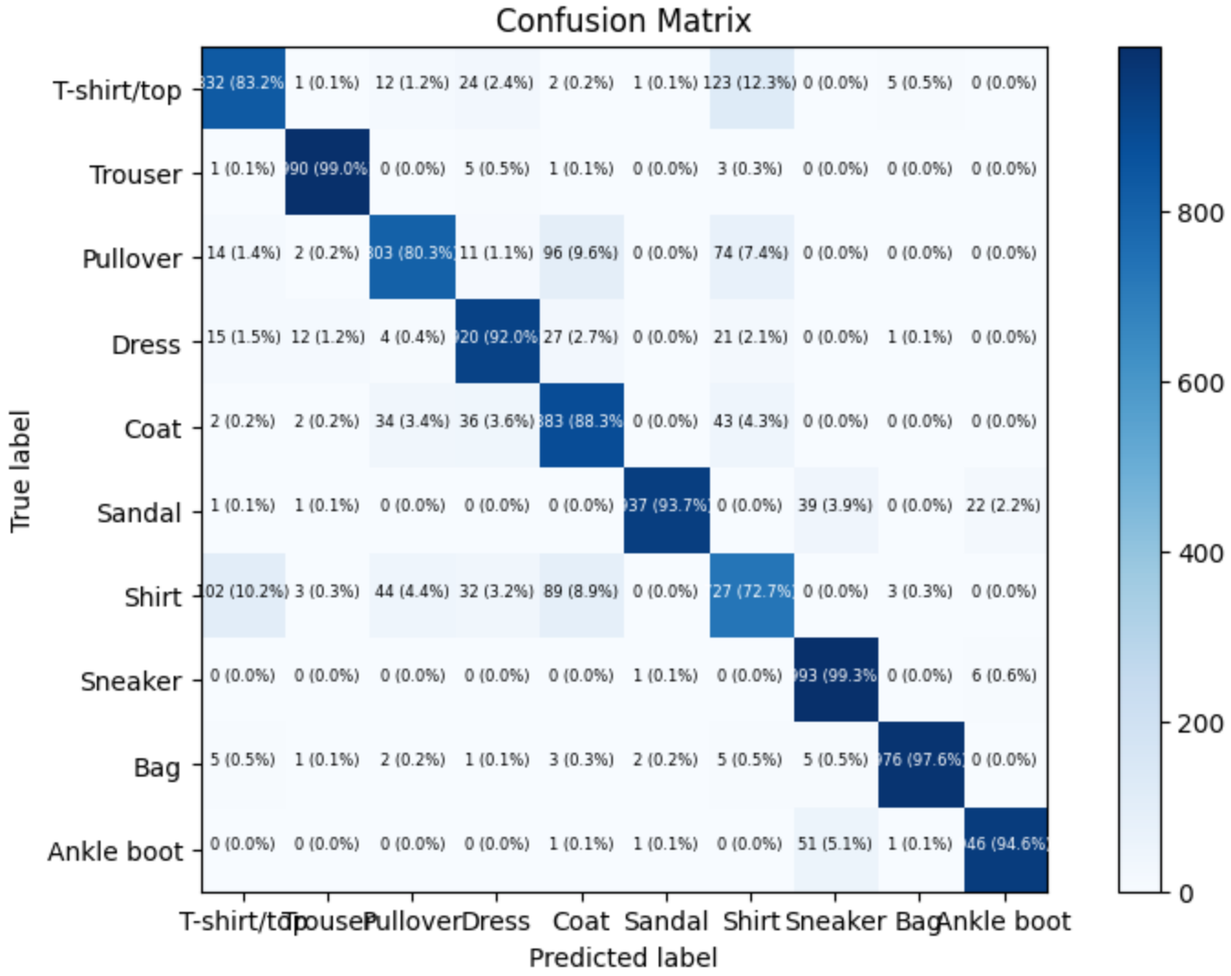
```
313/313 [==============================] - 1s 4ms/step
```

Out[13]:
```
array([9, 2, 1, ..., 8, 1, 5])
```

ploting confusion metrix

In [14]:

```python
def make_confusion_matrix(y_true, y_pred, classes=None, figsize=(10, 10), text_size=15):
    cm = confusion_matrix(y_true, y_pred)
    cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]
    n_classes = cm.shape[0]

    fig, ax = plt.subplots(figsize=figsize)
    cax = ax.matshow(cm, cmap=plt.cm.Blues)
    fig.colorbar(cax)

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

    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)

    ax.xaxis.set_label_position("bottom")
    ax.xaxis.tick_bottom()

    threshold = (cm.max() + cm.min()) / 2.

    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=text_size)

# Usage example
make_confusion_matrix(y_true=test_labels, y_pred=y_preds, classes=class_names, figsize=(10,6), text_size=6)
plt.show()

```


Confusion Matrix
T-shirt/top
Trouser
Pullover
Dress
Coat
Sandal
Shirt
Sneaker
Bag
Ankle boot
True label
Predicted label
832 (83.2%) 1 (0.1%) 12 (1.2%) 24 (2.4%) 2 (0.2%) 1 (0.1%) 123 (12.3%) 0 (0.0%) 5 (0.5%) 0 (0.0%)
1 (0.1%) 990 (99.0%) 0 (0.0%) 5 (0.5%) 1 (0.1%) 0 (0.0%) 3 (0.3%) 0 (0.0%) 0 (0.0%) 0 (0.0%)
14 (1.4%) 2 (0.2%) 803 (80.3%) 11 (1.1%) 96 (9.6%) 0 (0.0%) 74 (7.4%) 0 (0.0%) 0 (0.0%) 0 (0.0%)
15 (1.5%) 12 (1.2%) 4 (0.4%) 920 (92.0%) 27 (2.7%) 0 (0.0%) 21 (2.1%) 0 (0.0%) 1 (0.1%) 0 (0.0%)
2 (0.2%) 2 (0.2%) 34 (3.4%) 36 (3.6%) 883 (88.3%) 0 (0.0%) 43 (4.3%) 0 (0.0%) 0 (0.0%) 0 (0.0%)
1 (0.1%) 1 (0.1%) 0 (0.0%) 0 (0.0%) 0 (0.0%) 937 (93.7%) 0 (0.0%) 39 (3.9%) 0 (0.0%) 22 (2.2%)
102 (10.2%) 3 (0.3%) 44 (4.4%) 32 (3.2%) 89 (8.9%) 0 (0.0%) 727 (72.7%) 0 (0.0%) 3 (0.3%) 0 (0.0%)
0 (0.0%) 0 (0.0%) 0 (0.0%) 0 (0.0%) 0 (0.0%) 1 (0.1%) 0 (0.0%) 993 (99.3%) 0 (0.0%) 6 (0.6%)
5 (0.5%) 1 (0.1%) 2 (0.2%) 1 (0.1%) 3 (0.3%) 2 (0.2%) 5 (0.5%) 5 (0.5%) 976 (97.6%) 0 (0.0%)
0 (0.0%) 0 (0.0%) 0 (0.0%) 0 (0.0%) 1 (0.1%) 1 (0.1%) 0 (0.0%) 51 (5.1%) 1 (0.1%) 946 (94.6%)
800
600
400
200
0

Thank You 😊