In [16]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

#Load the training and testing data

train_data = pd.read_csv('C://Users//_//Downloads//archive//fashion-mnist_train.csv')
test_data = pd.read_csv('C://Users//_//Downloads//archive//fashion-mnist_test.csv')

X_train = train_data.iloc[:, 1:].values # Assuming the first column is the
y_train = train_data.iloc[:, 0].values
X_test = test_data.iloc[:, 1:].values
y_test = test_data.iloc[:, 0].values
```

In [17]:
```
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')

X_train = X_train / 255.0
X_test = X_test / 255.0

y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

In [18]:
```
# Define the model.

model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D (64, (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))
```

```
model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(Flatten())

model.add(Dense (64, activation='relu'))

model.add(Dropout(0.5))

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

```
C:\Users\bonde\anaconda3\Lib\site-packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not pass a
n `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object a
s the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [19]:
```
#Compile the model

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

In [26]:
```
#Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2, verbose=2)
```

```
Epoch 1/10
750/750 - 11s - 15ms/step - accuracy: 0.7359 - loss: 0.7245 - val_accuracy: 0.8427 - val_loss: 0.4301
Epoch 2/10
750/750 - 8s - 11ms/step - accuracy: 0.8354 - loss: 0.4624 - val_accuracy: 0.8756 - val_loss: 0.3496
Epoch 3/10
750/750 - 9s - 12ms/step - accuracy: 0.8622 - loss: 0.3911 - val_accuracy: 0.8833 - val_loss: 0.3170
Epoch 4/10
750/750 - 8s - 11ms/step - accuracy: 0.8769 - loss: 0.3479 - val_accuracy: 0.8879 - val_loss: 0.3108
Epoch 5/10
750/750 - 8s - 11ms/step - accuracy: 0.8858 - loss: 0.3228 - val_accuracy: 0.8897 - val_loss: 0.2933
Epoch 6/10
750/750 - 8s - 11ms/step - accuracy: 0.8931 - loss: 0.3007 - val_accuracy: 0.9020 - val_loss: 0.2793
Epoch 7/10
750/750 - 8s - 11ms/step - accuracy: 0.8988 - loss: 0.2844 - val_accuracy: 0.9018 - val_loss: 0.2717
Epoch 8/10
750/750 - 8s - 11ms/step - accuracy: 0.9062 - loss: 0.2639 - val_accuracy: 0.9065 - val_loss: 0.2667
Epoch 9/10
750/750 - 8s - 11ms/step - accuracy: 0.9089 - loss: 0.2505 - val_accuracy: 0.8978 - val_loss: 0.2721
Epoch 10/10
750/750 - 11s - 14ms/step - accuracy: 0.9138 - loss: 0.2374 - val_accuracy: 0.9109 - val_loss: 0.2560
```

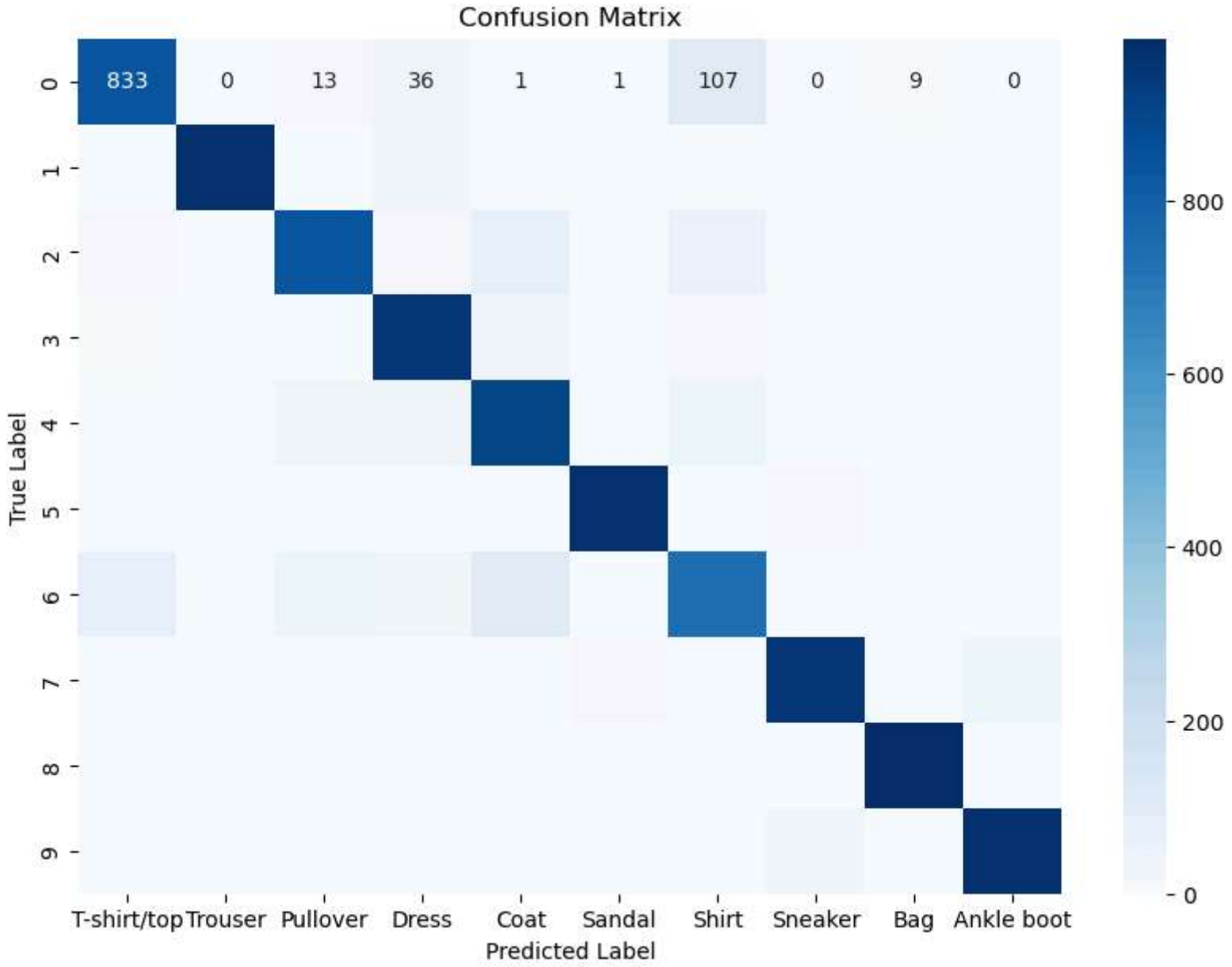
In [28]:
```
#Evaluate the model
test_loss, test_acc =model.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy: {test_acc:.2f}')
```

```
313/313 - 1s - 3ms/step - accuracy: 0.9125 - loss: 0.2458

Test accuracy: 0.91
```

In [35]:
```
#Make predictions

predictions=model.predict(X_test)

#Convert predictions and actual labels from one-hot encoding to integers

y_pred=np.argmax(predictions, axis=1)

y_true=np.argmax(y_test, axis=1)

print("Classification Report:")

print(classification_report(y_true, y_pred, target_names=['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sar
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
Classification Report:
              precision    recall  f1-score   support

 T-shirt/top       0.89      0.83      0.86      1000
     Trouser       1.00      0.97      0.99      1000
    Pullover       0.91      0.83      0.87      1000
       Dress       0.88      0.94      0.91      1000
        Coat       0.82      0.91      0.86      1000
      Sandal       0.98      0.98      0.98      1000
       Shirt       0.77      0.75      0.76      1000
     Sneaker       0.96      0.95      0.95      1000
         Bag       0.98      0.99      0.98      1000
  Ankle boot       0.95      0.97      0.96      1000

    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```

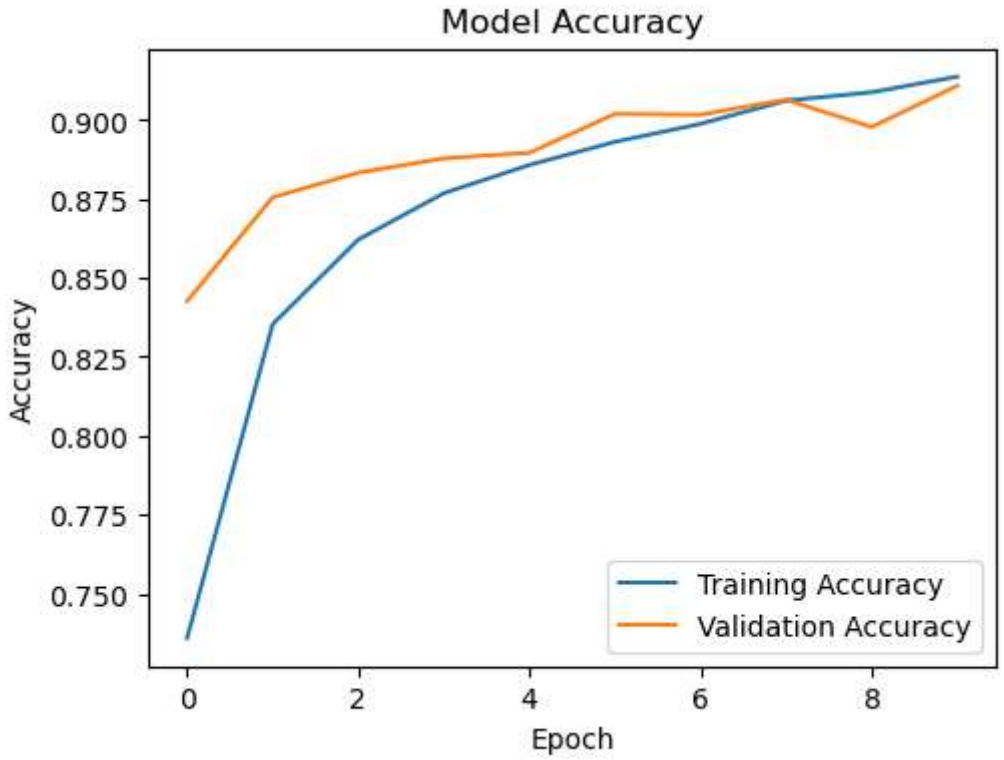
In [37]:
```
#Confusion matrix
cm=confusion_matrix(y_true, y_pred)
plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat
yticklabels=['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot

plt.xlabel('Predicted Label')
plt.ylabel('True Label')

plt.title('Confusion Matrix')

plt.show()
```



In [42]:
```
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
```

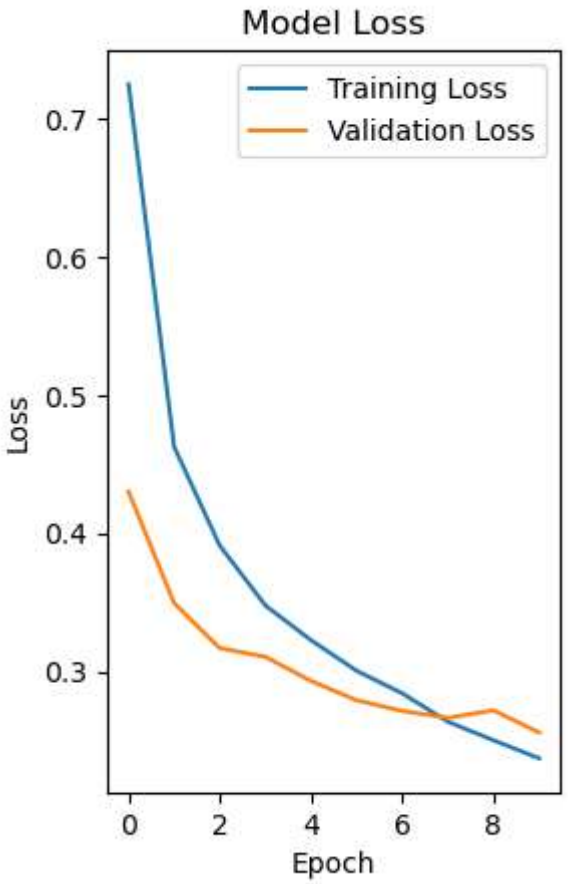
```
plt.title('Model Accuracy')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')

plt.legend(loc='lower right')
```

Out[42]: <matplotlib.legend.Legend at 0x1bd2f551a90>



In [44]:
```
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
```

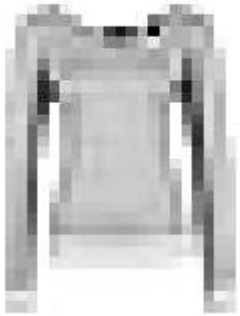
```
plt.show()
```



In [50]:
```
import matplotlib.pyplot as plt

labels = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
num_images_to_show = 7

# Number of images to display

for i in range(num_images_to_show):
    plt.figure(figsize=(2, 2))

    # Plot the image
```

```
    plt.imshow(X_test[i].reshape(28, 28), cmap=plt.cm.binary)
    plt.axis('off')  # Hide axes

    # Show the predicted and actual labels
    predicted_label = labels[y_pred[i]]
    actual_label = labels[y_true[i]]

    plt.title(f'Predicted: {predicted_label}\nActual: {actual_label}', fontsize=10)

    plt.show()
```

Predicted: Pullover
Actual: Pullover

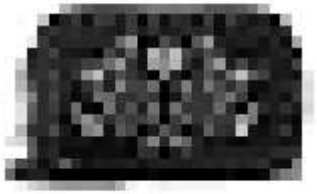


Predicted: Shirt
Actual: Pullover

Predicted: Dress
Actual: Dress

Predicted: Shirt
Actual: Pullover

Predicted: Bag
Actual: Bag



In [ ]: