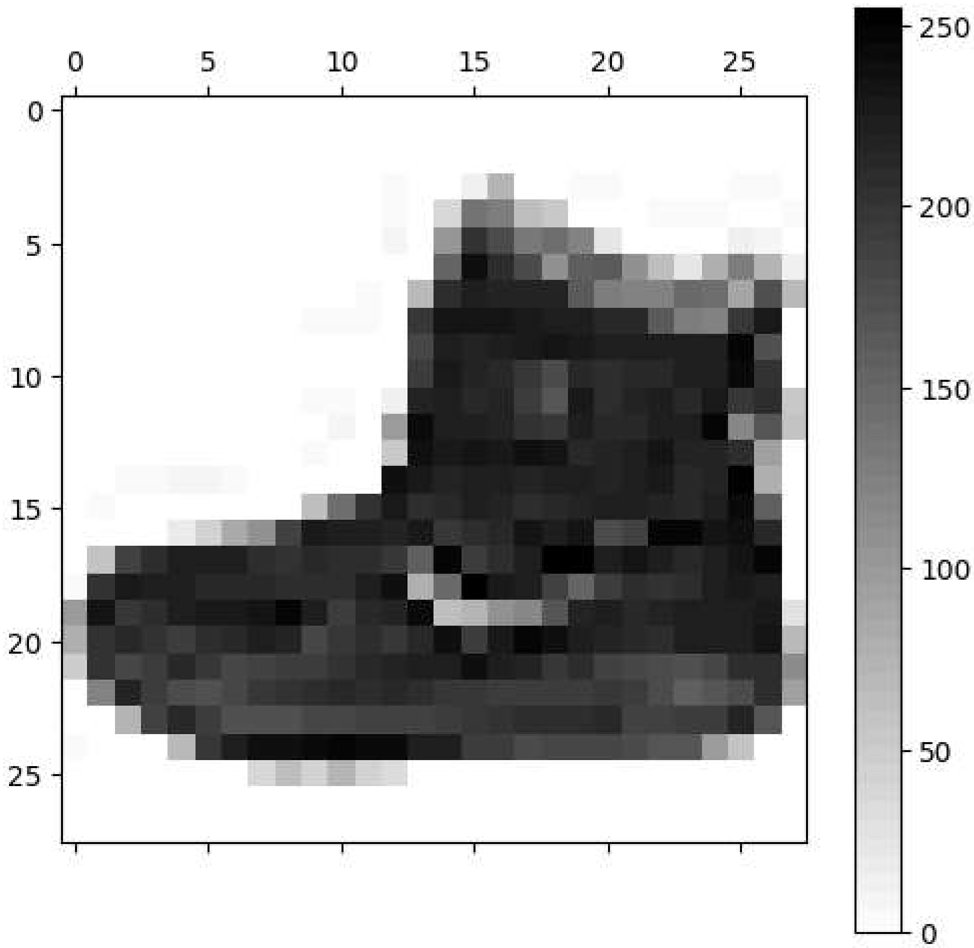
In [25]:
```
ID = 1631938
Name = 'MIN SOE HTUT'
```

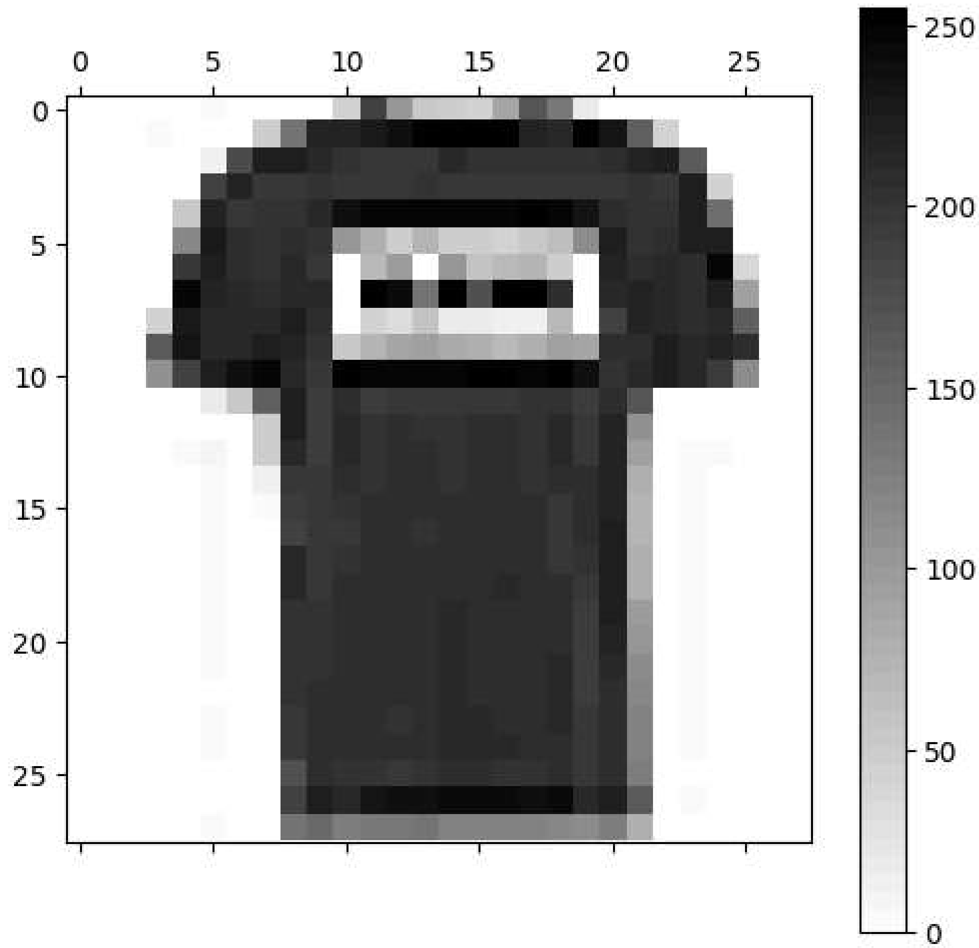
Loading the data and preparing the data

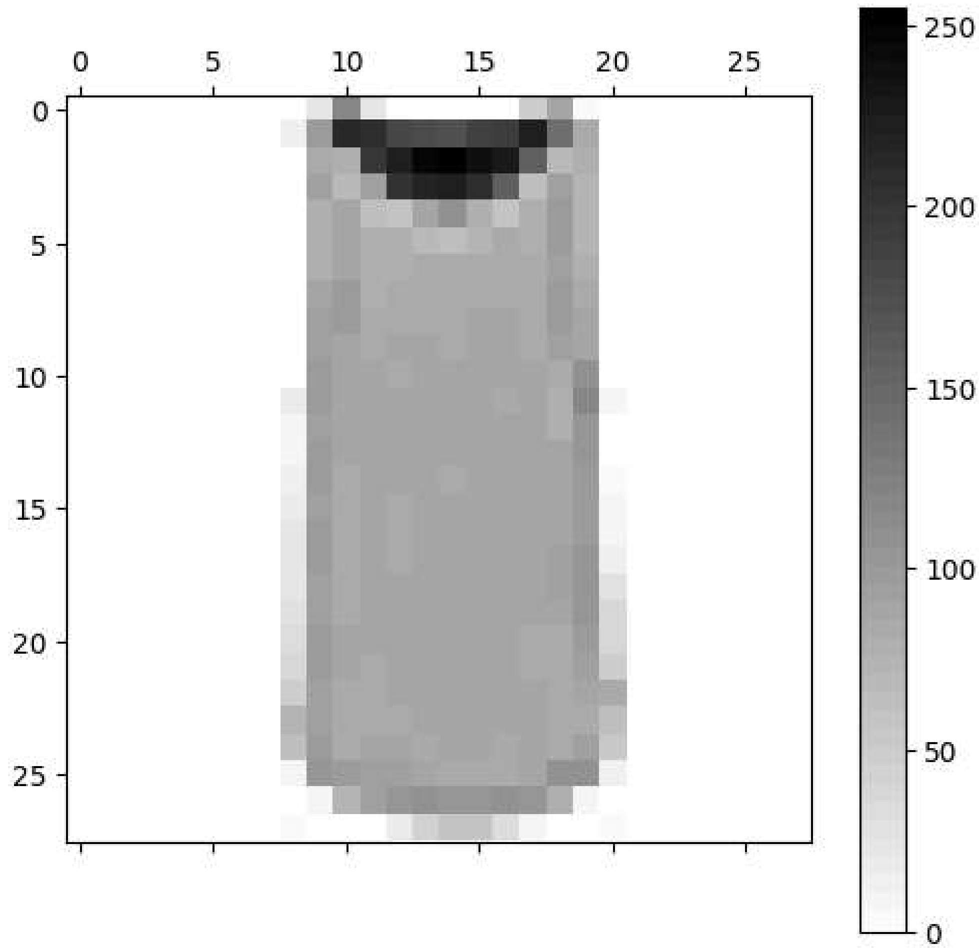
In [26]:
```
from keras.datasets import fashion_mnist
import numpy as np
import matplotlib.pyplot as plt

# Load and reshape the dataset
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
X_train = np.reshape(X_train, (-1, 784))  # Flattening 28x28 images to 784 fea
tures
X_test = np.reshape(X_test, (-1, 784))
```

Plot the first 3 images from X_train

In [27]:
```
def plot_matrix(m, target_names=None, cm=plt.cm.binary, shape=(28,28)):
    fig = plt.figure(figsize=(6,6))
    ax = fig.add_subplot(111)
    cax = ax.matshow(m.reshape(shape), cmap=cm)
    plt.colorbar(cax)
    plt.show()

# Plot the first three images
for i in range(3):
    plot_matrix(X_train[i])
```


0
5
10
15
20
25
250
200
150
100
50
0


0
5
10
15
20
25
250
200
150
100
50
0



a)

Grid Search for Hyperparameters (RandomForest and ExtraTrees)

In [28]:
```
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier

# Function to perform grid search over hyperparameters for a classifier
def grid_search(CLF, X_train, y_train, n_estimators=30):
    # Lists of hyperparameter values to try
    max_features_list = [4, 12, 'sqrt']  # Different values for the 'max_featu
res' parameter
    max_depth_list = [4, 12, None]       # Different values for the 'max_dept
h' parameter

    best_score = 0  # Initialize the best score to 0
    best_params = {}  # Dictionary to store the best hyperparameters

    # Loop over all combinations of max_features and max_depth
    for max_features in max_features_list:
        for max_depth in max_depth_list:
            # Initialize the classifier with the current hyperparameters
            clf = CLF(n_estimators=n_estimators, max_features=max_features,
                      max_depth=max_depth, oob_score=True, bootstrap=True,
                      random_state=123, n_jobs=-1)  # Enable out-of-bag score,
set random seed, and use all CPU cores

            # Train the classifier on the training data
            clf.fit(X_train, y_train)

            # Retrieve the out-of-bag score (which estimates generalization pe
rformance)
            oob_score = clf.oob_score_

            # Print current hyperparameter combination and its OOB score
            print(f"max_depth: {max_depth}, max_features: {max_features}, OOB
score: {oob_score}")

            # If the current OOB score is better than the best score seen so f
ar, update best_score and best_params
            if oob_score > best_score:
                best_score = oob_score
                best_params = {'max_depth': max_depth, 'max_features': max_fea
tures}

    # Return the best OOB score and corresponding hyperparameters
    return best_score, best_params
```

Run grid_search for RandomForestClassifier with (n_estimators=30), ExtraTreesClassifier with (n_estimators=30) and ExtraTreesClassifier with (n_estimators=90)

In [29]:
```
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier

# Grid search for RandomForestClassifier (n_estimators=30)
best_score_rf, best_params_rf = grid_search(RandomForestClassifier, X_train, y
_train, n_estimators=30)

# Grid search for ExtraTreesClassifier (n_estimators=30)
best_score_et30, best_params_et30 = grid_search(ExtraTreesClassifier, X_train,
y_train, n_estimators=30)

# Grid search for ExtraTreesClassifier (n_estimators=90)
best_score_et90, best_params_et90 = grid_search(ExtraTreesClassifier, X_train,
y_train, n_estimators=90)
```

```
max_depth: 4, max_features: 4, OOB score: 0.7210333333333333
max_depth: 12, max_features: 4, OOB score: 0.8270666666666666
max_depth: None, max_features: 4, OOB score: 0.8433
max_depth: 4, max_features: 12, OOB score: 0.7320166666666666
max_depth: 12, max_features: 12, OOB score: 0.85065
max_depth: None, max_features: 12, OOB score: 0.8580333333333333
max_depth: 4, max_features: sqrt, OOB score: 0.7313333333333333
max_depth: 12, max_features: sqrt, OOB score: 0.8568333333333333
max_depth: None, max_features: sqrt, OOB score: 0.8609833333333333
max_depth: 4, max_features: 4, OOB score: 0.6831
max_depth: 12, max_features: 4, OOB score: 0.7932166666666667
max_depth: None, max_features: 4, OOB score: 0.8259
max_depth: 4, max_features: 12, OOB score: 0.6993333333333334
max_depth: 12, max_features: 12, OOB score: 0.8219166666666666
max_depth: None, max_features: 12, OOB score: 0.8433333333333334
max_depth: 4, max_features: sqrt, OOB score: 0.7307833333333333
max_depth: 12, max_features: sqrt, OOB score: 0.84095
max_depth: None, max_features: sqrt, OOB score: 0.8522166666666666
max_depth: 4, max_features: 4, OOB score: 0.7189
max_depth: 12, max_features: 4, OOB score: 0.811
max_depth: None, max_features: 4, OOB score: 0.8514666666666667
max_depth: 4, max_features: 12, OOB score: 0.7303666666666667
max_depth: 12, max_features: 12, OOB score: 0.8343666666666667
max_depth: None, max_features: 12, OOB score: 0.8644166666666667
max_depth: 4, max_features: sqrt, OOB score: 0.7455166666666667
max_depth: 12, max_features: sqrt, OOB score: 0.8494833333333334
max_depth: None, max_features: sqrt, OOB score: 0.8727
```

Train a RandomForestClassifier with the best hyper-parameter settings as returned from grid_search

In [30]:

```
# Train RandomForest with best hyperparameters
rf_clf = RandomForestClassifier(n_estimators=30, **best_params_rf, oob_score=True, bootstrap=True, random_state=123, n_jobs=-1)
rf_clf.fit(X_train, y_train)

# Train ExtraTrees with best hyperparameters (30 estimators)
et_clf30 = ExtraTreesClassifier(n_estimators=30, **best_params_et30, oob_score=True, bootstrap=True, random_state=123, n_jobs=-1)
et_clf30.fit(X_train, y_train)

# Train ExtraTrees with best hyperparameters (90 estimators)
et_clf90 = ExtraTreesClassifier(n_estimators=90, **best_params_et90, oob_score=True, bootstrap=True, random_state=123, n_jobs=-1)
et_clf90.fit(X_train, y_train)
```

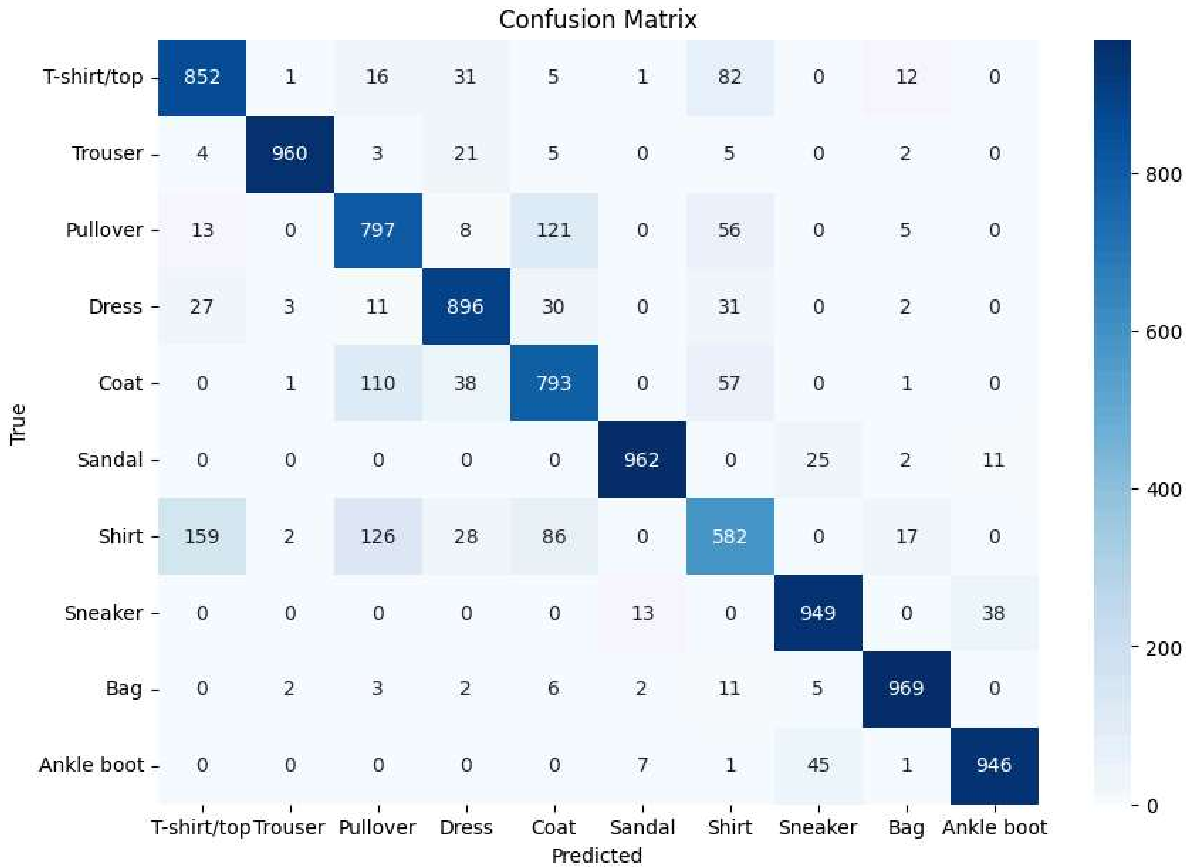
Out[30]:

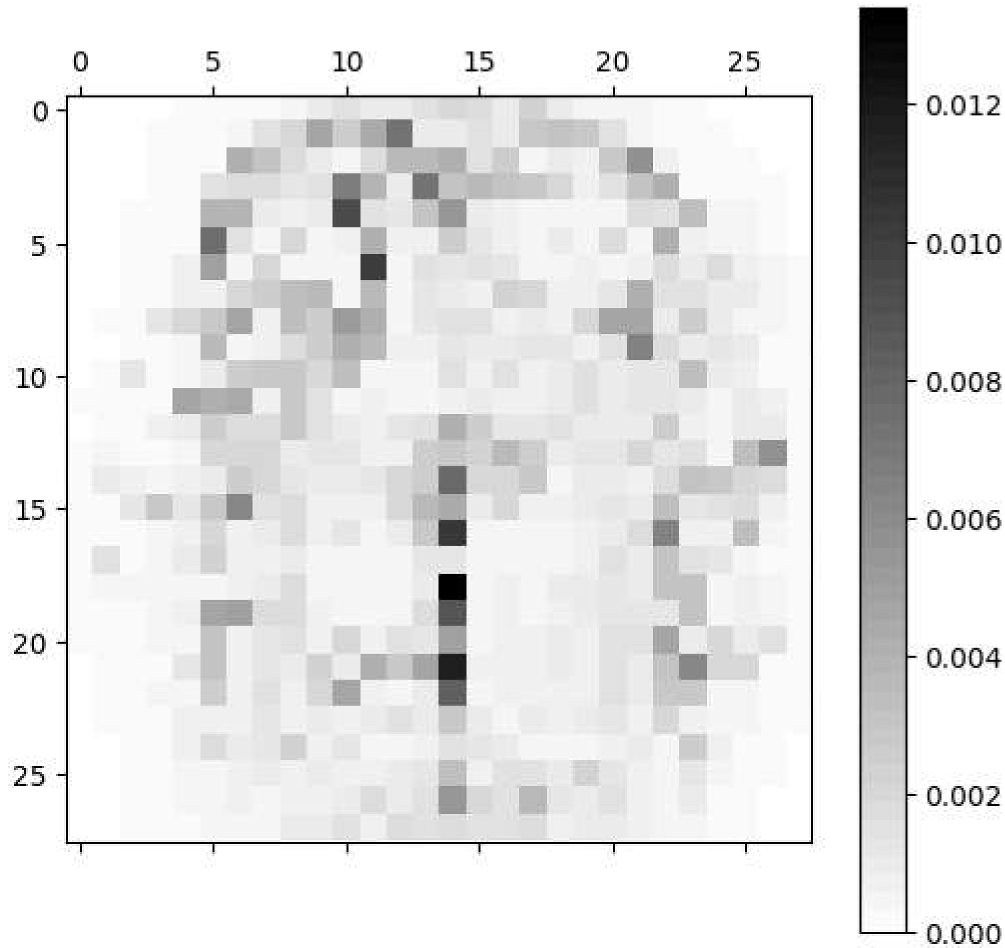
```
ExtraTreesClassifier
ExtraTreesClassifier(bootstrap=True, n_estimators=90, n_jobs=-1, oob_score=True,
                     random_state=123)
```

Evaluate the Classifiers (Accuracy, Confusion Matrix, Feature Importance)

In [31]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns

# Evaluate on the test set
def evaluate_model(clf, X_test, y_test, labels):
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)

    # Print accuracy
    print(f"Test Accuracy: {accuracy}")

    # Plot confusion matrix
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yti
cklabels=labels)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()

    # Plot feature importances as 28x28 image
    importances = clf.feature_importances_.reshape(28,28)
    plot_matrix(importances, shape=(28,28))

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

evaluate_model(rf_clf, X_test, y_test, labels)
```

```
Test Accuracy: 0.8706
```





b)

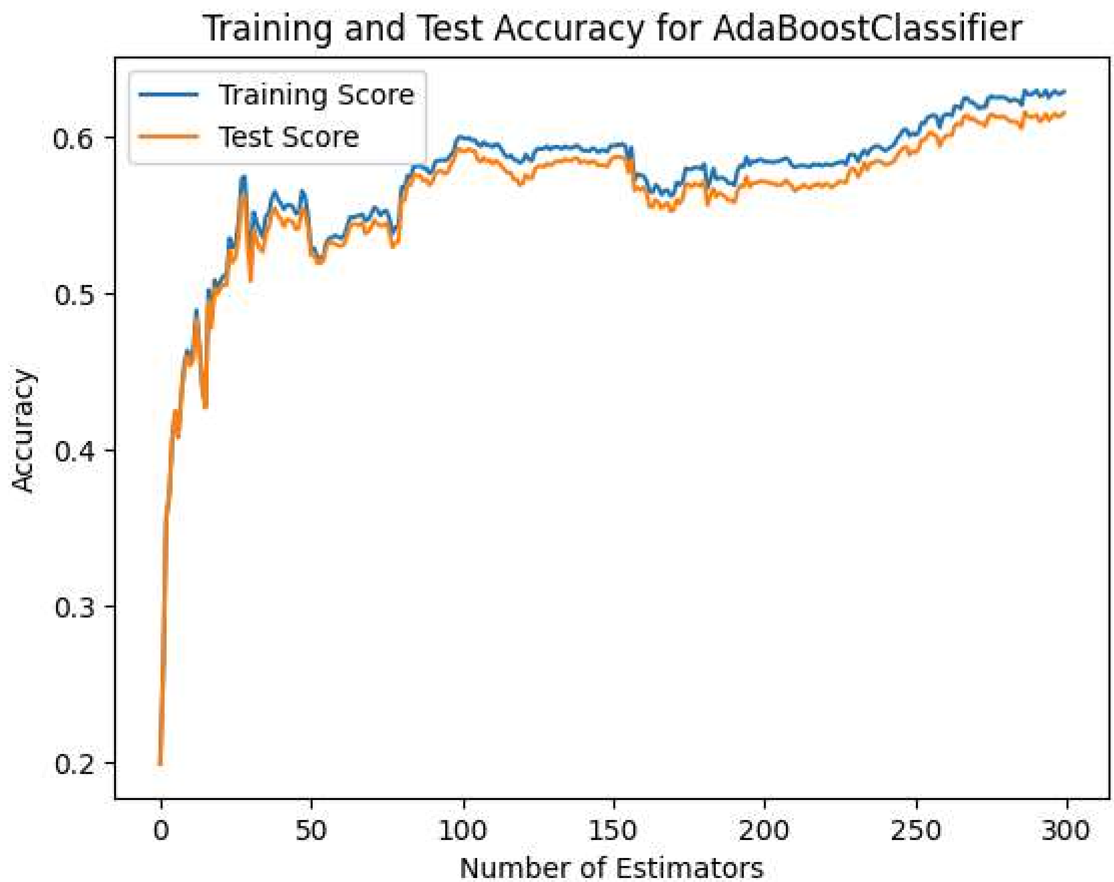
Train an AdaBoostClassifier with 300 estimators

In [32]:
```
from sklearn.ensemble import AdaBoostClassifier

# Initialize AdaBoostClassifier with 300 estimators
ada_clf = AdaBoostClassifier(n_estimators=300, random_state=123)

# Train the classifier on the training set
ada_clf.fit(X_train, y_train)
```

Out[32]:
```
▾ AdaBoostClassifier
AdaBoostClassifier(n_estimators=300, random_state=123)
```

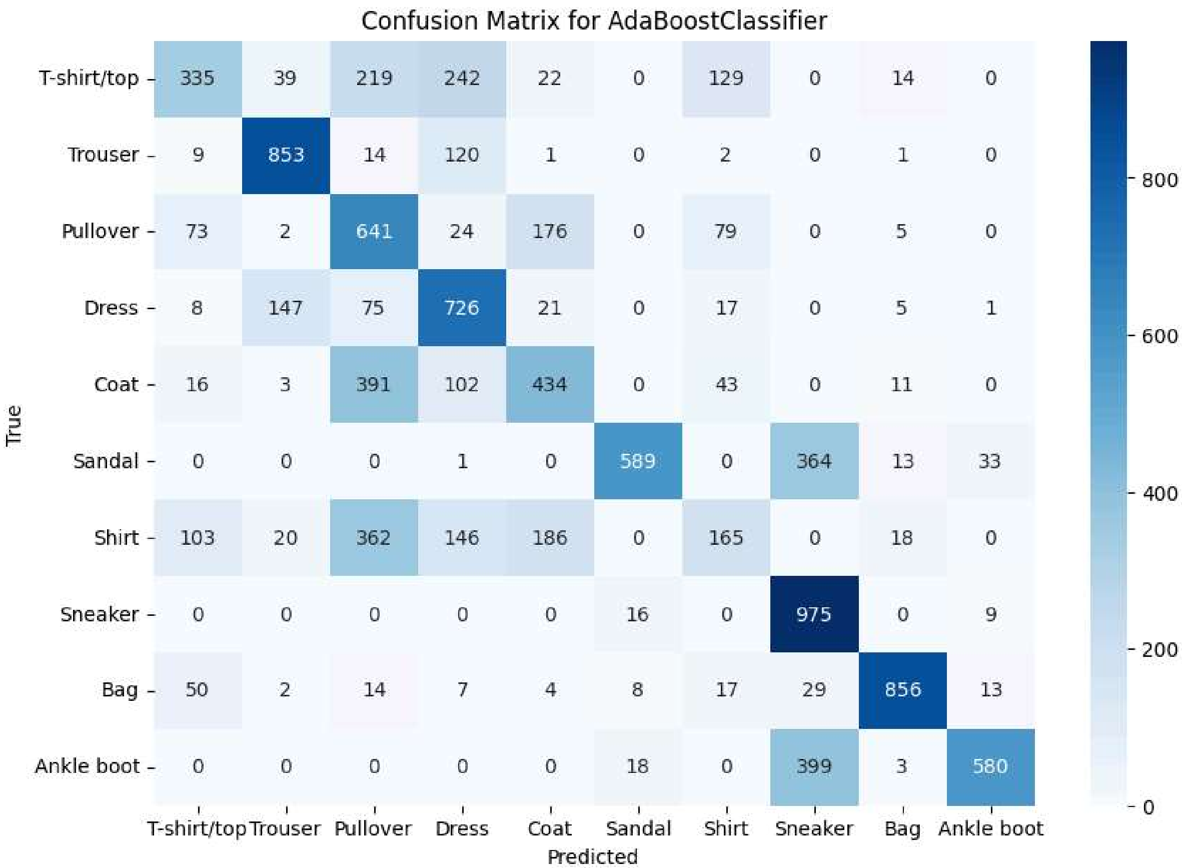
Plot the Training and Test Loss

In [33]:
```
# Collect staged training and test scores
train_scores = list(ada_clf.staged_score(X_train, y_train))
test_scores = list(ada_clf.staged_score(X_test, y_test))

# Plot training and test loss as a function of the number of estimators
plt.plot(train_scores, label='Training Score')
plt.plot(test_scores, label='Test Score')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.title('Training and Test Accuracy for AdaBoostClassifier')
plt.legend()
plt.show()
```



Compute and print the confusion matrix for the test set

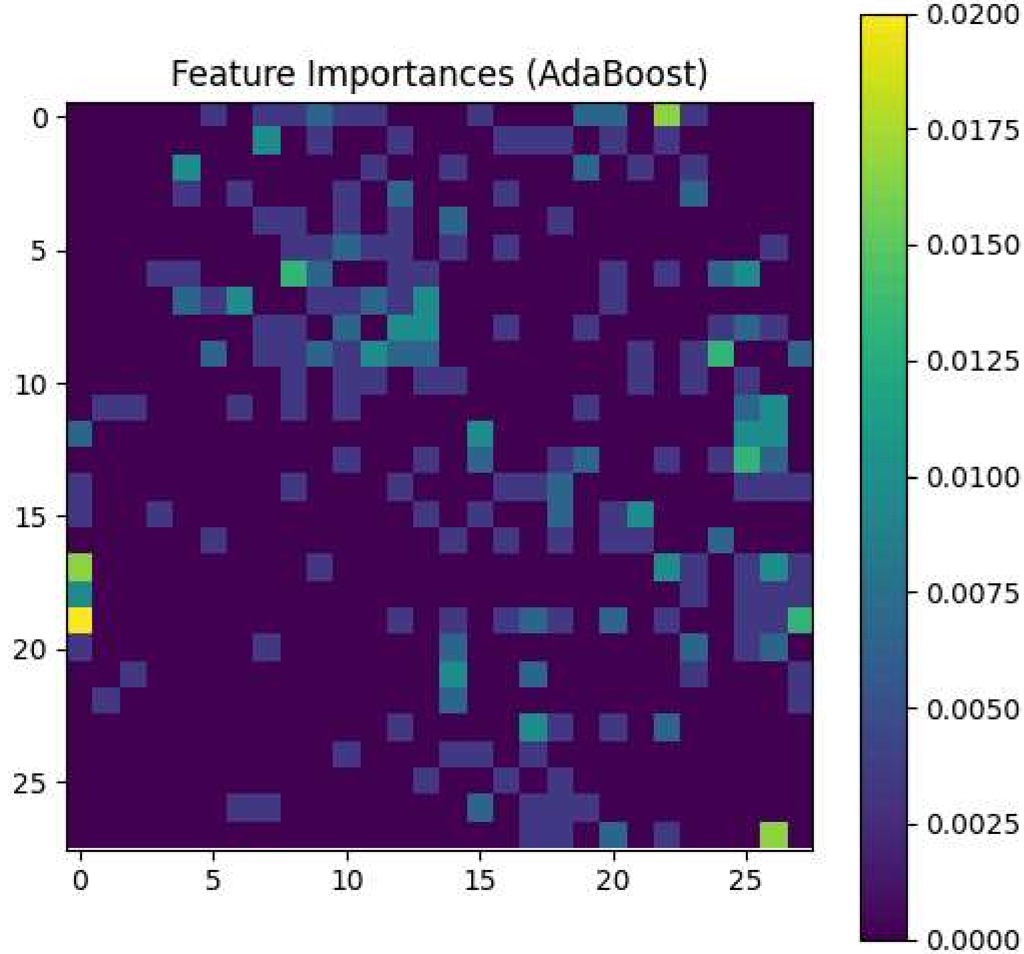
In [34]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Predict the labels for the test set
y_pred_ada = ada_clf.predict(X_test)

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_ada)

# Plot the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, ytickla
bels=labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for AdaBoostClassifier')
plt.show()
```



True \ Predicted	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	335	39	219	242	22	0	129	0	14	0
Trouser	9	853	14	120	1	0	2	0	1	0
Pullover	73	2	641	24	176	0	79	0	5	0
Dress	8	147	75	726	21	0	17	0	5	1
Coat	16	3	391	102	434	0	43	0	11	0
Sandal	0	0	0	1	0	589	0	364	13	33
Shirt	103	20	362	146	186	0	165	0	18	0
Sneaker	0	0	0	0	0	16	0	975	0	9
Bag	50	2	14	7	4	8	17	29	856	13
Ankle boot	0	0	0	0	0	18	0	399	3	580

Retrieve Feature Importances

In [35]:
```
# Retrieve feature importances
importances = ada_clf.feature_importances_.reshape(28, 28)

# Plot the feature importance matrix as a 28x28 grayscale image
def plot_matrix(m, cm=plt.cm.viridis, shape=(28, 28)):
    plt.figure(figsize=(6,6))
    plt.imshow(m, cmap=cm)
    plt.colorbar()
    plt.title("Feature Importances (AdaBoost)")
    plt.show()

plot_matrix(importances, shape=(28,28))
```



Discussion questions

a) Which of the classifiers is most accurate on the test data?

RandomForestClassifier

b) Look at your confusion matrix, what classes tends to be confused with each other, are they the same for the three classifiers. Are there any insights you can give regarding the classes that tend to be confused with each other?

T-shirt/top, Pullover, Shirt, and Coat classes tend to be the most confused across the classifiers.

T-shirt/top and Shirt are frequently confused with each other across all classifiers especially in AdaBoost.

Pullover and Coat also show consistent confusion across the classifiers.

Dress is sometimes confused with Coat but this happens more in AdaBoost compared to RandomForest or ExtraTrees.

c) Look at the feature importance matrix in part 1, what do you notice about the feature importance matrix? Is there anything that relates that to the classes that are easily confused?

T-shirt/top and Shirt tend to have overlapping or similar pixel importance regions. AdaBoost have a less distinct focus on the important pixels which makes it easier confusion between similar classes.

d) What do you notice about the the test loss for the AdaBoostClassifier? what about the confusion matrix and the feature importance matrix?

The AdaBoostClassifier test accuracy improves initially but then fluctuates and plateaus. This suggests that AdaBoost doesn't benefit significantly from increasing the number of estimators beyond a certain point and it struggles with stability in performance.

The confusion matrix for AdaBoost shows much more misclassification for several classes compared to RandomForest and ExtraTrees. AdaBoost especially struggles with T-shirt/top, Pullover and Shirt.

The feature importance matrix for AdaBoost is quite sparse and less concentrated on specific regions indicating that AdaBoost struggles to focus on the critical features needed for differentiating similar classes like T-shirt/top and Shirt.

e) Hypothetically, should you use the test loss to choose the optimum number of estimators for the AdaBoostClassifier? Why?

No, you shouldn't use the test loss to choose the optimal number of estimators. This is because the test set is meant to evaluate final model performance, not to guide model tuning. Using it for hyperparameter selection can lead to overfitting, where the model performs well on the test data but fails to generalize to new, unseen data.