In [1]:
```python
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

In [5]:
```python
# Step 1: Load the CSV files
train_data = pd.read_csv('fashion-mnist_train.csv')
test_data = pd.read_csv('fashion-mnist_test.csv')

# Step 2: Prepare the data
# Separate features (pixels) and labels
X_train = train_data.iloc[:, 1:].values
y_train = train_data.iloc[:, 0].values
X_test = test_data.iloc[:, 1:].values
y_test = test_data.iloc[:, 0].values

# Normalize the pixel values (scale them to range [0, 1])
X_train = X_train / 255.0
X_test = X_test / 255.0

# Reshape the data to 28x28x1 (as the images are 28x28 pixels and grayscale)
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

# Convert the labels to one-hot encoding
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

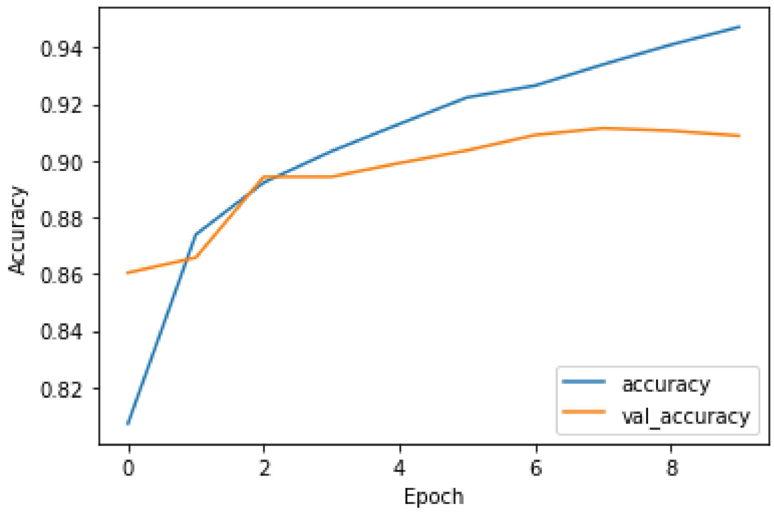
In [7]:
```python
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, rar

# Step 3: Build the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')  # 10 classes for Fashion MNIST
])
```

```
C:\Users\sd616\anaconda\lib\site-packages\keras\src\layers\convolutional\base_conv.p
y:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
n using Sequential models, prefer using an `Input(shape)` object as the first layer i
n the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [8]:
```python
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Step 4: Train the model
```

```
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val), batch

# Step 5: Evaluate the model on test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_acc}")

# Optional: Plot training history (accuracy and loss)
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

# Step 6: Display some random images with their predictions and true labels

# Assuming class_labels is a list of label names for Fashion MNIST
class_labels = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

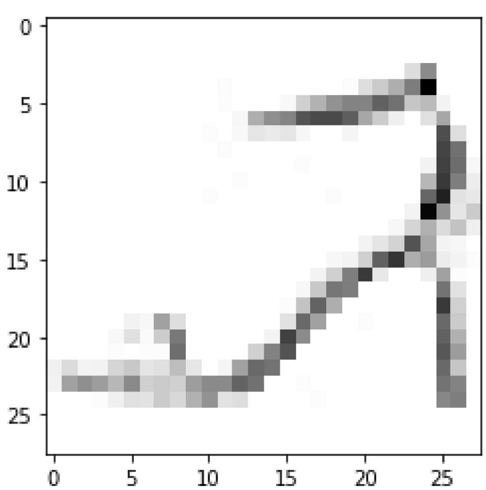
```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - accuracy: 0.7336 - loss: 0.7424 - val_ac
curacy: 0.8604 - val_loss: 0.3798
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.8709 - loss: 0.3582 - val_ac
curacy: 0.8658 - val_loss: 0.3683
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.8911 - loss: 0.3003 - val_ac
curacy: 0.8942 - val_loss: 0.2947
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.9041 - loss: 0.2638 - val_ac
curacy: 0.8942 - val_loss: 0.2861
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.9119 - loss: 0.2378 - val_ac
curacy: 0.8991 - val_loss: 0.2738
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.9252 - loss: 0.2078 - val_ac
curacy: 0.9036 - val_loss: 0.2596
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.9265 - loss: 0.1941 - val_ac
curacy: 0.9090 - val_loss: 0.2518
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - accuracy: 0.9355 - loss: 0.1731 - val_ac
curacy: 0.9114 - val_loss: 0.2503
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 11s 15ms/step - accuracy: 0.9425 - loss: 0.1574 - val_ac
curacy: 0.9105 - val_loss: 0.2521
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - accuracy: 0.9492 - loss: 0.1360 - val_ac
curacy: 0.9087 - val_loss: 0.2688
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9130 - loss: 0.2545
Test accuracy: 0.9147999882698059
```



In [11]:
```
def display_random_images(model, X_test, y_test, class_labels, num_images=10):
    plt.figure(figsize=(15, 15))
    random_indices = np.random.randint(0, X_test.shape[0], num_images)  # Randomly sel

    for i, index in enumerate(random_indices):
        plt.subplot(5, 5, i+1)
        plt.imshow(X_test[index].reshape(28, 28), cmap="Greys")
        plt.axis('off')

        # Get model prediction and the true label
        predicted_label = np.argmax(model.predict(np.expand_dims(X_test[index], axis=0
        true_label = np.argmax(y_test[index])

        # Set the title: Predicted and True label
        plt.title(f'Pred: {class_labels[predicted_label]}, True: {class_labels[true_la

    plt.tight_layout()
    plt.show()
```

In [14]:
```
plt.imshow(X_train[0],cmap="Greys")
```

Out[14]: <matplotlib.image.AxesImage at 0x17953551af0>



In [12]:
```
# Display 10 random images from the test set with predicted and true labels
display_random_images(model, X_test, y_test, class_labels, num_images=5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
```

In [ ]:

In [ ]: