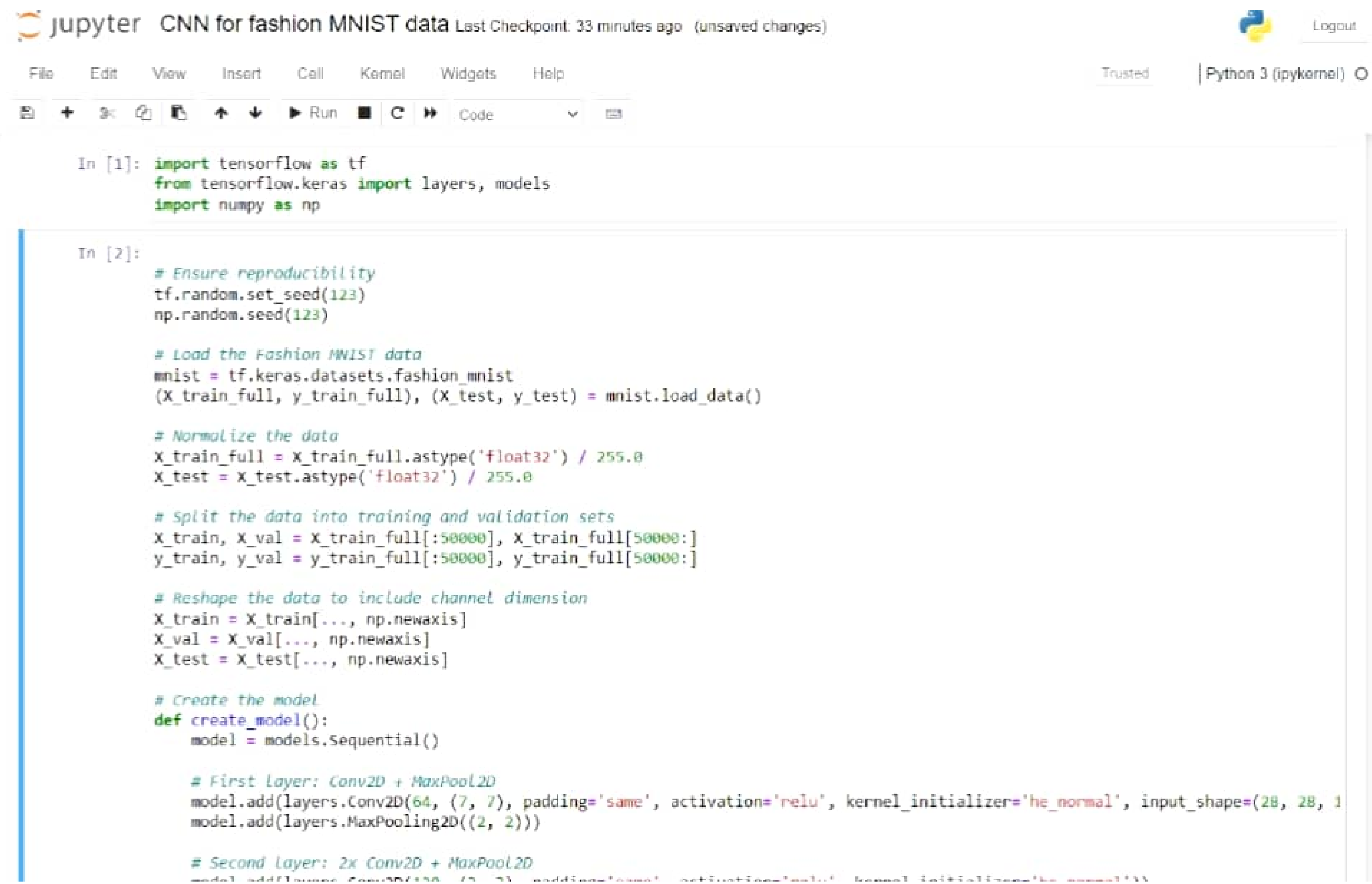
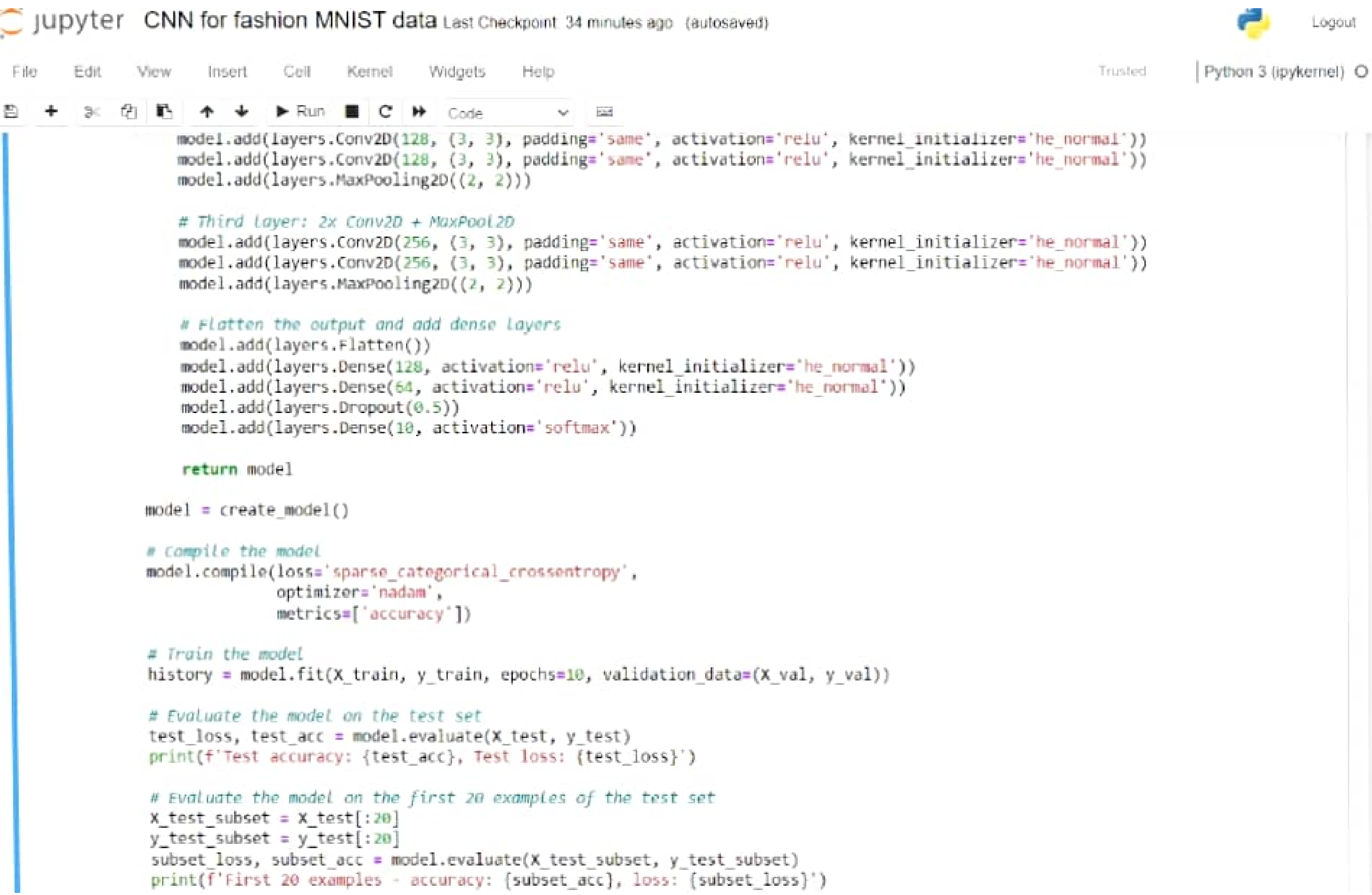


In [1]:

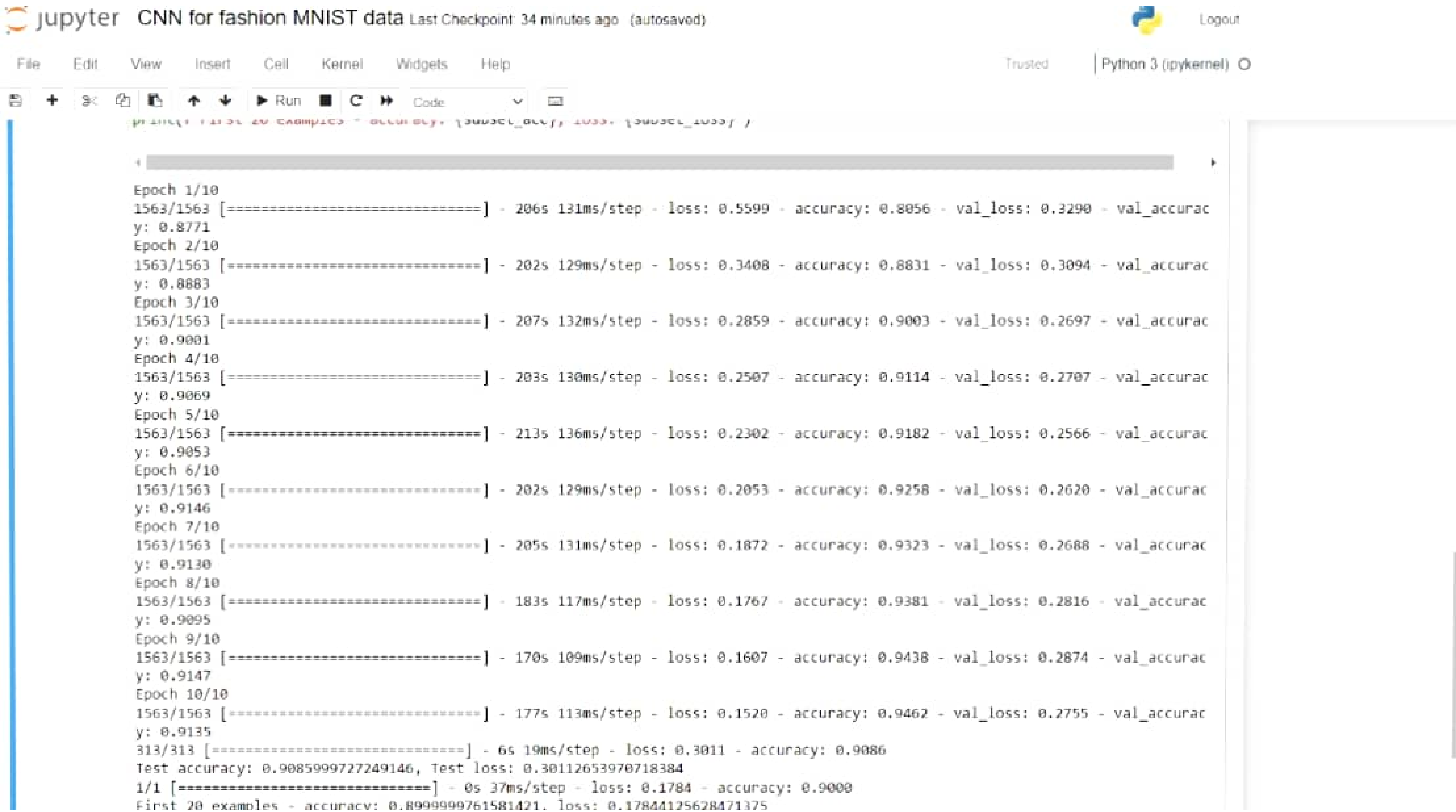
```
import tensorflow as tf
from tensorflow.keras import layers, models
import numpy as np
```

In [2]:

```
# Ensure reproducibility
tf.random.set_seed(123)
np.random.seed(123)

# Load the Fashion MNIST data
mnist = tf.keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = mnist.load_data()

# Normalize the data
X_train_full = X_train_full.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Split the data into training and validation sets
X_train, X_val = X_train_full[:50000], X_train_full[50000:]
y_train, y_val = y_train_full[:50000], y_train_full[50000:]

# Reshape the data to include channel dimension
X_train = X_train[..., np.newaxis]
X_val = X_val[..., np.newaxis]
X_test = X_test[..., np.newaxis]

# Create the model
def create_model():
    model = models.Sequential()

    # First layer: Conv2D + MaxPool2D
    model.add(layers.Conv2D(64, (7, 7), padding='same', activation='relu', kernel_initializer='he_normal', input_shape=(28, 28, 1
    model.add(layers.MaxPooling2D((2, 2)))

    # Second layer: 2x Conv2D + MaxPool2D
    model.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal'))
```



```
    model.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Third layer: 2x Conv2D + MaxPool2D
    model.add(layers.Conv2D(256, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(layers.Conv2D(256, (3, 3), padding='same', activation='relu', kernel_initializer='he_normal'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Flatten the output and add dense layers
    model.add(layers.Flatten())
    model.add(layers.Dense(128, activation='relu', kernel_initializer='he_normal'))
    model.add(layers.Dense(64, activation='relu', kernel_initializer='he_normal'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(10, activation='softmax'))

    return model

model = create_model()

# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='nadam',
              metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}, Test loss: {test_loss}')

# Evaluate the model on the first 20 examples of the test set
X_test_subset = X_test[:20]
y_test_subset = y_test[:20]
subset_loss, subset_acc = model.evaluate(X_test_subset, y_test_subset)
print(f'First 20 examples - accuracy: {subset_acc}, loss: {subset_loss}')
```



```
print(f'First 20 examples - accuracy: {subset_acc}, loss: {subset_loss}')
```

```
Epoch 1/10
1563/1563 [==============================] - 206s 131ms/step - loss: 0.5599 - accuracy: 0.8056 - val_loss: 0.3290 - val_accurac
y: 0.8771
Epoch 2/10
1563/1563 [==============================] - 202s 129ms/step - loss: 0.3408 - accuracy: 0.8831 - val_loss: 0.3094 - val_accurac
y: 0.8883
Epoch 3/10
1563/1563 [==============================] - 207s 132ms/step - loss: 0.2859 - accuracy: 0.9003 - val_loss: 0.2697 - val_accurac
y: 0.9001
Epoch 4/10
1563/1563 [==============================] - 203s 130ms/step - loss: 0.2507 - accuracy: 0.9114 - val_loss: 0.2707 - val_accurac
y: 0.9069
Epoch 5/10
1563/1563 [==============================] - 213s 136ms/step - loss: 0.2302 - accuracy: 0.9182 - val_loss: 0.2566 - val_accurac
y: 0.9053
Epoch 6/10
1563/1563 [==============================] - 202s 129ms/step - loss: 0.2053 - accuracy: 0.9258 - val_loss: 0.2620 - val_accurac
y: 0.9146
Epoch 7/10
1563/1563 [==============================] - 205s 131ms/step - loss: 0.1872 - accuracy: 0.9323 - val_loss: 0.2688 - val_accurac
y: 0.9130
Epoch 8/10
1563/1563 [==============================] - 183s 117ms/step - loss: 0.1767 - accuracy: 0.9381 - val_loss: 0.2816 - val_accurac
y: 0.9095
Epoch 9/10
1563/1563 [==============================] - 170s 109ms/step - loss: 0.1607 - accuracy: 0.9438 - val_loss: 0.2874 - val_accurac
y: 0.9147
Epoch 10/10
1563/1563 [==============================] - 177s 113ms/step - loss: 0.1520 - accuracy: 0.9462 - val_loss: 0.2755 - val_accurac
y: 0.9135
313/313 [==============================] - 6s 19ms/step - loss: 0.3011 - accuracy: 0.9086
Test accuracy: 0.9085999727249146, Test loss: 0.30112653970718384
1/1 [==============================] - 0s 37ms/step - loss: 0.1784 - accuracy: 0.9000
First 20 examples - accuracy: 0.8999999761581421, loss: 0.17844125628471375
```