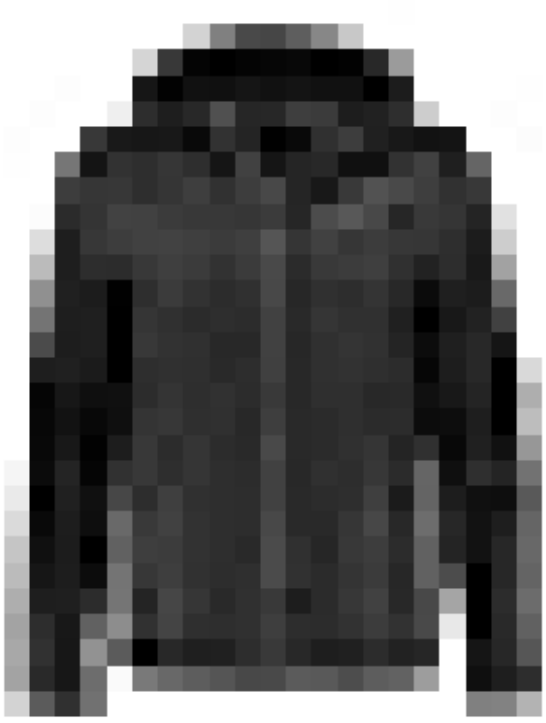
```
#
==========================================================================
============

# Fashion MNIST classification using Perceptron model

# --- Using Keras to load the dataset ---

from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score

fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) =
fashion_mnist.load_data()

print(X_train_full.shape)
print(X_train_full.dtype)

# --- Create a validation set & standardization ---

X_valid, X_train = X_train_full[:5000] / 255.0, X_train_full[5000:] /
255.0
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.0

# --- Create a list of class names ---

class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
"Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]
print(np.array(class_names)[0:10])

# --- Plot sample image ---

some_cloth = X_train[0]
some_cloth_image = some_cloth.reshape(28, 28)
plt.imshow(some_cloth_image, cmap="binary")
plt.axis("off")
plt.show()

# --- Creating the Perceptron model using the Sequential API ---

perceptron_model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(10, activation="softmax")  # Single layer
perceptron
])

# --- Display model information ---
```

```
perceptron_model.summary()

# --- Compiling the model ---

perceptron_model.compile(loss="sparse_categorical_crossentropy",
                         optimizer="sgd",
                         metrics=["accuracy"])

# --- Training and evaluating the model ---

history = perceptron_model.fit(X_train, y_train, epochs=30,
                               validation_data=(X_valid, y_valid))

# --- Learning curves ---

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1) # set the vertical range to [0-1]
plt.title("Learning Curves for Perceptron Model")
plt.show()

# --- Model evaluation on training set ---

train_loss, train_accuracy = perceptron_model.evaluate(X_train,
y_train)
print(f"Training accuracy: {train_accuracy:.4f}")

# --- Model evaluation on test set ---

test_loss, test_accuracy = perceptron_model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}")

# --- Using the model to make predictions ---

y_train_proba = perceptron_model.predict(X_train)
y_train_pred = np.argmax(y_train_proba, axis=1)
print(f"Detailed training set accuracy: {accuracy_score(y_train,
y_train_pred):.4f}")

y_test_proba = perceptron_model.predict(X_test)
y_test_pred = np.argmax(y_test_proba, axis=1)
print(f"Detailed test set accuracy: {accuracy_score(y_test,
y_test_pred):.4f}")

# --- Display some predictions ---

n_images = 5
plt.figure(figsize=(12, 4))
for i in range(n_images):
    plt.subplot(1, n_images, i+1)
```

```
    plt.imshow(X_test[i].reshape(28, 28), cmap="binary")
    plt.title(f"True: {class_names[y_test[i]]}\nPred:
{class_names[y_test_pred[i]]}")
    plt.axis("off")
plt.tight_layout()
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
(60000, 28, 28)
uint8
['T-shirt/top' 'Trouser' 'Pullover' 'Dress' 'Coat' 'Sandal' 'Shirt'
 'Sneaker' 'Bag' 'Ankle boot']
```

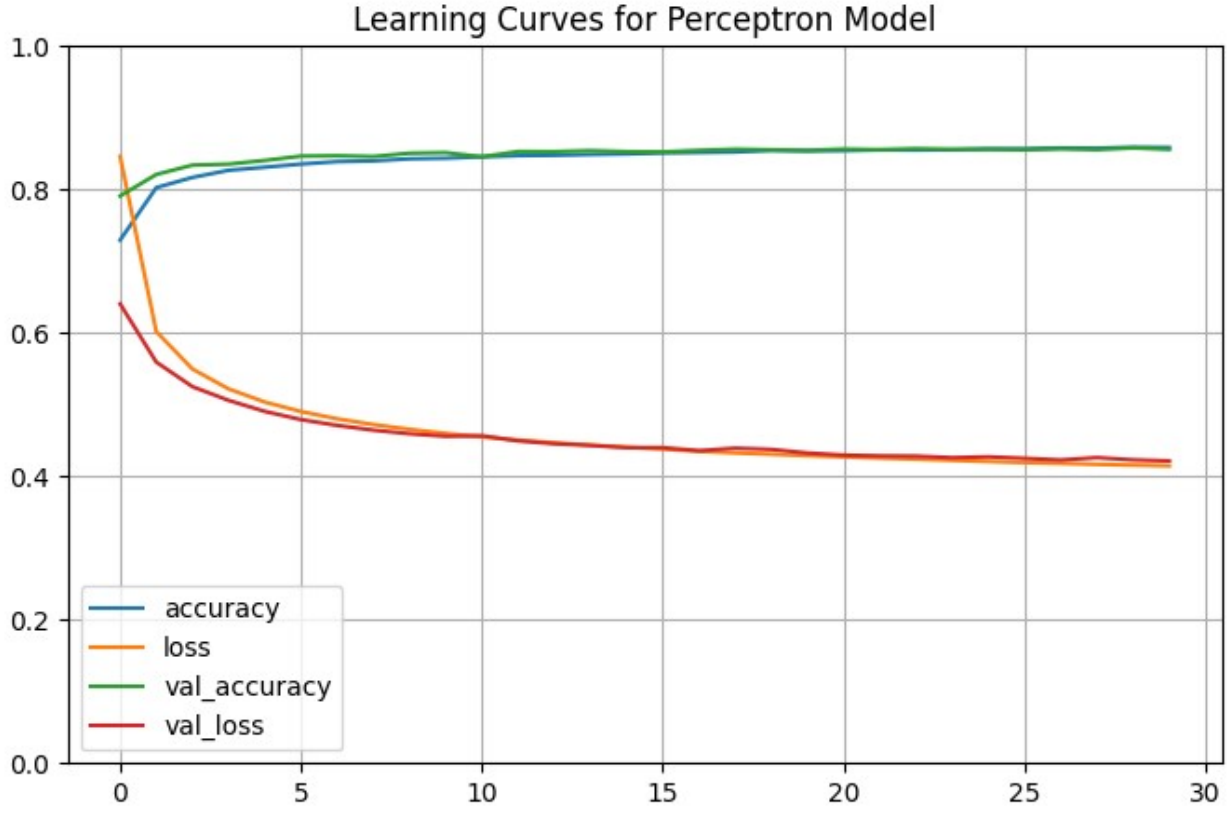


```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/
flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim`
```

```
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ flatten (Flatten)                    │ (None, 784)                 │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ dense (Dense)                        │ (None, 10)                  │
7,850 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────
┘

 Total params: 7,850 (30.66 KB)

 Trainable params: 7,850 (30.66 KB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 19s 10ms/step - accuracy: 0.6431 -
loss: 1.1190 - val_accuracy: 0.7910 - val_loss: 0.6405
Epoch 2/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7990 - loss:
0.6181 - val_accuracy: 0.8210 - val_loss: 0.5593
Epoch 3/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8128 - loss:
0.5605 - val_accuracy: 0.8340 - val_loss: 0.5251
Epoch 4/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8265 - loss:
0.5270 - val_accuracy: 0.8354 - val_loss: 0.5059
Epoch 5/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8304 - loss:
0.5051 - val_accuracy: 0.8406 - val_loss: 0.4906
Epoch 6/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8352 - loss:
0.4930 - val_accuracy: 0.8466 - val_loss: 0.4791
Epoch 7/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8390 - loss:
0.4792 - val_accuracy: 0.8470 - val_loss: 0.4710
Epoch 8/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8399 - loss:
0.4737 - val_accuracy: 0.8460 - val_loss: 0.4644
```

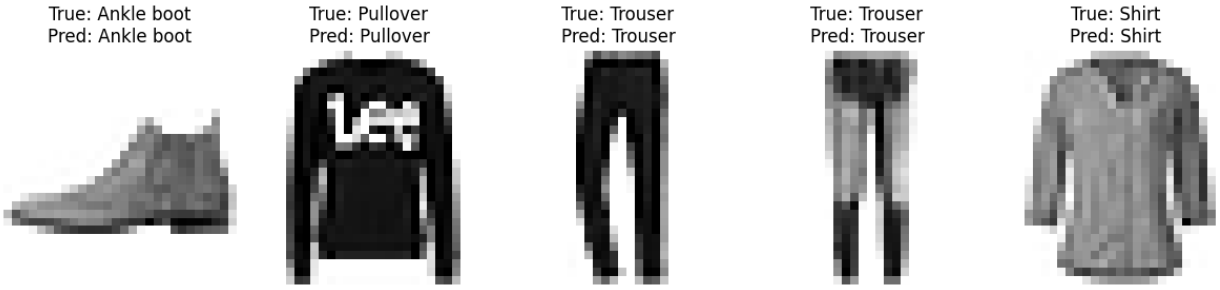
```
Epoch 9/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8437 - loss:
0.4623 - val_accuracy: 0.8506 - val_loss: 0.4594
Epoch 10/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8429 - loss:
0.4619 - val_accuracy: 0.8514 - val_loss: 0.4559
Epoch 11/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8476 - loss:
0.4524 - val_accuracy: 0.8456 - val_loss: 0.4566
Epoch 12/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8469 - loss:
0.4556 - val_accuracy: 0.8530 - val_loss: 0.4495
Epoch 13/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8474 - loss:
0.4485 - val_accuracy: 0.8528 - val_loss: 0.4455
Epoch 14/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8499 - loss:
0.4416 - val_accuracy: 0.8542 - val_loss: 0.4432
Epoch 15/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8504 - loss:
0.4378 - val_accuracy: 0.8528 - val_loss: 0.4401
Epoch 16/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8521 - loss:
0.4375 - val_accuracy: 0.8524 - val_loss: 0.4400
Epoch 17/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8531 - loss:
0.4311 - val_accuracy: 0.8546 - val_loss: 0.4358
Epoch 18/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8537 - loss:
0.4277 - val_accuracy: 0.8562 - val_loss: 0.4393
Epoch 19/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8535 - loss:
0.4293 - val_accuracy: 0.8554 - val_loss: 0.4375
Epoch 20/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8537 - loss:
0.4284 - val_accuracy: 0.8546 - val_loss: 0.4325
Epoch 21/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8533 - loss:
0.4319 - val_accuracy: 0.8564 - val_loss: 0.4296
Epoch 22/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.8556 - loss:
0.4261 - val_accuracy: 0.8556 - val_loss: 0.4285
Epoch 23/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8553 - loss:
0.4244 - val_accuracy: 0.8570 - val_loss: 0.4282
Epoch 24/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8569 - loss:
0.4177 - val_accuracy: 0.8560 - val_loss: 0.4260
Epoch 25/30
```

```
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8566 - loss:
0.4200 - val_accuracy: 0.8560 - val_loss: 0.4272
Epoch 26/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8578 - loss:
0.4140 - val_accuracy: 0.8554 - val_loss: 0.4251
Epoch 27/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8560 - loss:
0.4208 - val_accuracy: 0.8566 - val_loss: 0.4228
Epoch 28/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8552 - loss:
0.4227 - val_accuracy: 0.8556 - val_loss: 0.4261
Epoch 29/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8566 - loss:
0.4180 - val_accuracy: 0.8580 - val_loss: 0.4230
Epoch 30/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8597 - loss:
0.4142 - val_accuracy: 0.8560 - val_loss: 0.4215
```




```
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8579 - loss:
0.4140
Training accuracy: 0.8604
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8416 - loss:
0.4484
```

```
Test accuracy: 0.8397
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
Detailed training set accuracy: 0.8604
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Detailed test set accuracy: 0.8397
```



```
#
================================================================================
============

# Fashion MNIST classification using MLP with one hidden layer

# --- Using the same dataset from Question 1 ---

from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score

# Skip dataset loading if already done in Question 1
# Otherwise, uncomment these lines:

fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) =
fashion_mnist.load_data()

X_valid, X_train = X_train_full[:5000] / 255.0, X_train_full[5000:] /
255.0
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.0

class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
"Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"]

# --- Creating the MLP model with one hidden layer using the
Sequential API ---

mlp_model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(50, activation="relu"),  # One hidden layer
with 50 neurons
    keras.layers.Dense(10, activation="softmax")
])

# --- Display model information ---

mlp_model.summary()

# --- Compiling the model ---

mlp_model.compile(loss="sparse_categorical_crossentropy",
                  optimizer="sgd",
                  metrics=["accuracy"])

# --- Training and evaluating the model ---
```

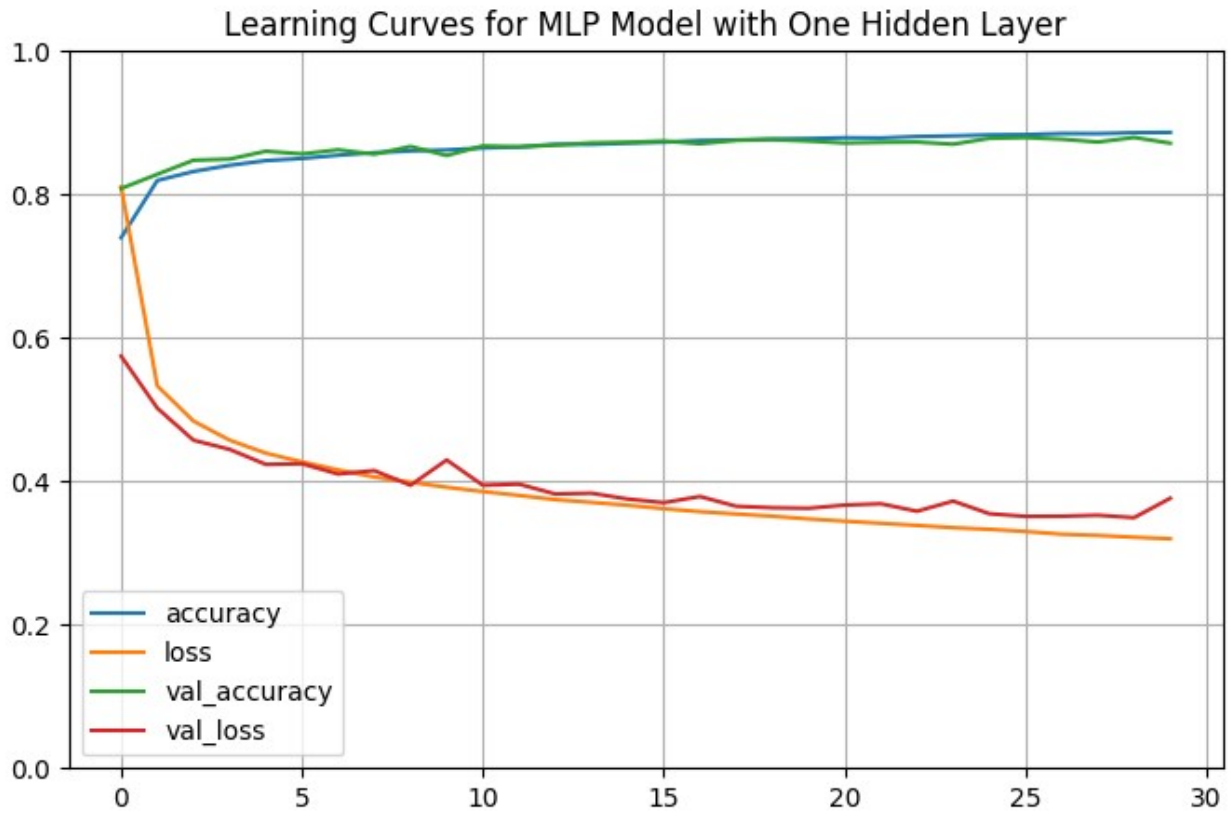
```
history = mlp_model.fit(X_train, y_train, epochs=30,
                        validation_data=(X_valid, y_valid))

# --- Learning curves ---

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1) # set the vertical range to [0-1]
plt.title("Learning Curves for MLP Model with One Hidden Layer")
plt.show()

# --- Model evaluation on training set ---

train_loss, train_accuracy = mlp_model.evaluate(X_train, y_train)
print(f"Training accuracy: {train_accuracy:.4f}")

# --- Model evaluation on test set ---

test_loss, test_accuracy = mlp_model.evaluate(X_test, y_test)
print(f"Test accuracy: {test_accuracy:.4f}")

# --- Using the model to make predictions ---

y_train_proba = mlp_model.predict(X_train)
y_train_pred = np.argmax(y_train_proba, axis=1)
print(f"Detailed training set accuracy: {accuracy_score(y_train,
y_train_pred):.4f}")

y_test_proba = mlp_model.predict(X_test)
y_test_pred = np.argmax(y_test_proba, axis=1)
print(f"Detailed test set accuracy: {accuracy_score(y_test,
y_test_pred):.4f}")

# --- Display some predictions ---

n_images = 5
plt.figure(figsize=(12, 4))
for i in range(n_images):
    plt.subplot(1, n_images, i+1)
    plt.imshow(X_test[i].reshape(28, 28), cmap="binary")
    plt.title(f"True: {class_names[y_test[i]]}\nPred:
{class_names[y_test_pred[i]]}")
    plt.axis("off")
plt.tight_layout()
plt.show()

# --- Comparison between Perceptron and MLP ---

print("\nComparison of Model Performance:")
print("-" * 50)
```

```
print(f"{'Model':<20} {'Training Accuracy':<20} {'Test
Accuracy':<20}")
print("-" * 50)
print(f"{'Perceptron':<20} {train_accuracy:.4f}{'':<14}
{test_accuracy:.4f}")

# If you have the perceptron results from Question 1, you can use them
directly:
# print(f"{'Perceptron':<20} {perceptron_train_accuracy:.4f}{'':<14}
{perceptron_test_accuracy:.4f}")
```

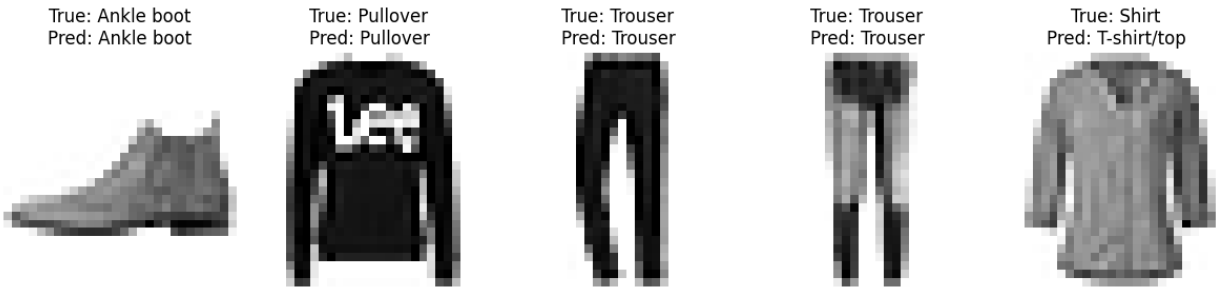
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 1us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step

/usr/local/lib/python3.11/dist-packages/keras/src/layers/reshaping/
flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ flatten (Flatten)                    │ (None, 784)                 │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ dense (Dense)                        │ (None, 50)                  │
39,250 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
│ dense_1 (Dense)                      │ (None, 10)                  │
510 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────
┘
```

```
 Total params: 39,760 (155.31 KB)

 Trainable params: 39,760 (155.31 KB)

 Non-trainable params: 0 (0.00 B)

Epoch 1/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 14s 7ms/step - accuracy: 0.6357 - loss:
1.1393 - val_accuracy: 0.8084 - val_loss: 0.5746
Epoch 2/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 12s 2ms/step - accuracy: 0.8175 - loss:
0.5412 - val_accuracy: 0.8282 - val_loss: 0.5022
Epoch 3/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8277 - loss:
0.4997 - val_accuracy: 0.8478 - val_loss: 0.4574
Epoch 4/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8431 - loss:
0.4541 - val_accuracy: 0.8494 - val_loss: 0.4446
Epoch 5/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8461 - loss:
0.4420 - val_accuracy: 0.8604 - val_loss: 0.4237
Epoch 6/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8505 - loss:
0.4300 - val_accuracy: 0.8566 - val_loss: 0.4245
Epoch 7/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8551 - loss:
0.4146 - val_accuracy: 0.8624 - val_loss: 0.4102
Epoch 8/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8576 - loss:
0.4057 - val_accuracy: 0.8562 - val_loss: 0.4150
Epoch 9/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8626 - loss:
0.3918 - val_accuracy: 0.8670 - val_loss: 0.3945
Epoch 10/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8593 - loss:
0.3945 - val_accuracy: 0.8544 - val_loss: 0.4298
Epoch 11/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8648 - loss:
0.3849 - val_accuracy: 0.8678 - val_loss: 0.3945
Epoch 12/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.8668 - loss:
0.3797 - val_accuracy: 0.8664 - val_loss: 0.3961
Epoch 13/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8718 - loss:
0.3702 - val_accuracy: 0.8690 - val_loss: 0.3824
Epoch 14/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8712 - loss:
0.3685 - val_accuracy: 0.8720 - val_loss: 0.3835
Epoch 15/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8721 - loss:
```

```
0.3671 - val_accuracy: 0.8728 - val_loss: 0.3753
Epoch 16/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8714 - loss:
0.3654 - val_accuracy: 0.8746 - val_loss: 0.3703
Epoch 17/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8767 - loss:
0.3563 - val_accuracy: 0.8708 - val_loss: 0.3789
Epoch 18/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8760 - loss:
0.3549 - val_accuracy: 0.8756 - val_loss: 0.3653
Epoch 19/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8797 - loss:
0.3455 - val_accuracy: 0.8766 - val_loss: 0.3632
Epoch 20/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8765 - loss:
0.3524 - val_accuracy: 0.8746 - val_loss: 0.3625
Epoch 21/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8811 - loss:
0.3410 - val_accuracy: 0.8718 - val_loss: 0.3669
Epoch 22/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8807 - loss:
0.3410 - val_accuracy: 0.8728 - val_loss: 0.3688
Epoch 23/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8818 - loss:
0.3378 - val_accuracy: 0.8732 - val_loss: 0.3583
Epoch 24/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8819 - loss:
0.3381 - val_accuracy: 0.8702 - val_loss: 0.3725
Epoch 25/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8853 - loss:
0.3321 - val_accuracy: 0.8784 - val_loss: 0.3548
Epoch 26/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8826 - loss:
0.3318 - val_accuracy: 0.8794 - val_loss: 0.3512
Epoch 27/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8869 - loss:
0.3225 - val_accuracy: 0.8772 - val_loss: 0.3514
Epoch 28/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8849 - loss:
0.3271 - val_accuracy: 0.8732 - val_loss: 0.3528
Epoch 29/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8878 - loss:
0.3196 - val_accuracy: 0.8796 - val_loss: 0.3494
Epoch 30/30
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8855 - loss:
0.3236 - val_accuracy: 0.8716 - val_loss: 0.3764
```




```
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8762 - loss:
0.3411
Training accuracy: 0.8788
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8584 - loss:
0.3976
Test accuracy: 0.8557
1719/1719 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
Detailed training set accuracy: 0.8788
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
Detailed test set accuracy: 0.8557
```




```

Comparison of Model Performance:
--------------------------------------------------
```

```
Model                Training Accuracy    Test Accuracy
-------------------------------------------------------
Perceptron           0.8788               0.8557
```