```
# prompt: First import the packages that create simple CNN

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import pandas as pd
import numpy as np
```

```

# Load Fashion MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

# Normalize pixel values
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# Expand grayscale channel to match CNN input (from (28, 28) → (28, 28, 1))
x_train = tf.expand_dims(x_train, axis=-1)
x_test = tf.expand_dims(x_test, axis=-1)

# Define input shape
input_shape = x_train.shape[1:]  # (28, 28, 1)

# Define CNN model
model = keras.Sequential([
    keras.Input(shape=input_shape),
    layers.Conv2D(128, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten()
])

model.summary()
```

```
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 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                 │ (None, 28, 28, 128)    │         1,280 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 14, 14, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 25088)          │             0 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 1,280 (5.00 KB)
 Trainable params: 1,280 (5.00 KB)
 Non-trainable params: 0 (0.00 B)
```

```
input_shape
```

```
TensorShape([28, 28, 1])
```

```
pd.DataFrame(y_train)[0].unique()
```

```
array([9, 0, 3, 2, 7, 5, 1, 6, 4, 8], dtype=uint8)
```

```
# prompt: Now train the model based on the labels

num_classes = len(pd.DataFrame(y_train)[0].unique())
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(num_classes, activation="softmax"))

model.summary()

batch_size = 128
epochs = 5

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

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━┓
┃ Layer (type)                 ┃ Output Shape          ┃     Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━┩
│ conv2d (Conv2D)              │ (None, 28, 28, 128)   │       1,280 │
├──────────────────────────────┼───────────────────────┼─────────────┤
│ max_pooling2d (MaxPooling2D) │ (None, 14, 14, 128)   │           0 │
├──────────────────────────────┼───────────────────────┼─────────────┤
│ flatten (Flatten)            │ (None, 25088)         │           0 │
├──────────────────────────────┼───────────────────────┼─────────────┤
│ dense (Dense)                │ (None, 128)           │   3,211,392 │
├──────────────────────────────┼───────────────────────┼─────────────┤
│ dense_1 (Dense)              │ (None, 10)            │       1,290 │
└──────────────────────────────┴───────────────────────┴─────────────┘
 Total params: 3,213,962 (12.26 MB)
 Trainable params: 3,213,962 (12.26 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 109s 254ms/step - accuracy: 0.7965 - loss: 0.5882 - val_accuracy: 0.8878 - val_loss: 0.3008
Epoch 2/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 108s 257ms/step - accuracy: 0.9017 - loss: 0.2745 - val_accuracy: 0.9090 - val_loss: 0.2598
Epoch 3/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 107s 254ms/step - accuracy: 0.9186 - loss: 0.2260 - val_accuracy: 0.9092 - val_loss: 0.2477
Epoch 4/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 105s 249ms/step - accuracy: 0.9307 - loss: 0.1922 - val_accuracy: 0.9150 - val_loss: 0.2367
Epoch 5/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 140s 244ms/step - accuracy: 0.9380 - loss: 0.1694 - val_accuracy: 0.9165 - val_loss: 0.2392
<keras.src.callbacks.history.History at 0x7e97bfc1ff90>
```

```
model.evaluate(x_test,y_test)[1]
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 13ms/step - accuracy: 0.9120 - loss: 0.2565
0.9115999937057495
```

```
# Save weights
model.save_weights("fashion_mnist_cnn.weights.h5")
```

```
# Load MNIST digits and preprocess
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

# Recreate base model
conv_base = keras.Sequential([
    keras.Input(shape=input_shape),
    layers.Conv2D(128, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten()
])
conv_base.load_weights("fashion_mnist_cnn.weights.h5")
conv_base.trainable = False  # Freeze base layers
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
# Add new head
transfer_model = keras.Sequential([
    conv_base,
    layers.Dense(64, activation="relu"),  # smaller dense layer to reduce overfitting
    layers.Dense(num_classes, activation="softmax")
])

# Compile and train
transfer_model.compile(optimizer="adam", loss="sparse_categorical_crossentropy", metrics=["accuracy"])
transfer_model.fit(x_train, y_train, epochs=3, validation_split=0.1)

# Evaluate
test_loss, test_acc = transfer_model.evaluate(x_test, y_test)
print(f"Transfer Learning Test Accuracy: {test_acc:.4f}")
```

```
Epoch 1/3
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 52s 30ms/step - accuracy: 0.9211 - loss: 0.2767 - val_accuracy: 0.9783 - val_loss: 0.0717
Epoch 2/3
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 81s 30ms/step - accuracy: 0.9819 - loss: 0.0584 - val_accuracy: 0.9835 - val_loss: 0.0589
Epoch 3/3
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9878 - loss: 0.0387 - val_accuracy: 0.9740 - val_loss: 0.0892
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - accuracy: 0.9639 - loss: 0.1218
Transfer Learning Test Accuracy: 0.9697
```