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

```
(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 = x_train.reshape((-1, 28, 28, 1))
x_test = x_test.reshape((-1, 28, 28, 1))

y_train = keras.utils.to_categorical(y_train, num_classes=10)
y_test = keras.utils.to_categorical(y_test, num_classes=10)
```

```
model = keras.Sequential(
    [
        layers.Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation="relu"),
        layers.MaxPooling2D((2, 2)),
        layers.Flatten(),
        layers.Dense(128, activation="relu"),
        layers.Dense(10, activation="softmax"),
    ]
)
```

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

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_3 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_1 (Flatten)	(None, 1600)	0
dense_2 (Dense)	(None, 128)	204,928
dense_3 (Dense)	(None, 10)	1,290

Total params: 225,034 (879.04 KB)

```
history = model.fit(x_train, y_train, epochs=10, validation_split=0.1)
```

```
Epoch 1/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 57s 32ms/step - accuracy: 0.9075 - loss: 0.3084 - val_accuracy: 0.9830 - val_loss: 0.0571
Epoch 2/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 79s 31ms/step - accuracy: 0.9858 - loss: 0.0472 - val_accuracy: 0.9883 - val_loss: 0.0398
Epoch 3/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 50s 29ms/step - accuracy: 0.9907 - loss: 0.0282 - val_accuracy: 0.9885 - val_loss: 0.0418
Epoch 4/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 80s 28ms/step - accuracy: 0.9930 - loss: 0.0203 - val_accuracy: 0.9892 - val_loss: 0.0410
Epoch 5/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 84s 30ms/step - accuracy: 0.9956 - loss: 0.0142 - val_accuracy: 0.9908 - val_loss: 0.0357
Epoch 6/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 50s 30ms/step - accuracy: 0.9962 - loss: 0.0120 - val_accuracy: 0.9877 - val_loss: 0.0432
Epoch 7/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9976 - loss: 0.0076 - val_accuracy: 0.9882 - val_loss: 0.0534
Epoch 8/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 82s 30ms/step - accuracy: 0.9980 - loss: 0.0067 - val_accuracy: 0.9915 - val_loss: 0.0422
Epoch 9/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 83s 30ms/step - accuracy: 0.9984 - loss: 0.0049 - val_accuracy: 0.9907 - val_loss: 0.0402
```

```
Epoch 10/10
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 80s 29ms/step - accuracy: 0.9981 - loss: 0.0055 - val_accuracy: 0.9915 - val_loss: 0.0438
```

```
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test loss: {loss:.4f}")
print(f"Test accuracy: {accuracy:.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.9903 - loss: 0.0408
Test loss: 0.0317
Test accuracy: 0.9919
```

```
predictions = model.predict(x_test)
print(predictions)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step
[[2.8182216e-15 1.6171050e-13 3.4985261e-12 ... 9.9999994e-01
  1.6728505e-17 4.6382453e-11]
 [2.3480274e-13 1.3916217e-10 9.9999994e-01 ... 4.6401180e-16
  3.2465846e-17 1.1698152e-21]
 [2.4907302e-11 9.9999911e-01 5.7571814e-10 ... 5.6047583e-07
  1.9578926e-07 4.4505510e-11]
 ...
 [1.1599857e-21 1.1680350e-15 2.2768531e-18 ... 2.6072008e-14
  2.6681270e-11 3.2994381e-15]
 [4.4203076e-16 3.5325142e-20 2.6083705e-21 ... 9.2329536e-21
  4.9647345e-11 4.7253544e-18]
 [1.2013277e-12 3.6340803e-14 2.0375839e-12 ... 2.0544706e-19
  3.3551479e-13 1.8980761e-18]]
```

```
from IPython.display import display
from google.colab import files
import numpy as np
from tensorflow.keras.preprocessing import image
```

```
def predict_image_class(model):
  """Predicts the class of an uploaded image using the given model."""

  uploaded = files.upload()

  for fn in uploaded.keys():
    # Preprocess the image
    img_path = fn
    img = image.load_img(img_path, target_size=(28, 28), color_mode='grayscale')
    img_array = image.img_to_array(img)
    img_array = img_array / 255.0  # Normalize
    img_array = img_array.reshape((-1, 28, 28, 1))  # Reshape

    # Make the prediction
    prediction = model.predict(img_array)
    predicted_class = np.argmax(prediction)

    # Get the class name
    class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                   'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
    predicted_class_name = class_names[predicted_class]

    # Display the result
    print(f"The predicted class for {fn} is: {predicted_class_name}")
```

```
# Call the function with your trained model
predict_image_class(model)
```

```
Choose Files images.jpg
• images.jpg(image/jpeg) - 7477 bytes, last modified: 4/4/2025 - 100% done
Saving images.jpg to images (1).jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
The predicted class for images (1).jpg is: T-shirt/top
```