Week-10

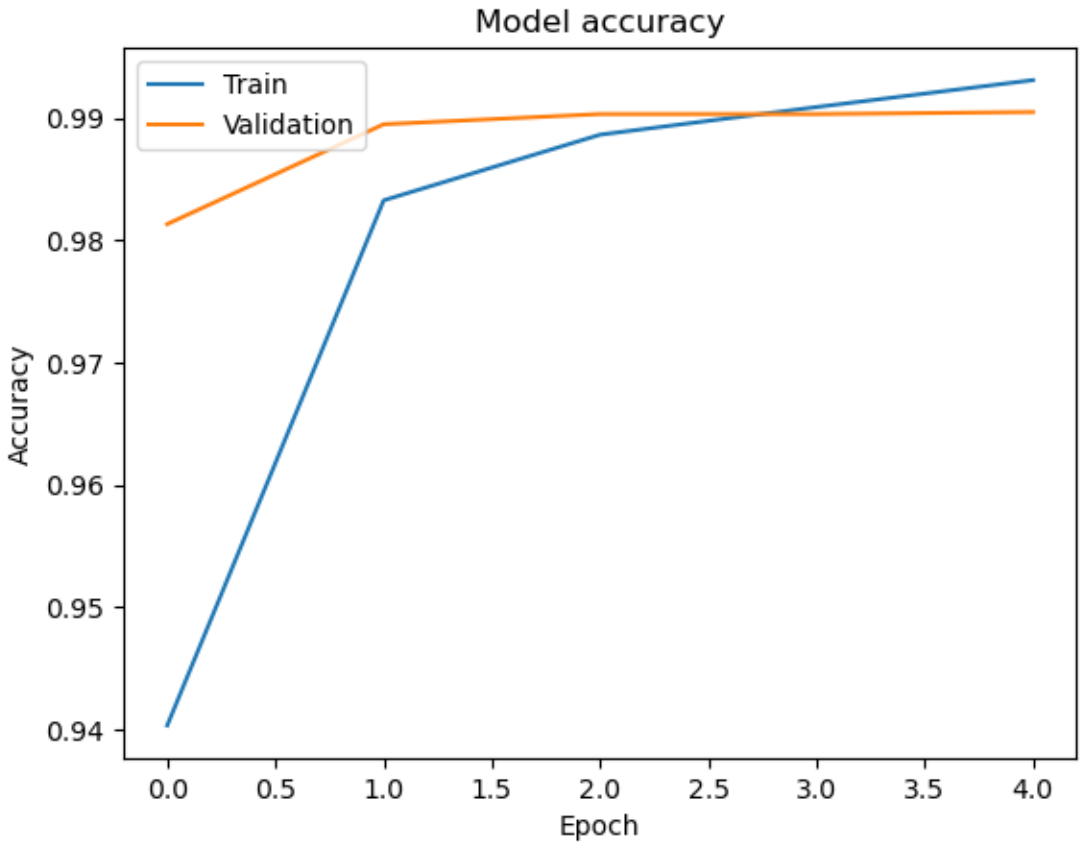
September 26, 2024

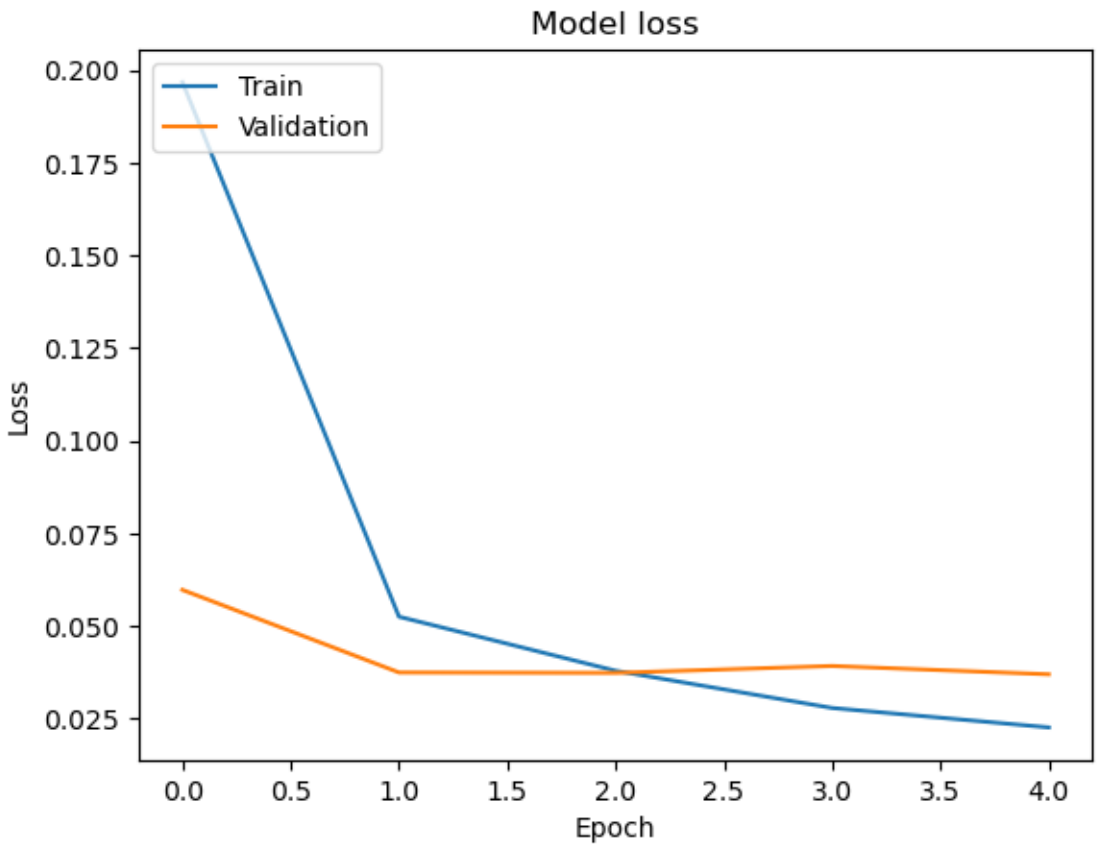
[2]:
```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocessing: Normalize the images to [0, 1] range and reshape them
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)

# Convert class labels to one-hot encoded vectors
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Build the neural network model
model = models.Sequential()

# First convolutional layer
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))

# Second convolutional layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Third convolutional layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

# Flatten the output and add a fully connected layer
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
```

```
# Output layer with 10 units (one for each digit)
model.add(layers.Dense(10, activation='softmax'))

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

# Train the model
history = model.fit(x_train, y_train, epochs=5, batch_size=64,␣
 ↪validation_split=0.1)

# Evaluate the model on the test dataset
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f'Test accuracy: {test_acc:.4f}')

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```

```
Epoch 1/5
844/844             34s 35ms/step -
accuracy: 0.8613 - loss: 0.4561 - val_accuracy: 0.9813 - val_loss: 0.0598
Epoch 2/5
844/844             29s 35ms/step -
accuracy: 0.9829 - loss: 0.0532 - val_accuracy: 0.9895 - val_loss: 0.0375
Epoch 3/5
844/844             29s 35ms/step -
accuracy: 0.9890 - loss: 0.0374 - val_accuracy: 0.9903 - val_loss: 0.0374
Epoch 4/5
844/844             29s 35ms/step -
accuracy: 0.9910 - loss: 0.0281 - val_accuracy: 0.9903 - val_loss: 0.0393
Epoch 5/5
844/844             28s 33ms/step -
```

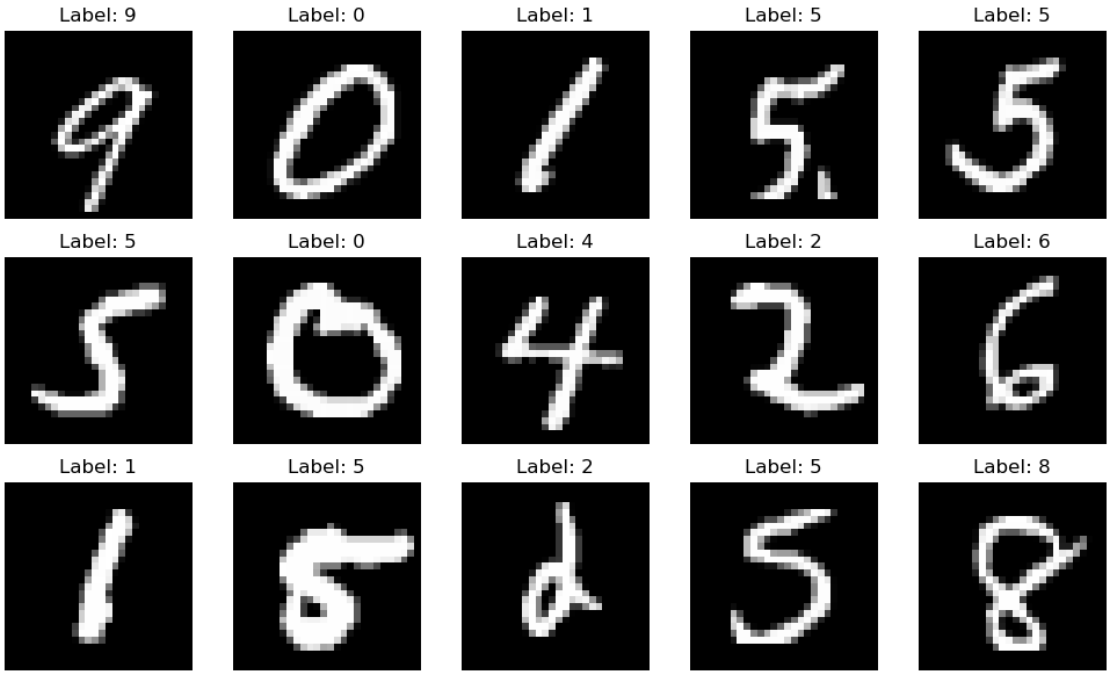
```
accuracy: 0.9933 - loss: 0.0219 - val_accuracy: 0.9905 - val_loss: 0.0370
313/313 - 3s - 8ms/step - accuracy: 0.9901 - loss: 0.0292
Test accuracy: 0.9901
```





[6]:
```python
import numpy as np
import matplotlib.pyplot as plt

def load_mnist_images(file_path):
    with open(file_path, 'rb') as f:
        # Skip the magic number and dimensions
        f.read(16)
        # Read the image data
        images = np.frombuffer(f.read(), dtype=np.uint8)
        images = images.reshape(-1, 28, 28, 1)  # Reshape to 28x28x1
    return images

def load_mnist_labels(file_path):
    with open(file_path, 'rb') as f:

        f.read(8)
        # Read the label data
        labels = np.frombuffer(f.read(), dtype=np.uint8)
    return labels
```

```
# Replace these paths with the correct paths to your extracted dataset files
train_images_path = 'train-images.idx3-ubyte'
train_labels_path = 'train-labels.idx1-ubyte'

# Load the data
train_images = load_mnist_images(train_images_path)
train_labels = load_mnist_labels(train_labels_path)

# Normalize the images to the range [0, 1]
train_images = train_images.astype('float32') / 255

# Display a few samples from the dataset
fig, axes = plt.subplots(3, 5, figsize=(10, 6))
axes = axes.flatten()

# Sample indices to visualize
sample_indices = np.random.choice(range(len(train_images)), 15, replace=False)

for i, idx in enumerate(sample_indices):
    image = train_images[idx].reshape(28, 28)
    label = train_labels[idx]  # Get the label as an integer

    axes[i].imshow(image, cmap='gray')
    axes[i].set_title(f'Label: {label}')
    axes[i].axis('off')

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



[13]:
```
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt


dataset_dir = 'C:/Users/HP/Documents/coin_dataset'
batch_size = 32
img_height, img_width = 224, 224
epochs = 5


train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)
```

```
train_generator = train_datagen.flow_from_directory(
    dataset_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training')

validation_generator = train_datagen.flow_from_directory(
    dataset_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation')


num_classes = len(train_generator.class_indices)
print(f"Detected number of classes: {num_classes}")


model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_width,␣
 ↪3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')  # Output layer with dynamic␣
 ↪num_classes
])

model.compile(optimizer=Adam(), loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

history = model.fit(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator)

model.save('inr_coin_model.h5')

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(epochs)


plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

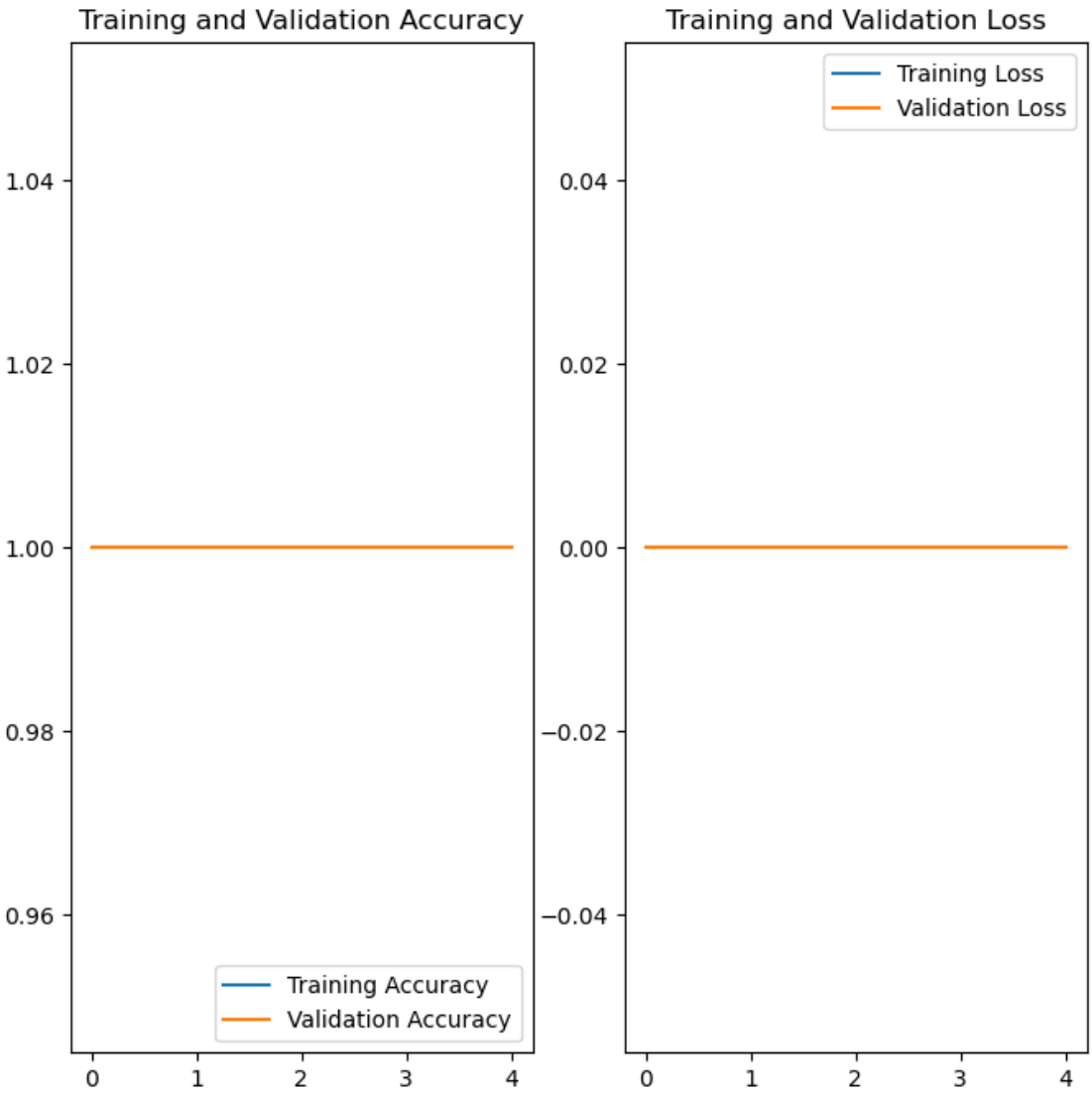
```
Found 720 images belonging to 1 classes.
Found 180 images belonging to 1 classes.
Detected number of classes: 1
```

```
C:\Users\HP\anaconda3\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: 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__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/5
23/23             136s 5s/step -
accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss:
0.0000e+00
Epoch 2/5
23/23             126s 4s/step -
accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss:
0.0000e+00
Epoch 3/5
23/23             125s 4s/step -
accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss:
0.0000e+00
Epoch 4/5
23/23             123s 4s/step -
accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss:
0.0000e+00
Epoch 5/5
23/23             122s 4s/step -
accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss:
```

```
0.0000e+00
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.
```



[16]:
```
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, load_model
```

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import cv2
import numpy as np
from tkinter import Tk, filedialog
from PIL import Image


dataset_dir = 'D:/AI&ML/DataSet'
model_path = 'inr_coin_model.h5'
batch_size = 32
img_height, img_width = 224, 224
epochs = 10


if not os.path.exists(model_path):
    print("Model not found, training a new model...")

    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        validation_split=0.2)

    train_generator = train_datagen.flow_from_directory(
        dataset_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical',
        subset='training')

    validation_generator = train_datagen.flow_from_directory(
        dataset_dir,
        target_size=(img_height, img_width),
        batch_size=batch_size,
        class_mode='categorical',
        subset='validation')

    num_classes = len(train_generator.class_indices)
    print(f"Detected number of classes: {num_classes}")
```

```
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(img_height,␣
↪img_width, 3)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    model.compile(optimizer=Adam(), loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

    history = model.fit(
        train_generator,
        epochs=epochs,
        validation_data=validation_generator)

    model.save(model_path)

    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs_range = range(epochs)

    plt.figure(figsize=(8, 8))
    plt.subplot(1, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')
    plt.title('Training and Validation Accuracy')

    plt.subplot(1, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.title('Training and Validation Loss')
    plt.show()

else:
    print("Model found, loading the model...")
```

```
model = load_model(model_path)
class_labels = ['1 INR', '2 INR', '5 INR', '10 INR']

def preprocess_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (224, 224))
    img = img.astype('float32') / 255.0
    img = np.expand_dims(img, axis=0)
    return img

def predict_coin(image_path):
    processed_image = preprocess_image(image_path)
    predictions = model.predict(processed_image)
    predicted_class = np.argmax(predictions, axis=1)[0]
    return class_labels[predicted_class]

def upload_image():
    root = Tk()
    root.withdraw()
    image_path = filedialog.askopenfilename(title="Select an Image",␣
 ↪filetypes=[("Image Files", ".jpg;.jpeg;*.png")])
    if image_path:
        coin_type = predict_coin(image_path)
        print(f'The coin is: {coin_type}')

        img = cv2.imread(image_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.imshow(img)
        plt.title(f"Coin Type: {coin_type}")
        plt.axis('off')
        plt.show()

upload_image()
```

```
Model found, loading the model…

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.

1/1             0s 118ms/step
The coin is: 1 INR
```

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import ModelCheckpoint

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((train_images.shape[0], 28, 28, 1)).
 ↪astype('float32') / 255
test_images = test_images.reshape((test_images.shape[0], 28, 28, 1)).
 ↪astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
```

```
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

checkpoint = ModelCheckpoint('mnist_cnn_model.keras', save_best_only=True)

history = model.fit(train_images, train_labels, epochs=5, batch_size=64,␣
 ↪validation_data=(test_images, test_labels), callbacks=[checkpoint])

test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc:.4f}')
```

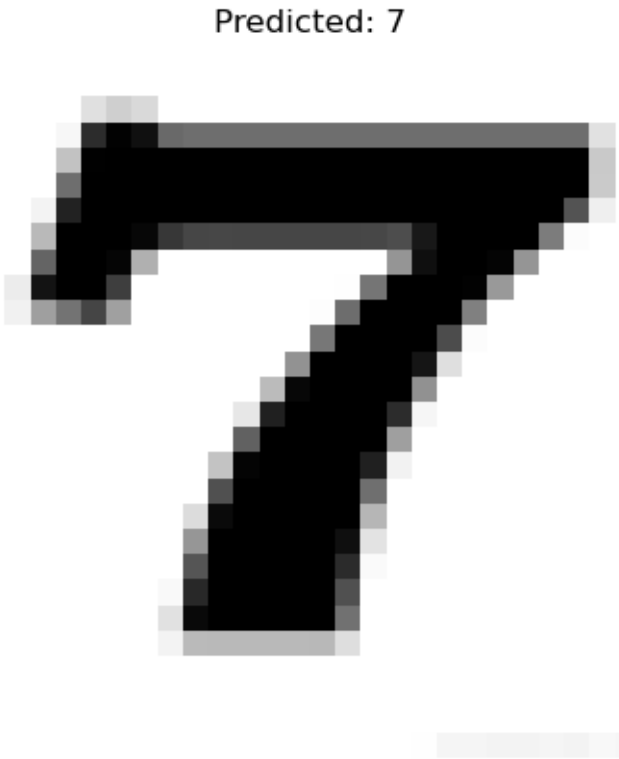
```
C:\Users\HP\anaconda3\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/5
938/938             14s 14ms/step -
accuracy: 0.8831 - loss: 0.3961 - val_accuracy: 0.9817 - val_loss: 0.0560
Epoch 2/5
938/938             14s 15ms/step -
accuracy: 0.9833 - loss: 0.0546 - val_accuracy: 0.9872 - val_loss: 0.0383
Epoch 3/5
938/938             13s 13ms/step -
accuracy: 0.9883 - loss: 0.0366 - val_accuracy: 0.9871 - val_loss: 0.0418
Epoch 4/5
938/938             13s 13ms/step -
accuracy: 0.9913 - loss: 0.0284 - val_accuracy: 0.9859 - val_loss: 0.0482
Epoch 5/5
938/938             13s 13ms/step -
accuracy: 0.9929 - loss: 0.0214 - val_accuracy: 0.9901 - val_loss: 0.0297
313/313             1s 3ms/step -
accuracy: 0.9868 - loss: 0.0400
Test accuracy: 0.9901
```

[4]:
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from PIL import Image

model = load_model('mnist_cnn_model.keras')
```

```python
def load_and_preprocess_image(img_path):
    img = Image.open(img_path).convert('L')
    img = img.resize((28, 28))
    img_array = np.array(img).astype('float32') / 255
    img_array = np.expand_dims(img_array, axis=0)
    img_array = np.expand_dims(img_array, axis=-1)
    return img_array

img_path = 'C:/Users/HP/Downloads/number7.png'
processed_image = load_and_preprocess_image(img_path)

prediction = model.predict(processed_image)
predicted_label = np.argmax(prediction)

print(f'The predicted label is: {predicted_label}')

plt.imshow(processed_image[0].squeeze(), cmap='gray')
plt.title(f'Predicted: {predicted_label}')
plt.axis('off')
plt.show()
```

```
1/1             0s 63ms/step
The predicted label is: 7
```



[ ]: