Deekonda Sairam
(232P4R1014)   LAB REPORT - 5

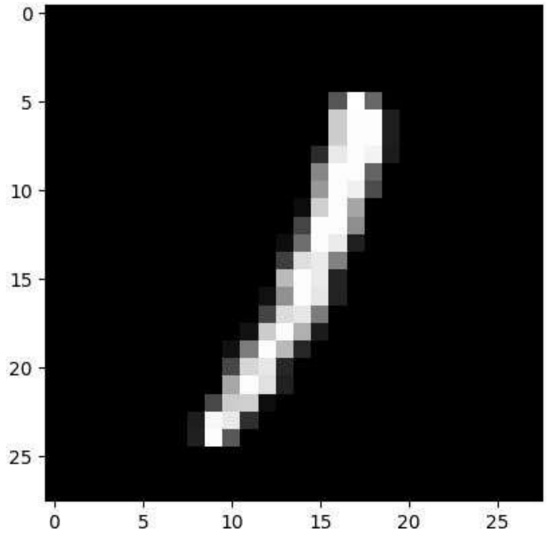
```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load and pnepnocess MNIST data
(train_images, train_labels), (test_images, test_labels) = mnist.1oad_data()

# Reshape aud normalize data
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

# One-hot encode labels
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Build CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layens.MaxPooling2D((2, 2)),

    layers.Couv2D(64, (3, 3), activation='relu'),
    layers.NaxPool lng2D((2, 2)),

    layens.Conv2D(64, (3, 3), activation='relu'),
    layens.Flatten(),
    layers.Deuse(64, activation='relu'),
    layers.Deuse(10, activation='softmax')


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

# Train model
model.fit(train_images, train_labels, epochs=5, batch_size=64, validatiou_split=0.1)

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.upz
l*4se434/1148e434━━━━━━━━━━━━━━━━━━━━ 1s Ous/step
Epoch 1/5
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an ’input_shape’/’inpu
  supen().finite(activity_negularizer-activity_regularizer, **kwargs)
8aa/844 ━━━━━━━━━━━━━━━━━━━━ 29s 33ms/step - accuracy: 0.8505 - loss: 0.4772 - val_accuracy: 0.9800 - val_loss: 0.B621
Epoch 2/5
844/844━━━━━━━━━━━━━━━━━━━━ 39s 30ms/step - accuracy: 0.9810 - loss: 0.0584 - val_accuracy: 0.9837 - val_loss: 0.B514
Epoch 3/5
844/844━━━━━━━━━━━━━━━━━━━━ 41s 30ms/step - accuracy: 0.9872 - loss: 0.0408 - val_accuracy: 0.9877 - val_loss: 0.0421
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 41s 30ms/step - accuracy: 0.9903 - loss: 0.0316 - val_accuracy: 0.9880 - val_loss: 0.0410
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 25s 29ms/step - accuracy: 0.9922 - loss: 0.0239 - val_accuracy: 0.9900 - val_loss: 0.0385
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9894 - loss: 0.0366
Test accuracy: 0.9917
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Dense, Dropout,Flatten,Conv2D,MaxPooling2D
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
(x_train,y_tnain),(x_test,y_test)=mnist.load_data()
print('Shape of training data',x_train.shape)
print('Shape of testing data',x_test.shape)
print('No. of training samples=',x_train.shape[0])
print('No. of testing samples=',x_test.shape[0])
x_train=x_traiu.reshape(60000,28,28,1).astype('float32') / 255
x_test=x_test.reshape(10000,28,28,1).astype('float32') / 255
from tensorflow.keras.utils import to_categorical
y_train-to_categorical(y_train,10)
```

```
y_test=to_categorical(y_test,10)
arr=y_tnain[99]

print(arr)
label=np.argmax(arr)
print(label)
import matplotlib.pyplot as plt
plt.imshow(x_train[99],cmap='gray')
plt.show()
model=Sequeutial()
#model.add(Deuse(32,input_shape=(28,28,1)))
model.add(Conv2D(32,(3,3),activation='relu’,input_shape=(28,28,1)))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.5))
model.add(Flatter())
model.add(Dense(10,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(x_tnain,y_train,epochs=10,batch_size=32,validation_data=(x_test,y_test))
val_loss,val_acc=model.evaluate(x_test,y_test)
print(val_loss,val_acc)
arr=model.predict([x_train[99].reshape(1,28,28,1)])
print(arr)
label=np.argmax(arr)
print(label)
```

```
Shape of training data (60000, 28, 28
Shape of testing data (10000, 28, 28)
No. of training samples= 60000
No. of testing samples= 10000
[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
1
```

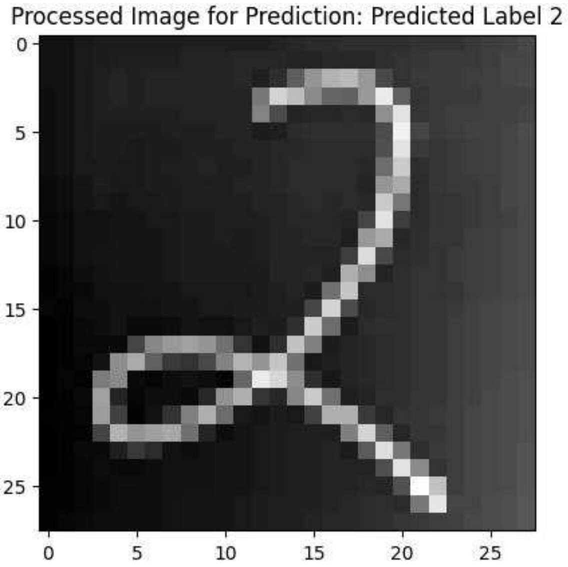


```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 84s 44ms/step - accuracy: 0.9015 - loss: 0.3161 - val_accuracy: 0.9830 - val_loss: B.0554
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 142s 44ms/step - accuracy: 0.9804 - loss: 0.0635 - val_accuracy: 0.9846 - val_loss: B.0479
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 141s 44ms/step - accuracy: 0.9839 - loss: 0.0512 - val_accuracy: 0.9869 - val_loss: B.0380
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 144s 45ms/step - accunacy: 0.9862 - loss: 0.0436 - val_accuracy: 0.9863 - val_loss: 8.0399
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 139s 43ms/step - accuracy: 0.9882 - loss: 0.0372 - val_accuracy: 0.9877 - val_loss: B.0370
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 83s 44ms/step - accuracy: 0.9900 - loss: 0.0318 - val_accuracy: 0.9885 - val_loss: B.0354
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 142s 44ms/step - accuracy: 0.9900 - loss: 0.0294 - val_accuracy: 0.9891 - val_loss: 8.0378
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 143s 44ms/step - accuracy: 0.9909 - loss: 0.0279 - val_accuracy: 0.9899 - val_loss: B.0365
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 142s 45ms/step - accuracy: 0.9923 - loss: 0.0242 - val_accuracy: 0.9892 - val_loss: 0.0344
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 141s 44ms/step - accuracy: 0.9923 - loss: 0.0238 - val_accuracy: 0.9893 - val_loss: B.0340
313/313 ━━━━━━━━━━━━━━━━━━━━ 4s l4ms/step - accuracy: 0.9871 - loss: 0.0430
0.034024473279714584 0.9893000l25885B1
*/* ━━━━━━━━━━━━━━━━━━━━ es s4ms/step
[[8.9165025e-10 9.9910849e-01 7.5B92402e-07 1.7847698e-07 2.2745512e-04
  1.3735315e-08 3.4700406e-11 6.6243258e-04 3.1780448e-07 4.3587787e-07]]

/usr/local/lib/python3.11/dist-packages/keras/src/models/functional.py:237: UserWarning: The structure of ’inputs’ doesn't match the
```

```
    Expected: keras_tensor_106
    Received: inputs=('Tensor(shape=(I, 28, 28, 1))',)
      warnings.warn(msg)
```

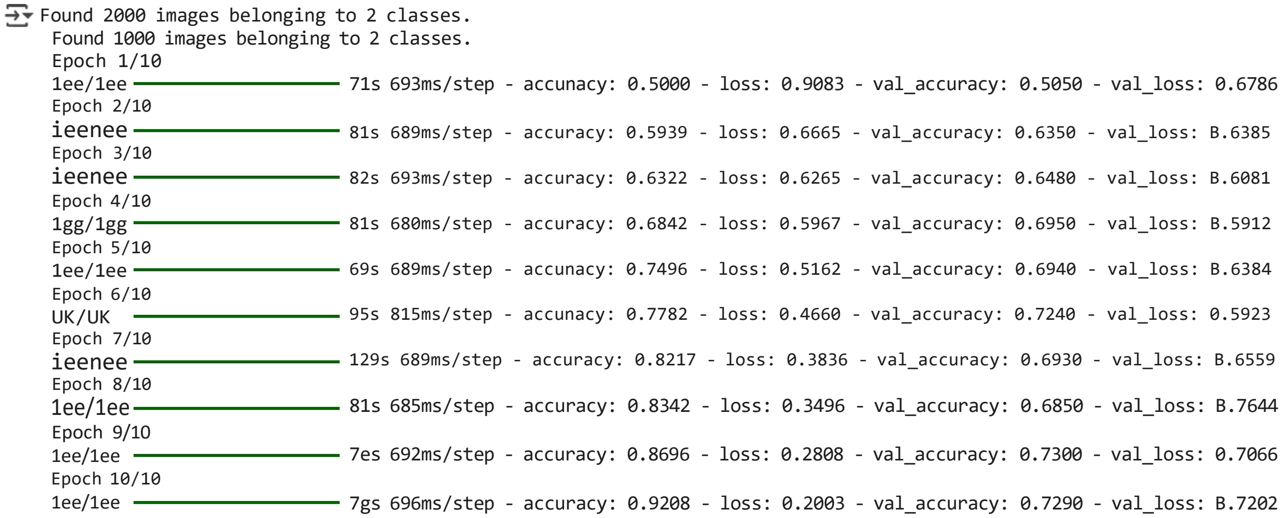
```
from PIL import Image,ImageOps
import numpy as np
import matplotlib.pyp1ot as plt
import tensorflow as tf # Import tensorflow

# Load the image and convert to grayscale
img = Image.open('/content/sample_data/imagetwo.jpg').convert('L')

# Invert the image (if needed for your model based on how MNIST data is structured - usually black digits on white background)
img_invented = Imageops.invert(img)

# Resize the image to 28x28
img_resized = im  inverted.resize((28, 28))

# Convert the resized image to a NumPy array
arr = np.array(im  resized)

# Normalize the pixel values to be between B and 1, similar to the training data
arr  = arr  / 255.0



# Reshape the array to match the model's expected input shape (batch_size, height, width, channels)
arr   = arr  . reshape( 1,  28,  28,  1)

# Predict using the trained model
predictions = model.predict(ann)
print(predictions)

# Get the predicted label
predicted_label = np.argmax(predictions)
print(f“Predicted label: {predicted_label}“)

# Display the processed image
plt.imshow(img_resized, cmap='gray')
plt.title(f“Processed Image for Prediction: Predicted Label {predicted_label}“)
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ is 25ms/step
[[0.06306896 0.02612549 0.36986455 0.2110239  0.B2089348 0.07947407
  0.06868032 0.02187605 0.11504704 0.02394615]]
Predicted label: 2
```

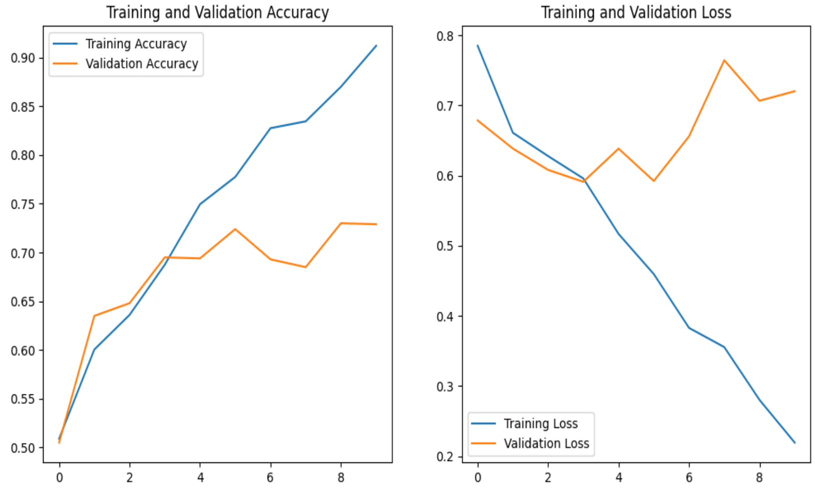


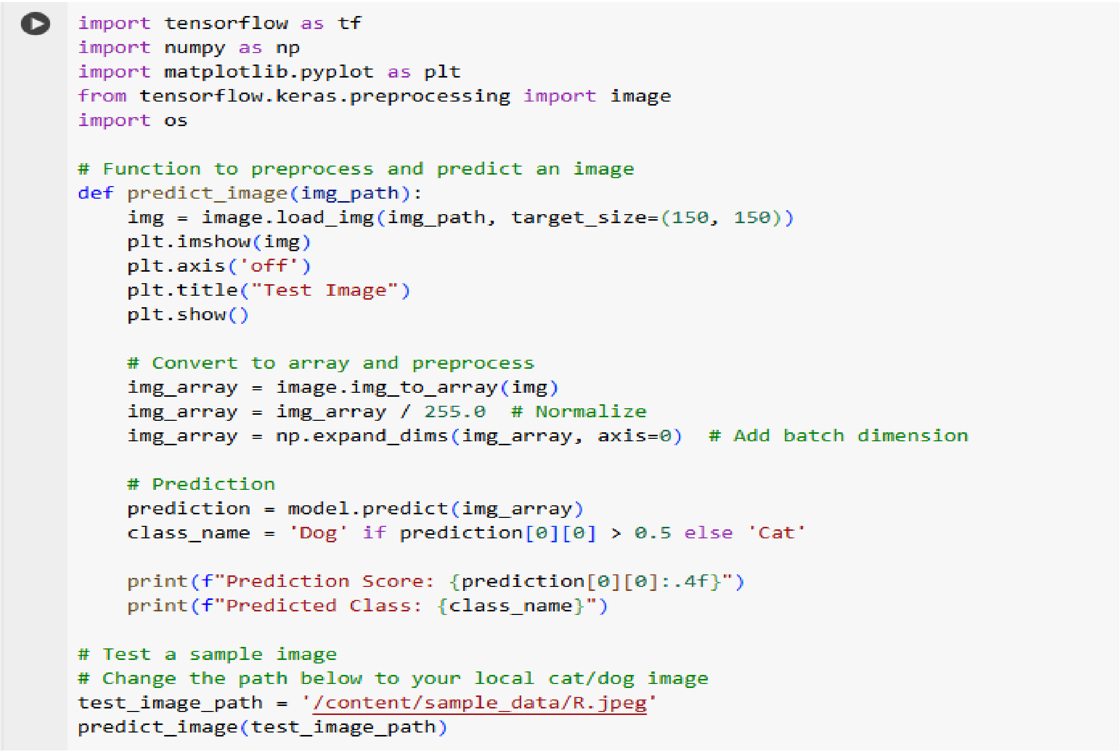
```
import tensonflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
import zipfile
from tensonflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
from teusorflow.keras.preprocessiug.image import ImageDataGenerator

# Step 1: Download and extract dataset
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtened.zip'
# Modified line: Use cache_dir to specify the directory
zip_path  = tf.keras.uti1s.get_fi1e('cats_and_dogs_filtered.zip', origin=_URL,  extract=True,  cache_dir='/content/datasets/')

base_dir = os.path.join('/content/datasets/cats and dogs filtered extracted/', 'cats_and_dogs_filtered') # Update base_dir to reflect the new
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

# Step 2: Set up ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1./255)
val_datageu = ImageDataGenerator(rescale=l./255)

train_generator = train_datagen.flow_from_directory(
    t ra ln_d lr,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'


validation_genenaton = val_datagen.flow_from_directony(
    validation_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'


# Step 3: Build CNN Model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(l5B, 150, 3)),
    MaxPooliug2D(2,  2),

    Conv2D(64, (3, 3), act lvatton='relu'),
    MaxPooling2D(2, 2),

    Conv2D(128, (3, 3), activation='relu'),
    HaxPooI1ng2D(2, 2),

    Dropout(0. 5),
    Flatten(),

    Dense(512, activatiou='relu'),
    Dense(1, activation='sigmoid')  # binary classification (cat/dog)


# Step 4: Compile the Model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Step 5: Train the Model
history = model.fit(
    traiu_geuerator,
    steps_per_epoch=100,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=50
```

```
# Step 6: Plot Accuracy & Loss
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_rauge = range(10)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
p1t.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legeud()
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()
plt.title('Training and Validation Loss')
plt.show()
```

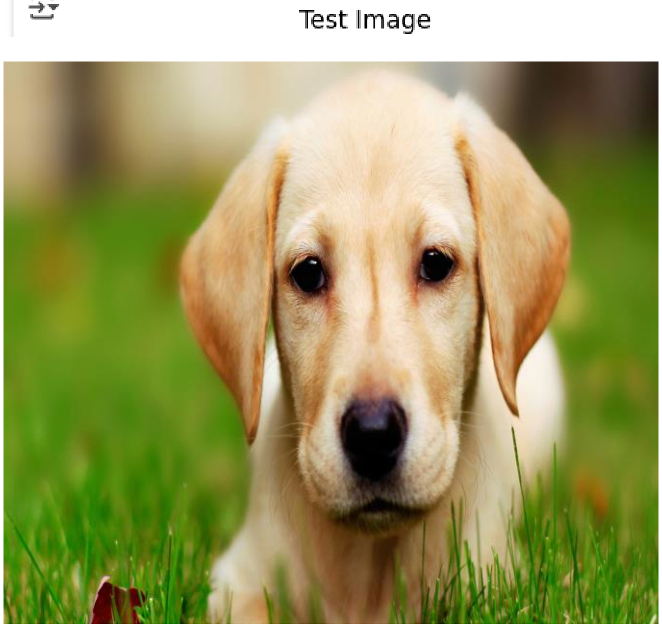


```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
Epoch 1/10
1ee/1ee ━━━━━━━━━━━━━━━━━━━━ 71s 693ms/step - accuracy: 0.5000 - loss: 0.9083 - val_accuracy: 0.5050 - val_loss: 0.6786
Epoch 2/10
ieenee ━━━━━━━━━━━━━━━━━━━━ 81s 689ms/step - accuracy: 0.5939 - loss: 0.6665 - val_accuracy: 0.6350 - val_loss: B.6385
Epoch 3/10
ieenee ━━━━━━━━━━━━━━━━━━━━ 82s 693ms/step - accuracy: 0.6322 - loss: 0.6265 - val_accuracy: 0.6480 - val_loss: B.6081
Epoch 4/10
1gg/1gg ━━━━━━━━━━━━━━━━━━━━ 81s 680ms/step - accuracy: 0.6842 - loss: 0.5967 - val_accuracy: 0.6950 - val_loss: B.5912
Epoch 5/10
1ee/1ee ━━━━━━━━━━━━━━━━━━━━ 69s 689ms/step - accuracy: 0.7496 - loss: 0.5162 - val_accuracy: 0.6940 - val_loss: B.6384
Epoch 6/10
UK/UK ━━━━━━━━━━━━━━━━━━━━ 95s 815ms/step - accuracy: 0.7782 - loss: 0.4660 - val_accuracy: 0.7240 - val_loss: 0.5923
Epoch 7/10
ieenee ━━━━━━━━━━━━━━━━━━━━ 129s 689ms/step - accuracy: 0.8217 - loss: 0.3836 - val_accuracy: 0.6930 - val_loss: B.6559
Epoch 8/10
1ee/1ee ━━━━━━━━━━━━━━━━━━━━ 81s 685ms/step - accuracy: 0.8342 - loss: 0.3496 - val_accuracy: 0.6850 - val_loss: B.7644
Epoch 9/10
1ee/1ee ━━━━━━━━━━━━━━━━━━━━ 7es 692ms/step - accuracy: 0.8696 - loss: 0.2808 - val_accuracy: 0.7300 - val_loss: 0.7066
Epoch 10/10
1ee/1ee ━━━━━━━━━━━━━━━━━━━━ 7gs 696ms/step - accuracy: 0.9208 - loss: 0.2003 - val_accuracy: 0.7290 - val_loss: B.7202
```





```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
import os

# Function to preprocess and predict an image
def predict_image(img_path):
    img = image.load_img(img_path, target_size=(150, 150))
    plt.imshow(img)
    plt.axis('off')
    plt.title("Test Image")
    plt.show()

    # Convert to array and preprocess
    img_array = image.img_to_array(img)
    img_array = img_array / 255.0   # Normalize
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    # Prediction
    prediction = model.predict(img_array)
    class_name = 'Dog' if prediction[0][0] > 0.5 else 'Cat'

    print(f"Prediction Score: {prediction[0][0]:.4f}")
    print(f"Predicted Class: {class_name}")

# Test a sample image
# Change the path below to your local cat/dog image
test_image_path = '/content/sample_data/R.jpeg'
predict_image(test_image_path)
```



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
Prediction Score: 0.5702
Predicted Class: Dog
```