Requirements:

1.Check Python version:

```
python - -version
```

Or

```
python3 - -version
```

2. Install Backend Dependencies (Flask, Pandas, TensorFlow, OpenCV, etc.):

For Windows (cmd/PowerShell)

```
pip install flask

pip install pandas

pip install numpy

pip install tensorflow

pip intsall keras

pip install opencv-python

pip install pillow

pip install flask-cors
```

For Mac/Linux (Terminal)

```
pip3 install flask

pip3 install pandas

pip3 install numpy

pip3 install tensorflow

pip3 install keras

pip3 install opencv-python

pip3 install  pillow

pip3 install flask-cors
```

commands to run the codes:

for app.py:-

```
python app.py
```

or

```
python3 app.py
```

for pcos_test.py:-

```
python3 pcos_test.py
```

or

```
python pcos_test.py
```

for test_model.py:-

```
python3 test_model.py
```

or

```
python test_model.py
```

to train the model in colab for pcos_model.py:-

```
python3 pcos_model.py
```

or

```
python pcos_model.py
```

```
pcos_detection/
|— static/              # Stores CSS, JavaScript, and images
|   ├— styles.css        # Your CSS file
|— templates/           # Stores HTML templates
|   ├— index.html        # Main page with upload form
|   ├— result.html       # Page to display results
|— uploads/             # Stores uploaded images (created dynamically)
|— pcos_model.h5         # Your trained model(not stored under same root folder)
|— app.py               # Flask backend
```

pcos_model.py code:{executed in colab }

```
import os
import numpy as np # type: ignore
from PIL import Image # type: ignore
from tensorflow.keras.models import Sequential # type: ignore
from tensorflow.keras.layers import Dense, Flatten # type: ignore
from tensorflow.keras.applications import VGG16 # type: ignore
from tensorflow.keras.preprocessing.image import ImageDataGenerator # type: ignore

# Define paths
train_path = "/content/drive/My Drive/pcos_dataset/train"
test_path = "/content/drive/My Drive/pcos_dataset/test"

# Function to validate image files
def validate_images(directory_path):
    """
    Removes invalid or corrupted images from the specified directory.
    """
    for root, _, files in os.walk(directory_path):
        for file in files:
            file_path = os.path.join(root, file)
            try:
                img = Image.open(file_path)
                img.verify()  # Check if the image is valid
            except (IOError, SyntaxError):
                print(f"Invalid image file detected and removed: {file_path}")
                os.remove(file_path)

# Validate images in train and test directories
validate_images(train_path)
validate_images(test_path)

# Data augmentation for training and testing
train_datagen = ImageDataGenerator(rescale=1.0 / 255)
test_datagen = ImageDataGenerator(rescale=1.0 / 255)
```

```
# Loading the data
train_data = train_datagen.flow_from_directory(
    train_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    color_mode='rgb'  # Ensure consistent color handling
)

test_data = test_datagen.flow_from_directory(
    test_path,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    color_mode='rgb'
)

# Load VGG16 pre-trained model without the top layers
vgg_base = VGG16(include_top=False, weights='imagenet', input_shape=(224, 224, 3))

# Freeze the base model layers
vgg_base.trainable = False

# Build the full model
model = Sequential([
    vgg_base,
    Flatten(),
    Dense(256, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification: PCOS detection
])
```

```
# Compile the model

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


# Train the model

history = model.fit(train_data, validation_data=test_data, epochs=10)


# Save the trained model

model.save("pcos_detection_model.h5")


print("Model training complete and saved successfully.")
```

O/p:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5

58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step

/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.

  self._warn_if_super_not_called()

Epoch 1/10

WARNING: All log messages before absl::InitializeLog() is called are written to STDERR

I0000 00:00:1739369872.236601    6701 service.cc:148] XLA service 0x7c9a28003a40 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:

I0000 00:00:1739369872.236749    6701 service.cc:156]   StreamExecutor device (0): Tesla T4, Compute Capability 7.5

I0000 00:00:1739369872.681577    6701 cuda_dnn.cc:529] Loaded cuDNN version 90300

I0000 00:00:1739369883.801089    6701 device_compiler.h:188] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.

61/61 ━━━━━━━━━━━━━━━━━━━━ 54s 687ms/step - accuracy: 0.8344 - loss: 0.7375 - val_accuracy: 1.0000 - val_loss: 8.5628e-05
```

```
Epoch 2/10

61/61 ━━━━━━━━━━━━━━━━━━━━ 23s 383ms/step - accuracy: 1.0000 - loss:
8.4523e-05 - val_accuracy: 1.0000 - val_loss: 7.3273e-05

Epoch 3/10

61/61 ━━━━━━━━━━━━━━━━━━━━ 23s 382ms/step - accuracy: 1.0000 - loss:
6.2634e-05 - val_accuracy: 1.0000 - val_loss: 6.3917e-05

Epoch 4/10

61/61 ━━━━━━━━━━━━━━━━━━━━ 23s 373ms/step - accuracy: 1.0000 - loss:
7.5144e-05 - val_accuracy: 1.0000 - val_loss: 5.5912e-05

Epoch 5/10

61/61 ━━━━━━━━━━━━━━━━━━━━ 24s 386ms/step - accuracy: 1.0000 - loss:
4.3588e-05 - val_accuracy: 1.0000 - val_loss: 4.9441e-05

Epoch 6/10

61/61 ━━━━━━━━━━━━━━━━━━━━ 23s 379ms/step - accuracy: 1.0000 - loss:
5.6029e-05 - val_accuracy: 1.0000 - val_loss: 4.3319e-05

Epoch 7/10

61/61 ━━━━━━━━━━━━━━━━━━━━ 24s 396ms/step - accuracy: 1.0000 - loss:
4.1863e-05 - val_accuracy: 1.0000 - val_loss: 3.8467e-05

Epoch 8/10

61/61 ━━━━━━━━━━━━━━━━━━━━ 40s 383ms/step - accuracy: 1.0000 - loss:
3.7175e-05 - val_accuracy: 1.0000 - val_loss: 3.4364e-05

Epoch 9/10

61/61 ━━━━━━━━━━━━━━━━━━━━ 32s 530ms/step - accuracy: 1.0000 - loss:
3.3562e-05 - val_accuracy: 1.0000 - val_loss: 3.0530e-05

Epoch 10/10

61/61 ━━━━━━━━━━━━━━━━━━━━ 24s 390ms/step - accuracy: 1.0000 - loss:
3.0715e-05 - val_accuracy: 1.0000 - val_loss: 2.7429e-05

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')`.

Model training complete and saved successfully.
```

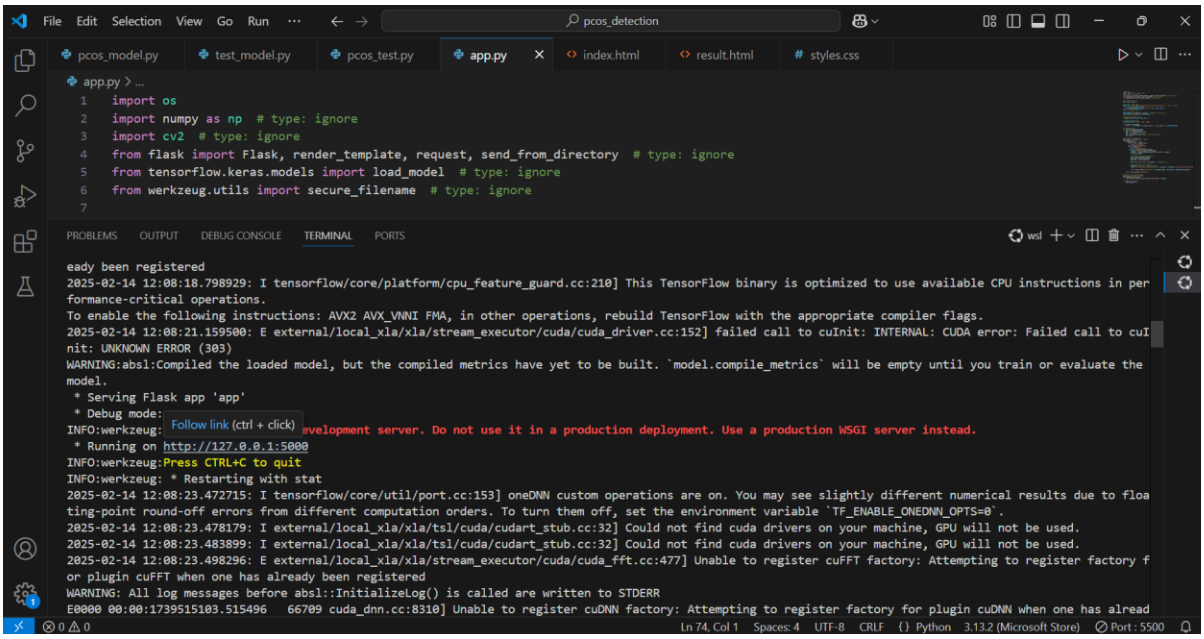
app.py code:

```
import os
import numpy as np  # type: ignore
import cv2  # type: ignore
from flask import Flask, render_template, request, send_from_directory  # type: ignore
from tensorflow.keras.models import load_model  # type: ignore
from werkzeug.utils import secure_filename  # type: ignore

# Initialize Flask app
app = Flask(__name__)

# Load the trained model
MODEL_PATH = "/mnt/c/Users/navya/Documents/pcos_detection_model.h5"  # Adjust if needed
if not os.path.exists(MODEL_PATH):
    raise FileNotFoundError(f"Model file not found: {MODEL_PATH}")
model = load_model(MODEL_PATH)

# Configure upload folder
UPLOAD_FOLDER = os.path.join(os.getcwd(), "uploads")  # Dynamic path
app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER

# Ensure upload folder exists
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

# Allowed file extensions
ALLOWED_EXTENSIONS = {"png", "jpg", "jpeg"}

def allowed_file(filename):
    return "." in filename and filename.rsplit(".", 1)[1].lower() in ALLOWED_EXTENSIONS

# Preprocess image
```

```
def preprocess_image(image_path):
    img = cv2.imread(image_path)
    img = cv2.resize(img, (224, 224))
    img = img / 255.0  # Normalize
    img = np.expand_dims(img, axis=0)  # Add batch dimension
    return img

# Home page with upload form
@app.route("/", methods=["GET", "POST"])
def upload_file():
    if request.method == "POST":
        if "file" not in request.files:
            return "No file part"
        file = request.files["file"]
        if file.filename == "":
            return "No selected file"
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            filepath = os.path.join(app.config["UPLOAD_FOLDER"], filename)
            file.save(filepath)

            # Preprocess image and make prediction
            img = preprocess_image(filepath)
            prediction = model.predict(img)
            raw_output = prediction[0][0]

            print(f"Raw model output: {raw_output}")  # Debugging line

            # Flip the condition if necessary
            threshold = 0.5  # Adjust based on training
            result = "Not Infected (No PCOS detected)" if raw_output >= threshold else "Infected (PCOS detected)"

            return render_template("result.html", filename=filename, result=result, probability=raw_output)

    return render_template("index.html")

# Route to serve uploaded images
@app.route("/uploads/<filename>")
def uploaded_file(filename):
    return send_from_directory(app.config["UPLOAD_FOLDER"], filename)

if __name__ == "__main__":
    app.run(debug=True)
```

o/p:

* Running on http://127.0.0.1:5000



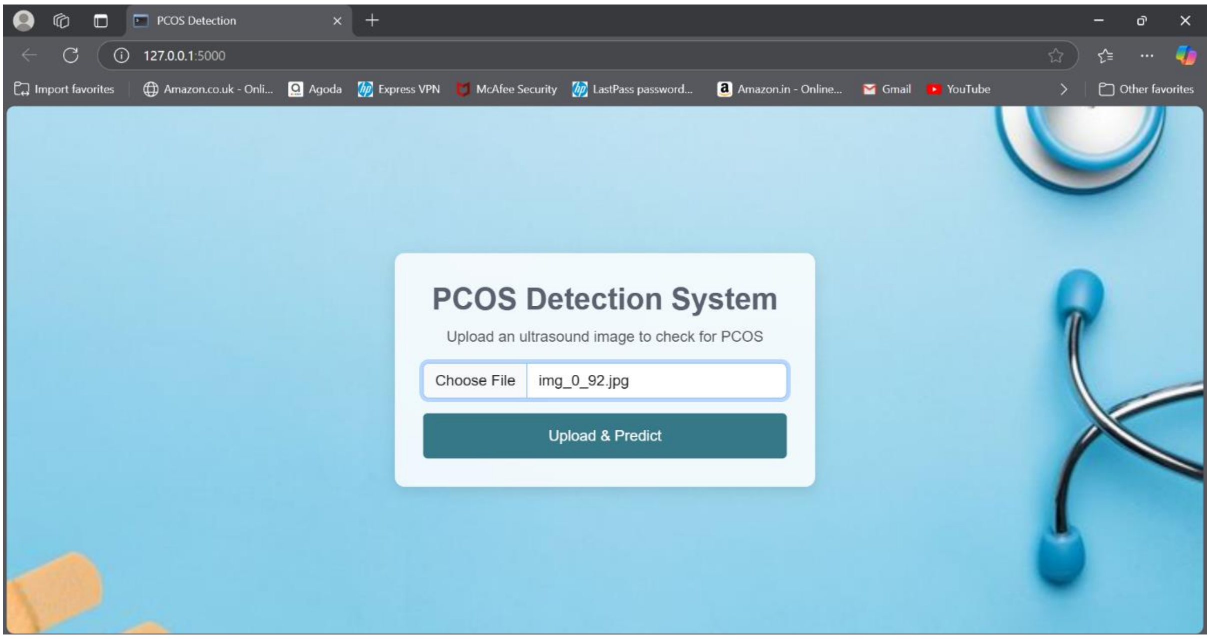
```
import os
import numpy as np  # type: ignore
import cv2  # type: ignore
from flask import Flask, render_template, request, send_from_directory  # type: ignore
from tensorflow.keras.models import load_model  # type: ignore
from werkzeug.utils import secure_filename  # type: ignore
```

```
eady been registered
2025-02-14 12:08:18.798929: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in per
formance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2025-02-14 12:08:21.159500: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152] failed call to cuInit: INTERNAL: CUDA error: Failed call to cuI
nit: UNKNOWN ERROR (303)
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.
 * Serving Flask app 'app'
 * Debug mode:
INFO:werkzeug:                        evelopment server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
2025-02-14 12:08:23.472715: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floa
ting-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-02-14 12:08:23.478179: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
2025-02-14 12:08:23.483899: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
2025-02-14 12:08:23.498296: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT factory: Attempting to register factory f
or plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1739515103.515496   66709 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has alread
```

index.html code:

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>PCOS Detection</title>

    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">

</head>

<body>

    <div class="container">

        <div class="upload-box">

            <h2 class="text-center">PCOS Detection System</h2>

            <p class="text-muted text-center">Upload an ultrasound image to check for PCOS</p>


            <form action="/" method="post" enctype="multipart/form-data" class="text-center">

                <input type="file" name="file" accept="image/*" class="form-control" required>

                <button type="submit" class="btn btn-primary mt-3 w-100">Upload & Predict</button>

            </form>

        </div>

    </div>

</body>

</html>
```

O/p:



PCOS Detection System

Upload an ultrasound image to check for PCOS

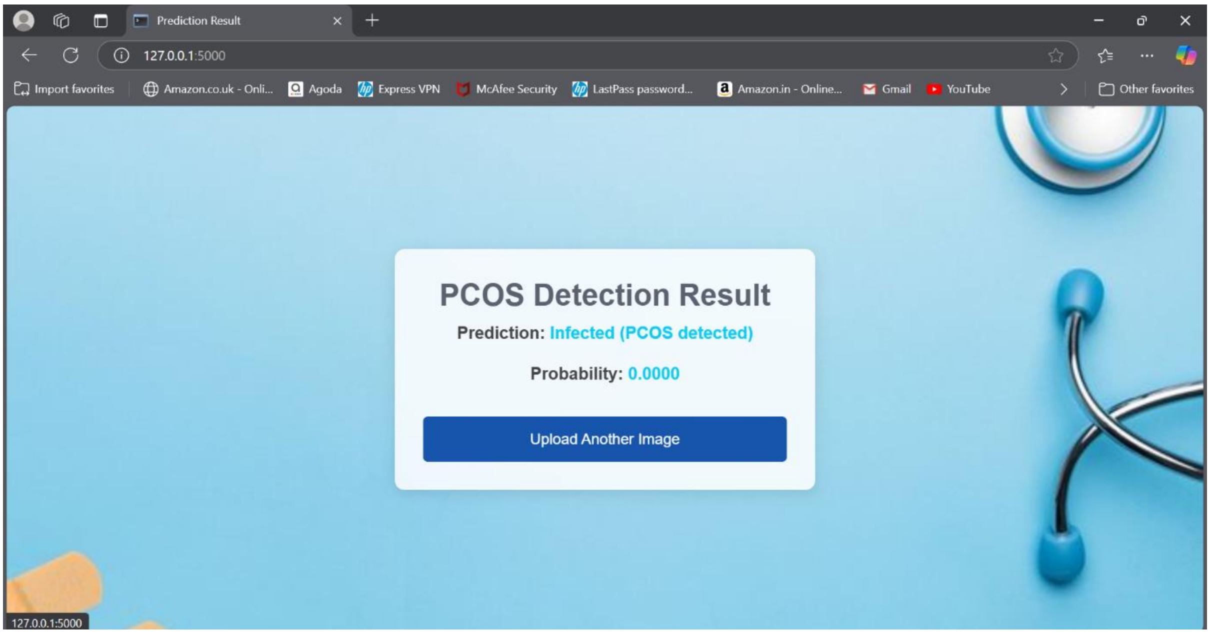
img_0_92.jpg

result.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Prediction Result</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <div class="container">
    <div class="result-box">
      <h2 class="text-center">PCOS Detection Result</h2>
      <p class="result-text"><strong>Prediction:</strong> <span class="text-info">{{ result }}</span></p>
      <p class="result-text"><strong>Probability:</strong> <span class="text-info">{{ "%.4f"|format(probability) }}</span></p>
```

```
            <div class="text-center">
                <a href="/" class="btn btn-success mt-3 w-100">Upload Another Image</a>
            </div>
        </div>
    </div>
</body>
</html>
```

o/p:



PCOS Detection Result

Prediction: Infected (PCOS detected)

Probability: 0.0000

style.css code:

```
body {
    background: url('https://iili.io/2pfCzSp.jpg') no-repeat center center/cover;
    font-family: 'Arial', sans-serif;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
```

```
  align-items: center;
  height: 100vh;
}

.container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

.upload-box, .result-box {
  background: rgba(255, 255, 255, 0.85); /* Soft white with transparency */
  padding: 30px;
  border-radius: 10px;
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
  width: 40%;
  text-align: center;
  backdrop-filter: blur(10px); /* Light blur for a soft effect */
}

h2 {
  color: #5a6270; /* Muted dark shade for contrast */
  font-weight: bold;
}

.result-text {
  font-size: 18px;
  color: #444;
}
```

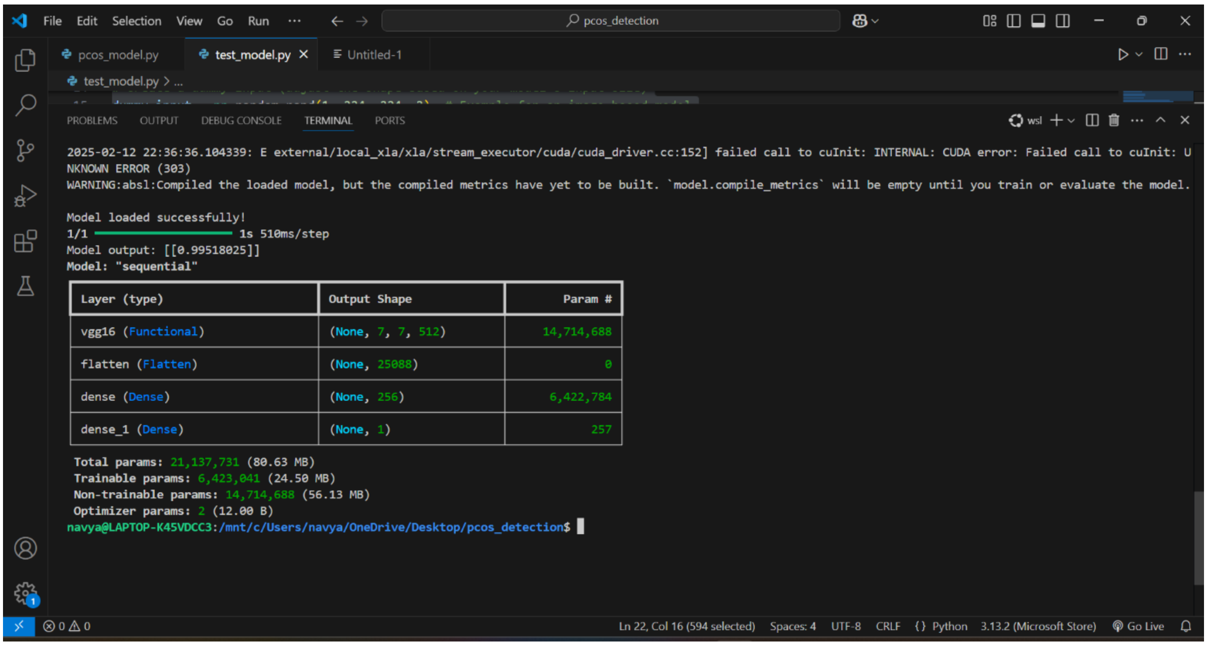
```
.text-info {
   font-weight: bold;
   color: #4ea69e; /* Soft pastel green for highlights */
}

button, .btn {
   padding: 12px 20px;
   font-size: 16px;
   border-radius: 5px;
   background: #377888; /* Soft pastel pink */
   color: white;
   border: none;
   cursor: pointer;
   transition: all 0.3s ease-in-out;
}

button:hover, .btn:hover {
   background: #0044a4; /* Slightly darker pink on hover */
   opacity: 0.9;
}
```

test_model.py

```
from tensorflow.keras.models import load_model # type: ignore

# Define the model path
model_path = "/home/navya/pcos_detection_model.h5"

# Load the trained model
model = load_model(model_path)
```

```
# Check if model loads successfully
print("Model loaded successfully!")

import numpy as np # type: ignore

# Create a dummy input (adjust the shape based on your model's input size)
dummy_input = np.random.rand(1, 224, 224, 3)  # Example for an image-based model

# Make a prediction
prediction = model.predict(dummy_input)

print("Model output:", prediction)

model.summary()
```

O/p:



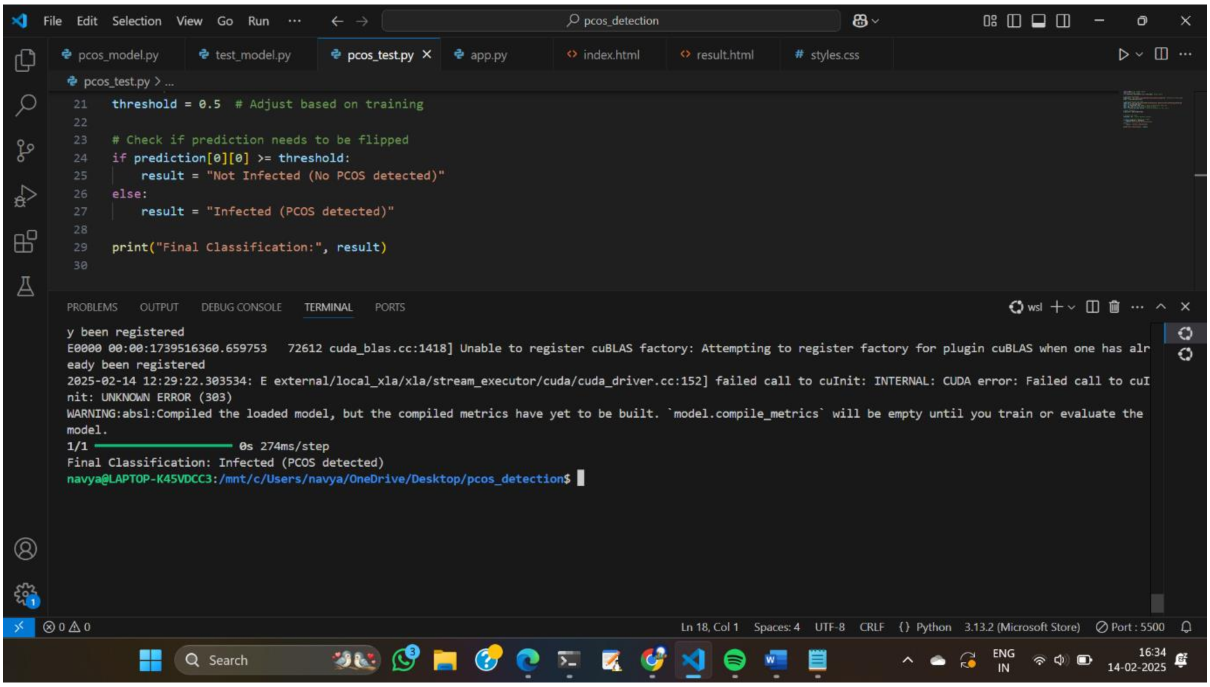
```
2025-02-12 22:36:36.104339: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152] failed call to cuInit: INTERNAL: CUDA error: Failed call to cuInit: U
NKNOWN ERROR (303)
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.

Model loaded successfully!
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 510ms/step
Model output: [[0.99518025]]
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━┓
┃ Layer (type)                 ┃ Output Shape         ┃     Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━┩
│ vgg16 (Functional)           │ (None, 7, 7, 512)    │  14,714,688 │
├──────────────────────────────┼──────────────────────┼─────────────┤
│ flatten (Flatten)            │ (None, 25088)        │           0 │
├──────────────────────────────┼──────────────────────┼─────────────┤
│ dense (Dense)                │ (None, 256)          │   6,422,784 │
├──────────────────────────────┼──────────────────────┼─────────────┤
│ dense_1 (Dense)              │ (None, 1)            │         257 │
└──────────────────────────────┴──────────────────────┴─────────────┘
 Total params: 21,137,731 (80.63 MB)
 Trainable params: 6,423,041 (24.50 MB)
 Non-trainable params: 14,714,688 (56.13 MB)
 Optimizer params: 2 (12.00 B)
navya@LAPTOP-K45VDCC3:/mnt/c/Users/navya/OneDrive/Desktop/pcos_detection$
```

pcos_test.py:

```
import numpy as np  # type: ignore
import cv2  # type: ignore
from tensorflow.keras.models import load_model  # type: ignore

# Load the trained model
model_path = "/mnt/c/Users/navya/Documents/pcos_detection_model.h5"  # Ensure the correct path
model = load_model(model_path)

# Load and preprocess the image
image_path = "/mnt/c/Users/navya/OneDrive/Desktop/pcos_detection/test/infected/img_0_245.jpg"
img = cv2.imread(image_path)
img = cv2.resize(img, (224, 224))  # Resize to match model input size
img = img / 255.0  # Normalize pixel values (0 to 1)
img = np.expand_dims(img, axis=0)  # Add batch dimension (1, 224, 224, 3)

# Make a prediction
prediction = model.predict(img)

# Interpret the result
threshold = 0.5  # Adjust based on training

# Check if prediction needs to be flipped
if prediction[0][0] >= threshold:
    result = "Not Infected (No PCOS detected)"
```

```
else:
    result = "Infected (PCOS detected)"

print("Final Classification:", result)
```

O/p:



```
threshold = 0.5  # Adjust based on training

# Check if prediction needs to be flipped
if prediction[0][0] >= threshold:
    result = "Not Infected (No PCOS detected)"
else:
    result = "Infected (PCOS detected)"

print("Final Classification:", result)
```

```
y been registered
E0000 00:00:1739516360.659753   72612 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has alr
eady been registered
2025-02-14 12:29:22.303534: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152] failed call to cuInit: INTERNAL: CUDA error: Failed call to cuI
nit: UNKNOWN ERROR (303)
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 274ms/step
Final Classification: Infected (PCOS detected)
navya@LAPTOP-K45VDCC3:/mnt/c/Users/navya/OneDrive/Desktop/pcos_detection$ 
```