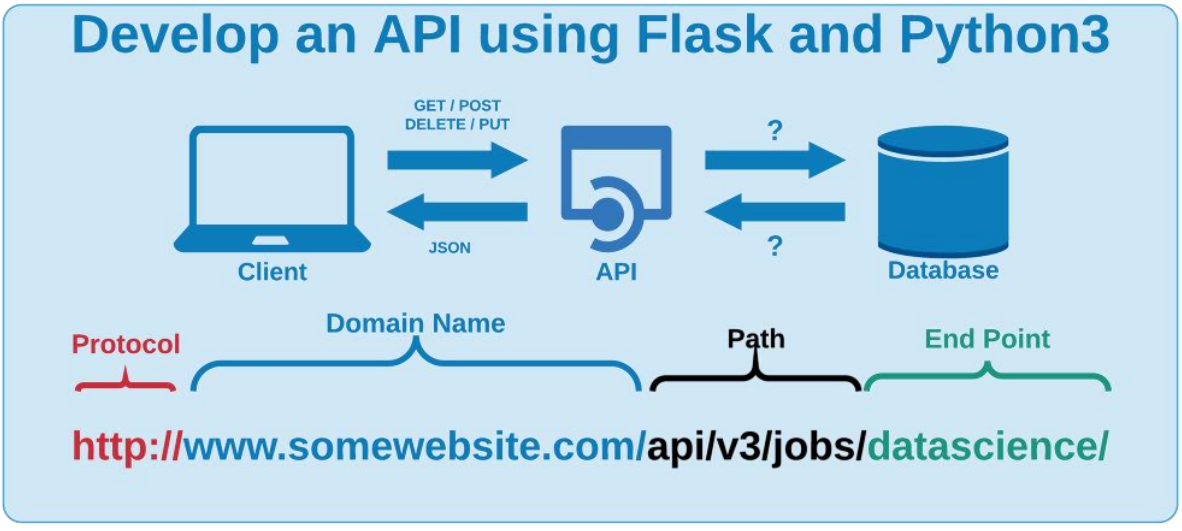
Deploying a python/TensorFlow model on web server using Flask

What is Flask?

Flask is a web framework that provides libraries to build lightweight web applications in python. It is developed by **Armin Ronacher** who leads an international group of python enthusiasts (POCCO). It is based on WSGI toolkit and jinja2 template engine. Flask is considered as a micro framework.



- ## Getting Started

1. We can install the flask by using the following command.

```
$ PIP INSTALL FLASK
```

2. The code lets us run a basic web application that we can serve, as if it were a website.

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Hello, World!"

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

This piece of code is stored in our main.py.

Line 1: Here we are importing the Flask module and creating a Flask web server from the Flask module.

Line 3: __name__ means this current file. In this case, it will be main.py. This current file will represent my web application.
We are creating an instance of the Flask class and calling it app. Here we are creating a new web application.

Line 5: It represents the default page

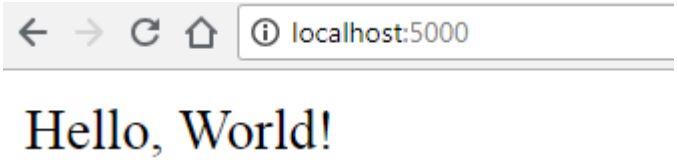
Line 9: When you run your Python script, Python assigns the name "__main__" to the script when executed.
If we import another script, the **if statement will prevent other scripts from running.** When we run main.py, it will change its name to __main__ and only then will that if statement activate.

Line 10: This will run the application. Having debug=True allows possible Python errors to appear on the web page. This will help us trace the errors.

Running main.py

```
In [*]: from flask import Flask

        app = Flask(__name__)

        @app.route("/")
        def home():
            return "Hello, World!"

        if __name__ == "__main__":
            app.run(debug=True, use_reloader=False)

         * Serving Flask app "__main__" (lazy loading)
         * Environment: production
           WARNING: This is a development server. Do not use it in a production deployment.
           Use a production WSGI server instead.
         * Debug mode: on

         * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

In [ ]:
```

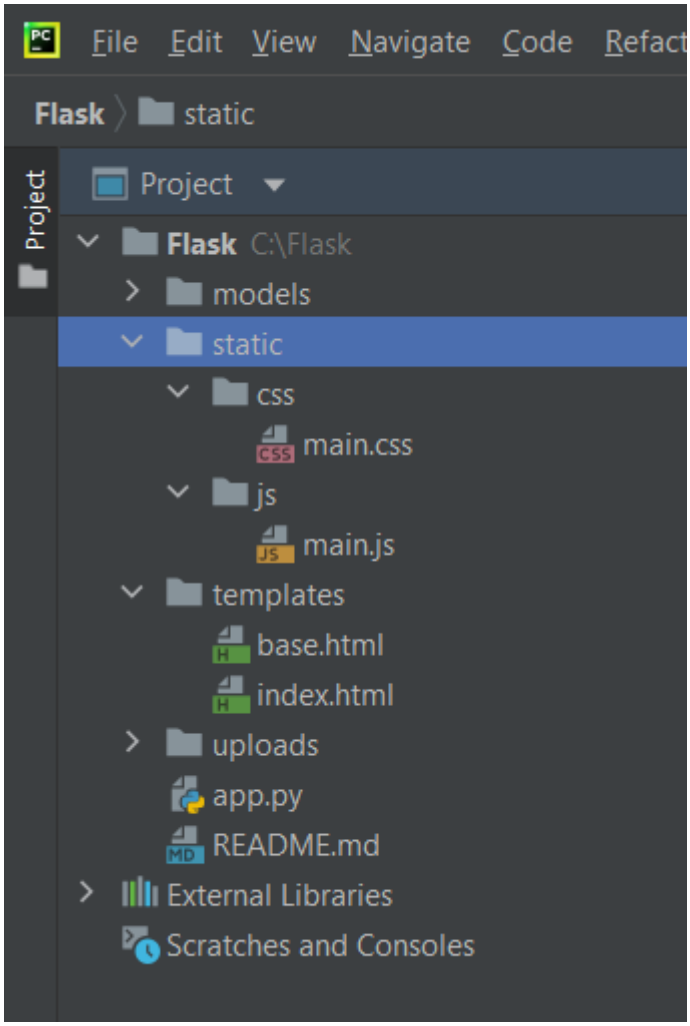
Go to that address and you should see the following:



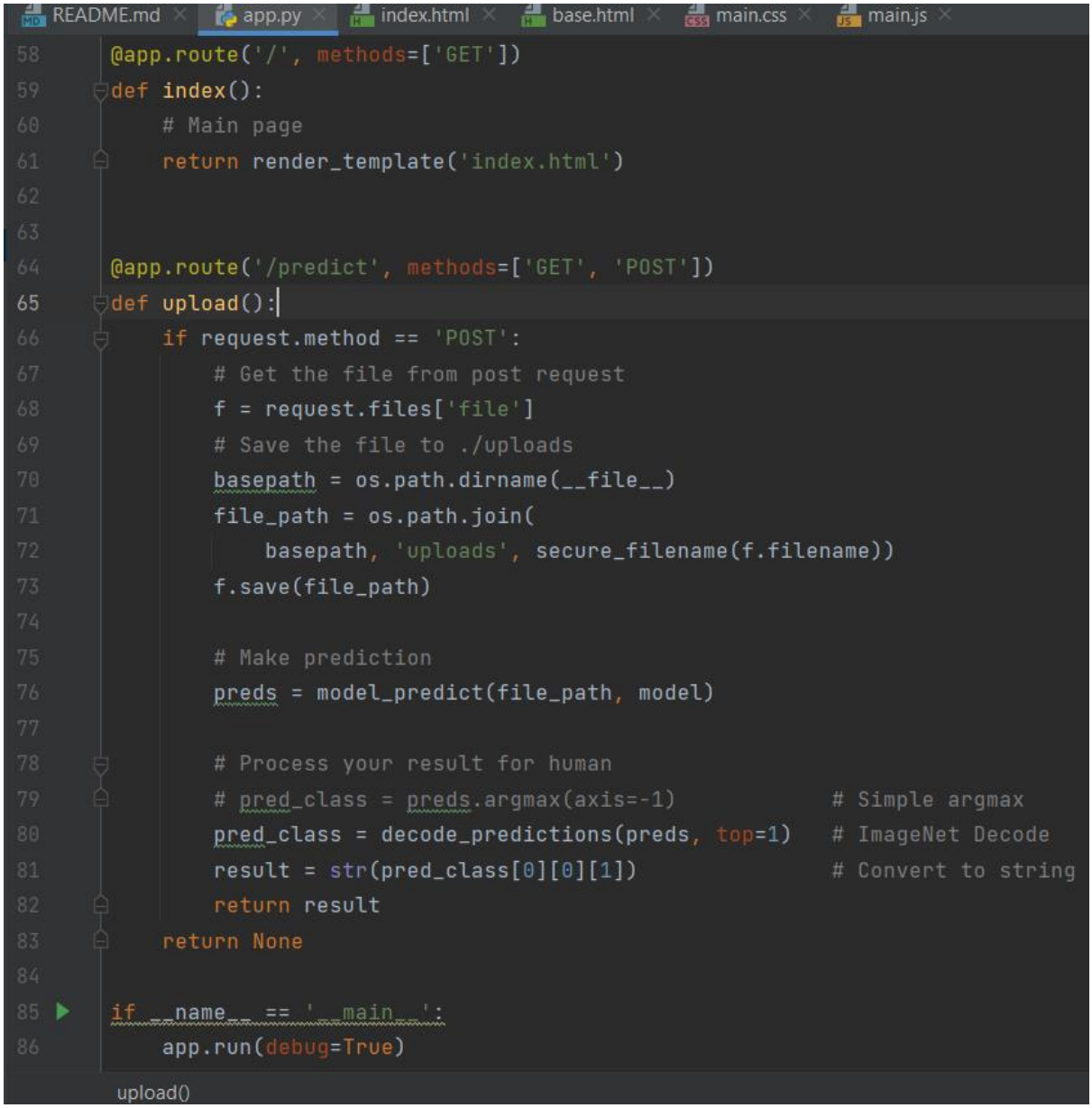
Deploying a Machine Learning Model of Animal Image Classifier to a web server using Flask

HTML, CSS, and Virtual Environments

We have to manually create following folders into your project folder and put all html, css and js files in respective folder as shown below. Remember to keep .py files outside the folders.

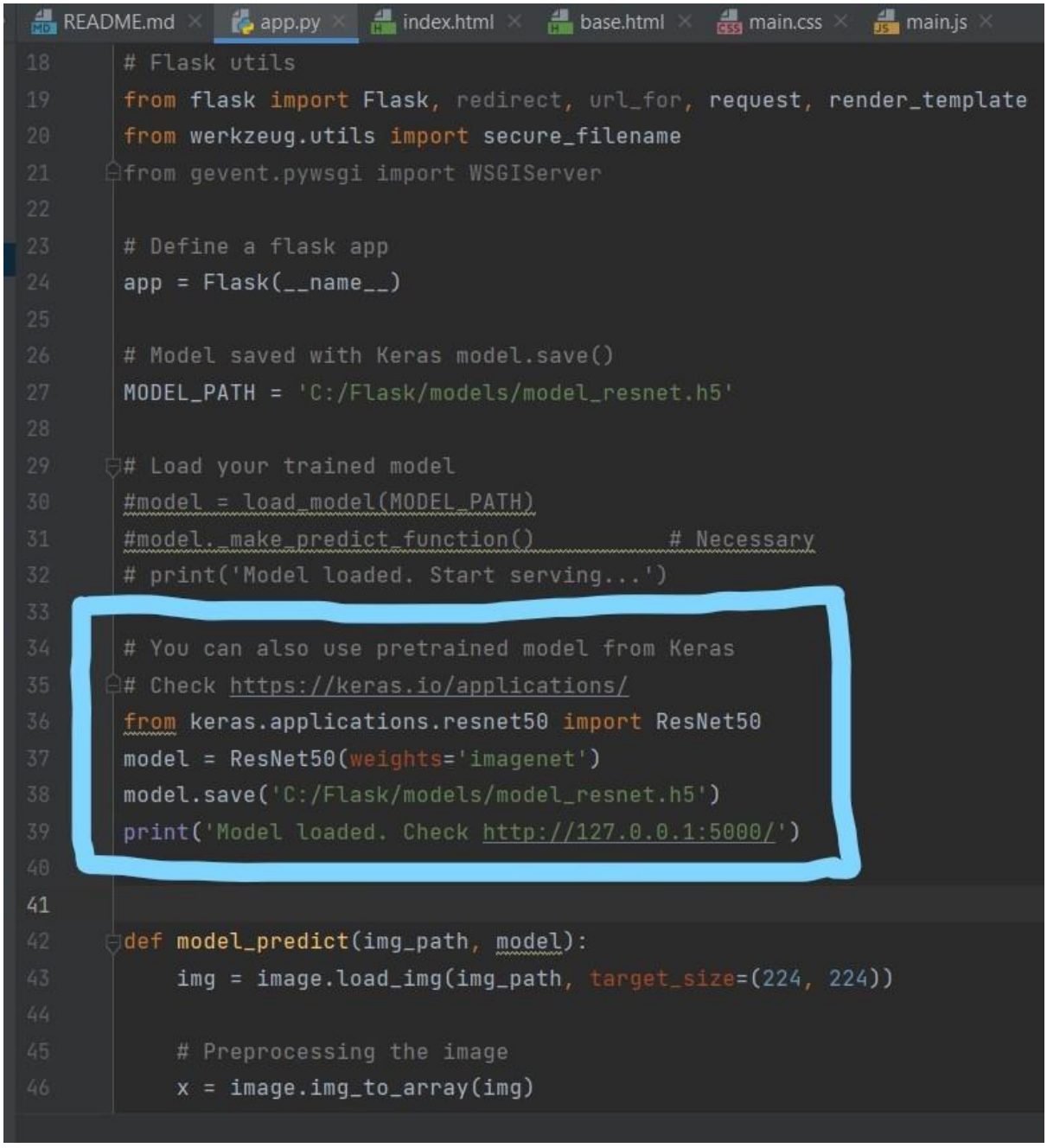


- Now we need to change our main.py so that we can view the HTML file we created. Also, we can use GET & POST to take input from user and fed it to code for processing.

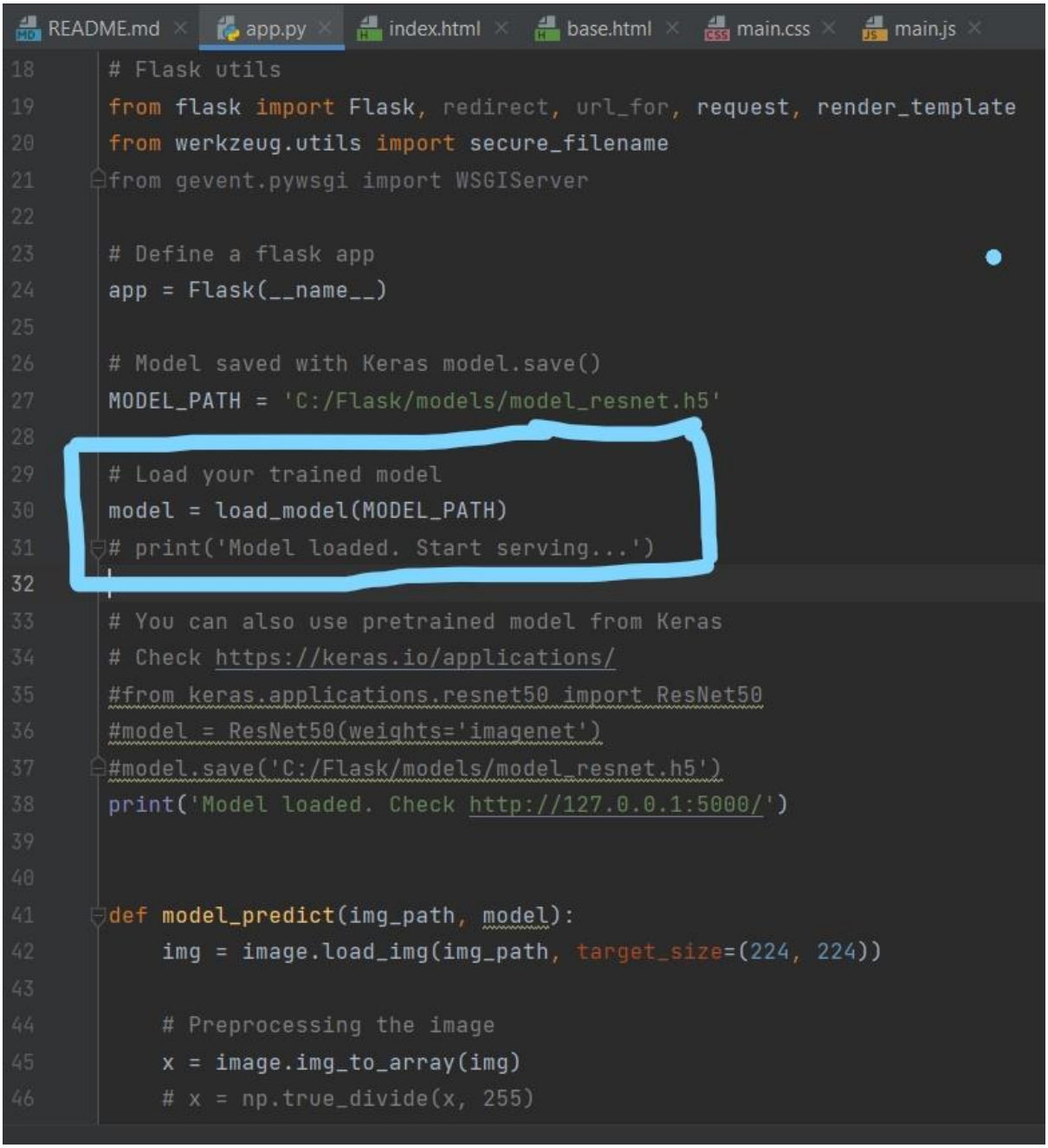


Inside the python code file i.e. app.py

-First, we have to import and save '**model_resnet50.h5**' file to 'models' folder. Below code will automatically do that once run at first.

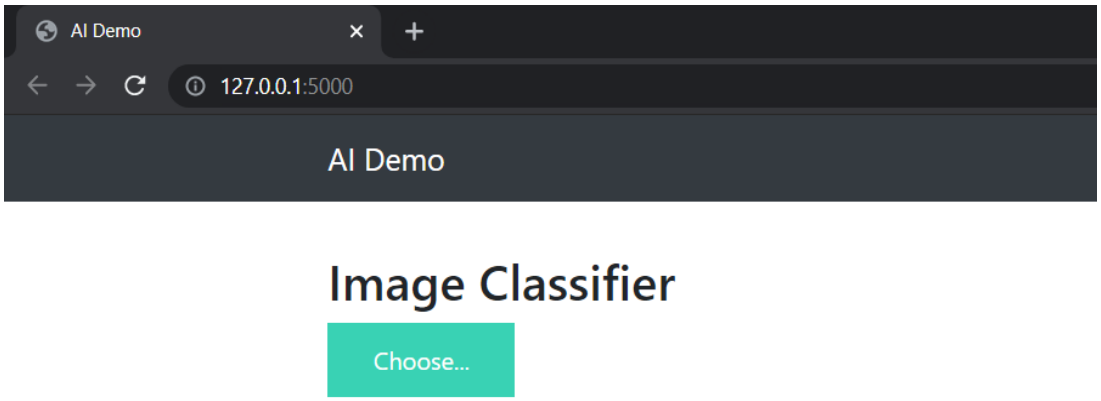


-Then after successfully executing the above code, we can comment the first 3 lines of that code and import model_resnet50.h5 file using below code.



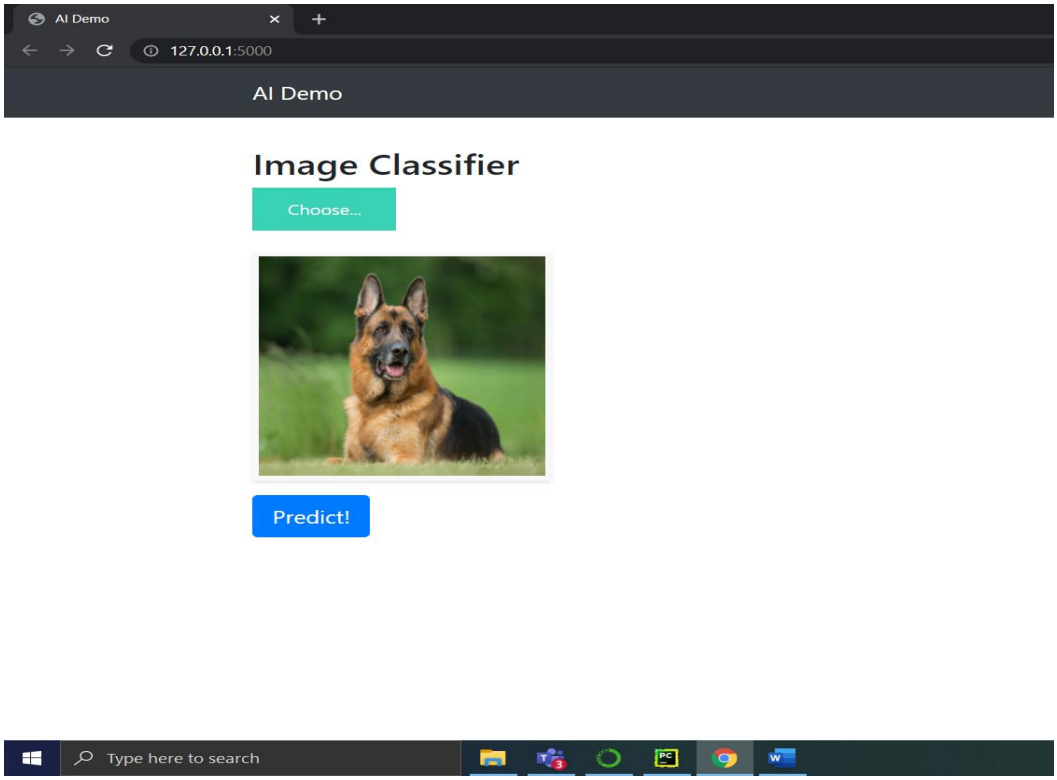
Finally, Run your application and go to http://localhost:5000/

It will open the following page:

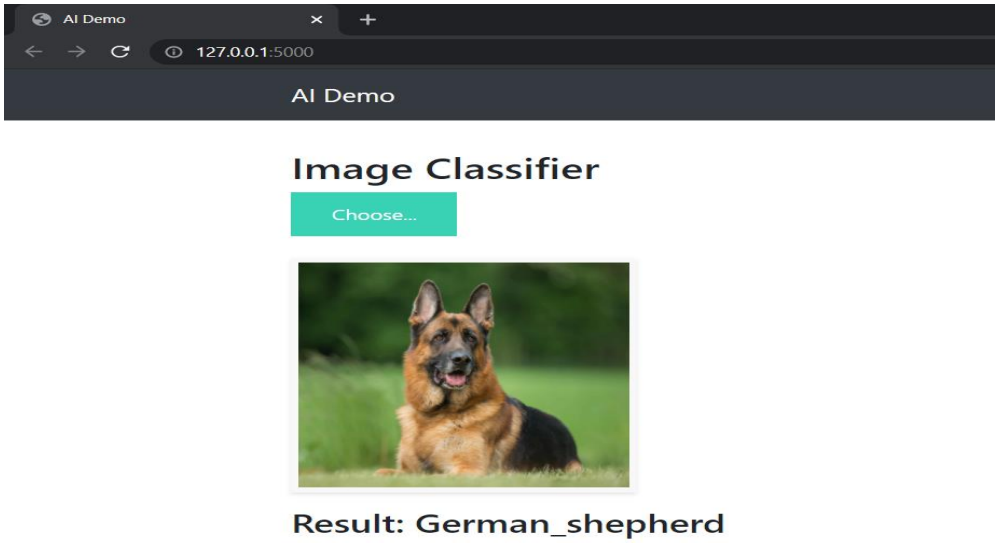


You can choose image of any animal or bird to identify the type and breed

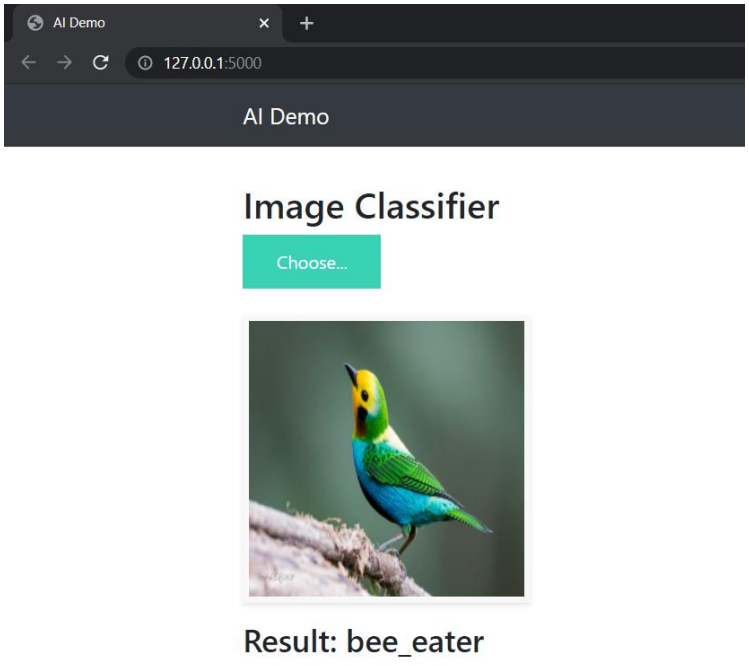
Then click on Predict button



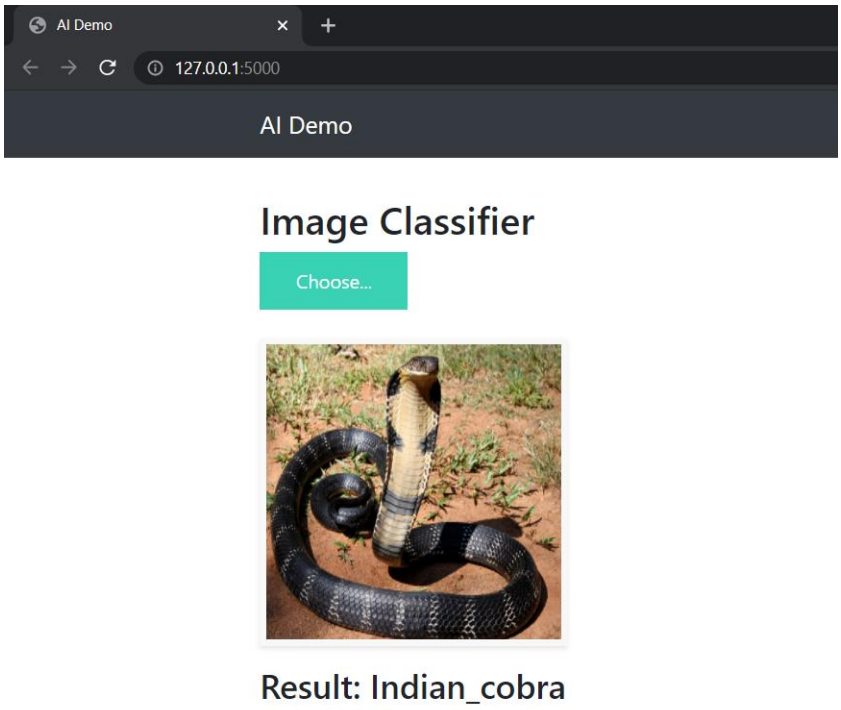
After you click on predict, it will show you the type and breed of the animal or bird you want.



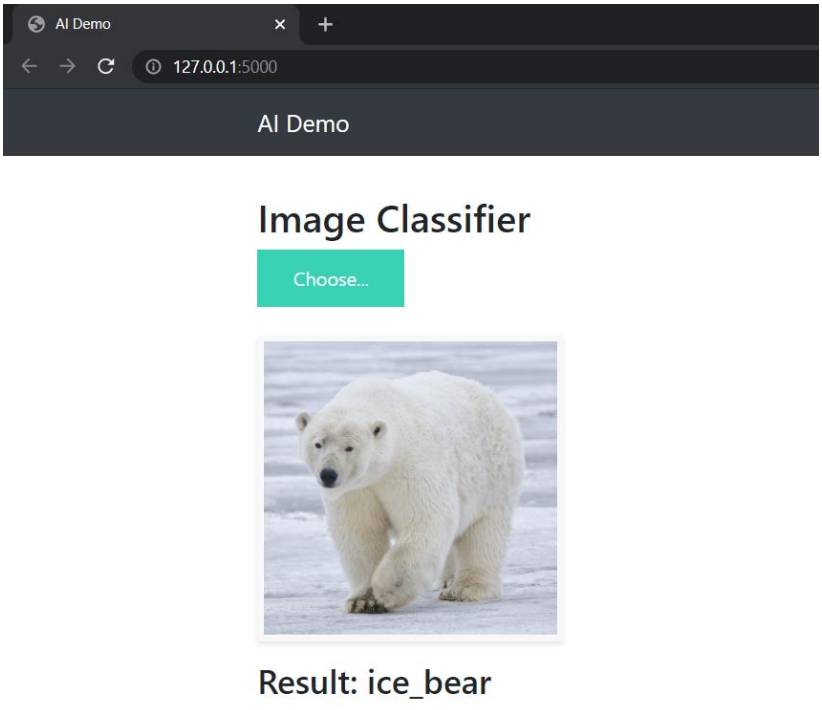
Some more results:

1. 

2.



3.



*Code will automatically save all the images you give under “uploads” folder inside your project.

Full Python code:

```
from __future__ import division, print_function
# coding=utf-8
import sys
import os
import glob
import re
import numpy as np

# Keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from keras.applications.imagenet_utils import preprocess_input,
decode_predictions
from keras.models import load_model
from keras.preprocessing import image
from keras.layers import LayerNormalization

# Flask utils
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer

# Define a flask app
app = Flask(__name__)

# Model saved with Keras model.save()
MODEL_PATH = 'C:/Flask/models/model_resnet.h5'

# Load your trained model
model = load_model(MODEL_PATH)
# print('Model loaded. Start serving...')

# You can also use pretrained model from Keras
# Check https://keras.io/applications/
#from keras.applications.resnet50 import ResNet50
#model = ResNet50(weights='imagenet')
#model.save('C:/Flask/models/model_resnet.h5')
print('Model loaded. Check http://127.0.0.1:5000/')


def model_predict(img_path, model):
    img = image.load_img(img_path, target_size=(224, 224))

    # Preprocessing the image
    x = image.img_to_array(img)
    # x = np.true_divide(x, 255)
    x = np.expand_dims(x, axis=0)

    # Be careful how your trained model deals with the input
    # otherwise, it won't make correct prediction!
    x = preprocess_input(x, mode='caffe')

    preds = model.predict(x)
    return preds


@app.route('/', methods=['GET'])
def index():
```

```
    # Main page
    return render_template('index.html')


@app.route('/predict', methods=['GET', 'POST'])
def upload():
    if request.method == 'POST':
        # Get the file from post request
        f = request.files['file']
        # Save the file to ./uploads
        basepath = os.path.dirname(__file__)
        file_path = os.path.join(
            basepath, 'uploads', secure_filename(f.filename))
        f.save(file_path)

        # Make prediction
        preds = model_predict(file_path, model)

        # Process your result for human
        # pred_class = preds.argmax(axis=-1)            # Simple argmax
        pred_class = decode_predictions(preds, top=1)   # ImageNet Decode
        result = str(pred_class[0][0][1])               # Convert to string
        return result
    return None

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