Flask Magic: Effortless Deployment of Deep Learning Models

Model deployment refers to the process of making machine learning or deep learning models accessible for use in real-world applications. It involves taking a trained model and integrating it into an environment where it can receive input data, make predictions, and provide output. There are various types of model deployment, including cloud-based deployment, edge deployment, and containerized deployment, each with its own advantages and considerations. Flask, a lightweight web framework for Python, plays a crucial role in deep learning model deployment by providing a straightforward way to create web APIs that can serve predictions from trained models. Flask's extensibility enables integration with other tools and frameworks commonly used in the deep learning ecosystem, such as TensorFlow and PyTorch, facilitating smooth deployment pipelines for machine learning practitioners. The journey of deploying your trained deep learning model using Flask entails assembling essential components. These components, ranging from your Flask application script to the necessary infrastructure, pave the way for seamless integration of your model into real-world applications.

Key Requirements:

- ***App.py Script***: This pivotal file serves as the foundation of your deployment, housing route definitions and logic for handling incoming HTTP requests. Within this script, you'll load your **trained model**, preprocess input data, and generate predictions.
- ***HTML Index File***: Complementing your Flask application, the HTML index file provides a user-friendly interface for interacting with your model. Rendered by Flask, this file empowers users to input data effortlessly and receive predictions.
- ***Python Environment with Flask Installed***: A robust Python environment, complete with Flask and necessary dependencies, forms the backbone of your deployment infrastructure. This environment ensures the seamless execution of your Flask application, facilitating smooth communication between users and your model.

Practical Example: Supermarket Tomato Classification

This tutorial focuses on deploying a pre-trained Convolutional Neural Network (CNN) model, named ***tomato_condition_classification.h5***, for classifying supermarket tomatoes as good or rotten (*Refer to the annexure link for comprehensive insights into the CNN model's architecture, training process, and performance evaluation*). Emphasizing deployment over model training, the ready availability of the model ensures efficiency in automating tomato quality assessment processes.. By leveraging Flask, users gain seamless access to this AI-driven solution, marking a pivotal step towards enhanced consumer satisfaction and operational excellence.

Procedure:

- ***Create App.py Script***: The **app.py** script serves as the core of the Flask application, handling route definitions and logic for processing HTTP requests, making it adaptable for diverse web development projects with minimal adjustments."

app.py C:\...\DL Deployment _Tomato classification ● app.py C:\...\catdog ●

C: > Users > Jayesh.P > Downloads > DL Deployment _Tomato classification > app.py

```python
from flask import Flask, render_template, request
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np

app = Flask(__name__)
dic = {0 : 'Good_Tomato', 1 : 'Rotten_Tomato'}
model = load_model('tomato_condition_classification.h5')


def predict_label(img_path):
    i = image.load_img(img_path, target_size=(150,150))
    i = image.img_to_array(i)/255.0
    i = i.reshape(1, 150,150,3)
    p = model.predict(i)
    result=p[0][0]
    npresult=round(result)
    return dic[npresult]

# routes
@app.route("/", methods=['GET', 'POST'])
def main():
    return render_template("index.html")

@app.route("/about")
def about_page():
    return "Welcome ....."

@app.route("/submit", methods = ['GET', 'POST'])
def get_output():
    if request.method == 'POST':
        img = request.files['my_image']

        img_path = "static/" + img.filename
        img.save(img_path)

        p = predict_label(img_path)

    return render_template("index.html", prediction = p, img_path = img_path)

if __name__ =='__main__':

    app.run(debug = True)
```

Code Explanation:

1. *Import Libraries*: Flask (A web framework for Python), render_template (render HTML template), request (for handling HTTP requests), load_model (load a pre-trained model), image (for image preprocessing.
2. *app = Flask(__name__):* Initializes a Flask application.
3. *model = load_model('tomato_condition_classification.h5')*: Loads the pre-trained deep learning model from the file 'tomato_condition_classification.h5'.
4. *Define predict_label Function*: The predict_label() function takes the path of an image file as input and predicts whether the image depicts a good or rotten tomato using a pre-trained deep learning model. It loads and preprocesses the image, reshapes it to match the model's input shape, and predicts the class probabilities. Then, it extracts and rounds the predicted probability and returns the corresponding label ('Good_Tomato' or 'Rotten_Tomato').
5. *Define Routes:* These route definitions in the Flask application handle different URLs and their corresponding actions. The main route ("/") is responsible for rendering the "index.html" template, accessible via both GET and POST methods. The about route ("/about") returns a simple welcome message when accessed via the GET method. The submit route ("/submit") is accessible via both GET and POST methods and handles image submissions. When a POST request is received, it retrieves the uploaded image, saves it to a static folder with a generated filename, predicts its label using the predict_label() function, and passes the prediction result along with the path to the saved image to the "index.html" template for display. These routes collectively define the behavior of the Flask application, ensuring proper handling of user requests and responses.
6. *Run the Flask App:* **`app.run(debug = True)`**
   Runs the Flask application with debugging enabled.

❖ ***Create HTML Index File***

This code is versatile and can be used for various purposes without concern, with only minor customization required for specific use cases or preferences.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Timespro Tomato Quality Classification</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
</head>
<body>

<div class="container">
  <h1 class="jumbotron bg-primary">Timespro Tomato Quality Classification</h1>
  <br><br>
  <form class="form-horizontal" action="/submit" method="post" enctype="multipart/form-data">

    <div class="form-group">
      <label class="control-label col-sm-2" for="pwd">Upload Your Image :</label>
      <div class="col-sm-10">
        <input type="file" class="form-control" placeholder="Hours Studied"  name="my_image" id="pwd">
      </div>
    </div>

    <div class="form-group">
      <div class="col-sm-offset-2 col-sm-10">
        <button type="submit" class="btn btn-success">Submit</button>
      </div>
    </div>
  </form>

  {% if prediction %}
  <img src="{{img_path}}" height="400px" width="400px">
  <h2> This is   : <i> {{prediction}} </i></h2>

  {% endif %}

</div>

</body>
</html>
```

Code Explanation:

1. *Document Type Declaration and Language*
2. *Head Section*: Contains metadata and links to external resources like character encoding , browser bar title, links etc.
3. *Body Section*: Contains the content of the web page.
    - A container ( ) to organize the content.
    - A heading (<h1 class="jumbotron bg-primary">) with a blue background.
    - A form (<form>) for image submission, configured to send data to "/submit" using the POST method.
    - An input field (<input type="file" class="form-control" name="my_image">) for uploading an image file.
    - A submit button (<button type="submit" class="btn btn-success">Submit</button>) to submit the form.
    - Conditional logic ({% if prediction %} ... {% endif %}) to check if a prediction exists.
    - An image element (<img src="{{img_path}}" height="400px" width="400px">) to display the uploaded image.
    - A heading (<h2> This is : {{prediction}} </h2>) to display the prediction result if available.

❖ ***Setting up the environment & running the script***

After creating the Python app.py script and HTML index script, the next step, step 3, involves setting up the environment and running the script. To do this, open Anaconda Prompt and navigate to the directory containing the project files using the command cd ***C:\Users\Jayesh.P\Downloads\DL Deployment _Tomato classification.***

```
(base) C:\Users\Jayesh.P>cd C:\Users\Jayesh.P\Downloads\DL Deployment _Tomato classification
```

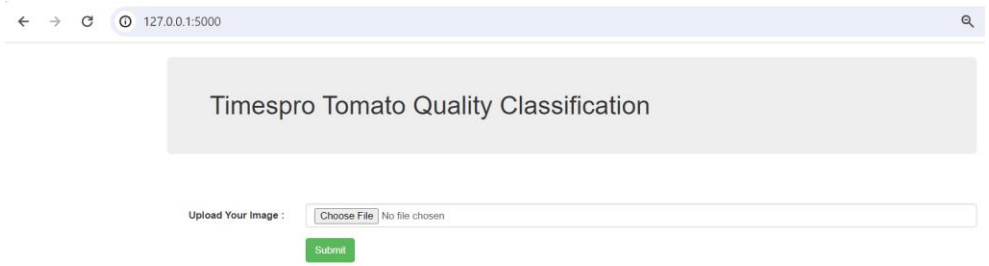
Then, install Flask by running the command pip install flask.

```
(base) C:\Users\Jayesh.P\Downloads\DL Deployment _Tomato classification>pip install flask
```

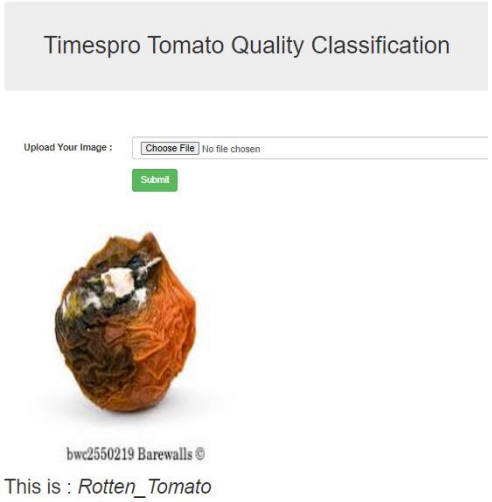
After successfully installing the flask, now execute the Python script with **python app.py.** Upon running the script, you may encounter warnings and commands in the terminal like below. Next we simply copy the link provided (e.g., http://127.0.0.1:5000) and paste it into a browser like Chrome or Windows. The URL "http://127.0.0.1:5000" typically refers to the local server or localhost, with "127.0.0.1" representing the loopback IP address of the local machine and "5000" indicating the port number on which the server is listening for incoming connections.

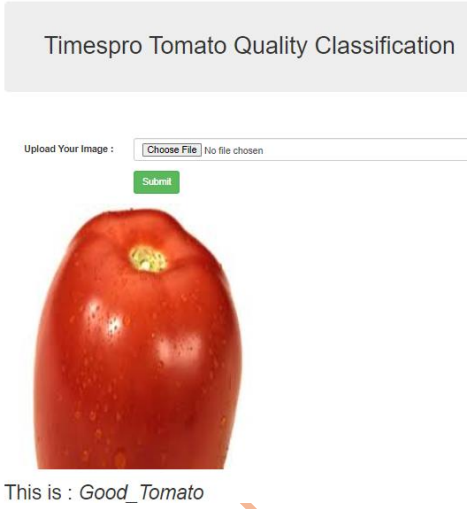
```
(base) C:\Users\Jayesh.P\Downloads\DL Deployment _Tomato classification>python app.py
2024-05-20 11:45:32.754528: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized wit
h oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:
 AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
2024-05-20 11:45:38.436522: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized wit
h oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:
 AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Debugger is active!
 * Debugger PIN: 132-187-973
```

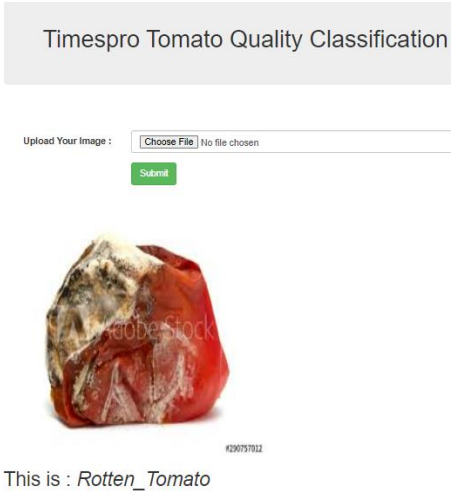
Once URL pasted, the application interface will be displayed like below.



With the setup nearly complete, proceed to open images from the 'static' folder mentioned in the HTML index. By loading these images, you can observe the classification results displayed on the window. Below are a couple of sample inferences for reference.







Conclusion

Look at the results—every prediction is spot-on! Deploying models in Flask API becomes effortless with practice. With minimal time, computation, and effort, you can seamlessly train and deploy models. We hope you found this article helpful. Refer to the annexure for datasets, code, and additional resource files to further enhance your practice.

Annexure

https://github.com/pjayeshkanayi/Flask-Magic-Effortless-Deployment-of-Deep-Learning-Models.git