Experiment No 10

Deployment of deep learning model for serving

Objective:

At the end of this practical session, student will be able to take a Keras model and deploy it as a REST API for serving.

Theory:

Serving machine learning models is the process of taking a trained model and making it available to serve prediction requests. When serving in production, it is important to make sure that environment is reproducible, enforces isolation, and is secure. Following steps can be used to serve keras model with flask.

A. Configuring environment

Keras need to be configured and installed on machine along with flask. Flask is a Python web framework. It is used to build API endpoint. Following command is used to install Flask with pip: *pip install flask gevent requests pillow*

B. Building your Keras REST API

Keras REST API is self-contained in a single file named run_keras_server.py. All the installation is placed in a single file for simplicity. It will have three following functionality:

- load_model: Used to load our trained Keras model and prepare it for inference. Function should include:

  global model
  model = ResNet50(weights="imagenet")

  As the name suggests, this method is responsible for instantiating network architecture and loading weights from disk. For the sake of simplicity, ResNet50 architecture is loaded on the ImageNet dataset. It is also possible to load custom model by loading architecture and weights from disk.
- prepare_image: This function preprocesses an input image prior to passing it through network for prediction. This function:
  - Accepts an input image
  - Converts the mode to RGB (if necessary)
  - Resizes it to 224x224 pixels (the input spatial dimensions for ResNet)
  - Preprocesses the array via mean subtraction and scaling

  Modification might be needed in this function based on any preprocessing, scaling, and/or normalization you need prior to passing the input data through the model.

```
if image.mode != "RGB":
    image = image.convert("RGB")

# resize the input image and preprocess it
image = image.resize(target)
image = img_to_array(image)
image = np.expand_dims(image, axis=0)
```

```
image = imagenet_utils.preprocess_input(image)
```

- predict: The actual endpoint of our API that will classify the incoming data from the request and return the results to the client.

```
data = {"success": False}
```

The data dictionary is used to store any data that we want to return to the client. Currently this includes a boolean used to indicate if prediction was successful or not —

To accept the incoming data it is required to check if: The request method is POST (enabling us to send arbitrary data to the endpoint, including images, JSON, encoded-data, etc.) An image has been passed into the files attribute during the POST

```
if flask.request.method == "POST":
    if flask.request.files.get("image"):
```

Then take the incoming data and:

- Read it in PIL format
- Preprocess it
- Pass it through our network
- Loop over the results and add them individually to the data["predictions"] list
- Return the response to the client in JSON format

```
image = flask.request.files["image"].read()
image = Image.open(io.BytesIO(image))

 # preprocess the image and prepare it for classification
image = prepare_image(image, target=(224, 224))

 # classify the input image and then initialize the list
# of predictions to return to the client
preds = model.predict(image)
 results = imagenet_utils.decode_predictions(preds)
data["predictions"] = []

 # loop over the results and add them to the list of
 # returned predictions
for (imagenetID, label, prob) in results[0]:
    r = {"label": label, "probability": float(prob)}
    data["predictions"].append(r)

# indicate that the request was a success
data["success"] = True
```

C. Launch the service

First call the load_model which loads Keras model from disk. The call to load_model is a blocking operation and prevents the web service from starting until the model is fully

loaded. It is important to check whether model is loaded completely otherwise system will be in situation where:

1. A request is POST'ed to the server.
2. The server accepts the request, preprocesses the data, and then attempts to pass it into the model
3. ...but since the model isn't fully loaded yet, our script will error out!

When building Keras REST APIs, ensure logic is inserted to guarantee your model is loaded and ready for inference prior to accepting requests. For this use following code:

```
if __name__ == "__main__":
    print(("* Loading Keras model and Flask starting server..."
        "please wait until server has fully started"))
    load_model()
    app.run()
```

D. Starting your Keras Rest API

Start REST API with command: python run_keras_server.py. After starting this server use the address http://127.0.0.1/<port_number>/predict to access the /predict endpoint via your browser. But it will provide error: a "Method Not Allowed" error. This error is due to the fact that browser is performing a GET request, but /predict only accepts a POST. To handle this use cURL as shown in following step.

E. Using cURL to test the Keras REST API

When testing and debugging your Keras REST API use cURL. We can use curl to pass this image to our API and find out what ResNet thinks the image contains:

```
curl -X POST -F image=@dog.jpg 'http://localhost:5000/predict'
```

It will get following output in command line


```
{
  "predictions": [
    {
      "label": "beagle",
      "probability": 0.9901360869407654
    },
    {
      "label": "Walker_hound",
      "probability": 0.002396771451458335
    },
    {
      "label": "pot",
      "probability": 0.0013951235450804234
    },
    {
      "label": "Brittany_spaniel",
      "probability": 0.001283277408219874
    },
    {
```

```
    "label": "bluetick",
    "probability": 0.0010894243605434895
   }
 ],
 "success": true
}
```

The -X flag and POST value indicates we're performing a POST request. We supply -F image=@dog.jpg to indicate we're submitting form encoded data. The image key is then set to the contents of the dog.jpg file. Supplying the @ prior to dog.jpg implies we would like cURL to load the contents of the image and pass the data to the request.

Finally, we have our endpoint: http://localhost:5000/predict

F. Consuming the Keras REST API programmatically

In all likelihood, you will be both submitting data to your Keras REST API and then consuming the returned predictions in some manner — this requires we programmatically handle the response from our server. This is a straightforward process using the requests Python package as shown in following code snippet.

```
import requests

# initialize the Keras REST API endpoint URL along with the input
# image path
KERAS_REST_API_URL = "http://localhost:5000/predict"
IMAGE_PATH = "dog.jpg"

# load the input image and construct the payload for the request
image = open(IMAGE_PATH, "rb").read()
payload = {"image": image}

# submit the request
r = requests.post(KERAS_REST_API_URL, files=payload).json()

# ensure the request was successful
if r["success"]:
    # loop over the predictions and display themzs
    for (i, result) in enumerate(r["predictions"]):
        print("{}. {}: {:.4f}".format(i + 1, result["label"],
            result["probability"]))

# otherwise, the request failed
else:
    print("Request failed")
```

The KERAS_REST_API_URL specifies our endpoint while the IMAGE_PATH is the path to our input image residing on disk. Using the IMAGE_PATH we load the image and then construct the payload to the request.

Given the payload we can POST the data to our endpoint using a call to requests.post. Appending .json() to the end of the call instructs requests that:

- The response from the server should be in JSON
- We would like the JSON object automatically parsed and deserialized for us
- Once we have the output of the request, r, we can check if the classification is a success (or not) and then loop over r["predictions"].

Keyword:
Serving Model, REST API, cURL, FLASK

Procedure:

1. Train your model and save it hard disk with network and weight.
2. Create environment by installing flask
3. Build keras rest api
4. Launch the service
5. Start keras REST API
6. Using cURL check predictions.