Image Classification with CNNs

Deep Learning Project Report
Submitted to

DEVI AHILYA VISHWAVIDHYALYA INDORE

In Partial Fulfillment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

in

BIG DATA ANALYTIC

by

Er. Manish Kumar Singh
DS7B-2208

Under the Supervision of

Mr. Puneet Gupta
Assistant Professor

SCHOOL OF DATA SCIENCE AND FORECASTING

2022-23

Image Classification with CNNs

May 5, 2023

[ ]:
```
from future import division, print_function
```

This line imports some features from future versions of Python to ensure compatibility with older versions. Specifically, the division operator / will return a float instead of integer division, and the print function will work as expected.

[ ]:
```
import os
```

This line imports the built-in Python module 'os', which provides a way to interact with the operating system.

[ ]:
```
import numpy as np
```

This line imports the third-party Python library 'numpy' (short for 'Numerical Python'), which provides support for multidimensional arrays, mathematical functions, and other numerical operations.

[ ]:
```
import keras.utils as image
```

This line imports the 'utils' module from the Keras library, which provides utility functions for loading and manipulating images.

[ ]:
```
from flask import Flask, redirect, url_for, request, render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer
```

These lines import several modules from the Flask and Werkzeug libraries. Flask is a micro web framework for building web applications, while Werkzeug provides utilities for handling HTTP requests and responses.

[ ]:
```
from keras.applications import ResNet50
from keras.applications.imagenet_utils
import preprocess_input as preprocess_input_resNet50, decode_predictions as
decode_predictions_resNet50
```

This line imports the ResNet50 CNN architecture, which is a deep neural network with 50 layers that has been pre-trained on the ImageNet dataset. It also imports two utility functions for preprocessing input images and decoding output predictions using ResNet50.

[ ]:
```
# VGG16
from keras.applications.vgg16 import VGG16
```

```
from keras.applications.vgg16 import preprocess_input as
preprocess_input_vgg16, decode_predictions as
decode_predictions_vgg16
```

These lines import the VGG16 CNN architecture, which is another deep neural network pre-trained on the ImageNet dataset. It also imports two utility functions for preprocessing input images and decoding output predictions using VGG16.

[ ]:
```
# Xception
from keras.applications.xception import Xception
from keras.applications.xception import preprocess_input as
preprocess_input_xception, decode_predictions as
decode_predictions_xception
```

These lines import the Xception CNN architecture, which is another deep neural network pre-trained on the ImageNet dataset. It also imports two utility functions for preprocessing input images and decoding output

predictions using Xception.

These import statements bring in the necessary modules and functions that the script will use to load and manipulate images and to run the pre- trained CNNs for image classification.

The remaining code of the script builds a Flask web application that allows users to upload an image and get the predicted class labels for the image using the pre-trained CNNs. The uploaded image is preprocessed using the corresponding preprocess_input function of each CNN before feeding it into the model for prediction. The predicted labels are then decoded using the corresponding decode_predictions function to get the top 5 most probable classes with their probabilities. Finally, Gevent is used to create a WSGI server that can handle multiple requests simultaneously.

[ ]:
```
app = Flask( name )
```

This line creates a new Flask web application instance named 'app' using the current module (name) as the name of the application.

[ ]:
```
modelResNet50 = ResNet50(weights='imagenet')
print('ResNet50 Model loaded.')
```

These lines load the pre-trained ResNet50 CNN model using the 'weights' parameter set to 'imagenet', which tells Keras to download the pre-trained weights from the ImageNet database. A message is printed to confirm that the model has been loaded.

[ ]:
```
modelVGG16 = VGG16(weights='imagenet', include_top=True)
print('VGG16 Model loaded.')
```

These lines load the pre-trained VGG16 CNN model using the 'weights' parameter set to 'imagenet', and the 'include_top' parameter set to True. The include_top parameter means to include the fully connected layer on top of the network, which is necessary for classification. A message is printed to confirm that the model has been loaded.

```
[ ]: modelXception = Xception(weights='imagenet', include_top=True)
print('Xception Model loaded.')
print('Running on http://localhost:5000')
```

These lines load the pre-trained Xception CNN model using the 'weights' parameter set to 'imagenet', and the 'include_top' parameter set to True. A message is printed to confirm that the model has been loaded, and another message is printed indicating the URL where the application can be accessed.

```
[ ]: def get_file_path_and_save(request):
    f = request.files['file']
basepath = os.path.dirname(  file  )
file_path = os.path.join(
basepath, 'uploads', secure_filename(f.filename))
f.save(file_path)
return file_path
```

This function takes a Flask 'request' object as input and extracts the file from the 'file' field of the POST request. It then saves the file to a specified directory ('./uploads') using the secure_filename function to ensure that the filename is safe and does not contain any potentially malicious content. The function returns the full file path of the saved file.

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

This line defines a route for the root URL path of the web application ("/") using the HTTP GET method.

```
[ ]: def index():
# Main page
return render_template('index.html')
```

This function defines the behavior of the root URL path. In this case, it returns a rendered HTML template file called "index.html" using the Flask `render_template()` function.

```
[ ]: @app.route('/predictResNet50', methods=['GET', 'POST'])
```

This line defines a route for the "/predictResNet50" URL path using both HTTP GET and POST methods.

```
[ ]: def predictResNet50():
if request.method == 'POST':
```

This function defines the behavior of the "/predictResNet50" URL path. It checks if the HTTP request method is POST.

```
[ ]: file_path = get_file_path_and_save(request)
```

This line calls a function called `get_file_path_and_save()` to retrieve the file path of the uploaded image file and saves it to the server.

```
[ ]: img = image.load_img(file_path, target_size=(224, 224))
```

This line loads the image file from the saved file path and resizes it to a target size of (224, 224) pixels.

```
[ ]: x = image.img_to_array(img)
     x = np.expand_dims(x, axis=0)
     x = preprocess_input_resNet50(x, mode='caffe')
```

These lines convert the image to a NumPy array, add an extra dimension to create a batch of one image, and preprocess the image data for use with the ResNet50 model.

```
[ ]: preds = modelResNet50.predict(x)
```

This line uses the ResNet50 model to make a prediction on the preprocessed image data.

```
[ ]: pred_class = decode_predictions_resNet50(preds, top=3)
```

This line decodes the predicted class probabilities into a list of class names and their corresponding probabilities using a function called `decode_predictions_resNet50()`.

```
[ ]: result1 = pred_class[0][0]
     f1 = "{0:.2f}".format(round(result1[2]*100,2))
     a = result1[1] +"        "+ str(f1)
```

These lines extract the top predicted class name and probability from the decoded predictions list and format them into a string.

```
[ ]: result2 = pred_class[0][1]
     f2 = "{0:.2f}".format(round(result2[2]*100,2))
     b = result2[1] +"        "+ str(f2)
```

These lines extract the second highest predicted class name and probability from the decoded predictions list and format them into a string.

```
[ ]: result3 = pred_class[0][2]
     f3 = "{0:.2f}".format(round(result3[2]*100,2))
     c = result3[1] +"        "+ str(f3)
```

These lines extract the third highest predicted class name and probability from the decoded predictions list and format them into a string.

```
[ ]: result = a+"\n"+b+"\n"+c return result
```

These lines combine the formatted class names and probabilities into a single string with newline characters and return it as the response to the HTTP POST request.

```
[ ]: return None
```

This line returns nothing as the response to an HTTP GET request.

a brief description of each line of code in the `predictVGG16()` function:

```
[ ]: @app.route('/predictVGG16', methods=['GET', 'POST'])
```

This is a Flask route decorator that specifies the endpoint `/predictVGG16` and the allowed HTTP methods (`GET` and `POST`) for accessing it.

[ ]:
```
def predictVGG16():
```

This function is executed when a POST request is sent to the `/predictVGG16` endpoint.

[ ]:
```
if request.method == 'POST':
```

Checks if the request method is `POST`, which is the method used to

submit the image file for prediction.

[ ]:
```
file_path = get_file_path_and_save(request)
```

Calls a function `get_file_path_and_save()` that retrieves the uploaded image file, saves it to the server, and returns the path to the saved file.

[ ]:
```
img = image.load_img(file_path, target_size=(224, 224))
```

Loads the saved image file using Keras `load_img()` function, with the target size set to 224 x 224 pixels.

[ ]:
```
img_data = image.img_to_array(img)
```

Converts the image data to a numpy array using Keras `img_to_array()` function.

[ ]:
```
img_data = np.expand_dims(img_data, axis=0)
```

Reshapes the image data by adding an extra dimension to represent the batch size.

[ ]:
```
img_data = preprocess_input_vgg16(img_data)
```

Preprocesses the image data using a function `preprocess_input_vgg16()` that applies VGG16-specific preprocessing, such as mean subtraction.

[ ]:
```
preds = modelVGG16.predict(img_data)
```

Makes a prediction on the preprocessed image data using the VGG16 model `modelVGG16`.

[ ]:
```
pred_class = decode_predictions_vgg16(preds, top=3)
```

Decodes the model predictions into a list of tuples representing the top 3 predicted classes, along with their descriptions and probabilities.

[ ]:
```
result1 = pred_class[0][0]
f1 = "{0:.2f}".format(round(result1[2]*100,2))
a = result1[1] +"        "+ str(f1)
```

Extracts the class name, description, and probability for the top predicted class, formats the probability as a percentage with two decimal places, and stores the result in a string variable `a`.

```
[ ]: result2 = pred_class[0][1]
     f2 = "{0:.2f}".format(round(result2[2]*100,2))
     b = result2[1] +"        "+ str(f2)
```

Extracts the class name, description, and probability for the second predicted class, formats the probability as a percentage with two decimal places, and stores the result in a string variable `b`.

```
[ ]: result3 = pred_class[0][2]
     f3 = "{0:.2f}".format(round(result3[2]*100,2))
     c = result3[1] +"        "+ str(f3)
```

Extracts the class name, description, and probability for the third predicted class, formats the probability as a percentage with two decimal places, and stores the result in a string variable `c`.

```
[ ]: result = a+"\n"+b+"\n"+c return result
```

Combines the three predicted class results into a single string, with each result on a separate line, and returns the string as the prediction result.

- '@app.route('/predictXception', methods=['GET', 'POST'])' : A decorator that maps the URL route '/predictXception' to this function 'predictXception'. It also specifies that the function can handle both GET and POST requests.
- `def predictXception():`: Defines a function 'predictXception' that handles the prediction using Xception model.
- `if request.method == 'POST':`: A conditional statement that checks if the request method is POST.
- `file_path = get_file_path_and_save(request)`: This line calls the 'get_file_path_and_save' function to save the uploaded image to a folder and return the file path.
- `img = image.load_img(file_path, target_size=(299, 299))`: Loads the image from the saved file path, resizes it to the target size of (299, 299) required for the Xception model.
- `img_data = image.img_to_array(img)`: Converts the image to a numpy array.
- `img_data = np.expand_dims(img_data, axis=0)`: Adds an extra dimension to the numpy array to make it compatible with the Xception model.
- `img_data = preprocess_input_xception(img_data)`: Preprocesses the image data using the preprocess_input_xception function to make it compatible with the Xception model.
- `preds = modelXception.predict(img_data)`: Makes a prediction using the Xception model on the preprocessed image data.
- `pred_class = decode_predictions_xception(preds, top=3)`: Decodes the predictions to a list of tuples (class, description, probability), sorted by probability, and returns the top 3 predictions.
- `result1 = pred_class[0][0]`: Extracts the first prediction tuple.
- `f1 = "{0:.2f}".format(round(result1[2]*100,2))`: Formats the probability of the first prediction tuple to two decimal places.

- `a = result1[1] +"  "+ str(f1)`: Concatenates the class and probability of the first prediction tuple.
- `result2 = pred_class[0][1]`: Extracts the second prediction tuple.
- `f2 = "{0:.2f}".format(round(result2[2]*100,2))`: Formats the probability of the second prediction tuple to two decimal places.
- `b = result2[1] +"  "+ str(f2)`: Concatenates the class and probability of the second prediction tuple.
- `result3 = pred_class[0][2]`: Extracts the third prediction tuple.
- `f3 = "{0:.2f}".format(round(result3[2]*100,2))`: Formats the probability of the third prediction tuple to two decimal places.
- `c = result3[1] +"  "+ str(f3)`: Concatenates the class and probability of the third prediction tuple.
- `result = a+"\n"+b+"\n"+c`: Concatenates the concatenated strings of the top 3 predictions separated by newline characters.
- `return result`: Returns the concatenated string of the top 3 predictions.
- 'if

name

== ' main ':': A conditional statement that checks if the

current module is being run as the main program.

- `http_server = WSGIServer(('', 5000), app)`: Creates a WSGI server object to serve the Flask app on port 5000.
- `http_server.serve_forever()`: Starts the server to serve the app forever until it is interrupted.