Running the prediction program

Download the code from https://github.com/juanluisrosaramos/CRNN_OCR.git

There is an available Docker image (with a model) here where we can avoid the steps of download the code and build the image.

The trained model is available here

Using Dockers

In the code folder we have a Dockerfile for building an image. The image base is an image with openCV stored in hub.docker.com (hubertlegec/opencv-python:1.0) and then runs the requirements.txt where the tensorflow library for CPU is required. In this case we use Tensorflow version 1.12

There is an already builded image for CPU available with:

docker pull gcr.io/juanluis-personal/crnn_ocr:cpu

CPU

1 Building the image for CPU

Build the image for CPU. In Ubuntu the command will be:

$docker build -f Dockerfile -t name_of_image:cpu .

2 Running the image for CPU

The purpose of the project is to build a CSV file with the predictions. In this case we need to provide an output file path and name. For keeping this csv file when the image is stopped we have to mount a volume in the container. Using this volume we can also test the code with a new folder of images.

$docker run -it --rm -v $(pwd):/files name_of_image:cpu

In this case, with $(pwd) we are setting the path where we are and mount it as /files in the running container

We can also avoid the step of building the image from the code and run the provided image

$docker run -it --rm -v $(pwd):/files gcr.io/juanluis-personal/crnn_ocr:cpu

If we are testing we can avoid this step and simply run the container without any mounting point.

$docker run -it --rm name_of_image:gpu

GPU

1 Building the image for GPU

If you can use a GPU you can build the image for using it. In this case the Dockerfile downloads a tensorflow GPU image (version 1.12) because the TF-gpu installed with pip does not provide CUDA9. OpenCV is installed via pip in requirementsgpu.txt. In Ubuntu the command will be:

There is a pre-builded image for GPU available at:

$docker build -f Dockerfilegpu -t name_of_image:gpu .

2 Running the image for GPU

docker run -it --rm -v $(pwd):/files --runtime=nvidia name_of_image:gpu

$docker run -it --rm -v $(pwd):/files --runtime=nvidia gcr.io/juanluis-personal/crnn_ocr:gpu

Running the detection or inference program

The code use Tensorflow framework and an already trained model for making predictions. In the image and code there is provided a model (crnn_dsc_2018-08-20.ckpt.) trained on a subset of Synth 90k

Once we run the Docker image we can execute the prediction running a python3 script called demo_batch.py

Testing the script

:/app# python demo_batch.py

And the script runs over the images in /data/test_images folder and must return

Restoring trained model
Predicting 17 images in chunks of 32
Prediction time for 32 images: 1.7642133235931396
Total prediction time: 1.7642252445220947
Predictions saved in file data/output.csv

Another test can be done:
:/app# python demo_batch.py -i data/bounding_box/

It will run the code against 3423 COCO text images.

And it should output
Restoring trained model
Predicting 3423 images in chuncks of 32
Prediction time for 32 images: 1.7391960620880127
Prediction time for 32 images: 1.6067194938659668
…
…
Total prediction time: 184.91092610359192
Predictions saved in file data/output.csv

The predictions are saved in a csv file if we mounted a volume the predictions it will be accessible when we stop the container.

Parameters of the script

The script has the following parameters:e

- dir of images “-i”
- output dir and name for the csv file “-o”
- trained model to use “-w”

/app# python demo_batch.py --help
usage: demo_batch.py [-h] [-i IMAGE_DIR] [-w WEIGHTS_PATH] [-o OUTPUT_FILE]

optional arguments:
  -h, --help   show this help message and exit
  -i IMAGE_DIR, --image_dir IMAGE_DIR
                        Where you store images
  -w WEIGHTS_PATH, --weights_path WEIGHTS_PATH
                        Where you store the weights
  -o OUTPUT_FILE, --output_file OUTPUT_FILE
                        Name of the csv file with the results

The script accepts a different model providing new trained weights it needs a path with the images to analyze and the name of an output file (name.csv) where we store the results of the inference.

Example of use

Having a container running with a volume named /files mounted in the root of the container
$docker run -it --rm -v $(pwd):/files name_of_image:cpu

We can run the program with:
/app# python demo_batch.py -i /files/data/test_images/ -o /files/predictions.csv

Or running our own model

/app# python demo_batch.py -w model/own_model.ckpt -i /files/data/test_images/ -o /files/predictions.csv

Changing number of predictions

This is not provided by parameter but we have to change it in the script demo_batch.py where there is a constant that can be changed
NUMBER_OF_PREDICTIONS = 10
In case of changing this number we will increase or decrease the number of predictions outputted in the csv file (and recomputing the softmax probabilities)

Changing the size of the batch

BATCH_SIZE = 32
For speeding the inference the script builds a np.array of 32 images and sent it to the model loaded by TF. Depending on memory and cpu we can increase this batch. The time of total computation is provided to see if batch_size increasing is rising the performance of running the inference of all the images.

Output csv file with the detections

The script produces a CSV file with the following format

Name_of_image,pred1,prob1,pred2,prob2,....,predN,probN

Where pred is a prediction and prob the probability of that prediction

COCO_train2014_000000042345.jpg,district,0.377022,pistrict,0.201334,districr,0.119883
COCO_train2014_000000448826.jpg,emirates,0.778889,enirates,0.042991,emiraies,0.041505,emnirates,0.037806,emiratos,0.031919

Details of the implementation

We use TensorFlow version 1.12 to implement a CRNN mainly based on the paper "An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition". You can refer to their paper for details http://arxiv.org/abs/1507.05717. The CRNN consists of convolutional layers (CNN) extracting a sequence of features and recurrent layers (RNN) propagating information through this sequence. It outputs character-scores for each sequence-element, which simply is represented by a probabilities matrix per each character. Finally the matrix is decoded by a CTC operation using the Tensorflow function called ctc_beam_search_decoder that provided an amount of possible paths in the predicted characters. We choose the best N paths.

The output of CTC beam search function is an sparse tensor. We need to decoded the matrix to a string using a characters map dictionary provided in data/chart_dict.json.So, the text from the image is recognized in a character-level having a maximum of 25 chars per image.

The characters are:
%'*+,-./:0123456789abcdefghijklmnopqrstuvwxyz

During inference (and for training) the images are resized to 100x32 pixels using opencv python library.

Probability computation

From the Tensorflow implementation of the method ctc_beam_search_decode we don't have a probability distribution because it would mean summing over all possible sequences of all admissible lengths. In that case, the scores, that are in log domain, are computed as Z = sum_k(exp(score_k)). To have a Softmax probability distribution in N (10) predictions we implemented the following function

$$S(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{j} e^{y_j}}$$

Where S(y_i) is the softmax function of y_i and e is the exponential and j is the no. of columns in the input vector Y.