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
And the trained model is available here

Using Dockers

From the code we have a Dockerfile for building an image that has to be built. The image uses an image with openCV stored in hub.docker.com and then install the requirements.txt where the tensorflow library is required. In this case we use TF version 1.12

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

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

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 .
```

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. We also need it to provide new images

In this case we set the dir where we are and mount it as /output in the running container

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

```
$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.

```
$docker run -it --rm name_of_image:gpu
```

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) as 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 pull gcr.io/juanluis-personal/crnn_ocr:gpu
```

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

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

The code use Tensorflow framework and an already trained model for making predictions. In the image 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/
```

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 will be accessible when we stop the command.

Parameters of the script

The script has the following parameters

```
/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 accept 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

In the script demo_batch.py 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 (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 depending on memory and cpu we can increase this batch. The time of total computation is provided to see if it increase or decrease the speed 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,....,pred10,prob10
```

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,0.0415
05,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 for consists of convolutional layers (CNN) to extract a sequence of features and recurrent layers (RNN) to propagate information through this sequence. It outputs character-scores for each sequence-element, which simply is represented by a matrix. 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 10 paths.

The output of CTC loss function is an sparse tensor and it is decoded to 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, then the scores are in log domain and are computed as Z = sum_k(exp(score_k)). To have a Softmax probabilty 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.