Konstantinos Mavromatakis
Assignment 1 - Chinese Character Detection

PART 1

The task at hand was to train a model on a dataset of images annotated with bounding box information, which represents the coordinates of the corners of a box that contains Chinese characters. The output is to provide a probability that shows whether or not each pixel is part of a bounding box that contains a Chinese character.

For the first part of the assignment, I loaded the image ids from the ‘info.json’ file. Then, I used the ‘train.jsonl’ file to load the images that were used as training data in the original task. The intersection of the image ids from these two files was used in this assignment, along with the metadata that were stored in the ‘train.jsonl’ file. These images and metadata were divided into train (80%), validation (10%), and test (10%) sets.

From the information provided by the metadata, I extracted the polygons, i.e., the coordinates of the corners of the boxes that contained a Chinese character. This information would be the features that I am planning to use to train the model.

The images are transformed into numpy arrays, while the gold standard is a 2048 * 2048 array that contains 1s and 0s, depending on whether a pixel is inside or outside the polygons.

After facing many memory errors, I asked my classmates for a hint. They suggested that I either resize the images or use joblib.Parallel to turn my code into parallel computing mode and, thus, increase the computing speed. Indeed, after implementing the latter approach, I managed to load the image arrays and the gold truth labels successfully, without encountering any memory errors.

PART 2

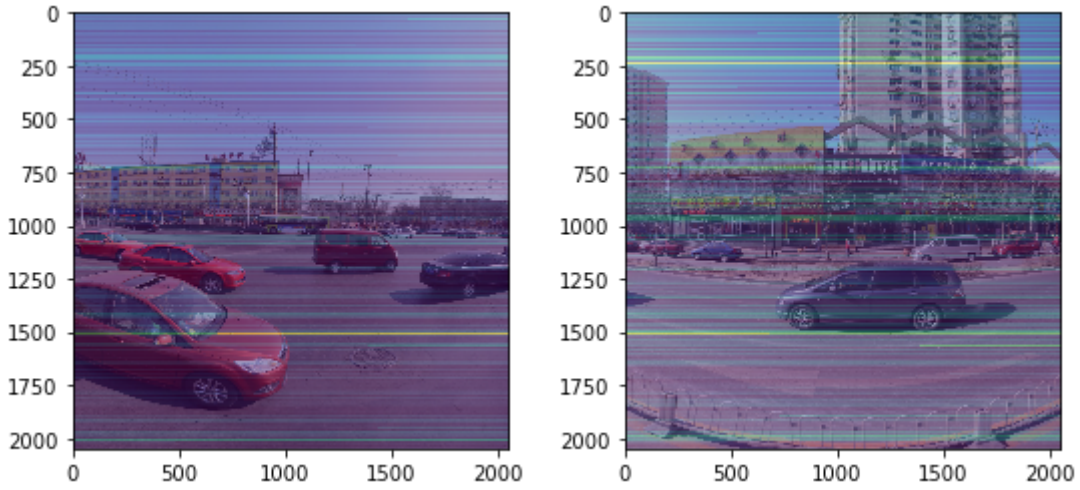
The architecture of the first model is inspired by the LeNet model, proposed by Yann LeCun et al. (1998), with some slight modifications. Instead of two, I used three CNN layers, which were followed by ReLU activation functions, and average pooling layers. Similar to the original LeNet model, two linear layers were also used, along with a Sigmoid function, to add non linearity to the model.

For the second model, I took inspiration from the VGG16 model proposed by K. Simonyan & A. Zisserman (2015). One of its main characteristics is that it uses multiple 3×3 kernel-sized filters one after another. I replaced the softmax function with a Sigmoid function, since the classification task here is binary. The images are passed through five stacks of multiple convolutional layers, followed by non linearities and max pooling layers. The convolution stride is set to 1 pixel. Three fully connected layers follow the stacks of the convolutional layers.

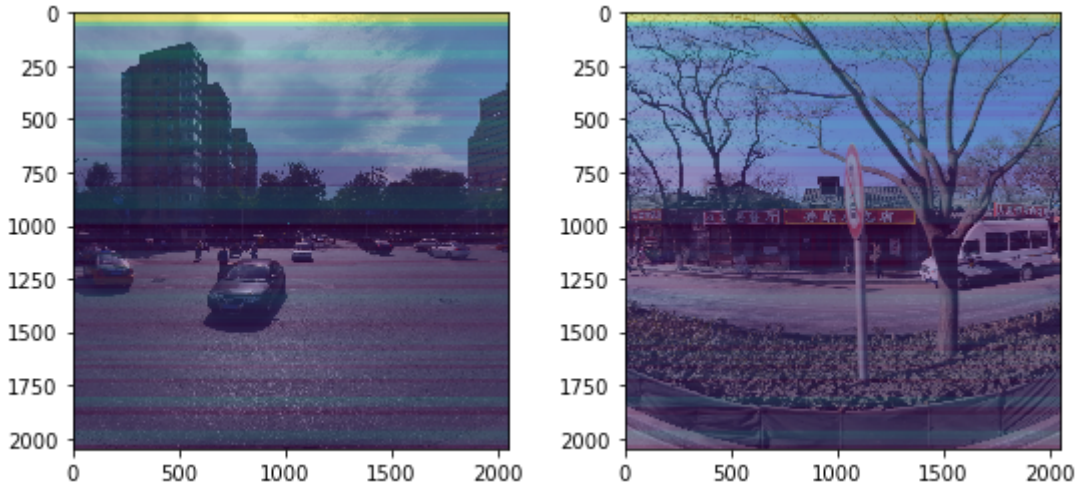
For the training, I used Binary Cross Entropy loss - since the classification task is binary - and Adam optimizer, as it is computationally efficient and has little memory requirements. I trained the models for 3 epochs with a batch size of 4 and a learning rate of 0.0001. In the training code, there is some permutation and upsampling happening to avoid memory errors.

PART 3

Evaluating the models consists of calculating their accuracy and total loss. The first model scored an accuracy of 0.14 with a loss of 0.05 on the validation data. On the test data, it scored an accuracy of 0.04 with a loss of 0.04. Such low accuracy is reasonable if we take into consideration that the model was trained on a very small sample of images. Perhaps, the training data was too similar, and thus, made it impossible for the model to learn anything substantial. Visualizing the model's predictions on separate images yields confusing results. The first model seems to detect Chinese characters all over the test images, which points to the fact that the model does not learn much. Below are two images, to which I have applied the first model.



The second model scored an accuracy of 0.08 with a loss of 0.03 on the validation data. On the test data, the second model's accuracy was 0.1. The small sample of training data is likely to have impacted the performance of the second model as well. Applying the second model on the test data appears to detect pixels corresponding to Chinese characters all over the images. The low accuracy of the second model, its architecture, along with the aforementioned pattern, allude that training the model on few images does not yield promising results for the task of Chinese character detection. Below are two images, to which I have applied the second model.



The performance of both models is not good enough for them to be deployed in real life applications. Additional finetuning needs to be done for them to successfully detect Chinese characters in images.

Resources & Materials

1) https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d
2) https://towardsdatascience.com/understanding-and-implementing-lenet-5-cnn-architecture-deep-learning-a2d531ebc342
3) https://towardsdatascience.com/implementing-yann-lecuns-lenet-5-in-pytorch-5e05a0911320?gi=8789606cb3f5
4) https://medium.com/@tioluwaniaremu/vgg-16-a-simple-implementation-using-pytorch-7850be4d14a1
5) https://github.com/hadikazemi/Machine-Learning/blob/master/PyTorch/tutorial/vgg16.py