Report: Chinese Character Detection

I started with exploring the dataset we were to use, the CTW dataset. This stage mainly consisted of me inspecting the folders the dataset consists of – especially the info.json, and train.jsonl files. To actually understand what was in them, and what would be useful information for our specific task, boundary box detection, I had to research the image segmentation field further.

Then it was time to start building my dataset class. As a dataset class usually (if not always) consists of an __init__ function, a __len__ function, and a __getitem__ function I created those methods for my class. The __init__ function initialise the image_size, img_files (a dictionary of the name of an image file as its key and the path to the file as its element, if the name of the file is also found in the training section in info.json), and samples (a list of dictionaries, where every dictionary is the annotation of an image file). The __len__ function returns the length of the dataset, based in this case on samples but could also be based on the length of img_files if one would want to. __getitem__ returns an image and a 'bi_mask' for every item in the dataset after having loaded the image and transformed it to a format which is readable for the computer. A bi_mask is an image consisting of 0:s and 1:s showcasing where the boundary boxes are in the image. Before returning an image and a bi_mask, the bi_mask has to be created by drawing a new image using the polygons from the annotated dataset to know where to place the boundary boxes. I chose to return an image and bi_mask because that would be what I would need later on when training my models (i.e. model is supposed to learn bi_mask from image), and based on the dataset architectures I had seen during my research returning the original image and its binary mask is the convention.

The only steps left for the dataset before training starts is to create a dataset object by initialising it, split the dataset object into a train, validation, and test set, and creating a dataloader for each set.

Then it was time to create a model. Throughout my research on image segmentation I constantly encountered a model called UNet, which seems to be the preferred model for tasks like these. Subsequently I decided to build a UNet model based on figure 1.

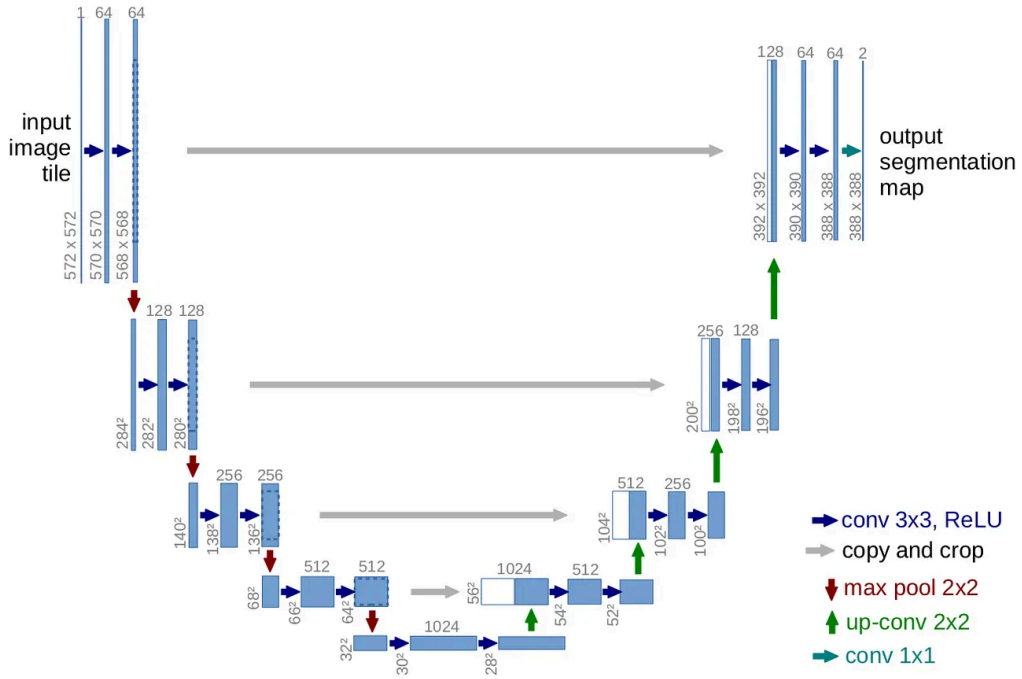

Figure 1
UNet architecture (Ronneberger et al. 2015) .

I modelled my second model on one of the other architectures I had seen being used for tasks like these – the SegNet model (see figure 2). There is no particular reason for why I chose the SegNet model, it was different enough from UNet for me to explore the existing architectures a bit more and further investigate the relationship between the encoding and decoding layer.

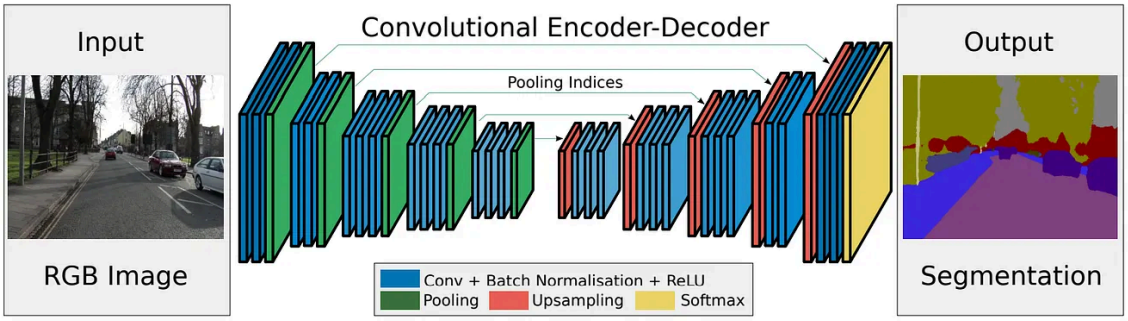

Figure 2
SegNet architecture (Badrinaryanan et al. 2017).

After having built the UNet model I moved on to develop the training loop, and evaluation part of my code. For said section I mainly replicated all other training, validation and evaluation loops I have created throughout the MLT program. The exception is the criterion, where I decided to use the dice as the loss function, instead of the binary cross entropy loss, due to the imbalance between the labels in the dataset — i.e. many more pixels are background (0) than in a boundary box (1). Before making the switch the models sometimes decided to predict 0 (background) for all/majority of pixels, which allowed for an accuracy of ~96% but all other statistics where 0 or close to 0.

Bibliography

Badrinarayanan, V., Kendall, A. and Cipolla, R. (2017) 'SegNet: A deep convolutional encoder-decoder architecture for image segmentation', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12), pp. 2481–2495. doi:10.1109/tpami.2016.2644615.

Ronneberger, O., Fischer, P. and Brox, T. (2015) 'U-Net: Convolutional Networks for Biomedical Image Segmentation', *Lecture Notes in Computer Science*, pp. 234–241. doi:10.1007/978-3-319-24574-4_28.