Data preparation

One of the main challenges of this assignment was the data, not necessarily implementing the architecture.The main issue was the class imbalance because the dataset contains large high resolutions images with 99,9% pixels being background and 1% Chinese characters.

The more simple mode,a plain CNN created noisy predictions, highlighting too much background, shadows, roads, tree markings. It did learn to predict but not the specific characters we wanted. The second model, a more complex U-Net model with ResNet architecture learned to simply predict the dominant background class for every pixel, as this was considered the easiest solution to minimize its loss and achieve high-accuracy. The output was not desired, displaying an entire black prediction (empty). The f1-score in this case was close to 0, given that the model was completely unable to predict the characters. Image resizing was also not sufficient to resolve this.

Given the results below I had to change the strategy and improve the curation of data to output better results.

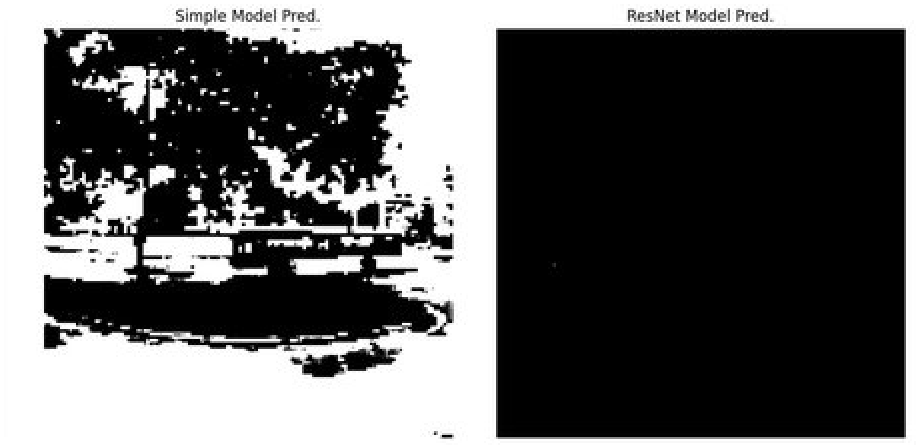


Figure 1, *Initial Model predictions*

At first i tried implementing YOLOV8 detector, a two stage, detect and segment pipeline. Soon it turned into a very laborious task because I did not implement it from the beginning and my code needed many changes to accommodate it. Thus, I abandoned the idea and turned to a more brute force method, the Sliding Window technique.

To prepare the data, I started by identifying the official list of training images followed by filtering (shuffling the list of usable files) and splitting the dataset : training (80%), validation (10%), and testing (10%).

The annotations in the *train.jsonl* file were parsed line-by-line and loaded into a Python dictionary (annotations_data) to be later used for lookouts using the image id. After inspecting the annotation data i noticed that the dataset included both non-Chinese and Chinese characters containing flags such as ‘is_chinese’ flag, attributes like ‘disorted’ or ‘occluded’. To make sure that the models process and generate masks properly I implemented a filter that allows only polygons specific to the Chinese characters to be kept.

Afterwards, within the ChineseDataset class, I implemented the Sliding Window technique which breaks down each image into hundreds of smaller patches. It works by pre-calculating the coordinates (x,y) for each possible windows across the entire dataset. The coordinates and source image path are stored in a master list that represent the new size of the dataset. During the training phase, the__getitem__ method retrieves these coordinates and image path for that specific index. It loads the full size images and is able to crop small patches for both ground truth and the image. I made sure to add padding for patches to guarantee that the output tensors look uniform.

The Models

I implemented two models called SimpleSegmentationModel, a simple Convolutional Neural Network and UNetResNetDeepSupervision which uses a U-Net architecture, and which will be referenced as Model 1 and Model 2 respectively within this report for brevity.

My motivation behind choosing two contrasting models,a baseline model and a complex one was to compare the difference in results and prove the advantages of using an U-net Architecture vs one with only 2 convolutional layers, which is obviously very ineffective as we will see in the results sections. Before delving deeper to discuss performance, it is crucial to explain how my models works so we can better asses their capabilities and failures.

Model 1 works by using a classic encoded-decoder structure. The model’s encoder uses a sequence of *Conv2d* and *MaxPool2d layers* for down-sampling the input images and extracting features. The decoder on the other hand uses *ConvTranspose2d* layers for the purpose of unsampling features. To be more specific, the features maps are gradually resized to the original dimensions of the image to reconstruct the segmentation mask. There is a final *Conv2d layer* with an applied Sigmoid function for generating a single-channel mask that represents the probability of each pixel. The model was trained using Binary Cross-Entropy (BCE) Loss with a standard Adam optimizer because i thought it could work well but combined with class imbalance it proved totally ineffective. Logits were highly negative which resulted in probability close to 0 when when applying the Sigmund function. I tried adjusting the decision threshold and lowered it to 0.05. It did perform better but not by much, because the loss was blurred by the large amount of background pixels. Because of this failure, I tried to find better solutions for Model 2 and implement a Dice Loss and Focal loss function to help combat the class imbalance which did slightly improve performance.

Model 2 as previously mentioned is based on U-Net Architecture but replaces the standard convultional encoder with a pre-trained ResNet-34 model. Thus, it utilizes transfer learning. The encoder, because it has been trained on a bigger dataset of images (ImageNet dataset) has better spatial knowledge, which i thought could bring us more accurate predictions scores. It also offers faster training, only needing a few epochs. What i liked about U-Net architecture was its use of skip connections, where features maps from the encoder are concatenated with the specific layers in the decoder. It adds spatial awareness and semantic information, very much needed for this project. The decoder uses ConvTranspose2d as the first model but it is able to to reconstruct a more precise and high-detail mask. The model was trained using deep supervision so that the shallower layers guide the deeper layers. I hoped that this technique would potentially help with the vanishing gradient problem. The frustrating problem of class imbalance as I previously mentioned was partly solved using these functions. DiceLoss is used to calculate the overlap between the predicted and true mask, thus making it great for getting solid positive predictions. Focal Loss was build with the purpose of refining the predictions at the pixel-level. The model needed to be forced to focus on a few, challenging to classify pixels that were obviously missing(some hard to read Chinese characters).

Practically, it diminished the loss contribution from the background pixels which allowed our model to focus on harder tasks. These techniques combined with the Sliding Window technique was my strategy for model 2.

Related to the training process, Model 1 was trained on a set of 15 epochs using a standard AdamW optimizer with a learning rate of 0.001.The input image were resized to 512x512 and a batch size of 32 was used to process them. We notice that the average training loss decreased over the 15 epochs beginning at 1.2266 and ending at 1.1297. From epoch 14 to 15 there is not much improvement in performance,dropping only 0.001. There was a significant improvement in the first few epochs (0.05) but after that the model struggled to learn. It reached its limit of what it is capable very early. Adding more epochs would have been redundant and not effective. In contrast with Model 2, it lacked features like having a rate scheduler and a two stage training strategy.This was done on purpose to analyze how more advanced technique can make a difference in the long run, purely for demonstration purposes.

Model 2 was trained on 3 epochs with a batch size of 16, a larger image size (768 x 768 pixels) and a standard AdamW optimizer with a learning rate scheduler ReduceLROnPlateau, used for monitoring the loss and reduce learning rate if we did not see any improvements for 3 epochs. A stopper was implemented (EARLY_STOPPING_PATIENCE = 5) to make sure we avoid over fitting and save waiting time. Thus, training would automatically stop if the validation loss wouldn’t improve for 5 epochs. Some data augmentation was needed here and to further prevent over-fitting we applied RandomHorizontalFlip, RandomRotation , and ColorJitter for adjusting of the brightness, contrast, and saturation. Normalisation was performed using the standard mean and deviation from ImageNet Dataset. The two stage training that was mentioned as being a superior attribute of the Model 2 needs some explanations here. In the first stage, the pre-trained ResNet encoder was frozen (the parameters requires_grad was set to False) and only the new decoder was trained with a learning rate of 0.001.This was a necessary step to prevent the gradient problem to negatively impact the pre-trained weights. In the second stage(fine-tuning) we unfroze the network and we set requires_grad to true, and trained for a few more epochs with a lower learning rate. (LEARNING_RATE_FINETUNE = 0.00005). This allowed the pre-trained decoder to adapt better to the specific task at hand.

The log of the training shows that sliding window strategy generated a lot of data from a limited amount of images. For example, from 676 training images we have 152,100 training windows, and for the 84 validation images a total of 18,900 validation windows were created. If we closely inspect, the model val loss dropped from 0.4430 after stage 1 to 0.3348 during stage 2, which does show some success. In epoch 3, while the training loss decreases, the validation loss unfortunately started to increase.This proves that although we did had improvements in comparison to model 1 we still struggle with over-fitting. Final validation loss is 0.3348.

Evaluation and testing

The models’ performances were assessed both qualitatively and quantitatively.

Quantitative Analysis

Methodology

The models' output is the probability map for each pixel that was converted into a binary segmentation mask. A probability threshold of 0.5 was applied. The predicted masks were compared against the ground truth. We calculated the following standard metrics: Accuracy, Precision, Recall, and F1-Score to have a thorough evaluation.

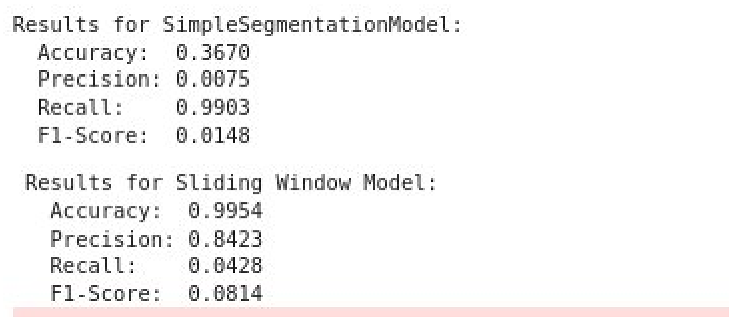
The *Accuracy* is high (but erroneous) for Model 2 (0.9954) and very low for Model 1 (0.3670).

Model 2, *F1 Score* shows a dramatic, but only relatively speaking, improvement compared to Model 1, reaching a score of 0.0814. *Precision* for model 2 (0.8432) indicates that the model is able to predict a character 70% correct. Model 1 has low *Precision* (0.0075) but higher R*ecall score* (0.9903)

What this means? Model 1 has the tendency to over-predict. The high recall means it does find every true character but the low precision score denotes they are false positive. The low accuracy shows that it is labeling many areas completely wrong.

Model 2 struggles with under-prediction. Although better,it still has many faults. The high precision tells us is correct approx. 84% of the time but the recall score is very low demonstrating that the model is very cautious and also fails to identify most characters.

Problem : Neither of them actually predicts properly.



```
Results for SimpleSegmentationModel:
  Accuracy:  0.3670
  Precision: 0.0075
  Recall:    0.9903
  F1-Score:  0.0148

 Results for Sliding Window Model:
   Accuracy:  0.9954
   Precision: 0.8423
   Recall:    0.0428
   F1-Score:  0.0814
```

Figure 2, *Evaluation metrics for Model 1 and 2*

Qualitative Analysis

The models, while performing poorly, do offer an interesting insight into how they operate, based on their outputs.

Looking at the simple Model 1, we can see that it wasn’t able to generalize the information about the Chinese characters, instead learning how to classify object outlines as belonging to the target category. This is corroborated by the high recall score and the 36% accuracy, which show how the model only overlaps with the ground truth by sheer coincidence, instead of a focused effort.

When it comes to the more complex, Model 2, using the Sliding Window approach, we can see that it behaves in a much more intentional way, albeit poorly, looking for what can indeed by defined as Chinese characters. It successfully ignores the background pixels as it managed to learn what to ignore, but it also isn’t sensitive enough to the information contained within the ground truth in order to properly detect the target feature, it instead detecting them only in the most visible conditions, hence the low evaluation metrics.

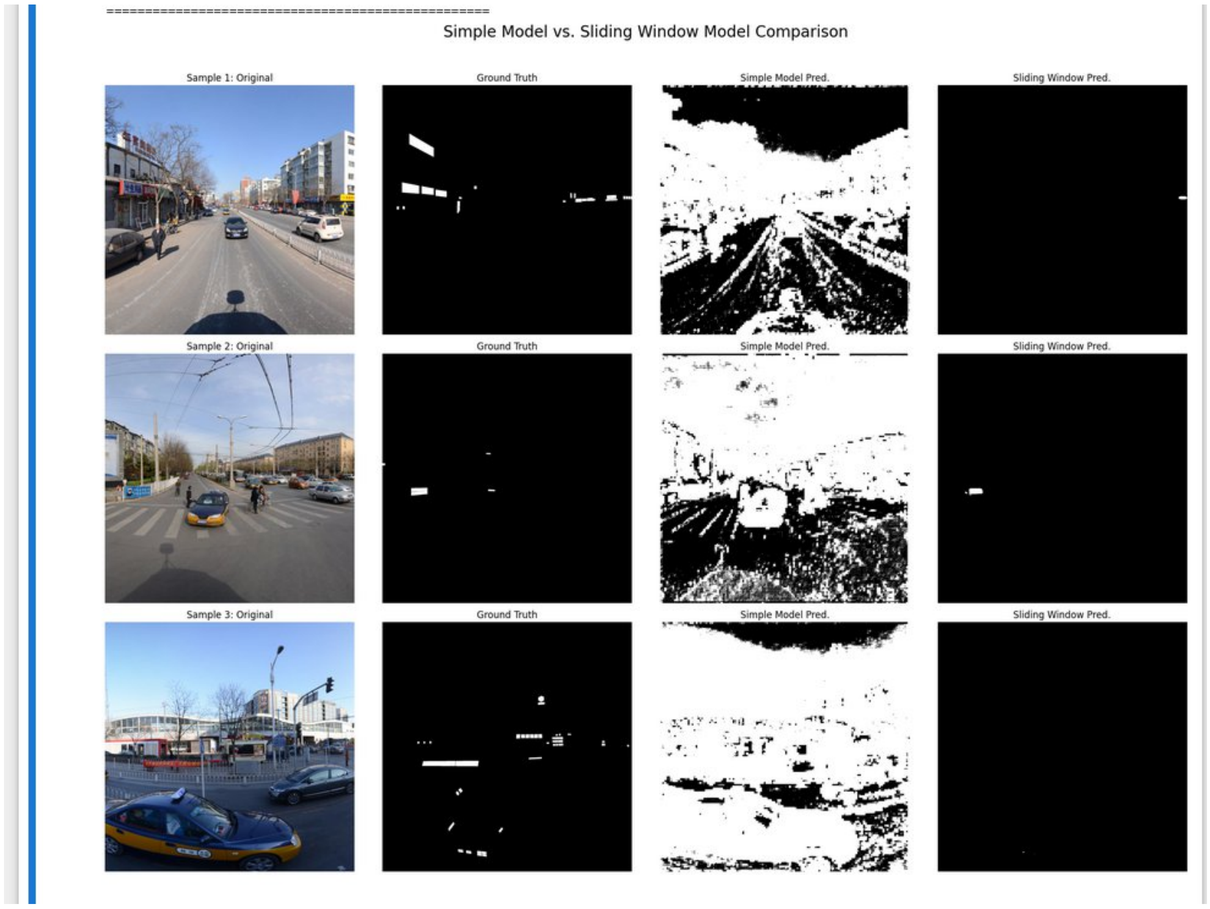


Figure 3, *Visual benchmark*