Assignment 1 Report

1. Part 1: Data Preparation

The dataset used here is a bit complex for me, and I spent a long time figuring out its structure to turn it into a usable dataset. I tried to use the whole dataset, but the efficiency was too low, so 500 images were selected for training. Moreover, both models learned almost nothing when using Dataset 1, which is the original full-image dataset where each sample is a whole image. The images and masks were resized to a fixed size (256×256).

Therefore, Dataset 2 was used instead. It splits each original image into dense overlapping patches of size 128×128 with a stride of 64 to ensure enough coverage and overlap. Patches with very small mask areas were filtered out using a threshold of 1% to remove invalid patches.

2. Part 2: Model

Two models were used here: a basic CNN model and a UNet model.

CNN Model

Encoder: 3 convolutional blocks, each including a convolution layer + ReLU + max pooling. The channels increase from 3→16→32→64 to extract features.

Decoder: 3 transpose convolution blocks, each including a transpose convolution + ReLU (the last layer outputs 1 channel) to upsample feature maps back to the original image size.

UNet

Encoder: 3 convolutional blocks, each with 2 convolutions + ReLU + max pooling. Channels increase from 3→64→128→256 to extract multi-scale features.

Bottleneck: convolution block 256→51.

Decoder (upsampling): 3 transpose convolution + convolution blocks, each step concatenates the upsampled features with corresponding encoder features through skip connections.

Output layer: 1×1 convolution, reducing channels to 1 to generate the segmentation mask.

The bottleneck channels from 1024 to 512 helps the model better preserve small target features during decoding.

3. Testing and Evaluation

The metrics used for evaluation include Accuracy, Precision, Recall, F1-score, Dice, and IoU.

Model	Accuracy	Precision	Recall	F1 Score	Dice Score	IoU
CNN pos_weight=150	23.60%	14.83%	98.60%	25.78%	25.78%	14.79%
CNN pos_weight = 20	65.01%	26.09%	83.71%	39.78%	39.78%	24.83%
UNet pos_weight = 120	68.28%	28.85%	92.57%	43.99%	43.99%	28.19%
UNet pose_weight = 20	89.00%	57.14%	72.95%	64.09%	64.09%	47.15%

The CNN model performs poorly when pos_weight is set too high, while UNet performs better under the same conditions. These results are obtained after repeated tuning. When using normal pos_weight values, the CNN model achieves acceptable results, and UNet performs much better. When pos_weight is set very high (150), the CNN achieves an extremely high recall (98.60%), but the precision is very low (14.83%), resulting in poor F1, Dice, and IoU scores. This indicates that the model predicts almost all pixels as target, which increases recall but leads to many false positives. Reducing pos_weight to 20 improves the precision (26.09%) and F1 (39.78%), Dice (39.78%), and IoU (24.83%). The accuracy increases by almost 180%.This shows that a appropriate weighting helps the model focus more on actual target pixels rather than overpredicting. With pos_weight of 120, UNet already achieves passable Accuracy(68.28%),F1 (43.99%), Dice (43.99%), and IoU (28.19%). Lowering pos_weight to 20 makes the accuracy reachs up to 0.89 and improves precision (57.14%) and overall F1 (64.09%), Dice (64.09%), and IoU (47.15%) .This demonstrates that UNet’s skip connections and encoder-decoder structure allow it to better capture small target areas.

Using Dataset 1, both models initially could not learn any meaningful information—during training, the loss decreased, but the Dice score remained very low or zero. This indicates that the models struggled to capture the target features.

Since the target pixels are much fewer than the background pixels, increasing pos_weight helps the models focus more on small target areas, improving detection performance. The recall values are very high mainly because the number of target pixels is very small compared to the background pixels.

4. Bonus

4.1 Dataset

The bonus part uses data from the same original database but processed differently. First, folders were created for each Chinese character, and each image was placed into the corresponding folder (over 70,000 characters). Then, only characters with more than 100 images were selected. Models were trained with 50-class and 200-class settings separately.

4.2 Model

A basic CNN model was used:

Encoder: 3 convolutional blocks (conv + ReLU + max pooling) with channels 3→32→64→128 to extract features and downsample.

Fully connected layers: 2 linear layers + ReLU + Dropout, outputting num_classes nodes for classification.

4.3 Test and Evaluation

Learning rate	classes	Accuracy	Precision	Recall	F1-score
0.001	50	85.24%	87.15%	74.14%	78.97%
0.001	200	74.84%	78.59%	62.63%	68.20%
0.001	500	66.12%	77.25%	51.29%	59.06%

Overall, the performance looks quite good, especially when the number of classes is small. With 50 classes, the model works the best and reaches the highest accuracy(85.24%),Precision(87.15%),Recall(74.14%) and F1-score(78.97%). Before filtering the data, the model did very poorly because there were too many classes and some characters had only a few images, which made it hard for the model to learn.

When the number of classes increases from 50 to 200 and then to 500, the performance becomes lower. This is expected, because more classes mean the task becomes more difficult and the model needs to tell apart many similar characters. Even so, the results are still acceptable to me, and the model manages to handle a large number of classes reasonably well.