COMS4036A & COMS7050A
Computer Vision

Prof. Richard Klein

Lab 3: Segmentation with Deep Learning

1 Introduction

Tino is quite pleased with the quality of our segmentation system. He was surprised we could get such good results from such a small amount of data. He is convinced that the hand-crafted features are sensible starting points, especially when the amount of data we have is limited or when we want to explicitly encourage the model to focus on specific aspects of the data.

Nevertheless, he thinks it would be a good idea to try some segmentation models based on Deep Learning, which he recently heard about on TikTok.

In this lab, we will investigate the efficacy of some deep learning segmentation models. You will implement the UNet model from scratch using Pytorch and experiment with more advanced architectures from popular libraries.

Tino wants you to use Pytorch for this lab. If you have not used Pytorch before, then these links should help you get started:

- Installation: https://pytorch.org/get-started/locally/
- Basics: https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

2 Data

Tino suspects we might need more data to train these larger deep learning models. He carefully labelled a few more images, which are provided in the Moodle handout.

There are 10 training images, 4 validation images, and 4 test images.

The training images from the last labs are included under training.

Tino is unsure whether this will be enough, so we should take extra care that our models do not overfit. When training your models, you should perform any data augmentation that may help. You may use the built-in PyTorch data loaders and augmentation pipelines.

3 UNet Model Construction

Implement the UNet model from scratch in PyTorch. UNet is a popular image segmentation architecture that captures spatial and contextual information.

You should consider two main variants:

- **Variant 1:** Use `torch.nn.ConvTranspose2d` for upsampling.
- **Variant 2:** Use `torch.nn.Upsample` for bilinear upsampling.

3.1 Tino's UNet Design

We will implement a slightly modified version of the UNet.

- The original UNet paper takes in a grayscale image, but the colour information is important for the puzzle. So, our model input will be `3x512x512` pixels instead.
- The normal UNet architecture does not use padding and the final output size is smaller than the input size. In this lab, we would like to maintain the original input size. We should use `padding=(1,1)` with each `Conv2d` to ensure that the feature maps remain the same size. We should `MaxPool2d` with `kernel_size=2` and `stride=2`. This will exactly halve the size of the feature maps with each downsample.
- The `ConvTranspose2d/Upsample` should exactly double the size of the feature maps. For the `ConvTranspose2d`, we can use `kernel_size=(2,2)` and `stride=(2,2)`. For the `Upsample` layers, we should use `scale_factor=2`.
- The output should have two layers. A `1` in layer 0 should indicate a background pixel, while a `1` in layer 1 should indicate a foreground pixel. These values should sum to 1 via a softmax that runs across the channels at each pixel.

The model should follow the architecture and feature map sizes shown in Figure 1. This architecture matches the original UNet architecture in terms of convolution, pooling, and upsampling, but with the correct padding we ensure that the feature map size halves or doubles exactly. The spatial dimensions of each feature map are given below the blocks.

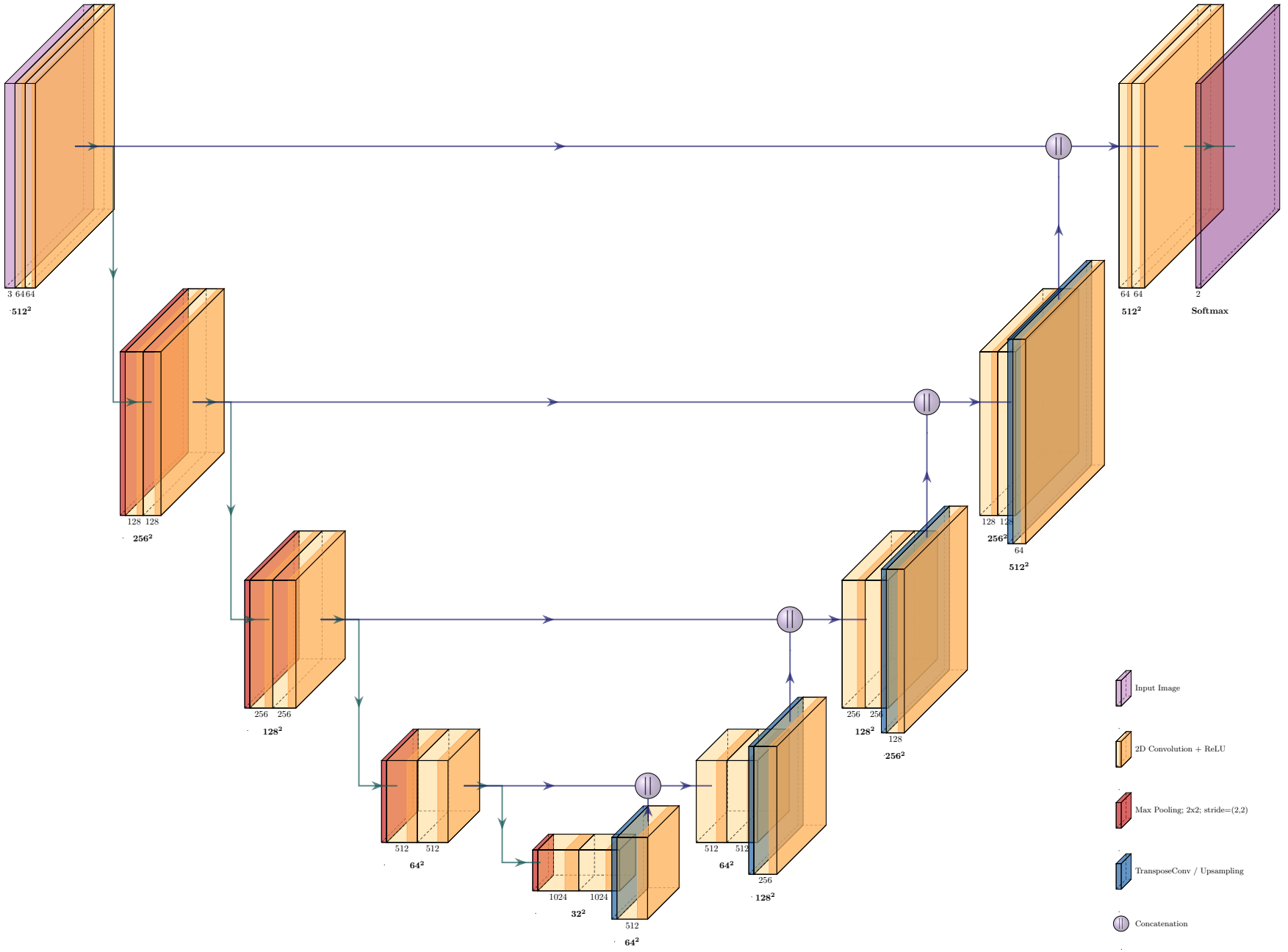

Figure 1: UNet Architecture

3.2 Training and Evaluation

Training

- Implement the training loop using **binary cross-entropy (BCE) loss** for pixel-wise classification.
- Use the **Adam optimizer** to update model parameters.
- Regularly save checkpoints and evaluate your model on the validation set to avoid overfitting.

Weights and Biases Integration

You must integrate with **Weights and Biases (wandb)** to log and visualise the following metrics:

- **Training and Validation Loss:** Track the BCE loss over epochs.
- **IoU (Intersection-over-Union):** Calculate IoU for both training and validation sets to monitor segmentation performance.

Use the wandb dashboard to create and submit the following plots:

- Loss curves (training and validation loss over epochs).
- IoU curves (training and validation IoU over epochs).

Based on the training and validation loss curves, comment on whether the model is overfitting and how you could recognise and deal with this.

Evaluation

Once training is complete, evaluate your model on the test set. Report the following metrics: Accuracy, Precision, Recall, F1-Score, and IoU.

Select the best model based on validation IoU, then report its performance on the test set.

4 Other Architectures

After building your baseline model, explore more advanced architectures and compare their performance.

UNet and DeepLab v3+ Variants

- **UNet Variants:** Experiment with at least 3 different backbone architectures (e.g., ResNet, EfficientNet) using the *Segmentation Models* pytorch library.
- **DeepLab v3+:** Train a DeepLab v3+ model across at least 3 backbone architectures to compare performance.

For these experiments, use the pre-built models in the *Segmentation Models* PyTorch library:

- GitHub: https://github.com/qubvel-org/segmentation_models.pytorch

Use the validation loss to identify overfitting. Use the model with the best validation loss from each of the 6 models above (make a model checkpoint each time the validation loss improves, and then use the best checkpoint when you test the model at the end). Comment on whether you noticed overfitting in the different architectures.

Comparison

Compare the performance of different models using the same evaluation metrics (accuracy, precision, recall, F1-score, and IoU) – summarise these in a table. Visualise the differences in the segmentation masks produced by each model.

5 Submission

Your submission must include:

- **Jupyter Notebooks:** Clearly organised notebooks with code, results, and explanations for:
  - The UNet baseline and its two variants.
  - Comments on overfitting, with various graphs generated by **wandb**.
  - Experiments with UNet and DeepLab v3+ using different backbones, performance metrics and commentary as requested above.
- **Comments** A concise summary of your findings, including:
  - Insights on model performance.
  - Challenges faced during implementation.
  - Suggestions for further improvements.

Good luck! Tino is counting on you!