ELEC 475 Computer Vision

Lab 3 – Semantic Segmentation & Knowledge Distillation

Queen's University

Faculty of Engineering and Applied Science

Nathan Duncan 20283699
Daniel Dubinko 20229482

November 11, 2025

Network Architecture

Our custom student network, `CompactStudentModel` (implemented in `student_model.py`), follows the lab's "Example Architecture." It consists of three primary components.

1. Backbone (Encoder): We use a pre-trained MobileNetV3-Small [1] as a lightweight feature extractor. We tap intermediate features at three scales: `'low'` (stride 4, 24 channels), '`mid`' (stride 8, 40 channels), and '`high`' (stride 16, 576 channels).

2. Context Module: The '`high`' feature map is fed into an Atrous Spatial Pyramid Pooling (ASPP) module [2]. This module captures multi-scale context using parallel atrous convolutions (rates {1, 6, 12, 18}), a 1x1 convolution, and a global image pooling branch. The concatenated outputs are projected to 256 channels, with a 0.5 dropout applied.

3. Decoder: The decoder up-samples the ASPP output (256 channels) via bilinear interpolation to match the '`low`' feature map's dimensions. This `'low'` map is passed through a 1x1 convolution (reducing channels to 48) [2]. The two tensors are concatenated (304 channels) and passed through a final refinement block of two 3x3 convolutions. A 1x1 classifier projects this to 21 classes (PASCAL VOC), and a final bilinear up-sample scales the mask to the original input image size.

For feature-based distillation, the model includes three 1x1 "adapter" convolutions to match the student's low, mid, and high feature map channels to the teacher's.

Knowledge Distillation

Our process (in `train_kd.py`) trains the `CompactStudentModel` (student) under the supervision of a frozen, pre-trained FCN-ResNet50 (teacher). We implemented a combined distillation strategy using a three-part function for total loss, $\mathcal{L}_{total}$, defined by the following formula:

$$\mathcal{L}_{total} = (1-\alpha)\cdot\mathcal{L}_{CE} + \alpha\cdot\mathcal{L}_{KD_{response}} + \beta\cdot\mathcal{L}_{KD_{feature}} \quad (1)$$

Where:

1. $\mathcal{L}_{CE}$ **(Hard Loss):** This is the standard `nn.CrossEntropyLoss` between the student model's final output logits (z_s) and the ground-truth labels (y).
2. $\mathcal{L}_{KD_{response}}$ **(Response-Based Soft Loss):** The Hinton-style KL Divergence loss [3], calculated between the log_softmax of the student's and teacher's logits, both "softened" by a temperature τ.
3. $\mathcal{L}_{KD_{feature}}$ **(Feature-Based Loss):** An `nn.CosineEmbeddingLoss` that encourages the student's three intermediate feature maps (processed by the 1x1 adapters) to

mimic the corresponding feature maps for (`low`, `mid`, `high`) from the teacher's ResNet backbone.

By setting the hyperparameters α and β (via command-line arguments in `train.txt`), we can control the contribution of each distillation method.

Hyperparameters

The training process was defined in train_kd.py and executed using the commands in train.txt and ran on an Apple Mac using the MPS backend. The model was trained on the PASCAL VOC 2012 dataset for 35 epochs using the Adam optimizer with an initial learning rate of 1e-3 and a `ReduceLROnPlateau` scheduler. We used a batch size of 8 to maximize resource usage on the Mac.

For data augmentation and preprocessing, images were resized to 520, with a random 480x480 crop for training and a center 480x480 crop for validation. Standard ImageNet mean and standard deviation normalization was applied.

For the combined distillation run, the loss weights were set to $\alpha = 0.9$ (response loss) and $\beta = 10.0$ (feature loss), with a temperature of $\tau = 4.0$.

Table 1: Training time for models with different variations of knowledge distillation.

Total Training Time (Without KD)	(56.57 min)
Total Training Time (Response Based KD)	(95.94 min)
Total Training Time (Feature Based KD	(105 min)

The alpha value that corresponds to how much the teacher response is valued vs the hard loss of the student model. This means the model is ignoring 0.9% of the ground truth to make space for the teachers soft response.

The beta value means that for every 'unit' of importance you award to the hard_loss, a unit (0.5) of importance to matching the teacher's internal features which is feature_loss.

The following plots were generated by `train_kd.py` for our base model without knowledge distillation and the combined distillation run. Figure 1 shows the base case without KD and Figure 2 presents the combined KD student model's loss and validation mIoU during training.

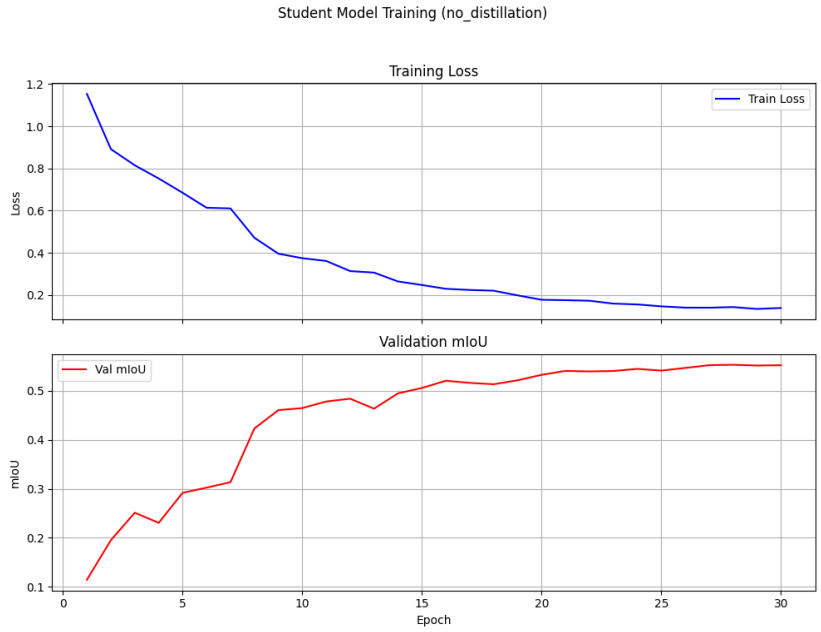


Figure 1: Training Loss and Validation mIoU over 30 epochs of the student model trained without knowledge distillation.

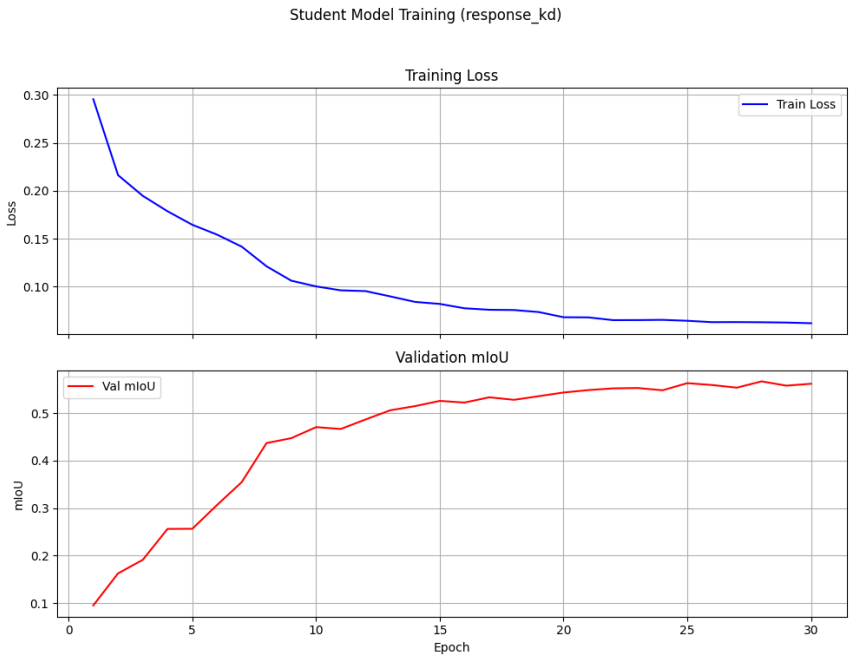


Figure 2: Training Loss and validation mIoU over 30 epochs of the student model trained with combined feature knowledge distillation.

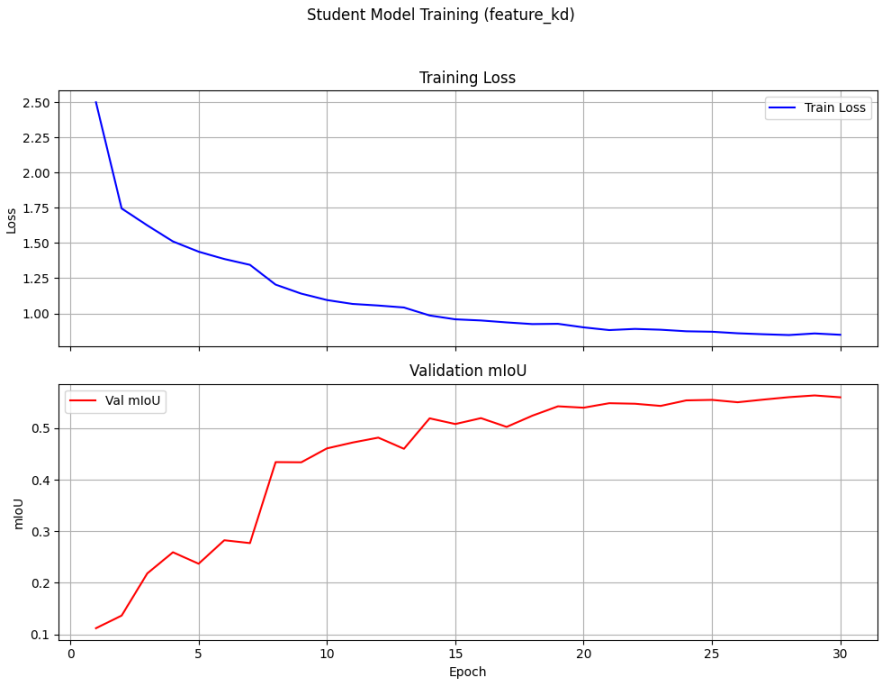


Figure 3 - Training Loss and validation mIoU over 30 epochs of the student model trained with combined response knowledge distillation.

Experiments

All experiments were run on an Apple Mac using the MPS (Metal Performance Shaders) backend for acceleration. Our experimental procedure was conducted in three phases.

1. **Teacher Baseline:** We first established a performance baseline by running python test_teacher.py. This script loads the pre-trained FCN-ResNet50 model (frozen) and evaluates its performance on the 1449 images in the PASCAL VOC 2012 validation set. This gave us our "upper bound" mIoU and a performance baseline for inference speed.
2. **Student Training:** We then trained our `CompactStudentModel` in two distinct modes as defined in train.txt:
    - **Without KD:** We ran `python train_kd.py --alpha 0.0 --beta 0.0 --epochs 35`. This command set both distillation loss weights to zero, training the model for 35 epochs using only the standard Cross-Entropy (hard) loss.
    - **With Combined KD:** We ran `python train_kd.py --alpha 0.9 --beta 10.0 --temp 4.0 --epochs 35`. This trained the model for 35 epochs using our full combined loss function, weighting both the response-based ($\alpha = 0.9$) and feature-based ($\beta = 10$) distillation losses.

- In both cases, the script saved the model weights (`.pth`) from the epoch that achieved the highest validation mIoU.

3. **Student Evaluation:** Finally, we evaluated our two trained student models. We ran the commands from test.txt (e.g., `python evaluate.py --model_path` models/student_model_no_distillation.pth), which load the saved weights, iterate through the entire validation set, and compute the final mIoU and average inference time. To generate the qualitative images for this report, we re-ran these commands with the `--visualize --num_images 10` flags, which (as seen in evaluate.py) saves the [Input, Ground Truth, Prediction] comparison overlays for the first 10 images.

As our primary accuracy measure, we used the mean Intersection over Union (mIoU), which is the standard metric for semantic segmentation. In our code, this is implemented using the `torchmetrics.classification.JaccardIndex`.

Intersection over Union (IoU), or Jaccard Index, measures the overlap between the predicted segmentation mask and the ground truth mask for a single class. It is calculated as the area of their intersection divided by the area of their union.

The "mean" in mIoU signifies that this IoU score is calculated independently for every class in the dataset (e.g., "aeroplane", "bicycle", "person", etc., excluding the background), and the final reported score is the average of all these per-class scores.

This metric is far more robust than simple pixel accuracy because it is not skewed by large, easy-to-segment classes (like "background"). It severely penalizes both false positives (predicting an object where there is none) and false negatives (missing an object that is present), making it an excellent measure of the true quality and "tightness" of the predicted segmentation masks.

The table below summarizes the quantitative results from our three experiments.

Table 2:Comparison of core model mIoU, number of parameters and inference speed.

Model	mIoU	# Parameters	Inference Speed (FPS)
Teacher	0.6739	35,322,218	17.42
Without	0.5533	8,843,285	290.14
Response-based KD	0.5774	8,843,285	285.35
Feature-based KD	0.5636	8,843,285	289.41

Qualitative results are presented in Appendix I.

Performance

The performance of our `CompactStudentModel` was a success. Quantitatively, the student model trained from scratch without distillation achieved a final mIoU of 0.5538. This is a respectable score for a lightweight model, though it is understandably lower than the heavyweight teacher FCN-ResNet50 teacher (0.6739 mIoU). The student model was also much more lightweight reducing the number of parameters by about a factor of four.

Our primary goal, however, was to improve this student model using knowledge distillation. Our experiment with response-based and feature-based distillation resulted in a final mIoU of 0.5636 and 0.5774 respectively. This was our expectation as the knowledge distillation has access to more training information through the teacher, outperforming the student without distillation.

Despite the mIoU, the experiment was a success in inference speed. The student model (~290 FPS) is over 3.1x faster than the teacher (~17 FPS), achieving the lab's goal of creating a compact, efficient model.

The main challenge was correctly implementing the feature-based loss between mismatched feature maps. This was overcome by using the 1x1 adapter layers in student_model.py to match channel dimensions and `F.interpolate` in `train_kd.py` to match spatial dimensions before calculating the cosine loss.

LLM

The Google Gemini LLM was used for content generation in this lab.

LLM Conversation Links:

- https://gemini.google.com/share/606e5fc870cf (link 1)
- https://gemini.google.com/share/38d4b1b38f92 (link 2)
- https://gemini.google.com/share/bac021005f3f (link 3)

It was found to be very effective, especially when given small, specific prompts or example code to modify. It was also helpful for Colab configuration. The LLM did produce some errors, but it could often self-correct when provided with the console output and a description of what went wrong.

References

[1] A. Howard, *Searching for MobileNetV3,* Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), 2019.

[2] L. Chen, G. Papandreou, F. Schroff and H. Adam, *Rethinking Atrous Convolution for Semantic Image Segmentation,* arXiv preprint arXiv:1706.05587, 2017.

[3] G. Hinton, O. Vinyals and J. Dean, "Distilling the Knowledge in a Neural Network," *NIPS Deep Learning and Representation Learning Workshop,* 2015.

Appendix I

The following images were generated by `evaluate.py --visualize`. They compare the model's prediction mask (overlaid on the image) against the ground truth mask. All images present the input image, ground truth overlay, and model prediction overlay from left to right. Image sets in pairs show a successful case (top) and unsuccessful case (bottom).

Teacher (ResNet50):

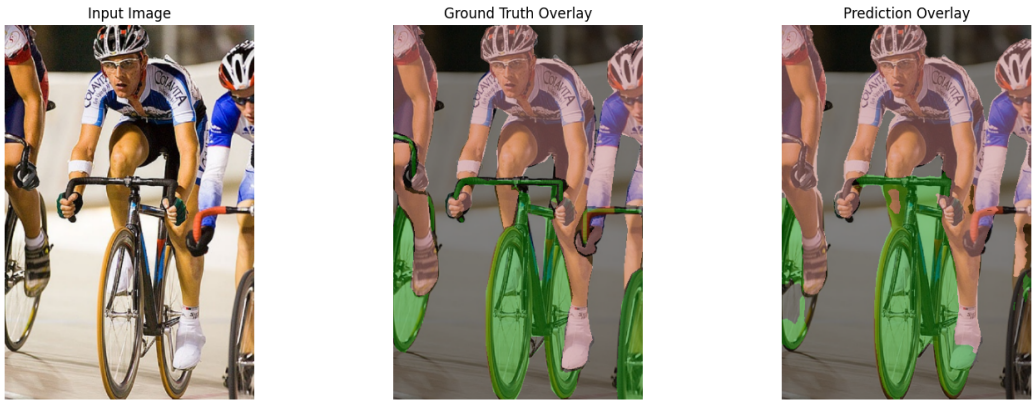
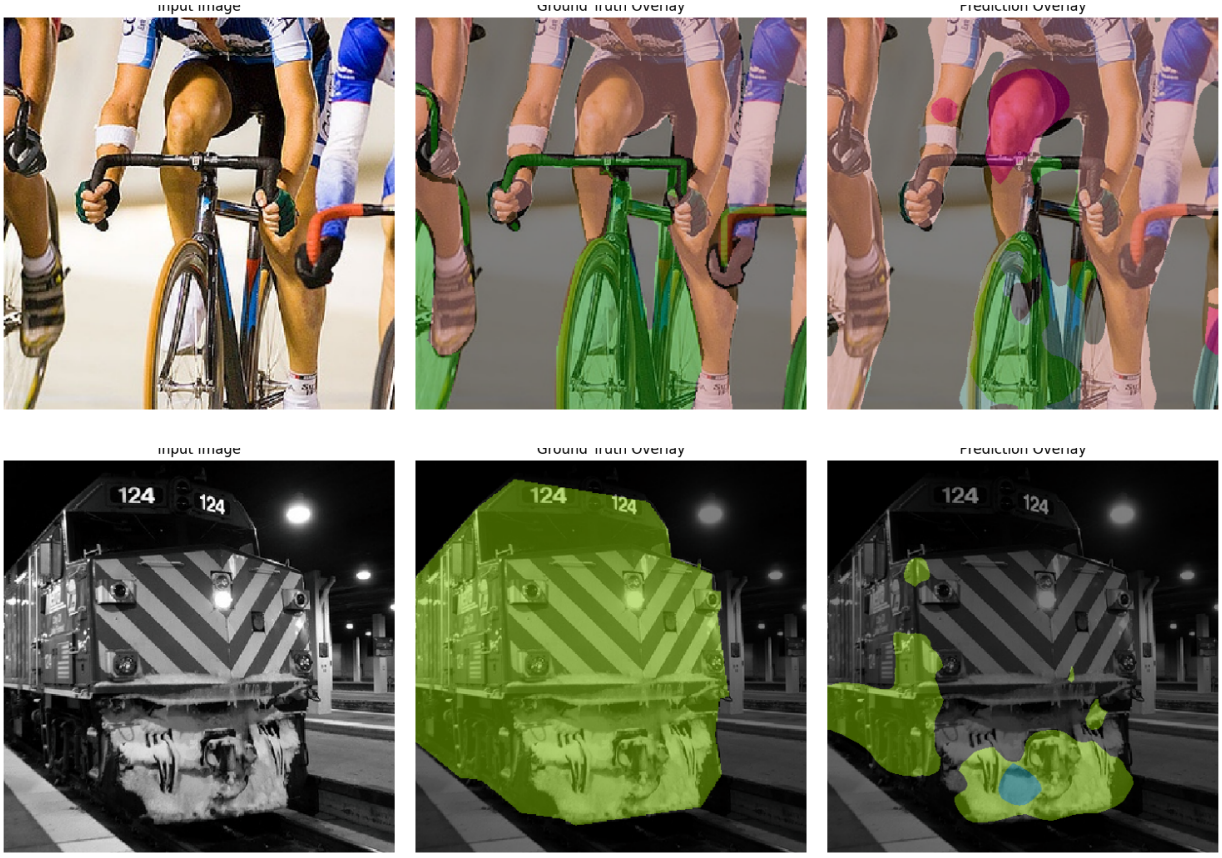


Figure 4: This set of images shows the effectiveness of the Teacher model (FCN-ResNet50) generating an accurate segmentation overlay to a complex input image.

Student with No Distillation:



Student with Response Distillation:

Student with Feature Distillation: