ELEC 475 Project

Ethan Parliament

20220776

2024-12-03

Network Architecture

The model consists of an encoder/decoder structure, accepting a 3 channel RGB image of size (256, 256). The encoder is made up of 4 encoder blocks. Each encoder block includes a convolutional layer, followed by batch normalization, and finally ReLU. The first block outputs a channel depth of 32, then 64, then 128, and finally outputs a bottleneck with channel depth of 256. Before the tensor is passed to the next layer in the encoder, a max pool operation is performed with a kernel size of 2. Each encoder block also performs some dilation on the input tensor, starting at a dilation of size 1, and doubling until size 8.

The decoder portion of the model utilizes skip connections from the encoder layer. Each decoder block performs a convolution, followed by batch normalization, and ReLU. There are 3 decoder blocks being used, each receiving input from the encoder blocks (not the bottleneck). The encoder input is concatenated with the output from the previous decoder layer before being passed through the model. After each decoder layer, a boundary refinement block is used to help the model pick up on the boundary of objects in the image [1]. This consists of a series of convolutions, batch normalizations, and ReLU. Finally, the decoder structure ends with a final convolution, resulting in an output with channel depth of 3, and back to the original (256, 256) size.

Commented [EP1]: Source?

Knowledge Distillation

2 different types of knowledge distillation were used: response-based distillation, and feature based distillation. Both take the same approach to training a model without distillation, but used different loss functions.

Response Based Distillation

Response based distillation works by calculating the difference between the softmaxed logits of the student and the teacher models. In this case, the total loss of the model was a mix of both Cross Entropy Loss and Response Loss. A portion of the code used to calculate response loss can be seen below.

```
soft_student = nn.functional.log_softmax(student_output / temperature, dim=1)
soft_teacher = nn.functional.softmax(teacher_output / temperature, dim=1)
response_loss = F.kl_div(soft_student, soft_teacher, reduction='batchmean') *
temperature**2
```

The mixture of Cross Entropy Loss and Response Loss was controlled by 2 scalars

```
total_loss = alpha * ce_loss + beta * response_loss
```

Feature Based Distillation

Feature based distillation is a technique where the student model learns to mimic the intermediate feature maps of the teacher model. Like response-based learning, a mixture of Cross Entropy Loss and Feature Loss was used to find the total loss.

One challenge with feature distillation was being able to access the intermediate layers of the teacher model (In this case ResNet50). To do this, PyTorch's register_forward_hook method was used to extract the feature maps of the backbone layer, which were then used along with the feature maps of the student's encoder layers for feature-based distillation. Because of the student model's smaller size, only the first 3 layers were used for feature distillation.

Once the feature maps for both models were flattened, the cosine similarity function was used to compute the similarity between each feature map.

```
feature_loss += 1 - F.cosine_similarity(student_features_flat,
teacher_features_flat, dim=1).mean()
# Total loss = Cross-Entropy loss + feature distillation loss
total_loss = alpha * ce_loss + beta * feature_loss
```

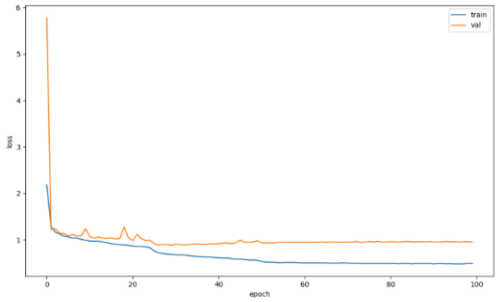
Hyperparameters

A table of the hyperparameters used for each of the models can be seen below.

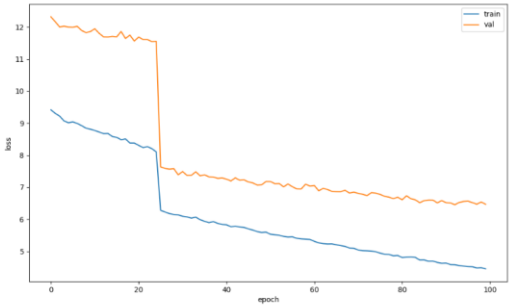
Hyperparameter	Without distillation	Response-based	Feature-based
Epochs	100	100	60
Batch size	32	32	16
Learning rate	0.01	1e-5	1e-4
Weight decay	0.005	0	0.0005
Transforms	Resize (256,256), ToTensor()	Resize (256,256), ToTensor()	Resize (256,256), ToTensor()
Optimizer	SGD	Adam	Adam
Scheduler	StepLR(step size=25, gamma = 0.1)	ReduceLROnPlateau	ReduceLROnPlateau
Alpha (CE loss)	1	5, after 25 epochs: 1	0.7
Beta (distillation)	0	0.00005	0.3
Temperature	NA	5	2

Shown below are the loss plots for each of the distillation methods. They all show a smooth train loss continuously decreasing, with the validation loss following along slightly higher. In the response-based distillation loss, there is a sharp decline in epoch 25 – that is when the values of alpha and beta changed to promote more distillation over cross entropy. After the drop, both the train and validation loss continue to decrease as expected.

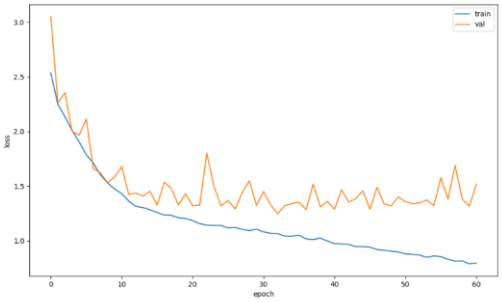
No distillation loss plot:



Response-based distillation loss plot:



Feature-based distillation loss plot:



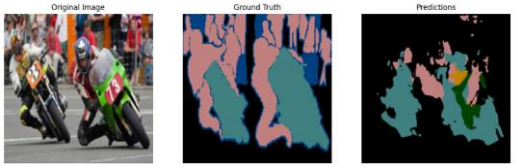
Experiments

Each of the models were valuated using the mean intersection over union metric (mIoU). The mIoU was calculated as the mean of the IoU from each class. Each model performed relatively the same, with similar accuracies and similar inference times. The results of each of the models can be seen in the table below.

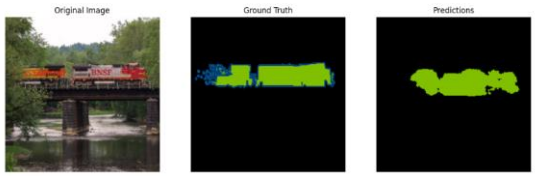
Knowledge distillation	mIoU	# Parameters	Inference Speed
Without	0.136	2,349,141	110msec/image
Response-based	0.128	2,349,141	113msec/image
Feature-based	0.103	2,349,141	120msec/image

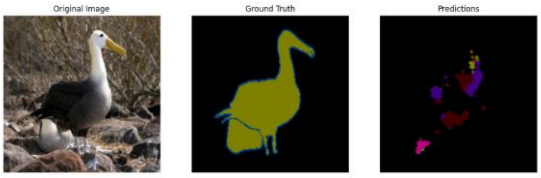
Each of the models was trained on Google Colab's T4 GPU. When training, the model without distillation took around 25 seconds per epoch (17msec/image), both distillation models took just over 2 minutes per epoch (81msec/image). For testing the models, an NVDIA mx250 GPU was used, taking just under 3 minutes to test each model. Some example images from each model are shown below:

No distillation:

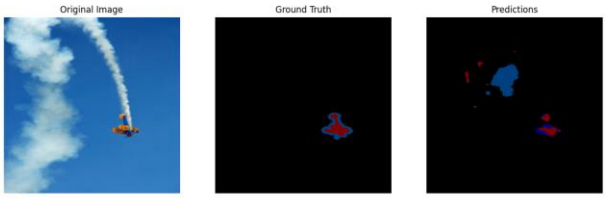


Response-based distillation





Feature-based distillation



Discussion

The performance of my model was much poorer than what was anticipated when I first started. The distillation methods did not seem to increase the performance of the model at all, with the most successful model being the one trained without distillation.

I trialed 6 or 7 different model architectures, each with their own benefits and drawbacks, and none seemed to perform as well as I had hoped. Even in combination with different hyperparameters, not much seemed to work.

One of the biggest challenges I faced was with class imbalance – especially with the background class taking over the predictions, which I tried dealing with in a few ways. At first, I computed what the class weights should be to perform a weighted cross entropy loss. This worked somewhat well, and stopped the background class from dominating the predictions, however, this made the shape of the predictions much worse than before, and they became blob like instead of sharper edges like what the ground truth had. I also tried working with a combination of cross entropy loss and dice

loss to help the predictions improve on the shapes, but that did not work very well either. Finally, I added boundary refinement modules to my model architecture, which greatly improved dealing with the object shapes [1].

Another challenge that I faced was during feature distillation. Trying to access the intermediate feature maps of resnet50 was not so straight forward. The model comes with an easy way to access the last feature map before prediction, but it took a lot of research to find a way to access the middle layers of the model. Eventually I came across a way to do this from deep inside an old discussion post on the PyTorch website, using the register_forward_hook method [2].

Overall, I expected the model to perform much better given the vast number of hours I put into training and development (~100 hours). Although there is a bright side to this, and that is all the different learning I did on how to tackle some of the problems that I faced. Even though distillation learning did not improve the model's performance, I now have a much deeper understanding of the topic, and strategies on how I can implement it in the future.

References

[1] SConsul, "Global Convolutional Network," 2018. [Online]. Available: https://github.com/SConsul/Global_Convolutional_Network/blob/master/model/BR.py. [Accessed 1 12 2024].

[2] "How can I load my best model as a feature extractor/evaluator," April 2019. [Online]. Available: https://discuss.pytorch.org/t/how-can-l-load-my-best-model-as-a-feature-extractor-evaluator/17254/23. [Accessed 2 12 2024].