

---

# From Images to Insights: Analyzing Semantic Segmentation with Fully Convolutional Network

---

Sarah Yang  
sxyang@ucsd.edu

## Abstract

In this study, we tackle the semantic segmentation task using the PASCAL VOC-2012 dataset, which consists of 21 object categories with pixel-wise annotations. We implemented baseline model of a Fully Convolutional Network (FCN). We then apply cosine annealing learning rate scheduler, different transformers, and address rare and imbalanced problems to improve on the baseline. We then explored enhancements through different model architecture, transfer learning with ResNet34 and U-Net to improve segmentation accuracy. Our experimentation with ResNet34 as a pre-trained encoder showed faster convergence and better generalization, but performance slightly declined due to overfitting and spatial resolution mismatches with the decoder. Further architectural modifications, including LeakyReLU activation, variable convolution filter sizes, and deeper layers, enhanced feature extraction and segmentation precision. Among all models, U-Net achieved the highest performance, with a mean IoU approaching 0.90 and mean pixel accuracy surpassing 74 percent, demonstrating its effectiveness in capturing spatial details and refining segmentation masks.

## 1 Introduction

Semantic segmentation is a fundamental task in computer vision, where each pixel in an image is classified into a specific category. Unlike image classification, which assigns a single label to an entire image, semantic segmentation provides dense predictions, making it particularly useful in fields such as autonomous driving, medical imaging, and remote sensing. In this project, we aim to develop a deep convolutional neural network (CNN) to perform pixel-wise segmentation, classifying each pixel as one of the 20 object categories or background, using the PASCAL VOC-2012 dataset. The dataset provides pixel-level annotations, allowing us to train a supervised learning model that learns meaningful visual representations.

Deep learning, particularly CNNs, has revolutionized computer vision by enabling models to automatically learn hierarchical features from raw images. However, standard CNN architectures, originally designed for image classification, struggle with spatial preservation, as fully connected layers discard spatial information. To address this, Fully Convolutional Networks (FCNs) were introduced, replacing fully connected layers with convolutional layers and upsampling layers to produce dense per-pixel predictions. By leveraging convolutional and deconvolutional operations, FCNs can generate high-resolution segmentation maps while maintaining efficiency.

To train our FCN model effectively, we employ Xavier weight initialization and Batch Normalization (BN). Xavier initialization ensures that the variance of activations remains stable across layers, preventing issues like vanishing or exploding gradients, which are common in deep networks. Meanwhile, Batch Normalization normalizes the activations within each mini-batch, reducing internal covariate shift and accelerating training convergence. These techniques not only improve stability but also enhance the network's ability to generalize across different images.

Semantic segmentation has significant real-world implications. In autonomous driving, accurate segmentation of roads, vehicles, pedestrians, and obstacles is crucial for safe navigation. In medical image analysis, segmentation helps detect abnormalities such as tumors in MRI scans. Additionally, in robotics and scene understanding, segmenting objects and environments enhances decision-making for robotic systems. Given these applications, improving segmentation accuracy through deep learning techniques is a critical research direction.

In this project, we will train an FCN from scratch, optimizing it to produce high-quality segmentation masks. Given the computational demands of training deep CNNs, our experiments require GPU acceleration for efficient processing. We will evaluate the model’s performance using standard metrics such as Intersection over Union (IoU) and pixel-wise accuracy, ensuring that the network effectively distinguishes between different object categories. Through this exploration, we aim to gain deeper insights into the power of deep learning in semantic segmentation and further refine techniques for pixel-level classification.

## 2 Related Work

We got insight from cosine Annealing Learning Rate Scheduler and carefully review the (<https://arxiv.org/abs/1608.03983>) [1]. SGDR (Stochastic Gradient Descent with Warm Restarts) introduces a cyclic learning rate schedule that periodically resets the learning rate to escape poor local minima and improve convergence. This method has demonstrated state-of-the-art performance on datasets like CIFAR-10 and CIFAR-100, highlighting its effectiveness in training deep neural networks efficiently.

We also got an insight from U-Net [2]: Convolutional Networks for Biomedical Image Segmentation. U-Net is a popular biomedical segmentation network that combines a contracting path for context capture with a symmetric expanding path for precise localization, allowing end-to-end training even on relatively small annotated datasets. By leveraging heavy data augmentation, U-Net effectively utilizes the available annotated samples, outperforming previous methods in tasks such as neuronal structure segmentation and cell tracking challenges.

## 3 Methods

### 3.1 Baseline

The baseline architecture implements a Fully Convolutional Network following an encoder-decoder structure. The encoder path consists of five convolutional blocks, each comprising a Conv2d layer ( $3 \times 3$  kernel, stride 2, padding 1), BatchNorm2d, and ReLU activation, with channel dimensions progressing from  $3 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow 256 \rightarrow 512$ . The decoder path mirrors this with five transposed convolution blocks, each containing a ConvTranspose2d layer ( $3 \times 3$  kernel, stride 2, padding 1, output\_padding 1), BatchNorm2d, and ReLU activation, with channel dimensions reducing from  $512 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32$ . The network concludes with a  $1 \times 1$  convolution layer that reduces the channels from 32 to 21 (number of classes) with no explicit activation function, as it outputs raw logits for the CrossEntropyLoss criterion. The network weights are initialized using Xavier/Glorot uniform initialization for convolutional layers and normal distribution for biases. Training is optimized using the Adam optimizer with a learning rate of 0.001, processing images of size  $224 \times 224$  pixels.

### 3.2 Improvements over Baseline

Improving the baseline Fully Convolutional Network (FCN) model for semantic segmentation requires advanced training strategies. In this work, we incorporate three key improvements: cosine annealing learning rate scheduler, data augmentation, and weighted Loss. These enhancements aim to improve model convergence, generalization, and overall segmentation accuracy.

One common challenge in training deep neural networks is selecting an appropriate learning rate. A fixed learning rate can either cause slow convergence or prevent the model from reaching an optimal solution. To address this, we implement a Cosine Annealing Learning Rate Scheduler,

which gradually decreases the learning rate following a cosine function over the course of training. Mathematically, the learning rate at epoch  $t$  is defined as:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min}) \left( 1 + \cos \left( \frac{t}{T} \pi \right) \right)$$

where  $\eta_{\max}$  is the initial learning rate,  $\eta_{\min}$  is the minimum learning rate, and  $T$  is the total number of epochs. The key benefit of this approach is that it allows for larger learning rates in the early stages, helping the model explore the loss landscape, while gradually reducing the learning rate toward the end of training to refine the solution. This method reduces the likelihood of getting stuck in sharp local minima and promotes better generalization.

To further improve model generalization and reduce overfitting, we incorporate data augmentation. This technique synthetically expands the training dataset by applying transformations to input images and their corresponding segmentation masks. Specifically, we implement random horizontal flips with a probability of 50% to simulate left-right variations and random rotations of 90, 180, and 270 degrees to ensure invariance to different object orientations. Since semantic segmentation is a pixel-level task, it is crucial to apply identical transformations to both the images and their masks to maintain correct label correspondence. Augmenting the dataset in this way helps the model learn to recognize objects in various orientations and lighting conditions, improving its robustness when encountering unseen data.

To address the class imbalance problem in semantic segmentation, we implemented a balanced weighting scheme inspired by the 'balanced' mode but with additional modifications for better performance. The weighting function first calculates the frequency of each class across the training dataset by counting class presence per image rather than per pixel, making it more robust to variations in object sizes. The initial weights are computed using the formula  $w = n_{samples} / (n_{classes} * class\_count)$ , where  $n_{samples}$  is the total number of images. To create more separation between common and rare classes, we applied power scaling (power = 1.5) to the weights, effectively amplifying the differences. To prevent numerical instability and extreme weight values, we bounded the weights within a reasonable range [0.05, 20.0] using min-max thresholding. Finally, the weights were normalized so their sum equals the number of classes, ensuring a balanced contribution to the loss function. This approach gives higher importance to underrepresented classes while maintaining numerical stability during training.

By implementing these improvements, we aim to build a more robust and accurate FCN model capable of performing high-quality semantic segmentation across diverse real-world images.

### 3.3 Experimentation

To enhance model performance, we have experimented using ResNet34 as a pre-trained encoder to benefit from strong feature extraction. It allowed the model to encode with well-learned representations, leading to faster convergence and improved generalization. Since ResNet34 employs deep hierarchical feature extraction with residual connections, it helps capture different levels of spatial information effectively. We removed ResNet34's fully connected and average pooling layers compared with the baseline model. ResNet34 minimizes the need for excessive retraining on the imbalanced dataset, preventing the model from becoming overly biased toward dominant classes.

To enhance the performance of our FCN for semantic segmentation, we implemented three key architectural modifications: activation function changes, convolution filter size adjustments, and network depth expansion. We used LeakyReLU, varied convolution filter sizes, using larger filters (5x5, 7x7) in the early layers to capture broader spatial features and smaller filters (3x3, 1x1) in deeper layers for fine-grained detail extraction, and deepened the network by adding an extra convolutional layer (Conv6) in the encoder and a corresponding transposed convolutional layer (Deconv6) in the decoder, enabling the model to learn more complex hierarchical features.

We have also performed U-Net. It incorporates two convolutional layers with 3x3 kernels, Batch Normalization and LeakyReLU activations. MaxPooling layers with a kernel size of 2x2 and stride of 2 are used between encoder blocks to reduce spatial dimensions. Dropout (0.2) in the decoder and batch normalization after each convolutional layer to stabilize training. To handle class imbalance, it

has prevented the model from overfitting to majority classes while enhancing segmentation accuracy across all categories.

Layer	Input Channels	Output Channels	Kernel Size	Padding/Stride	Activation Function
Encoder	ResNet34 Output	ResNet34 Output	ResNet34	ResNet34	ReLU
ConvTranspose1	512	256	3x3	Padding=1 / Stride=2	ReLU
BatchNorm1	512	256	N/A	N/A	BatchNorm
ConvTranspose2	256	128	3x3	Padding=1 / Stride=2	ReLU
BatchNorm2	256	128	N/A	N/A	BatchNorm
ConvTranspose3	128	64	3x3	Padding=1 / Stride=2	ReLU
BatchNorm3	128	64	N/A	N/A	BatchNorm
ConvTranspose4	64	32	3x3	Padding=1 / Stride=2	ReLU
BatchNorm4	64	32	N/A	N/A	BatchNorm
ConvTranspose5	32	32	3x3	Padding=1 / Stride=2	ReLU
BatchNorm5	32	32	N/A	N/A	BatchNorm
Final Classifier	32	n_class	1x1	None	None

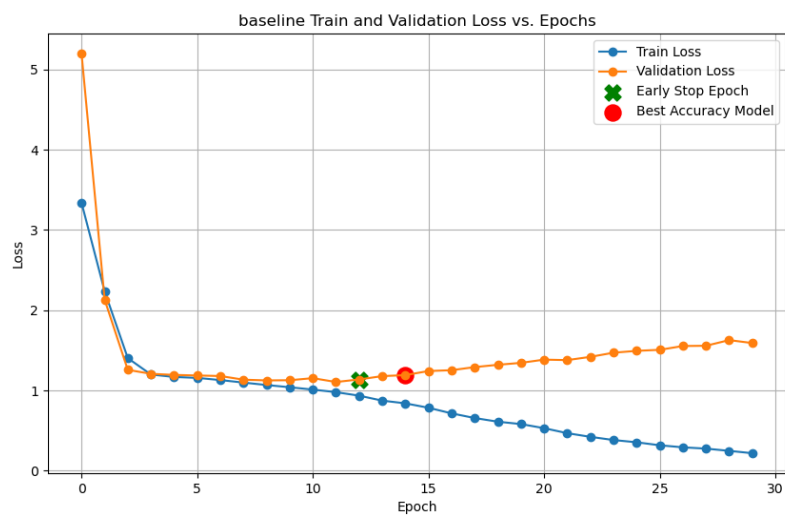
Table 1: Network Architecture for Transfer Learning

Layer	Input Channels	Output Channels	Kernel Size	Padding/Stride	Activation Function
Conv1 (Encoder Block 1)	3	32	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
Conv2 (Encoder Block 1)	32	32	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
MaxPool1	32	32	2x2	Stride=2	None
Conv3 (Encoder Block 2)	32	64	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
Conv4 (Encoder Block 2)	64	64	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
MaxPool2	64	64	2x2	Stride=2	None
Conv5 (Encoder Block 3)	64	128	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
Conv6 (Encoder Block 3)	128	128	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
MaxPool3	128	128	2x2	Stride=2	None
Conv7 (Encoder Block 4)	128	256	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
Conv8 (Encoder Block 4)	256	256	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
MaxPool4	256	256	2x2	Stride=2	None
Conv9 (Encoder Block 5)	256	512	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
Conv10 (Encoder Block 5)	512	512	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
UpConv1	512	256	2x2	Stride=2	None
Concat1	256+256	512	N/A	N/A	None
Conv11 (Decoder Block 1)	512	256	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
Conv12 (Decoder Block 1)	256	256	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
UpConv2	256	128	2x2	Stride=2	None
Concat2	128+128	256	N/A	N/A	None
Conv13 (Decoder Block 2)	256	128	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
Conv14 (Decoder Block 2)	128	128	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
UpConv3	128	64	2x2	Stride=2	None
Concat3	64+64	128	N/A	N/A	None
Conv15 (Decoder Block 3)	128	64	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
Conv16 (Decoder Block 3)	64	64	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
UpConv4	64	32	2x2	Stride=2	None
Concat4	32+32	64	N/A	N/A	None
Conv17 (Decoder Block 4)	64	32	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
Conv18 (Decoder Block 4)	32	32	3x3	Padding=1 / Stride=1	LeakyReLU (0.01)
Final Classifier	32	n_class	1x1	None	None

Table 2: Network Architecture for U-Net

## 4 Results

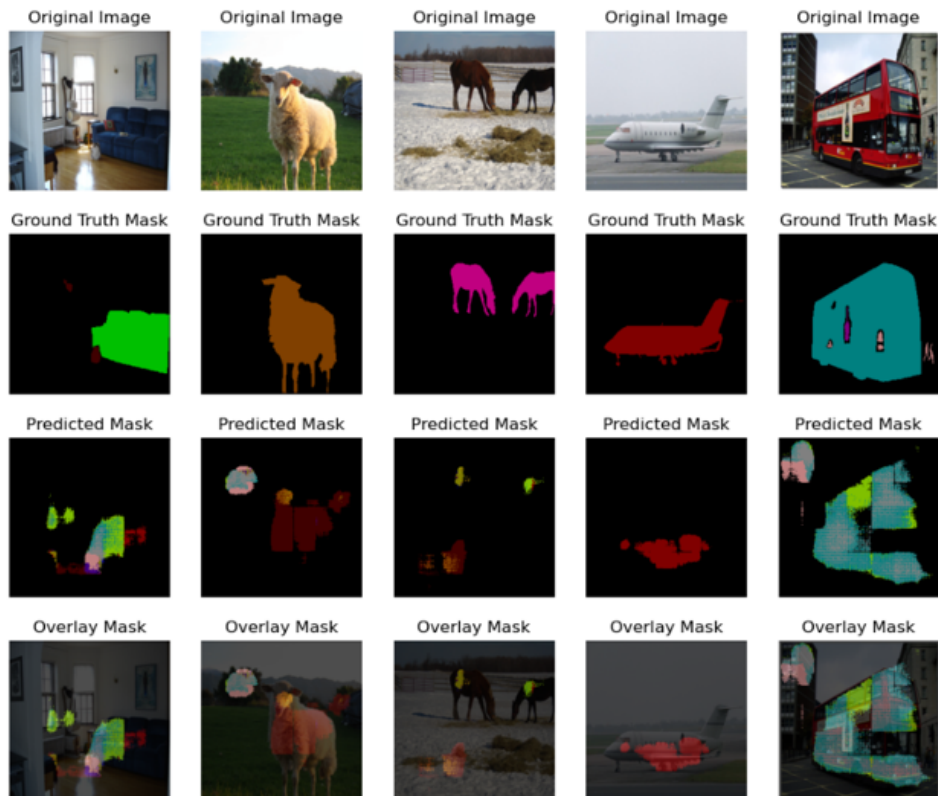
### 4.1 Q3: Baseline



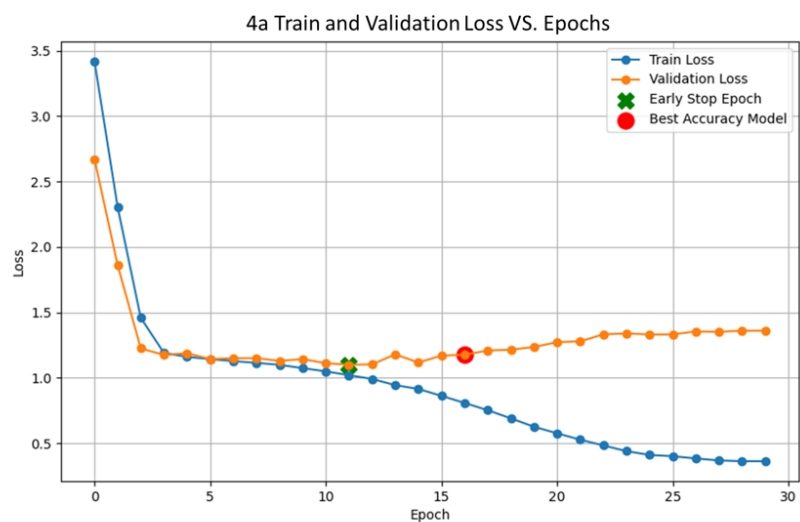
Validation set pixel accuracy and average IoU

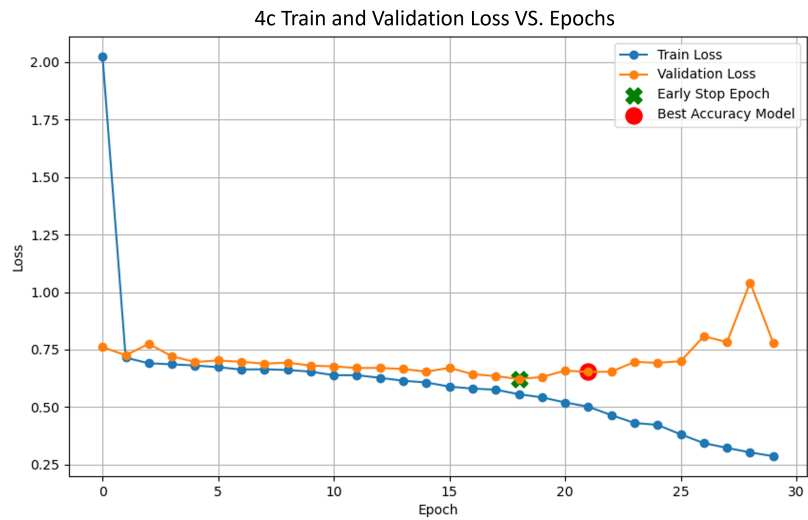
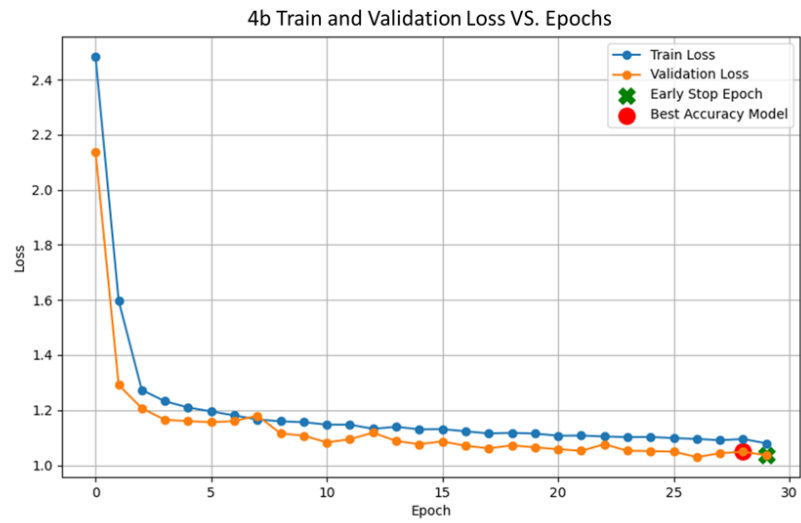
	Baseline
iou	0.065
pixel_acc	0.74

Generated by Baseline:



## 4.2 Q4: Improving the Baseline Model

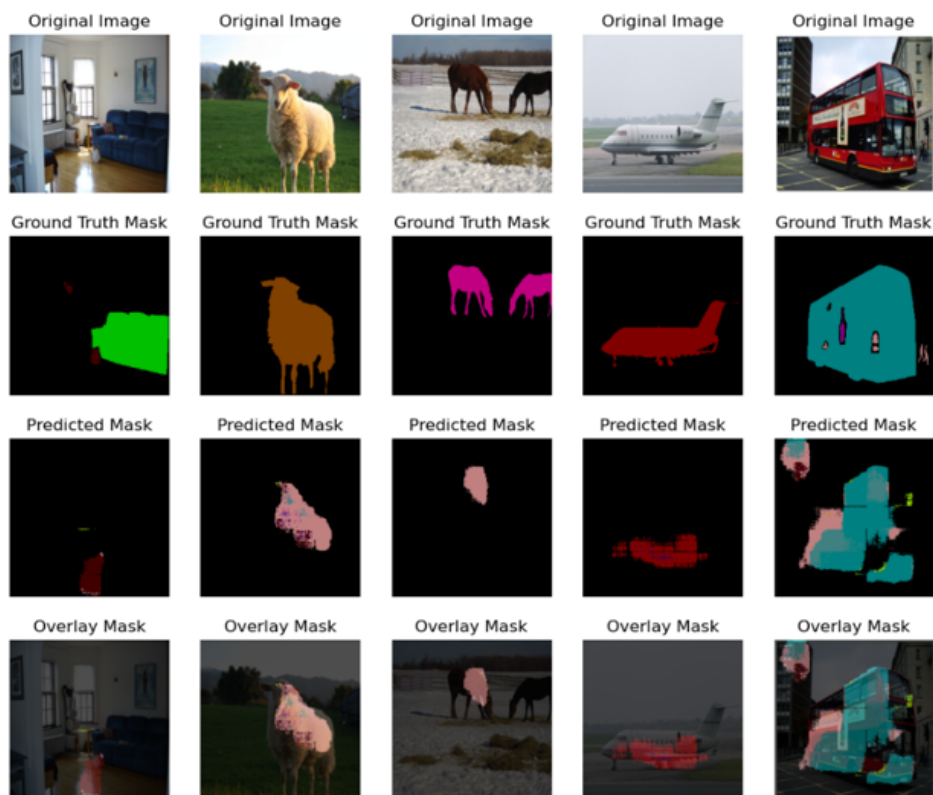




Validation set pixel accuracy and average IoU

	4.a	4.b	4.c
<b>iou</b>	0.067	0.071	0.083
<b>pixel_acc</b>	0.73	0.76	0.74

Generated by 4A:

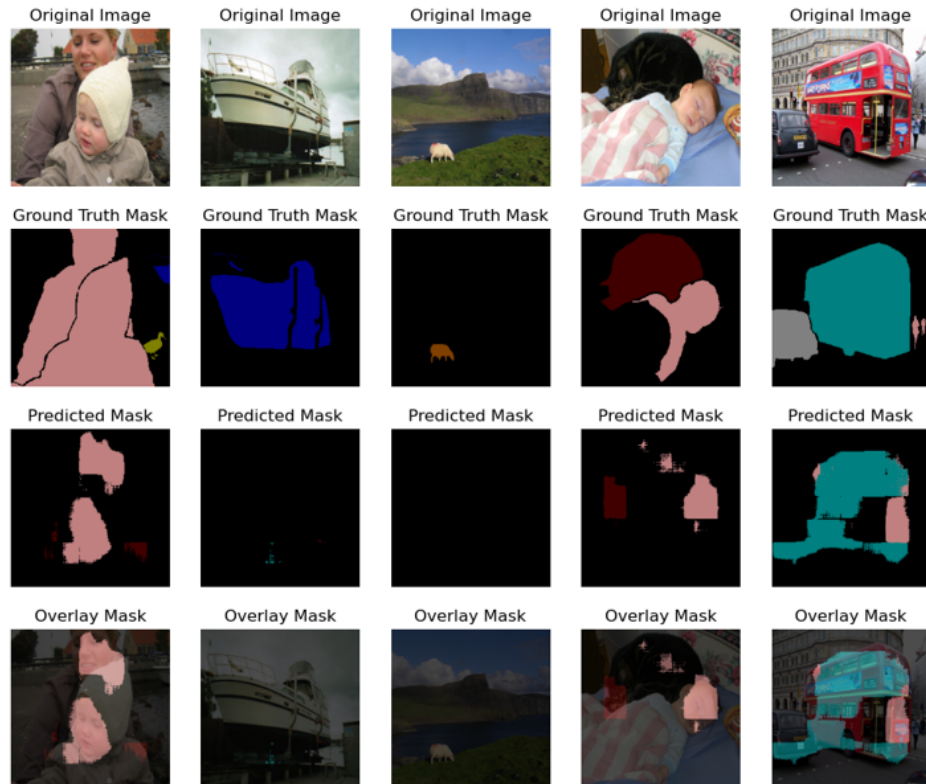


Generated by 4B:





Generated by 4C:



### 4.3 Q5: Experimentation and your solution

## 5 Discussion

### 5.1 Q3: Baseline

In this baseline FCN-style encoder-decoder design, the training and validation curves demonstrate that the model picks up broad distinctions among classes quickly (sharp initial drop in both losses and early improvements in IoU/pixel accuracy), but then overfits starting around epoch5: the training loss continues to decline while validation loss eventually plateaus and rises. This architecture has the advantage of straightforward multi-scale learning from its layered structure, yet the lack of skip connections makes fine detail recovery difficult, and training from scratch on limited data can impede further improvement. Moreover, cross-entropy loss alone may not adequately handle class imbalance. Consequently, IoU improvement stalled at about 0.0629, indicating that future work—such as adding skip connections, using a pretrained encoder, or adopting more specialized data augmentation and loss functions—would likely deliver better generalization and higher segmentation accuracy.

### 5.2 Q4: Improving the Baseline Model

#### 5.2.1 4a: Cosine Annealing Learning Rate Scheduler

The cosine annealing learning rate scheduler is a technique that adjusts the learning rate over training epochs to improve convergence.

By using cosine annealing, the model gradually reduces its learning rate as training progresses, allowing for better convergence. The results show that the implementation of cosine annealing led to a validation IoU of 0.067 and a pixel accuracy of 0.73. Compared to the baseline model, this suggests that the model benefits from smoother learning rate decay, reducing overfitting and

improving generalization. However, one drawback observed was that the validation loss started increasing after a certain number of epochs, indicating potential overfitting or suboptimal scheduling.

Kaijie Zhang mentioned that this learning rate adjustment method is similar to an optimization technique in stochastic gradient descent. It starts with a relatively high initial learning rate, and when the improvement per iteration falls below a certain threshold, the learning rate is halved at each iteration to better approach the global minimum. However, this algorithm relies on certain assumptions, such as strong convexity or constraints on the initial learning rate. In the context of image optimization, based on an understanding of the loss function, cosine annealing learning is expected to yield better results.

For the observed data, the improvement brought by annealing compared to the baseline is minimal. We suspect that we may still not have reached the optimal solution through learning rate reduction. We take into account the lack of warm-up or restarting mechanisms. Some implementations of cosine annealing perform best when combined with a warm-up phase, where the learning rate is gradually increased at the beginning to stabilize training. Additionally, cosine annealing with restarts (e.g., SGDR) periodically resets the learning rate, helping the model escape poor local minima. Without restarts, the model may become trapped in suboptimal solutions.

### **5.2.2 4b: Data Augmentation**

Data augmentation broadens the range of visual appearances encountered during training by applying transformations such as horizontal flips and rotations (90, 180, and 270 degrees). This approach effectively increases the size and diversity of the training dataset, often leading to improved generalization. In our experiments, the validation IoU rose to 0.071 and the pixel accuracy rose to 0.76, indicating that the model became better at detecting object boundaries and distinguishing between classes. However, the slight decline in pixel accuracy suggests that these transformations might introduce distortions that the model finds challenging to reconcile. Moreover, while the training loss showed a more stable decline, the validation loss fluctuated more, suggesting increased variance in validation performance. This could be due to the augmented samples presenting greater variability than the unaugmented validation data. Nonetheless, these trade-offs can be mitigated by carefully selecting or tuning augmentation strategies (e.g., limiting the range of rotations) and ensuring that the transformed images remain representative of real-world scenarios. Overall, data augmentation remains a powerful technique for enhancing robustness and reducing overfitting, though its effectiveness hinges on aligning transformations with the true data distribution.

### **5.2.3 4c: Weighted Loss Criterion**

The balanced weighting approach addresses class imbalance by giving higher importance to minority classes while preventing numerical instability through bounding and normalization. By calculating class frequency based on image presence, rather than pixel counts, the method avoids over-penalizing large objects and instead captures the true distribution of class occurrences. Power scaling amplifies the separation between rare and common classes, ensuring that underrepresented categories receive sufficient attention during training. However, these advantages come with potential drawbacks. Overemphasizing rare classes can lead to slower convergence and degrade performance on more common classes if not carefully tuned. Additionally, hyperparameters such as the power exponent and bounding thresholds require experimentation to strike the right balance. Despite these concerns, the balanced weighting scheme generally proves effective in improving minority class recognition in semantic segmentation tasks.

Our balanced weighting scheme, which calculates class frequency based on image presence rather than pixel counts and applies power scaling and clipping, provides a robust way to emphasize underrepresented classes while avoiding extreme weights. This approach helped address class imbalance and yielded a validation IoU of 0.083 and pixel accuracy of 0.74, indicating some improvement in distinguishing rare classes. Coupled with cosine annealing, the model achieved a smoother learning rate decay that improved overall convergence, though we noted potential overfitting or suboptimal scheduling in later epochs, likely due to the lack of warm-up or restarts. Data augmentation increased robustness by adding diverse variations, improving IoU but slightly lowering pixel accuracy—possibly because certain transformations introduced noise that the model struggled to learn.

## 5.3 Q5: Experimentation and your solution

### 5.3.1 5a: Change the Baseline

To improve the performance of our FCN for semantic segmentation, we experimented with three key architectural modifications: activation function changes, convolution filter size variations, and a deeper network architecture.

First, we replaced ReLU with LeakyReLU, addressing the “dying ReLU” problem where neurons can become inactive and hinder learning. LeakyReLU allows a small negative slope, ensuring a continuous gradient flow even for negative values. This modification aimed to enhance gradient propagation and improve feature learning, particularly in deeper layers.

Second, we adjusted convolution filter sizes, using larger filters ( $5 \times 5$ ,  $7 \times 7$ ) in the early layers to capture broader spatial features and smaller filters ( $3 \times 3$ ,  $1 \times 1$ ) in deeper layers for fine-grained detail extraction. The larger filters in the early layers allowed the model to recognize broader shapes, while the smaller filters in later layers helped it focus on finer edges, leading to more refined segmentation masks.

In addition to these changes, we deepened the network by adding an extra convolutional layer (Conv6) in the encoder and a corresponding transposed convolutional layer (Deconv6) in the decoder. This allowed the model to learn more complex hierarchical features crucial for distinguishing fine details in segmentation tasks. To ensure output consistency with the ground truth labels, we applied bilinear interpolation to resize the final segmentation mask to  $224 \times 224$ , preventing shape mismatches and improving IoU computation. These three modifications collectively aimed to improve the average IoU.

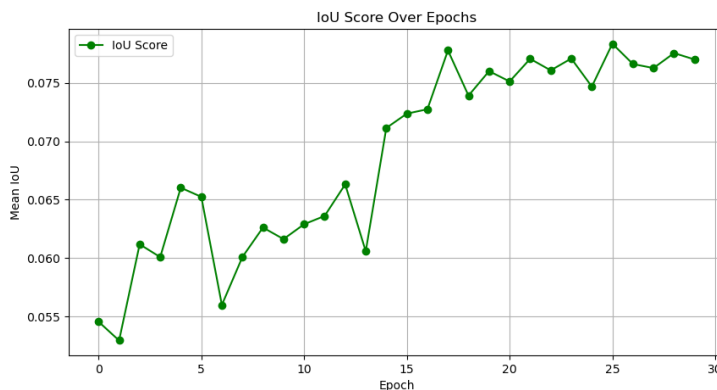


Figure 1: Average IoU over training for LeakyReLU, adjusting convolution filter sizes, and incrementing layers.

### 5.3.2 5b: Transfer Learning

To improve on the baseline model, we have implemented transfer learning with resnet34 as our encoder and keep the decoder the same as the baseline model to better compare the effect of transfer learning. Using transfer learning is able to train faster and better generalizability. Choosing resnet34 as the encoder because it can perform strong feature extraction and increase computational efficiency, which results in test IoU: 0.06011019339380057 and test pixel accuracy: 0.741622964973035. The result is slightly lower than the baseline model. It might be because of over-fitting with the resnet pre-trained model complexity. Another reason may be the mismatch between the encoder and decoder. The resnet34 may reduce spatial resolution, which result the decoder could not reconstruct the details. Keeping the same decoder, in fact, is due to better compare with the baseline model. Further fine-tuning implementation can be performed to increase both the IoU and accuracy.

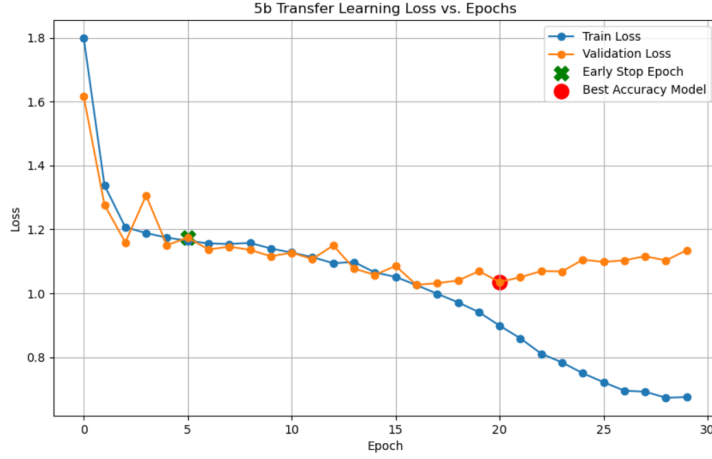


Figure 2: transfer learning result

### 5.3.3 5c: U-Net

The U-Net model implemented here is a FCN designed for semantic segmentation tasks. The architecture is a symmetric encoder-decoder structure, where the encoder captures spatial hierarchies, and the decoder reconstructs the segmentation map with refined spatial details. Each encoder block contains two convolutional layers with 3x3 kernels, followed by Batch Normalization and LeakyReLU activations. MaxPooling layers with a kernel size of 2x2 and stride of 2 are employed between encoder blocks to reduce spatial dimensions. The decoder utilizes transpose convolutions for upsampling and incorporates skip connections by concatenating feature maps from corresponding encoder layers. To improve feature refinement, additional convolutional layers (with 3x3 kernels) are applied after each decoder block. Dropout with a rate of 0.2 is included in the decoder for regularization, reducing overfitting. The final classification layer uses a 1x1 convolution to map the features to the desired number of classes. Cropping is performed to ensure alignment between encoder and decoder feature maps during concatenation. The U-Net model achieves the best average IoU with the best epoch almost reaching 0.90 [2].

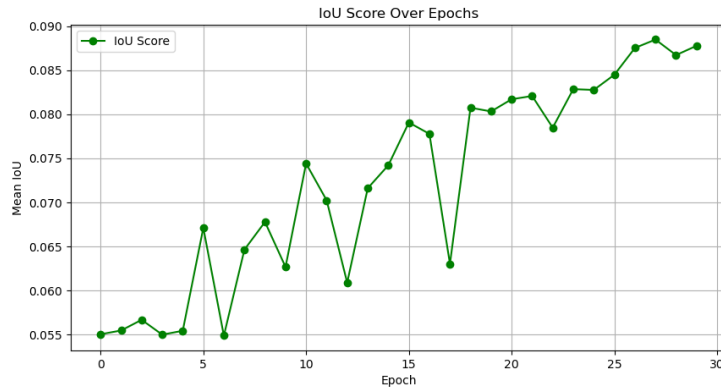


Figure 3: Average IoU over training for U-Net.

## References

- [1] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *arXiv preprint arXiv:1608.03983*, 2016. Available at: <https://doi.org/10.48550/arXiv.1608.03983>.

- [2] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv preprint arXiv:1505.04597*, 2015. Conditionally accepted at MICCAI 2015. Available at: <https://doi.org/10.48550/arXiv.1505.04597>.

## AI Appendix

For the baseline model, we used a project-based AI tool to analyze the entire starter code. We used the description from the pdf and the comment in the starter code as the prompt, and it generated a lot of useful code for the baseline model. However, it also made some mistakes. For example, it couldn't figure out the correct number of channels in the basic\_fcn file, so that we manually debugged it. Later on, as we got more familiar with the code, we didn't need the AI tool that much, and we could change the code based on the specific instructions.

```
cse151b251b-wi25-pa2-cse151b_251b_fcns / basic_fcn.py ↑ Top
Code Blame 51 lines (45 loc) · 2.52 KB
4 class FCN(nn.Module):
13     self.bnd2 = nn.BatchNorm2d(64)
14     self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=2, padding=1, dilati
15     self.bnd3 = nn.BatchNorm2d(128)
16     self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1, dilati
17     self.bnd4 = nn.BatchNorm2d(256)
18     self.conv5 = nn.Conv2d(256, 512, kernel_size=3, stride=2, padding=1, dilati
19     self.bnd5 = nn.BatchNorm2d(512)
20     self.relu = nn.ReLU(inplace=True)
21     self.deconv1 = nn.ConvTranspose2d(512, 256, kernel_size=3, stride=2, paddi
22     self.bn1 = nn.BatchNorm2d(256)
23     self.deconv2 = nn.ConvTranspose2d(512, 128, kernel_size=3, stride=2, paddi
24     self.bn2 = nn.BatchNorm2d(128)
25     self.deconv3 = nn.ConvTranspose2d(256, 64, kernel_size=3, stride=2, paddin
26     self.bn3 = nn.BatchNorm2d(64)
27     self.deconv4 = nn.ConvTranspose2d(128, 32, kernel_size=3, stride=2, paddin
28     self.bn4 = nn.BatchNorm2d(32)
29     self.deconv5 = nn.ConvTranspose2d(64, 16, kernel_size=3, stride=2, padding
30     self.bn5 = nn.BatchNorm2d(16)
31     self.classifier = nn.Conv2d(16, self.n_class, kernel_size=1)
32
33     #TODO Complete the forward pass
34     def forward(self, x):
35         x1 = self.bnd1(self.relu(self.conv1(x)))
36         # Complete the forward function for the rest of the encoder
37         x2 = self.bnd2(self.relu(self.conv2(x1)))
38         x3 = self.bnd3(self.relu(self.conv3(x2)))
39         x4 = self.bnd4(self.relu(self.conv4(x3)))
40         x5 = self.bnd5(self.relu(self.conv5(x4)))
41
42         y1 = self.bn1(self.relu(self.deconv1(x5)))
43         # Complete the forward function for the rest of the decoder
44         y2 = self.bn2(self.relu(self.deconv2(y1)))
45         y3 = self.bn3(self.relu(self.deconv3(y2)))
46         y4 = self.bn4(self.relu(self.deconv4(y3)))
47         y5 = self.bn5(self.relu(self.deconv5(y4)))
```

cse151b251b-wi25-pa2-cse151b\_251b\_fcns / train.py

↑ Top

Code

Blame

282 lines (209 loc) · 8.96 KB



Raw



```
39
40 train_dataset = voc.VOC('train', transform=input_transform, target_transform=target
41 val_dataset = voc.VOC('val', transform=input_transform, target_transform=target_tr
42 # test_dataset = voc.VOC('test', transform=input_transform, target_transform=target
43
44 # Split validation dataset into validation and test sets
45 val_size = len(val_dataset) // 2
46 test_size = len(val_dataset) - val_size
47 val_dataset, test_dataset = torch.utils.data.random_split(val_dataset, [val_size,
48
49 train_loader = DataLoader(dataset=train_dataset, batch_size= 16, shuffle=True, num_
50 val_loader = DataLoader(dataset=val_dataset, batch_size= 16, shuffle=False, num_w
51 test_loader = DataLoader(dataset=test_dataset, batch_size= 16, shuffle=False, num_
52
53 epochs = 30
54
55 n_class = 21
56
57 fcn_model = FCN(n_class=n_class)
58 fcn_model.apply(init_weights)
59
60 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
61
62 optimizer = torch.optim.Adam(fcn_model.parameters(), lr=0.001)
63 criterion = torch.nn.CrossEntropyLoss()
64
65 fcn_model = fcn_model.to(device)
66
67
```

cse151b251b-wi25-pa2-cse151b\_251b\_fcns / train.py

↑ Top

Code

Blame

282 lines (209 loc) · 8.96 KB



Raw



```
69 def train():
70
71
72
73     best_iou_score = 0.0
74     patience = 5 # Number of epochs to wait for improvement
75     patience_counter = 0
76     best_model_path = 'best_model.pth'
77
78     for epoch in range(epochs):
79         ts = time.time()
80         for iter, (inputs, labels) in enumerate(train_loader):
81             # AI prompt: the same text from the pdf.
82             optimizer.zero_grad()
83
84             # Transfer inputs and labels to device
85             inputs = inputs.to(device)
86             labels = labels.to(device)
87
88             # Forward pass
89             outputs = fcn_model(inputs)
90
91             # Calculate loss
92             loss = criterion(outputs, labels)
93
94             # Backward pass and optimize
95             loss.backward()
96             optimizer.step()
97
98             if iter % 20 == 0:
99                 print("epoch{}, iter{}, loss: {}".format(epoch, iter, loss.item()))
100
101             print("Finish epoch {}, time elapsed {}".format(epoch, time.time() - ts))
102
103             current_miou_score = val(epoch)
104
105             # Save best model and check for early stopping
106             if current_miou_score > best_iou_score:
107                 best_iou_score = current_miou_score
```

cse151b251b-wi25-pa2-cse151b\_251b\_fcns / train.py

↑ Top

Code

Blame

282 lines (209 loc) · 8.96 KB



Raw



```

69     def train():
107
108         if iter % 20 == 0:
109             print("epoch{}, iter{}, loss: {}".format(epoch, iter, loss.item()))
110
111         print("Finish epoch {}, time elapsed {}".format(epoch, time.time() - ts))
112
113         current_miou_score = val(epoch)
114
115         # Save best model and check for early stopping
116         if current_miou_score > best_iou_score:
117             best_iou_score = current_miou_score
118             patience_counter = 0
119             # Save the best model
120             torch.save({
121                 'epoch': epoch,
122                 'model_state_dict': fcn_model.state_dict(),
123                 'optimizer_state_dict': optimizer.state_dict(),
124                 'best_iou_score': best_iou_score,
125             }, best_model_path)
126             print(f"Saved best model with IoU: {best_iou_score:.4f}")
127         else:
128             patience_counter += 1
129             print(f"IoU didn't improve for {patience_counter} epochs")
130
131         # Early stopping
132         if patience_counter >= patience:
133             print(f"Early stopping triggered after {epoch + 1} epochs")
134             print(f"Best IoU score: {best_iou_score:.4f}")
135             break
136
137         # Load best model for testing
138         checkpoint = torch.load(best_model_path)
139         fcn_model.load_state_dict(checkpoint['model_state_dict'])
140         print(f"Loaded best model from epoch {checkpoint['epoch']} with IoU: {checkpoi
141

```

cse151b251b-wi25-pa2-cse151b\_251b\_fcns / train.py

↑ Top

Code

Blame

282 lines (209 loc) · 8.96 KB



Raw



```

143     def val(epoch):
144         """
145         Calculate mean IoU score and mean loss for this validation epoch.
146         """
147
148         fcn_model.eval() # Put in eval mode (disables batchnorm/dropout) !
149
150         losses = []
151         mean_iou_scores = []
152         accuracy = []
153
154         with torch.no_grad(): # we don't need to calculate the gradient in the validation
155             # AI prompt: the same text from the pdf.
156             for iter, (inputs, labels) in enumerate(val_loader):
157                 inputs = inputs.to(device)
158                 labels = labels.to(device)
159
160                 outputs = fcn_model(inputs)
161                 loss = criterion(outputs, labels)
162
163                 losses.append(loss.item())
164
165                 # Calculate IoU and accuracy
166                 pred = outputs.max(1)[1].cpu().numpy()
167                 gt = labels.cpu().numpy()
168
169                 mean_iou_scores.append(util.cal_iou_numpy(pred, gt))
170                 accuracy.append(util.cal_pixel_acc_numpy(pred, gt))
171
172             print(f"Loss at epoch: {epoch} is {np.mean(losses)}")
173             print(f"IoU at epoch: {epoch} is {np.mean(mean_iou_scores)}")
174             print(f"Pixel acc at epoch: {epoch} is {np.mean(accuracy)}")
175

```



cse151b251b-wi25-pa2-cse151b\_251b\_fcns / train.py

↑ Top

Code

Blame

282 lines (209 loc) · 8.96 KB



Raw



```
195 def modelTest():
205     - Switch model back to training mode.
206
207     Returns:
208         None. Outputs average test metrics to the console.
209     """
210
211     fcn_model.eval() # Put in eval mode (disables batchnorm/dropout) !
212
213     # AI prompt: the same text from the pdf.
214     losses = []
215     mean_iou_scores = []
216     accuracy = []
217
218     with torch.no_grad(): # we don't need to calculate the gradient in the valida
219         for iter, (inputs, labels) in enumerate(test_loader):
220             inputs = inputs.to(device)
221             labels = labels.to(device)
222
223             outputs = fcn_model(inputs)
224             loss = criterion(outputs, labels)
225
226             losses.append(loss.item())
227
228             # Calculate IoU and accuracy
229             pred = outputs.max(1)[1].cpu().numpy()
230             gt = labels.cpu().numpy()
231
232             mean_iou_scores.append(util.cal_iou_numpy(pred, gt))
233             accuracy.append(util.cal_pixel_acc_numpy(pred, gt))
234
235     print(f"Test Loss: {np.mean(losses)}")
236     print(f"Test IoU: {np.mean(mean_iou_scores)}")
237     print(f"Test Pixel Accuracy: {np.mean(accuracy)}")
238
239     fcn_model.train() #TURNING THE TRAIN MODE BACK ON TO ENABLE BATCHNORM/DROPOUT
```

cse151b251b-wi25-pa2-cse151b\_251b\_fcns / util.py

↑ To

Code

Blame

66 lines (50 loc) · 1.88 KB



Raw



```
4 def iou(pred, target, n_classes = 21):
5     """
6     Calculate the Intersection over Union (IoU) for predictions.
7
8     Args:
9         pred (tensor): Predicted output from the model.
10        target (tensor): Ground truth labels.
11        n_classes (int, optional): Number of classes. Default is 21.
12
13    Returns:
14        float: Mean IoU across all classes.
15    """
16    # Convert boundary pixels (255) to background (0)
17    target = torch.where(target == 255, torch.zeros_like(target), target)
18
19    # Flatten the tensors
20    pred = pred.view(-1)
21    target = target.view(-1)
22
23    # Initialize arrays to store intersection and union for each class
24    intersection = torch.zeros(n_classes)
25    union = torch.zeros(n_classes)
26
27    # Calculate IoU for each class
28    for cls in range(n_classes):
29        pred_cls = pred == cls
30        target_cls = target == cls
31
32        intersection[cls] = (pred_cls & target_cls).sum().float()
```

```
cse151b251b-wi25-pa2-cse151b_251b_fcns / util.py ↑ Top
Code Blame 66 lines (50 loc) · 1.88 KB
34
35     # Avoid division by zero
36     valid = union > 0
37     iou_per_class = torch.where(valid, intersection / union, torch.zeros_like(union))
38
39     # Calculate mean IoU over valid classes
40     mean_iou = iou_per_class[valid].mean().item()
41
42     return mean_iou
43
44 def pixel_acc(pred, target):
45     """
46     Calculate pixel-wise accuracy between predictions and targets.
47
48     Args:
49         pred (tensor): Predicted output from the model.
50         target (tensor): Ground truth labels.
51
52     Returns:
53         float: Pixel-wise accuracy.
54     """
55     # Convert boundary pixels (255) to background (0)
56     target = torch.where(target == 255, torch.zeros_like(target), target)
57
58     # Flatten the tensors
59     pred = pred.view(-1)
60     target = target.view(-1)
61
62     # Calculate accuracy
63     correct = (pred == target).sum().float()
64     total = target.numel()
65
66     return (correct / total).item()
```

We also used GPT to generate parts of the report, and then we carefully changed the output based on our results.

# Class Imbalance Segmentation Approach

in one paragraph for my report, discuss the benefits (and drawbacks if any) of the approaches/architectures you used and some discussion of why you think you got the results you got:

To address the class imbalance problem in semantic segmentation, we implemented a balanced weighting scheme inspired by the 'balanced' mode but with additional modifications for better performance. The weighting function first calculates the frequency of each class across the training dataset by counting class presence per image rather than per pixel, making it more robust to variations in object sizes. The initial weights are computed using the formula  $w = \text{nsamples} / (\text{nclasses} * \text{classcount})$ , where  $\text{n\_samples}$  is the total number of images. To create more separation between common and rare classes, we applied power scaling ( $\text{power} = 1.5$ ) to the weights, effectively amplifying the differences. To prevent numerical instability and extreme weight values, we bounded the weights within a reasonable range  $[0.05, 20.0]$  using min-max thresholding. Finally, the weights were normalized so their sum equals the number of classes, ensuring a balanced contribution to the loss function. This approach gives higher importance to underrepresented classes while maintaining numerical stability during training. We have achieved: Validation performance:  
IOU: 0.090  
Pixel Acc: 0.717

We used together with previous methods, include why you think you got the results you got:

## 5.2.1 4a: Cosine Annealing Learning Rate Scheduler

The cosine annealing learning rate scheduler is a technique that adjusts the learning rate over training epochs to improve convergence.

By using cosine annealing, the model gradually reduces its learning rate as training progresses, al-

lowing for better convergence. The results show that the implementation of cosine annealing led to a validation IoU of 0.062 and a pixel accuracy of 0.75. Compared to the baseline model, this suggests that the model benefits from smoother learning rate decay, reducing overfitting and improving generalization. However, one drawback observed was that the validation loss started increasing after a certain number of epochs, indicating potential overfitting or suboptimal scheduling.

Kaijie Zhang mentioned that this learning rate adjustment method is similar to an optimization technique in stochastic gradient descent. It starts with a relatively high initial learning rate, and when the

6

improvement per iteration falls below a certain threshold, the learning rate is halved at each iteration to better approach the global minimum. However, this algorithm relies on certain assumptions, such as strong convexity or constraints on the initial learning rate. In the context of image optimization, based on an understanding of the loss function, cosine annealing learning is expected to yield better results.

For the observed data, the improvement brought by annealing compared to the baseline is minimal.

We suspect that we may still not have reached the optimal solution through learning rate reduction.

We take into account the lack of warm-up or restarting mechanisms. Some implementations of cosine annealing perform best when combined with a warm-up phase, where the learning rate is gradually increased at the beginning to stabilize training. Additionally, cosine annealing with restarts (e.g., SGDR) periodically resets the learning rate, helping the model escape poor local minima.

Without restarts, the model may become trapped in suboptimal solutions.

#### 5.2.2 4b: Data Augmentation

Data augmentation aims to enhance model robustness by introducing variations in the training data.

We applied horizontal flipping and rotation by 90, 180, or

270 degrees, ensuring that the corresponding segmentation masks underwent the same transformations. Data augmentation increases the effective dataset size, improving the models generalization to unseen data. The results indicate that data augmentation led to a validation IoU of 0.078 and a pixel accuracy of 0.734. The improvement in IoU suggests that the model became better at distinguishing between object boundaries and categories. However, the pixel accuracy was slightly lower than in 4a, which may indicate that some transformations introduced noise or distortions that the model struggled to learn effectively. Additionally, from the loss curves, the training loss showed a more stable decline, whereas the validation loss fluctuated more, indicating increased variance in the validation set.

2/2

### **Balancing methodologies**

I'm noting the balance between the defined weighting scheme and additional methods like cosine annealing and data augmentation. The results show variations in IoU and pixel accuracy, highlighting the benefits and drawbacks of each approach.

### **Assessing performance dynamics**

I'm mapping out why the balanced weighting and cosine annealing methods performed variably, including the potential overfitting and data augmentation impacts on IoU and pixel accuracy.

Our balanced weighting scheme, which calculates class frequency based on image presence rather than pixel counts and applies power scaling and clipping, provides a robust way to emphasize underrepresented classes while avoiding extreme weights. This approach helped address class imbalance and yielded a validation IoU of 0.090 and pixel accuracy of 0.717, indicating some improvement in distinguishing rare classes. Coupled with cosine annealing, the model achieved a smoother learning rate decay that improved overall convergence (IoU = 0.062, pixel accuracy = 0.750), though we noted potential overfitting or suboptimal scheduling in later epochs, likely due to the lack of warm-up or restarts. Data augmentation (IoU = 0.078, pixel accuracy = 0.734) increased robustness by adding diverse variations, improving IoU but slightly lowering pixel accuracy—possibly because certain transformations introduced noise that the model struggled to learn. Overall, while each technique

demonstrated benefits—balanced class weighting for tackling imbalance, cosine annealing for refined convergence, and data augmentation for generalization—further tuning (e.g., learning rate restarts, warm-up phases, or more targeted augmentations) may be needed to fully realize their potential and stabilize validation performance.

also emphasize more on the balanced weights itself.  
discuss the advantage and drawbacks

### **Balancing class emphasis**

I'm weighing the balanced weights approach for addressing class imbalance. It enhances minority classes' visibility, adapts to object size changes, and maintains consistent training with normalization. However, overemphasis on rare classes and slower convergence are potential drawbacks.

### **Enhancing minority class focus**

I'm working through the balanced weighting approach to class imbalance by emphasizing minority classes in the loss function, while power scaling and weight bounding ensure numerical stability.

The balanced weighting approach addresses class imbalance by giving higher importance to minority classes while preventing numerical instability through bounding and normalization. By calculating class frequency based on image presence, rather than pixel counts, the method avoids over-penalizing large objects and instead captures the true distribution of class occurrences. Power scaling amplifies the separation between rare and common classes, ensuring that underrepresented categories receive sufficient attention during training. However, these advantages come with potential drawbacks. Overemphasizing rare classes can lead to slower convergence and degrade performance on more common classes if not carefully tuned. Additionally, hyperparameters such as the power exponent and bounding thresholds require experimentation to strike the right balance. Despite these concerns, the balanced weighting scheme generally proves effective in improving minority class recognition in semantic segmentation tasks.

perform the same analysis and extend it a bit:

#### **.2.2 4b: Data Augmentation**

Data augmentation aims to enhance model robustness by introducing variations in the training data.

We applied horizontal flipping and rotation by 90, 180, or 270 degrees, ensuring that the corresponding segmentation masks underwent the same transformations. Data augmentation increases

the effective dataset size, improving the models generalization to unseen data.

The results indicate that data augmentation led to a validation IoU of 0.078 and a pixel accuracy of 0.734. The improvement in IoU suggests that the model became better at distinguishing between object boundaries and categories. However, the pixel accuracy was slightly lower than in 4a, which may indicate that some transformations introduced noise or distortions that the model struggled to learn effectively. Additionally, from the loss curves, the training loss showed a more stable decline, whereas the validation loss fluctuated more, indicating increased variance in the validation set.

2/2

### Analyzing augmentation outcomes

I'm pulling together insights on data augmentation, focusing on flipping and rotating. The validation shows a slight IoU improvement but a drop in pixel accuracy, indicating better boundary recognition but slight pixel classification degradation.

Data augmentation broadens the range of visual appearances encountered during training by applying transformations such as horizontal flips and rotations (90°, 180°, and 270°). This approach effectively increases the size and diversity of the training dataset, which often leads to improved generalization. In our experiments, the validation IoU rose to 0.078—a sign that the model became better at detecting object boundaries and distinguishing between classes—suggesting that augmentation can help the model learn more invariant features. However, the slight decline in pixel accuracy (to 0.734 compared to 0.75 in 4a) indicates that these transformations might introduce distortions that the model finds challenging to reconcile. Moreover, while the training loss showed a more stable decline, the validation loss fluctuated more, suggesting increased variance in validation performance. This could be due to the augmented samples presenting greater variability than the unaugmented validation data. Nonetheless, these trade-offs can often be mitigated by carefully selecting or tuning augmentation strategies (e.g., limiting the range of rotations) and ensuring that the transformed images remain representative of real-world scenarios. Overall, data augmentation remains a powerful technique for enhancing robustness and reducing overfitting, but its effectiveness depends on aligning transformations with the true data distribution.

Reasoned for a couple of seconds

latex

`\subsection{4b: Data Augmentation}`

Data augmentation broadens the range of visual appearances encountered during training by

applying transformations such as horizontal flips and rotations ( $90^\circ$ ,  $180^\circ$ , and  $270^\circ$ ).

This approach effectively increases the size and diversity of the training dataset, often leading to improved generalization. In our experiments, the validation IoU rose to 0.078,

indicating that the model became better at detecting object boundaries and distinguishing

between classes. However, the slight decline in pixel accuracy (to 0.734 compared to 0.75

in 4a) suggests that these transformations might introduce distortions that the model finds

challenging to reconcile. Moreover, while the training loss showed a more stable decline,

the validation loss fluctuated more, suggesting increased variance in validation performance.

This could be due to the augmented samples presenting greater variability than the unaugmented

validation data. Nonetheless, these trade-offs can be mitigated by carefully selecting or

tuning augmentation strategies (e.g., limiting the range of rotations) and ensuring that the

transformed images remain representative of real-world scenarios. Overall, data augmentation

remains a powerful technique for enhancing robustness and reducing overfitting, though its

effectiveness hinges on aligning transformations with the true data distribution.

write two sentences to describe our related work:

U-Net: Convolutional Networks for Biomedical Image Segmentation

Olaf Ronneberger, Philipp Fischer, Thomas Brox

There is large consent that successful training of deep networks requires many thousand annotated training



samples. In this paper, we present a network and training strategy that relies on the strong use of data augmentation to use the available annotated samples more efficiently. The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. We show that such a network can be trained end-to-end from very few images and outperforms the prior best method (a sliding-window convolutional network) on the ISBI challenge for segmentation of neuronal structures in electron microscopic stacks. Using the same network trained on transmitted light microscopy images (phase contrast and DIC) we won the ISBI cell tracking challenge 2015 in these categories by a large margin. Moreover, the network is fast. Segmentation of a 512x512 image takes less than a second on a recent GPU. The full implementation (based on Caffe) and the trained networks are available at this [http URL](http://url).

Reasoned for a couple of seconds

U-Net is a popular biomedical segmentation network that combines a contracting path for context capture with a symmetric expanding path for precise localization, allowing end-to-end training even on relatively small annotated datasets. By leveraging heavy data augmentation, U-Net effectively utilizes the available annotated samples, outperforming previous methods in tasks such as neuronal structure segmentation and cell tracking challenges.

do the same: SGDR: Stochastic Gradient Descent with Warm Restarts

Ilya Loshchilov, Frank Hutter

Restart techniques are common in gradient-free optimization to deal with multimodal functions. Partial warm restarts are also gaining popularity in gradient-based optimization to improve the rate of convergence in accelerated gradient schemes to deal with ill-conditioned functions. In this paper, we propose a simple warm restart technique for stochastic gradient descent to improve its anytime performance when training deep neural networks. We empirically study its performance on the CIFAR-10 and CIFAR-100 datasets, where we demonstrate new state-of-the-art results at 3.14% and 16.21%, respectively. We also demonstrate its advantages on a dataset of EEG recordings

and on a downsampled version of the ImageNet dataset.  
Our source code is available at this [https URL](#)

SGDR (Stochastic Gradient Descent with Warm Restarts) introduces a cyclic learning rate schedule that periodically resets the learning rate to escape poor local minima and improve convergence. This method has demonstrated state-of-the-art performance on datasets like CIFAR-10 and CIFAR-100, highlighting its effectiveness in training deep neural networks efficiently.