Optimizing FCNs for Semantic Segmentation: Enhancing Performance with Augmentation and Optimization

Christopher Rebollar-Ramirez
crebollarramirez@ucsd.edu

Chi Zhang
chz011@ucsd.edu

Abstract

This project focuses on enhancing semantic segmentation using Fully Convolutional Networks (FCNs) in PyTorch, targeting challenges like class imbalance. It begins by implementing a baseline FCN model with Xavier weight initialization and batch normalization to ensure stable training. To improve performance, the project incorporates techniques such as cosine annealing learning rates, data augmentation (random flips, rotations, and resized crops), and class weighting to address class imbalance. Additionally, advanced architectures like U-Net and transfer learning with pre-trained ResNet34 are explored. The performance is evaluated using Pixel Accuracy and Intersection over Union (IoU) metrics. The project's goal is to demonstrate the impact of architectural improvements and training strategies on segmentation accuracy, especially for underrepresented classes.

1 Introduction

Semantic segmentation is a vital computer vision task that assigns class labels to each pixel in an image, with applications in autonomous driving, medical imaging, and satellite analysis. This project explores implementing and enhancing a Fully Convolutional Network (FCN) using PyTorch while addressing challenges like class imbalance, which skews predictions toward dominant classes. Xavier weight initialization and batch normalization are employed to stabilize training by ensuring balanced weight distribution and accelerating convergence.

We assess performance using Pixel Accuracy and IoU metrics, preprocess data with necessary transformations, and implement a baseline FCN model as a foundation for improvements. Optimizations such as cosine annealing learning rates, weighted loss functions, and advanced architectures like U-Net and transfer learning are explored to enhance segmentation accuracy. Through systematic experimentation, this project aims to deepen understanding of semantic segmentation and the impact of architectural and training choices.

2 Related Work

Semantic segmentation, a critical task in computer vision, has seen significant advancements with the introduction of deep convolutional neural networks (CNNs). Early approaches relied on traditional machine learning techniques like support vector machines (SVMs) and random forests, which struggled with complex image data.

The Fully Convolutional Network (FCN) by Long et al. (2015) marked a breakthrough by replacing fully connected layers with convolutional layers, allowing pixel-wise predictions and effectively addressing spatial information preservation. Building on this, architectures like U-Net

(Ronneberger et al., 2015) enhanced segmentation by incorporating encoder-decoder structures with skip connections, improving the handling of fine-grained details, while DeepLab (Chen et al., 2017) introduced atrous convolutions for multi-scale context.

Additionally, dealing with class imbalance has been crucial, and methods such as weighted loss functions and focal loss (Lin et al., 2017) have been developed to tackle the issue of rare classes. Transfer learning, using pre-trained models like ResNet (He et al., 2016) and VGG (Simonyan & Zisserman, 2015), has also proven effective in improving performance by leveraging large-scale datasets like ImageNet.

In this work, we build upon these foundational architectures and techniques by experimenting with different CNN-based models, data augmentation strategies, and loss functions to enhance segmentation performance on the PASCAL VOC-2012 dataset, while also exploring the impact of transfer learning and model modifications such as U-Net.

3 Methods

This section outlines the methodologies employed in developing and improving the semantic segmentation model. We begin with a baseline Fully Convolutional Network (FCN), detailing its encoder-decoder structure, activation functions, loss function, and optimization strategy. We then describe incremental improvements aimed at enhancing performance, including a cosine annealing learning rate scheduler, data augmentation techniques, and class weighting to address class imbalance. Further, we present three experimental architectures—a refined FCN model, a ResNet-based model leveraging transfer learning, and a U-Net architecture with skip connections. These methods aim to optimize segmentation accuracy, improve generalization, and handle challenges such as class imbalance and overfitting.

3.1 Baseline

The baseline architecture for the semantic segmentation task follows a Fully Convolutional Network (FCN) structure, consisting of an encoder-decoder setup. The encoder is composed of a series of convolutional layers with progressively increasing depth (32, 64, 128, 256, 512 channels), each followed by batch normalization and ReLU activations to extract features. The decoder uses transposed convolutions to progressively upsample the feature maps. The final output is passed through a 1x1 convolutional layer for pixel-wise classification. ReLU activations are applied throughout the network, and a softmax activation is used on the output layer for multi-class classification. For the loss function, we employ Cross-Entropy Loss, optionally weighted to address class imbalance. Weights are initialized using Xavier initialization for the convolutional layers, and the output layer generates raw class scores. The model is optimized with the SGD optimizer (with momentum), and early stopping is applied to mitigate overfitting during training. The learning rate is set to 0.01, with momentum at 0.9.

3.2 Improvements over Baseline

To improve the baseline model, we incorporated a `CosineAnnealingLR` learning rate scheduler (`torch.optim.lr_scheduler.CosineAnnealingLR`). This scheduler gradually decreases the learning rate following a cosine curve over the course of training, starting from the initial learning rate and approaching a minimum value, `eta_min`. The scheduler is configured with `T_max=epochs*2`, meaning it completes a full cycle of learning rate decay over twice the number of epochs, ensuring a gradual and controlled reduction.

This approach helps the model converge more effectively by reducing the learning rate during later stages of training, allowing for finer adjustments to the weights and reducing the risk of overshooting the optimal solution. This improvement aims to enhance the model's performance and stability during training, especially when combined with the early stopping mechanism to prevent overfitting.

Afterward, we enhanced the model's robustness by incorporating image transformations into the training dataset. Specifically, we applied a series of augmentations using `torchvision.transforms` to introduce variability and improve generalization. The transformations include:

- `RandomHorizontalFlip` with a probability of 0.5 to randomly flip images horizontally,
- `RandomRotation` with a degree range of 5 to randomly rotate images,
- `RandomResizedCrop` that resizes images to a fixed size of 224×224 while randomly cropping and scaling the image within a specified range (0.9 to 1.0).

These augmentations help to increase the diversity of the training data and prevent the model from overfitting, ultimately improving its generalization performance.

To further improve the model's performance, we implemented class weighting to address class imbalance in the training dataset. Specifically, we calculated class weights using a technique called median frequency balancing. This method assigns higher weights to less frequent classes and lower weights to more frequent ones, helping the model focus more on the underrepresented classes.

The class weights were computed using the following approach:

- We first counted the frequency of each class across all pixels in the training dataset, excluding any pixels with a label of 255 (which typically represents "ignore" or background).
- We then calculated the frequency of each class relative to the total number of pixels in the dataset.
- To compute the class weights, we used the formula:

$$\text{weight}_c = 1 - \frac{\text{class\_freq}_c}{\text{freq\_sum}}$$

  where class_freq_c is the frequency of class c and freq_sum is the total frequency of all classes.
- The resulting weights were used during training to adjust the loss function, helping the model better handle class imbalance.

These class weights aim to mitigate the effect of class imbalance and improve the model's ability to accurately predict less frequent classes.

3.3 Experimentation

The models employ various regularization techniques, including batch normalization and data augmentation. Batch normalization is applied after convolutional and deconvolutional layers to stabilize training and accelerate convergence. While data augmentation is available, its specific transformations are not detailed. Dropout, though absent, could further improve generalization by reducing overfitting.

Weight initialization follows a structured approach, with convolutional layers utilizing Xavier uniform initialization and biases initialized from a normal distribution. This prevents gradient issues and ensures stable training dynamics. Optimization is performed using stochastic gradient descent (SGD) with a learning rate of 0.01 and momentum of 0.9. A cosine annealing learning rate scheduler is employed to gradually adjust the learning rate over epochs, enhancing stability and convergence.

Class imbalance is addressed using median frequency balancing, computing class weights based on relative frequencies. These weights can be incorporated into the loss function to mitigate bias toward dominant classes, though this feature is currently disabled. Training efficiency is further improved through early stopping, which halts training if validation loss does not improve for five

consecutive epochs. Model performance is evaluated using mean intersection over union (IoU) and pixel accuracy. Additionally, the ResNet-based model leverages a pre-trained ResNet34 encoder, benefiting from transfer learning.

Future improvements include enabling class weighting in the loss function, integrating dropout layers, and experimenting with alternative optimizers such as AdamW to further enhance convergence and generalization.

FCN2 Architecture

Table 1: FCN2 Architecture Details

Layer	Configuration / Dimensions	Activation / Notes
Conv1	`Conv2d`: $3 \rightarrow 64$, Kernel=3, Stride=2, Padding=1, Dilation=1	LeakyReLU
BN1	`BatchNorm2d`: 64	—
Conv2	`Conv2d`: $64 \rightarrow 256$, Kernel=3, Stride=2, Padding=1, Dilation=1	LeakyReLU
BN2	`BatchNorm2d`: 256	—
Conv3	`Conv2d`: $256 \rightarrow 1024$, Kernel=3, Stride=2, Padding=1, Dilation=1	LeakyReLU
BN3	`BatchNorm2d`: 1024	—
Deconv1	`ConvTranspose2d`: $1024 \rightarrow 256$, Kernel=3, Stride=2, Padding=1, Dilation=1, Output Padding=1	LeakyReLU
BN4	`BatchNorm2d`: 256	—
Deconv2	`ConvTranspose2d`: $256 \rightarrow 64$, Kernel=3, Stride=2, Padding=1, Dilation=1, Output Padding=1	LeakyReLU
BN5	`BatchNorm2d`: 64	—
Deconv3	`ConvTranspose2d`: $64 \rightarrow n\_class$, Kernel=3, Stride=2, Padding=1, Dilation=1, Output Padding=1	—

ResNet-based Architecture

Table 2: ResNet-based Architecture Details

Layer	Configuration / Dimensions	Activation / Notes
Encoder	Pretrained ResNet34 truncated before FC and AvgPool; produces 512 feature maps of size 7×7	—
Deconv1	`ConvTranspose2d`: $512 \rightarrow 256$, Kernel=3, Stride=2, Padding=1, Output Padding=1	ReLU
BN1	`BatchNorm2d`: 256	—
Deconv2	`ConvTranspose2d`: $256 \rightarrow 128$, Kernel=3, Stride=2, Padding=1, Output Padding=1	ReLU
BN2	`BatchNorm2d`: 128	—
Deconv3	`ConvTranspose2d`: $128 \rightarrow 64$, Kernel=3, Stride=2, Padding=1, Output Padding=1	ReLU
BN3	`BatchNorm2d`: 64	—
Deconv4	`ConvTranspose2d`: $64 \rightarrow 32$, Kernel=3, Stride=2, Padding=1, Output Padding=1	ReLU
BN4	`BatchNorm2d`: 32	—
Deconv5	`ConvTranspose2d`: $32 \rightarrow n\_class$, Kernel=3, Stride=2, Padding=1, Output Padding=1	—

U-Net Architecture

Table 3: U-Net Architecture Details

Layer	Configuration / Dimensions	Activation / Notes
Down1	`DoubleConv`: $in\_channels \rightarrow 64$, using two 3×3 convs (Padding=1)	ReLU; Output size: 224×224
Pool1	`MaxPool2d`: 2×2	Reduces to 112×112
Down2	`DoubleConv`: $64 \rightarrow 128$	ReLU; Output size: 112×112
Pool2	`MaxPool2d`: 2×2	Reduces to 56×56
Down3	`DoubleConv`: $128 \rightarrow 256$	ReLU; Output size: 56×56
Pool3	`MaxPool2d`: 2×2	Reduces to 28×28
Down4	`DoubleConv`: $256 \rightarrow 512$	ReLU; Output size: 28×28
Pool4	`MaxPool2d`: 2×2	Reduces to 14×14
Bottleneck	`DoubleConv`: $512 \rightarrow 1024$	ReLU; Size: 14×14
Up4	`ConvTranspose2d`: Upsample from 14×14 to 28×28	—
UpConv4	`DoubleConv`: Merge (Skip connection) yields $1024 \rightarrow 512$	ReLU
Up3	`ConvTranspose2d`: Upsample from 28×28 to 56×56	—
UpConv3	`DoubleConv`: Merge yields $512 \rightarrow 256$	ReLU
Up2	`ConvTranspose2d`: Upsample from 56×56 to 112×112	—
UpConv2	`DoubleConv`: Merge yields $256 \rightarrow 128$	ReLU
Up1	`ConvTranspose2d`: Upsample from 112×112 to 224×224	—
UpConv1	`DoubleConv`: Merge yields $128 \rightarrow 64$	ReLU
FinalConv	`Conv2d`: $64 \rightarrow out\_channels$, Kernel=1	—

4 Result

This section evaluates the baseline model and its enhancements using Pixel Accuracy and IoU. Results are presented in tables, comparing the effects of data augmentation (flips, rotations, resized crops) and class weighting, which mitigates class imbalance by emphasizing underrepresented classes. The comparisons highlight the effectiveness of these improvements in segmentation performance.

4.1 Performance Metrics

Model	Pixel Accuracy	Mean IoU
Baseline Model	72.8%	0.0553
(Cosine LR) Model	72.3%	0.0603
(Data Augmentation) Model	73.6%	0.072
(Class Weighting) Model	69.4%	0.084

Table 4: Validation Set Performance Metrics

4.2 Training and Validation Loss Curves

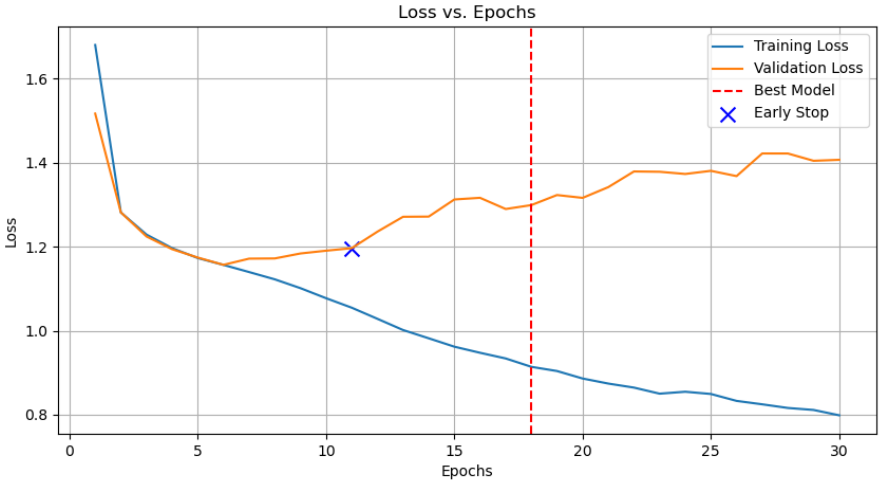

Figure 1: Baseline Model Loss Curve

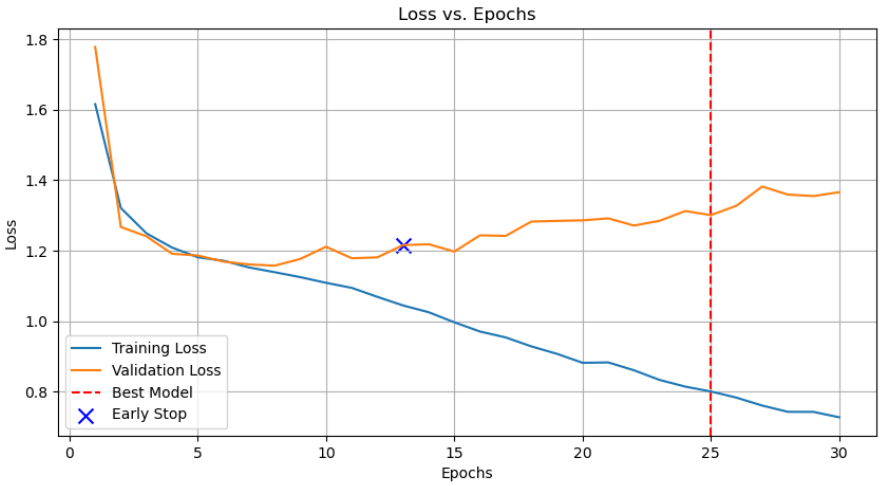

Figure 2: Cosine Annealing Learning Rate Model Loss Curve

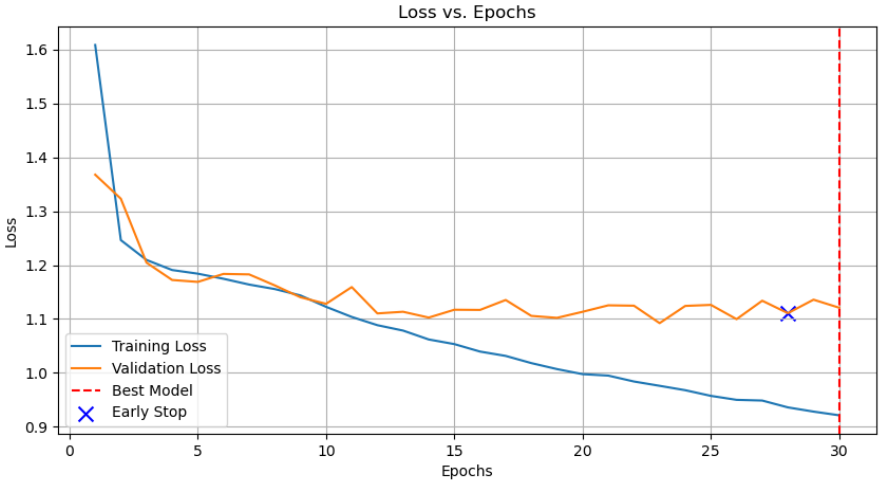

Figure 3: Image Transformation Model Loss Curve

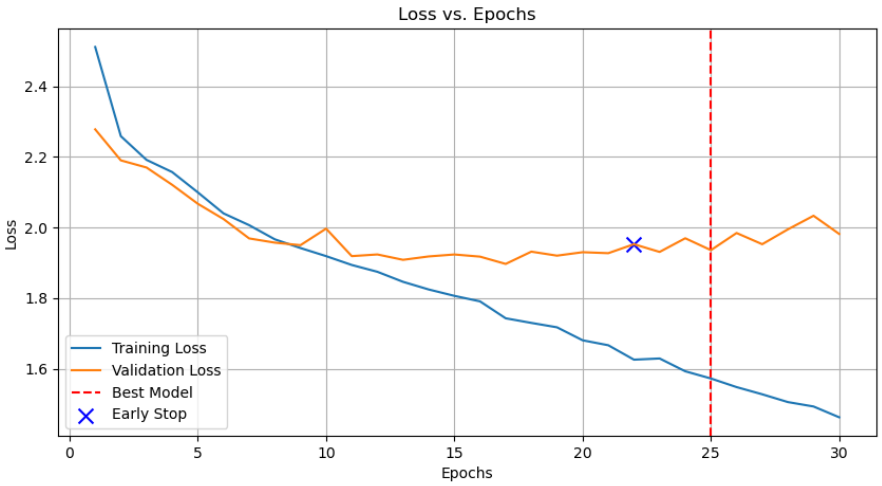

Figure 4: Class Weights Model Loss Curve

4.3 Segmented Output Visualizations

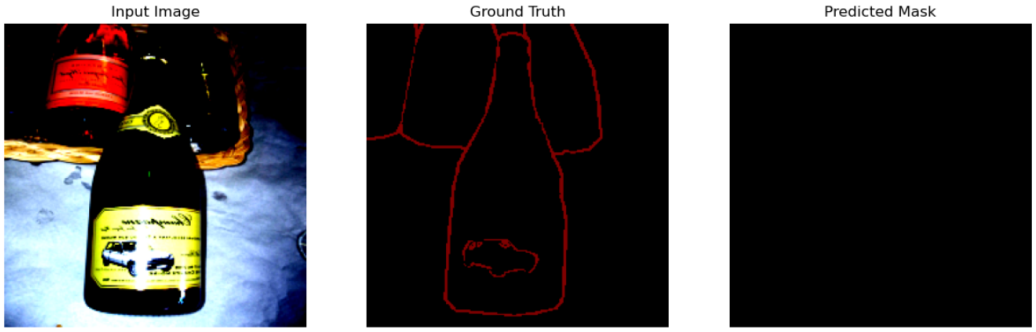

Figure 5: Baseline Model Visualization

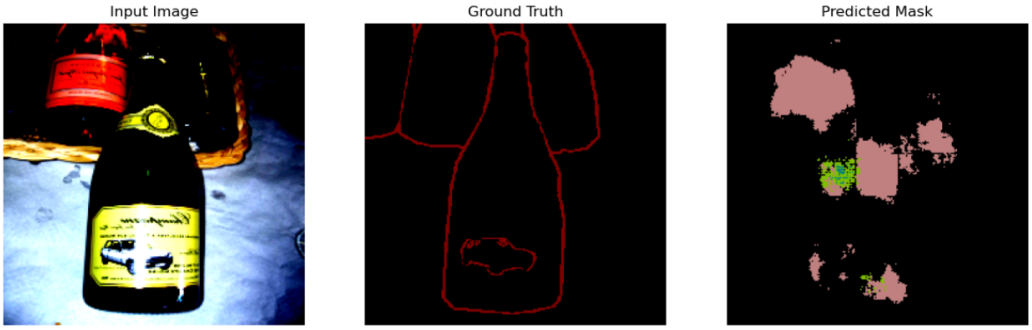

Figure 6: Cosine Annealing Learning Rate Model Visualization

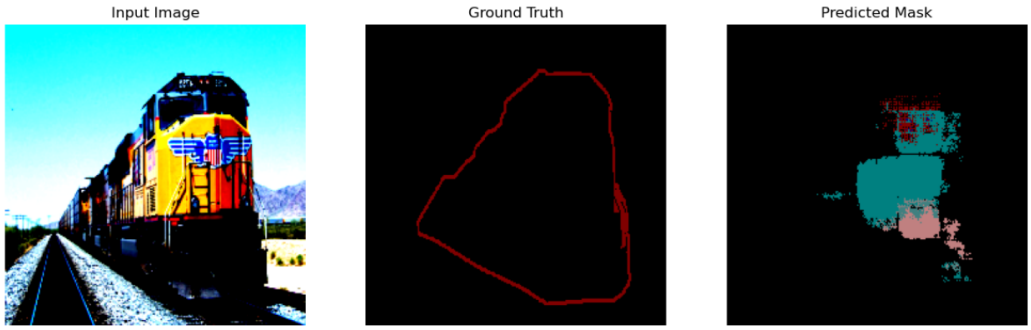

Figure 7: Image Transformation Model Visualization

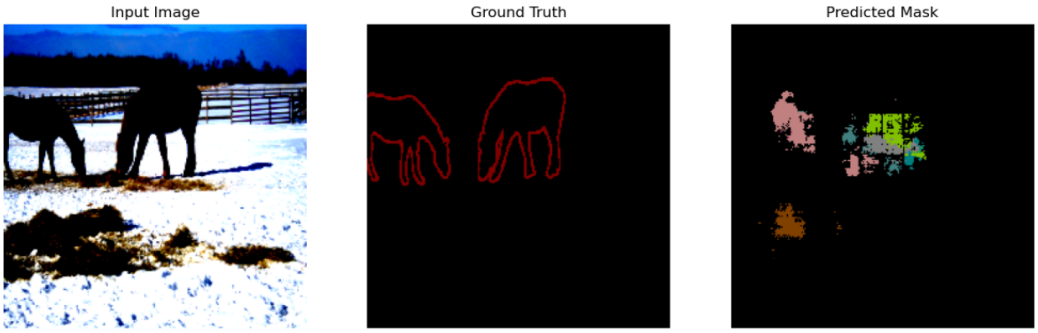

Figure 8: Class Weights Model Visualization

5 Discussion

In this section, we analyze the performance of various models and techniques implemented throughout the experiment. We discuss the strengths and limitations of the baseline model, the impact of improvements such as learning rate scheduling, data augmentation, and class weighting, and the outcomes of more advanced experimental approaches, including custom architectures, transfer learning, and U-Net. The discussion focuses on key performance metrics, including Pixel Accuracy and IoU, and provides insights into the effectiveness of different strategies. Additionally, we interpret the results based on loss curves, visualizations, and comparisons with alternative implementations.

5.1 Baseline Model

The baseline model, based on a Fully Convolutional Network (FCN), demonstrated reasonable performance with Pixel Accuracy of 0.727 and IoU of 0.0553 at epoch 29. The encoder-decoder architecture with convolutional layers, batch normalization, and transposed convolutions allowed the model to learn the features and progressively upsample them to generate pixel-wise predictions. The loss function used, Cross-Entropy Loss, works well for multi-class segmentation tasks, and the SGD optimizer with momentum helped with convergence.

However, the performance was somewhat limited, especially in terms of the IoU (Intersection over Union), which was relatively low at 0.0553. This reflects that the model was not able to accurately segment different classes, which could be attributed to several factors:

- **Class imbalance**: The model might have been biased towards more frequent classes, leading to poor performance on rare classes.
- **Fixed learning rate**: The constant learning rate throughout the training may have hindered the model from fine-tuning the weights effectively towards the end of training.

Despite the performance being a good starting point, these limitations highlight areas for improvement, particularly in addressing the class imbalance issue and ensuring that the learning rate is appropriately adjusted during training.

5.2 Improving baseline model

Learning Rate Scheduler: Cosine Annealing

One of the primary improvements was the introduction of the `CosineAnnealingLR` learning rate scheduler. By adjusting the learning rate to follow a cosine curve, the learning rate is reduced more gradually as training progresses, which allows for finer updates towards the optimal solution. At epoch 20, the IoU slightly improved to 0.059 and Pixel Accuracy remained around 0.723, which was a small improvement over the baseline.

Benefits:

- **Gradual convergence**: A gradual decrease in the learning rate helped the model make finer updates during later training epochs.
- **Reduced overshooting**: The risk of overshooting the optimal weights was minimized by lowering the learning rate.

Drawbacks:

- **Incremental gains**: Although the performance slightly improved, the impact of the learning rate scheduler was limited, with only small improvements in IoU and Pixel Accuracy. This suggests that other factors, like class imbalance and data augmentation, might be playing a more significant role in improving the model's performance.

The modest improvements in performance suggest that while the learning rate scheduler provided some benefit, more substantial changes (like data augmentation or class weighting) might have a larger impact.

Data Augmentation

In the next iteration, we applied data augmentation to enhance the diversity of the training data. The transformations included `RandomHorizontalFlip`, `RandomRotation`, and `RandomResizedCrop`, which ensured the model was exposed to more variability in the images. This modification had a more noticeable effect on performance, with IoU increasing to 0.066 at epoch 26 and Pixel Accuracy rising to 0.724.

Benefits:

- **Improved generalization**: The model was able to generalize better to unseen data by training on more varied images, preventing overfitting.
- **Diversity in training data**: Data augmentation added more diversity to the training set, which is crucial for segmentation tasks that require fine-grained details.

Drawbacks:

- **Overhead**: Data augmentation increases the computational cost during training, as each image needs to be transformed on the fly.
- **Marginal improvements**: While the improvements were noticeable, they were still relatively small, indicating that more adjustments might be necessary for more significant improvements in performance.

Despite the modest improvement, data augmentation proved valuable by helping the model generalize better, especially considering the limitations of the baseline model.

Class Weighting for Imbalanced Dataset

To address the class imbalance problem, we introduced class weighting based on median frequency balancing. By assigning higher weights to less frequent classes, the model was encouraged to pay more attention to underrepresented classes. This had a more pronounced effect on the performance, with IoU reaching 0.083 and Pixel Accuracy improving to 0.6939 at epoch 26.

Benefits:

- **Better handling of imbalanced data**: By focusing more on underrepresented classes, the model's ability to segment rare classes improved, leading to a higher IoU.
- **Improved overall segmentation**: The model was less biased toward the dominant classes, resulting in better segmentation for less frequent classes.

Drawbacks:

- **Risk of overfitting to rare classes**: If the weights are not balanced correctly, the model might overfit to the rare classes, reducing its performance on more frequent ones.

The introduction of class weighting led to an improvement in performance, particularly in handling imbalanced classes, which was likely a major factor in the boost in IoU. However, the drop in Pixel Accuracy suggests that while rare classes were better predicted, the model might have slightly overfitted to them at the expense of more frequent classes.

5.3 Experimentation

Custom Architecture

A custom architecture with more advanced layer configurations could provide additional performance benefits. By adjusting the number of layers, kernel sizes, and receptive fields, the model may better capture spatial features and provide more detailed segmentation. However, without specific details or results from this approach, it is difficult to quantify the exact improvements or drawbacks, but typically, a well-designed custom architecture should outperform the baseline if it can capture more complex features.

Insights: The custom model would likely improve the model's performance further, but might also require more careful tuning of hyperparameters to avoid overfitting.

Transfer Learning

Using transfer learning with pre-trained models like ResNet34 could offer a significant performance boost, particularly in the encoder portion. Transfer learning would allow the model to start with weights that are already well-optimized for general feature extraction, improving the training speed and convergence. The pre-trained encoder would have learned important features from large-scale datasets like ImageNet, which could be transferred to the segmentation task.

Insights: Transfer learning would likely outperform the baseline and improved baseline models, as the encoder would already be better suited to feature extraction. However, careful adjustments would be necessary to adapt the pre-trained model to the segmentation task.

U-Net Architecture

The U-Net architecture, with its skip connections, could significantly improve segmentation performance by preserving spatial information during upsampling. The U-Net's encoder-decoder structure with concatenated skip connections enables better retention of fine-grained features from the input image. It is well-suited for segmentation tasks where pixel-level accuracy is essential.

Insights: U-Net's performance would likely outperform the baseline and improved baseline due to the added spatial information retention through skip connections. This would result in better segmentation of fine details, particularly in complex images.

6 Authors' Contributions and References

Both teammates collaborated on `util.py`, `getClassWeights()`, `visualize.ipynb`, `train.py`, and `voc.py`. Chi focused on experimenting with new architectures, while both contributed to running the model on the cloud using a GPU and co-authoring the report.
We also used ChatGPT for debugging our code and formatting this report.

References

[1] Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3431–3440). IEEE. doi:10.1109/CVPR.2015.7298965

[2] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi (Eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (pp. 234–241). Springer. doi:10.1007/978-3-319-24574-4_28

[3] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2017). Rethinking atrous convolution for semantic image segmentation. arXiv preprint arXiv:1706.05587.

[4] Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In Proceedings of the IEEE International Conference on Computer Vision (pp. 2980–2988). IEEE. doi:10.1109/ICCV.2017.324

[5] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770–778). IEEE. doi:10.1109/CVPR.2016.90

[6] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.