Deep Learning Techniques for Semantic Segmentation Tasks

Krishna Daamini Ellendula
Department of Electrical & Computer Engineering
University of California, San Diego
kellendula@ucsd.edu

Nishanth Rachakonda
Department of Computer Science
University of California, San Diego
nrachakonda@ucsd.edu

Pranav Khanna
Department of Computer Science
University of California, San Diego
pkhanna@ucsd.edu

Soumya Ganguly
Department of Mathematics
University of California, San Diego
s1gangul@ucsd.edu

Abstract

This project paper introduces the use of semantic segmentation, a deep learning algorithm that associates a label or category with every pixel in an image, for the classification of objects in images. The method is applied to the PASCAL VOC-2007 dataset, which contains images of everyday objects and is commonly used for image classification and object detection. In this project we experimented with different types of model architecture and techniques like Transfer Learning, the Unet architecture, a custom architecture as well as demonstrated the effect of different hyperameters on the efficiency on models. On the test set we achieved a best accuracy of 0.726 and an IOU of 0.0744.

1 Introduction

Deep learning models have proven to be effective in modeling complex real-world data and have found widespread use in a variety of applications. However, one of the most popular use of deep learning models is the classification of images or rather, the classification of objects in the images. For this purpose, we will use a method called *semantic segmentation*. Semantic segmentation is a deep learning algorithm that associates a label or category with every pixel in an image. It is used to recognize a collection of pixels that form distinct categories. It can separate the main object in the image from the background by classifying the pixels and thus currently have widespread applications in the field of automated driving (where we need the vehicle to identify vehicles, pedestrians, traffic signs, road features, etc. from images), medical imaging, industrial inspection, satellite imagery, robotic vision etc.

Our model is trained to classify the PASCAL VOC-2007, dataset, a huge data set with images of everyday objects. It is a dataset that is used for image classification and object detection. We use this dataset to apply semantic segmentation on it so that we can recognize objects from a number of visual object classes in realistic scenes, which are not pre-segmented. The twenty object classes present in this dataset are:

- *Person:* Person
- *Animal:* Bird, Cat, Cow, Dog, Horse, Sheep
- *Vehicle:* Aeroplane, Bicycle, Boat, Bus, Car, Motorbike, Train
- *Indoor:* Bottle, Chair, Dining table, Potted plant, Sofa, Tv/Monitor

We consider another extra object class as 'background', which makes total of 21 object classes in this dataset. We are also considering the class 'background' as a separate class from the object classes above. The data has been split into 50% for training/validation and 50% for testing. The distributions of images and objects by class are approximately equal across the training/validation and test sets. In total there are 9,963 images, containing 24,640 annotated objects.

The effectiveness of the experiments carried out in this paper is measured in terms of two metrics: i. Pixel Accuracy and ii. IoU (Intersection over Union). Pixel accuracy is defined as the fraction of correct prediction $= \frac{\text{total correct predictions}}{\text{total number of samples}}$. IoU is the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground truth. It is computed on a per-class basis, and then these are averaged over the classes. Mathematically, IoU $= \frac{TP}{TP+FP+FN}$, where TP, FP, and FN are the numbers of true positive, false positive, and false negative pixels, respectively, determined over the whole validation set.

As the best result among all the experiments conducted, we were able to achieve a test IoU of 0.0744 and an accuracy of 0.726 using Transfer learning architecture.

2 Related Work

In the lecture slides of CSE 251B by Cottrell [3], [2], the essential theory behind convolutional neural nets is discussed which provides us with the base structure of the networks used in this project. Xavier initialization method was proposed by Glorot and Bengio in [4]. Batch normalization is a technique proposed by Sergey Ioffe and Christian Szegedy in [5]. The theory and first appearance of Adam Optimizer can be found in [6]. Cosine learning rate scheduler was first introduced in [7] by Loshchilov and Hutter. The first introduction to transfer learning techniques were described by Stevo Bozinovski and Ante Fulgosi in 1976, details of which can be found in [1]. UNet is an architecture developed by Olaf Ronneberger et al [8], for Biomedical Image Segmentation in 2015 at the University of Freiburg, Germany.

3 Methods

3.1 Weight Initialization:

We are using *Xavier weight initialization* ([4]) in this project. Xavier weight initialization assigns layer's weights to values chosen from a random uniform distribution that's bounded between $\pm\sqrt{6}/(\sqrt{n_i + n_{i+1}})$, where n_i is the number of incoming network connections, or 'fan-in', to the layer, and n_{i+1} is the number of outgoing network connections from that layer, also known as the 'fan-out'. It maintains the variance of activations and back-propagated gradients all the way up or down the layers of a network, leading to a substantially quicker convergence (as it solves the vanishing gradient problem effectively) and higher accuracy.

3.2 Batch Normalization:

The batch normalization technique ([5]) makes the training of neural networks faster and more stable through the normalization of the weighted sum of the inputs coming from the previous layer, before entering the next layer. These weighted sums of input are normalized over a mini-batch thus giving this process the name batch normalization. Specifically, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

However, after this shift/scale of activation outputs by some randomly initialized parameters, the weights in the next layer are no longer optimal. SGD (Stochastic gradient descent) undoes this normalization if needed, for it to minimize the loss function. To undo this normalization, we get two trainable parameters to each layer, so the normalized output is multiplied by a "standard deviation" parameter (gamma) and subsequently a "mean" parameter (beta) is added. SGD does this denormalization by changing only these two weights for each activation, instead of losing the stability of the network by changing all the weights.

3.3 Adam Optimizer:

In order to converge faster, we used the Adam optimizer (for detailed theory, please see [6]), which inherits properties from AdaGrad and RmsProp. This technique computes adaptive learning rates for each parameter of the model using estimates of the first and last moment of the gradient. This method is implemented in Pytorch as torch.optim.Adam(). For non-convex optimization, Adam is preferred because i. it is straightforward to implement, ii. it is computationally efficient, iii. it has little memory requirements, iv. it is invariant to diagonal rescale of the gradients, v. it is well suited for problems that are large in terms of data and/or parameters, vi. it is appropriate for problems with very noisy/or sparse gradients, and vii. here hyper-parameters have intuitive interpretation and typically require little tuning.

3.4 Baseline:

In the baseline model, we are adopting mini-batch stochastic gradient descent with a batch size of 16, Xavier initialization for weights, and Adam gradient descent optimizer. The batches are taken from a shuffled training set. The encoder consists of 5 convolution layers each with 32, 64, 128, 256, & 512 output channels respectively, and a kernel size of 3, stride of 2, padding of 1, and dilation of 1. The output from each convolution layer is activated using the ReLU function and then batch normalized before passing it down to the next layer. The input to the encoder is a batch of 16 images each of size 3x224x224 and the output from the encoder is of size 16x512x7x7. To increase the resolution of the output, this is passed to the decoder which again consists of 5 deconvolution layers each with 512, 256, 128, 64, & 32 output channels. The final output from the decoder is of size 16x224x224. It is activated using the Softmax function which gives an output of 16x21x224x224, where 21 corresponds to the number of classes.

The CNN architecture of the 'baseline' model here can be visualized using the table below:

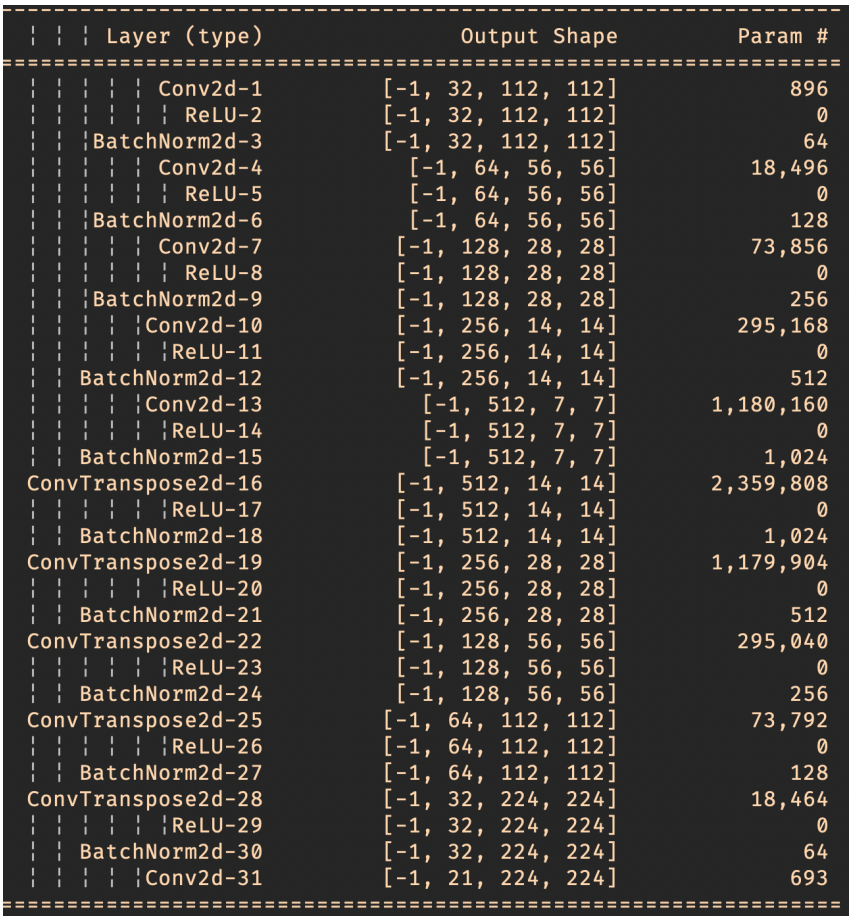


Figure 1: Architecture of Baseline model

3.5 Improvements over Baseline:

The CNN architecture of the 'Improved baseline' model in general, can be visualized using the table below:

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 112, 112]	896
ReLU-2	[-1, 32, 112, 112]	0
BatchNorm2d-3	[-1, 32, 112, 112]	64
Conv2d-4	[-1, 64, 56, 56]	18,496
ReLU-5	[-1, 64, 56, 56]	0
BatchNorm2d-6	[-1, 64, 56, 56]	128
Conv2d-7	[-1, 128, 28, 28]	73,856
ReLU-8	[-1, 128, 28, 28]	0
BatchNorm2d-9	[-1, 128, 28, 28]	256
Conv2d-10	[-1, 256, 14, 14]	295,168
ReLU-11	[-1, 256, 14, 14]	0
BatchNorm2d-12	[-1, 256, 14, 14]	512
Conv2d-13	[-1, 512, 7, 7]	1,180,160
ReLU-14	[-1, 512, 7, 7]	0
BatchNorm2d-15	[-1, 512, 7, 7]	1,024
ConvTranspose2d-16	[-1, 512, 14, 14]	2,359,808
ReLU-17	[-1, 512, 14, 14]	0
BatchNorm2d-18	[-1, 512, 14, 14]	1,024
ConvTranspose2d-19	[-1, 256, 28, 28]	1,179,904
ReLU-20	[-1, 256, 28, 28]	0
BatchNorm2d-21	[-1, 256, 28, 28]	512
ConvTranspose2d-22	[-1, 128, 56, 56]	295,040
ReLU-23	[-1, 128, 56, 56]	0
BatchNorm2d-24	[-1, 128, 56, 56]	256
ConvTranspose2d-25	[-1, 64, 112, 112]	73,792
ReLU-26	[-1, 64, 112, 112]	0
BatchNorm2d-27	[-1, 64, 112, 112]	128
ConvTranspose2d-28	[-1, 32, 224, 224]	18,464
ReLU-29	[-1, 32, 224, 224]	0
BatchNorm2d-30	[-1, 32, 224, 224]	64
Conv2d-31	[-1, 21, 224, 224]	693

Figure 2: Architecture of Improvement over Baseline model

3.5.1 Learning rate Scheduler :

In optimization via gradient descent, it is often a good idea to adjust your learning rate with the number of epochs during training. That is why we often use a learning rate scheduler. Cosine Annealing is a type of learning rate scheduler that resembles the nature of a cosine curve starting from zero. It starts with a large learning rate which relatively rapidly decreases and then increases again. The resetting of the learning rate is like a restart of the learning process. If the good weights are reused at the starting point of the restart, it is called a 'warm restart' otherwise, if a new set of small numbers are used at this stage, it is called a 'cold restart'. This learning rate scheduler was first introduced in [7] in order to perform SGDR (Stochastic Gradient Descent with Warm Restarts). The formula for the Cosine Learning rate Scheduler is given in the following :

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right), \qquad T_{cur} \neq (2k+1)T_{max}$$

$$\eta_{t+1} = \eta_t + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{1}{T_{max}}\pi\right)\right), \qquad T_{cur} = (2k+1)T_{max}$$

Here η_{min} and η_{max} are ranges for the learning rate and η_{max} is set to the initial learning rate at every epoch. We note that the initial learning rate at the first epoch is the learning rate set by us. T_{cur} is the number of epochs since the last restart in SGDR. If the learning rate is set solely by this scheduler, the learning rate at each step becomes:

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

In Pytorch, the main syntax for using this learning rate scheduler is : *torch.optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max, eta_min=0, last_epoch=-1, verbose=False)*.

3.5.2 Data Augmentation :

Most of the time, the images to be classified might not be well aligned with the train images. The images might be scaled, flipped, rotated, etc and we expect the model to classify them. For this purpose, we augmented the data set by applying transforms to the train images like Random Horizontal Flips, Random Vertical Flips, Random Affines, and Random Crops, etc.

There are in fact multiple advantages of doing data augmentation: i. it reduces the cost of collection of data and reduces the scarcity of data, ii. it reduces the cost of labeling data, iii. it improves the model prediction accuracy, iv. it reduces data overfitting, v. it creates variability and flexibility in data models, thus increasing the generalization ability of the data models, vi. it helps in resolving the class imbalance issue in the classification to some extent as well.

3.5.3 Class Imbalance:

The class Imbalance problem occurs when a few classes are more frequent than others. Due to fewer data on the minority classes, the model fails to learn as well as the majority class. This can be overcome by penalizing the loss from minority classes more than the loss from majority classes. We do this by assigning higher weights to the loss from minority classes. We weighed the loss as the inverse of the class frequency to which they belong.

$$weight_c = \frac{1/\text{no. of samples in class c}}{\sum_c (1/\text{no. of samples in class c})}.$$

3.5.4 Focal Loss function:

In the first part of these experiments, we used the Cross-Entropy loss function. In simple words, it is an improved version of cross-entropy loss. It is given as:

$$L = -\sum_{n,i} (1 - y_i^n)^\gamma t_i^n \log(y_i^n)$$

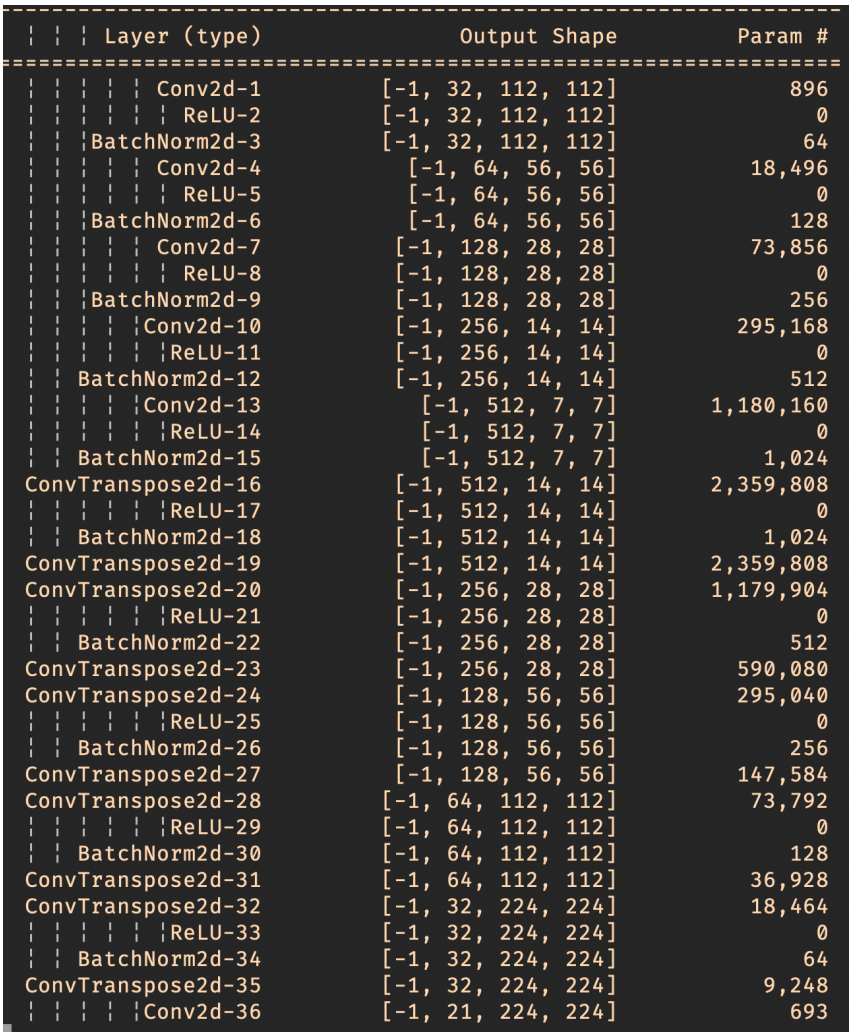
where $y_\cdot^\cdot$ denotes the outputs, $t_\cdot^\cdot$ denotes the labels, n ranges over the samples and i ranges over the classes. $\gamma \geq 0$ is the 'focusing parameter'. The idea of using 'Focal Loss' is, that if a sample is already well-classified, we can significantly decrease or down-weigh its contribution to the loss. This allows the model to incorporate the losses made on the misclassifications arising only from the very low probability examples i.e. hard examples.

3.6 Experimentation :

3.6.1 Custom Architecture:

We propose a conventional autoencoder model with skip connections. Our autoencoder has 4 convolution layers and 4 de-convolution layers. Similar to UNet, we propose to use the features from the shallower networks and features from deeper networks. The former features capture the boundary information of the segments and the latter captures segmentation information. We combine both these features using residual connections in de-convolution layers. We call this architecture as 'Residual Autoencoder'.

The CNN architecture of the 'Custom Archiecture' model here can be visualized using the table below:



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 112, 112]	896
ReLU-2	[-1, 32, 112, 112]	0
BatchNorm2d-3	[-1, 32, 112, 112]	64
Conv2d-4	[-1, 64, 56, 56]	18,496
ReLU-5	[-1, 64, 56, 56]	0
BatchNorm2d-6	[-1, 64, 56, 56]	128
Conv2d-7	[-1, 128, 28, 28]	73,856
ReLU-8	[-1, 128, 28, 28]	0
BatchNorm2d-9	[-1, 128, 28, 28]	256
Conv2d-10	[-1, 256, 14, 14]	295,168
ReLU-11	[-1, 256, 14, 14]	0
BatchNorm2d-12	[-1, 256, 14, 14]	512
Conv2d-13	[-1, 512, 7, 7]	1,180,160
ReLU-14	[-1, 512, 7, 7]	0
BatchNorm2d-15	[-1, 512, 7, 7]	1,024
ConvTranspose2d-16	[-1, 512, 14, 14]	2,359,808
ReLU-17	[-1, 512, 14, 14]	0
BatchNorm2d-18	[-1, 512, 14, 14]	1,024
ConvTranspose2d-19	[-1, 512, 14, 14]	2,359,808
ConvTranspose2d-20	[-1, 256, 28, 28]	1,179,904
ReLU-21	[-1, 256, 28, 28]	0
BatchNorm2d-22	[-1, 256, 28, 28]	512
ConvTranspose2d-23	[-1, 256, 28, 28]	590,080
ConvTranspose2d-24	[-1, 128, 56, 56]	295,040
ReLU-25	[-1, 128, 56, 56]	0
BatchNorm2d-26	[-1, 128, 56, 56]	256
ConvTranspose2d-27	[-1, 128, 56, 56]	147,584
ConvTranspose2d-28	[-1, 64, 112, 112]	73,792
ReLU-29	[-1, 64, 112, 112]	0
BatchNorm2d-30	[-1, 64, 112, 112]	128
ConvTranspose2d-31	[-1, 64, 112, 112]	36,928
ConvTranspose2d-32	[-1, 32, 224, 224]	18,464
ReLU-33	[-1, 32, 224, 224]	0
BatchNorm2d-34	[-1, 32, 224, 224]	64
ConvTranspose2d-35	[-1, 32, 224, 224]	9,248
Conv2d-36	[-1, 21, 224, 224]	693

Figure 3: Architecture of Custom Residual Autoencoder model

3.6.2 Transfer Learning :

Transfer learning is a method in machine learning where one uses a model developed for a task (base task) as a starting point for a model on a second task (target task, say). This process works if the features in the first task are general i.e. suitable to both base and target tasks, instead of specific to the base task. This is a popular approach in deep learning which is mainly used to tackle problems in natural language processing and computer vision. Transfer learning techniques were first introduced by Stevo Bozinovski and Ante Fulgosi in 1976, details of which can be found in [1].

On the baseline architecture provided, we replaced the encoder layer with a ResNet-34 model which is pre-trained on the ImageNet dataset. Before adding this model, we drop its last two layers. On top of it, the ResNet-34 module is frozen at the time of training. We get the following results after the experiments:

3.6.3 UNet :

It is one of the most popularly used approaches in any semantic segmentation task today. UNet works really well when we do not have a lot of training samples. It has a -shaped encoder-decoder network architecture, which consists of four encoder blocks and four decoder blocks that are connected via

bridges. At each encoder layer, the number of spatial dimensions is halved and the number of filters is doubled. Then at each decoder layer, the number of spatial dimensions is doubled and the number of filters is halved. The encoder part extracts features of the images and learns an abstract representation of the input image through a sequence of the encoder blocks. The decoder network is used to take the abstract representation and generate a semantic segmentation. But after an image is passed through the encoder network once, a lot of information is lost - leading to possibly erroneous semantic segmentation labels while decoding. The bridges here connect the encoder and the decoder network and provide additional information that helps the decoder to generate better semantic features. They also help in mitigating the 'vanishing gradient' problem while backpropagating through such a deep network. There are many applications of U-Net in biomedical image segmentation, such as brain image segmentation and semantic segmentation in general.

The figure below gives an idea of how a UNet architecture looks like :

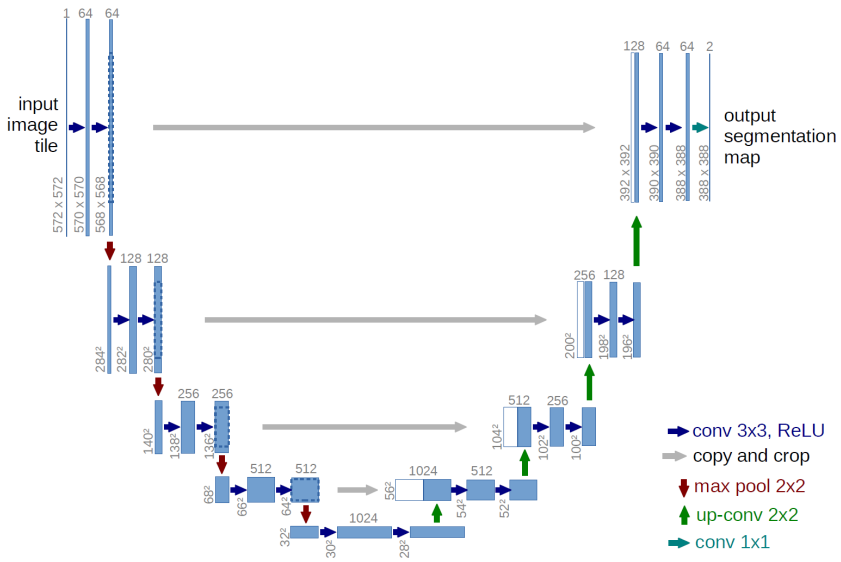
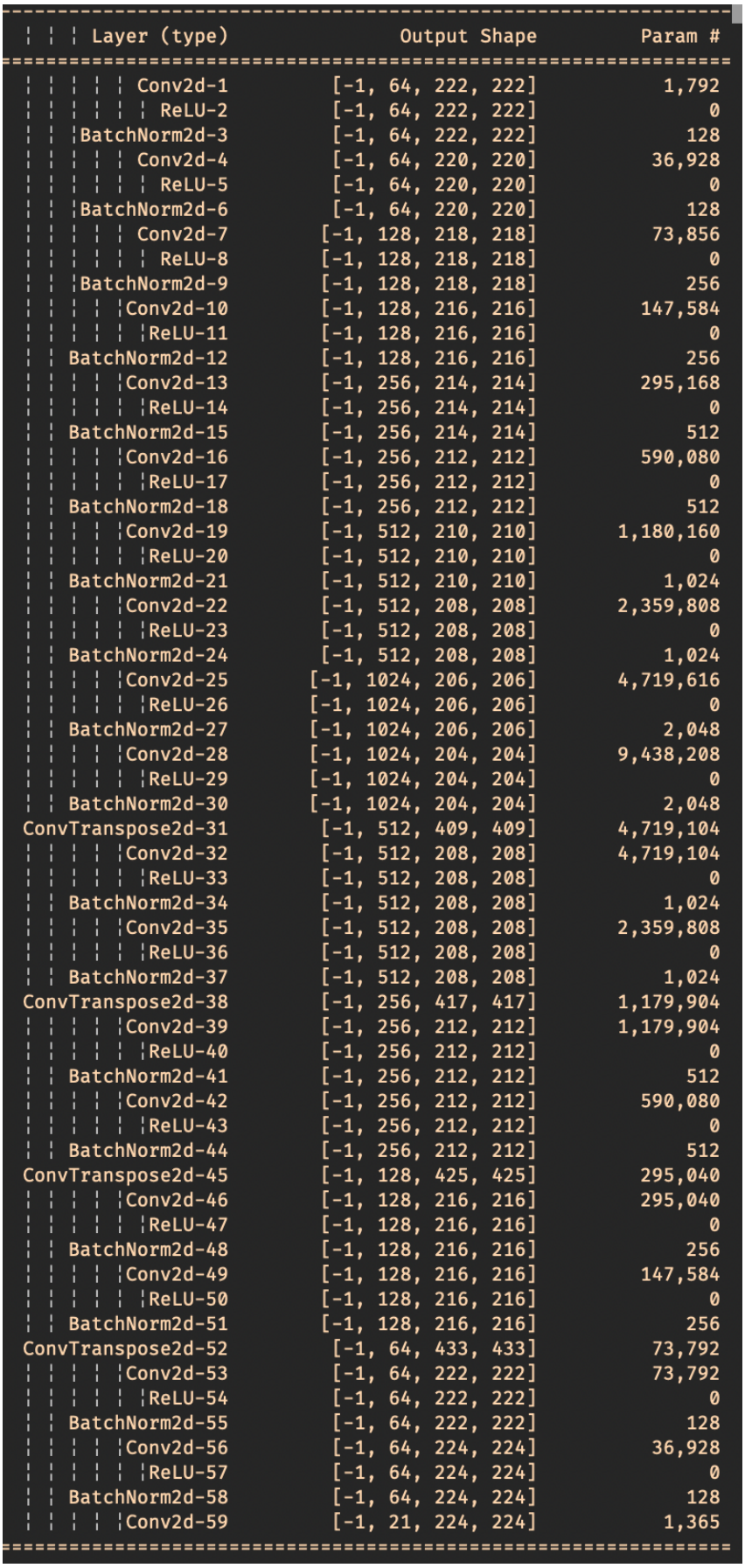


Figure 4: U-net architecture (example for 32X32 pixels in the lowest resolution)[8]. Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

The CNN architecture of the 'UNet' model here can be visualized using the table below:



Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 222, 222]	1,792
ReLU-2	[-1, 64, 222, 222]	0
BatchNorm2d-3	[-1, 64, 222, 222]	128
Conv2d-4	[-1, 64, 220, 220]	36,928
ReLU-5	[-1, 64, 220, 220]	0
BatchNorm2d-6	[-1, 64, 220, 220]	128
Conv2d-7	[-1, 128, 218, 218]	73,856
ReLU-8	[-1, 128, 218, 218]	0
BatchNorm2d-9	[-1, 128, 218, 218]	256
Conv2d-10	[-1, 128, 216, 216]	147,584
ReLU-11	[-1, 128, 216, 216]	0
BatchNorm2d-12	[-1, 128, 216, 216]	256
Conv2d-13	[-1, 256, 214, 214]	295,168
ReLU-14	[-1, 256, 214, 214]	0
BatchNorm2d-15	[-1, 256, 214, 214]	512
Conv2d-16	[-1, 256, 212, 212]	590,080
ReLU-17	[-1, 256, 212, 212]	0
BatchNorm2d-18	[-1, 256, 212, 212]	512
Conv2d-19	[-1, 512, 210, 210]	1,180,160
ReLU-20	[-1, 512, 210, 210]	0
BatchNorm2d-21	[-1, 512, 210, 210]	1,024
Conv2d-22	[-1, 512, 208, 208]	2,359,808
ReLU-23	[-1, 512, 208, 208]	0
BatchNorm2d-24	[-1, 512, 208, 208]	1,024
Conv2d-25	[-1, 1024, 206, 206]	4,719,616
ReLU-26	[-1, 1024, 206, 206]	0
BatchNorm2d-27	[-1, 1024, 206, 206]	2,048
Conv2d-28	[-1, 1024, 204, 204]	9,438,208
ReLU-29	[-1, 1024, 204, 204]	0
BatchNorm2d-30	[-1, 1024, 204, 204]	2,048
ConvTranspose2d-31	[-1, 512, 409, 409]	4,719,104
Conv2d-32	[-1, 512, 208, 208]	4,719,104
ReLU-33	[-1, 512, 208, 208]	0
BatchNorm2d-34	[-1, 512, 208, 208]	1,024
Conv2d-35	[-1, 512, 208, 208]	2,359,808
ReLU-36	[-1, 512, 208, 208]	0
BatchNorm2d-37	[-1, 512, 208, 208]	1,024
ConvTranspose2d-38	[-1, 256, 417, 417]	1,179,904
Conv2d-39	[-1, 256, 212, 212]	1,179,904
ReLU-40	[-1, 256, 212, 212]	0
BatchNorm2d-41	[-1, 256, 212, 212]	512
Conv2d-42	[-1, 256, 212, 212]	590,080
ReLU-43	[-1, 256, 212, 212]	0
BatchNorm2d-44	[-1, 256, 212, 212]	512
ConvTranspose2d-45	[-1, 128, 425, 425]	295,040
Conv2d-46	[-1, 128, 216, 216]	295,040
ReLU-47	[-1, 128, 216, 216]	0
BatchNorm2d-48	[-1, 128, 216, 216]	256
Conv2d-49	[-1, 128, 216, 216]	147,584
ReLU-50	[-1, 128, 216, 216]	0
BatchNorm2d-51	[-1, 128, 216, 216]	256
ConvTranspose2d-52	[-1, 64, 433, 433]	73,792
Conv2d-53	[-1, 64, 222, 222]	73,792
ReLU-54	[-1, 64, 222, 222]	0
BatchNorm2d-55	[-1, 64, 222, 222]	128
Conv2d-56	[-1, 64, 224, 224]	36,928
ReLU-57	[-1, 64, 224, 224]	0
BatchNorm2d-58	[-1, 64, 224, 224]	128
Conv2d-59	[-1, 21, 224, 224]	1,365

Figure 5: Architecture of UNet model

4 Results:

4.1 Baseline Model:

In our experiments, we employed various modifications to improve the performance of models trained on the VOC Segmentation dataset. These modifications comprised data augmentation, activation functions, hyperparameter values, and optimizers, such as Adam and Xavier. Data augmentation was implemented to augment the training dataset and improve the generalization ability of the model. Hyperparameters were tuned to determine the optimal learning rate and regularization strength for the model, while different optimizers were utilized to enhance the convergence rate of the model during training. Through these modifications, we were able to determine the best combination of parameters that resulted in the highest accuracy and best performance for the model. The best results of our experiments are listed below.

Table 1: Baseline Model

Activation	**lr**	**L^2 reg**	**Batch size**	**Val. Acc.**	**Val. IOU**	**Test Acc.**	**Test IOU**
Leaky ReLU	0.0005	0	16	0.7396	0.0657	0.7038	0.0599
Leaky ReLU	0.0005	0.001	16	0.7218	0.0584	0.6990	0.0571
Leaky ReLU	0.0005	0.0005	16	0.6843	0.0567	0.6590	0.0556
ReLU	0.001	0	16	0.74657	0.0605	0.7213	0.0588
ReLU	0.002	0.001	16	0.7486	0.0613	0.7209	0.0592

The Loss-plot (with no. of epochs) and segmentation masks of the images for the best (in terms of IOU) row of hyperparameters above can be found in the following :

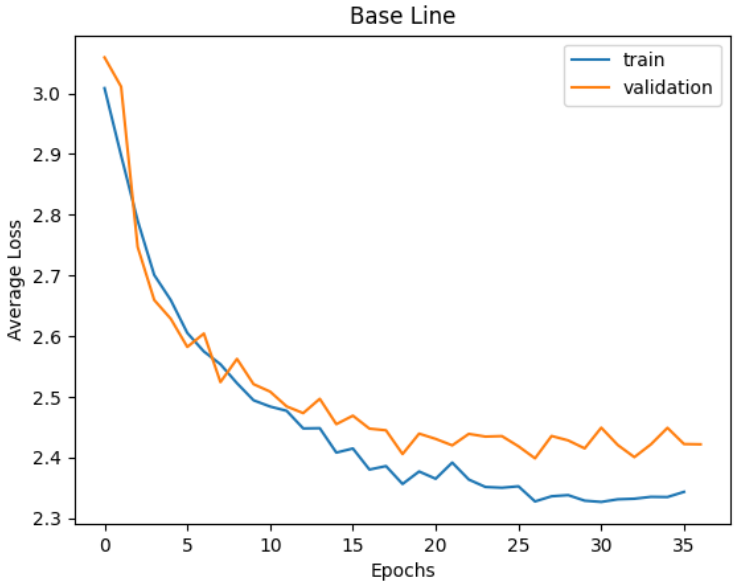


Figure 6: Loss plot for Baseline model

(a) Ground Truth (b) Predicted

Figure 7: Segmentation mask of images predicted by baseline model

4.2 Improvements over Baseline:

4.2.1 Cosine Lr. Scheduler:

First, we try to include Cosine Learning Rate Scheduler as an improvement over our baseline architecture. Keeping the basic setup the same as above unless mentioned otherwise, the best results of the experiments carried out can be found in the following:

Table 2: Improvements over baseline with Cosine Lr Scheduler

Activation	**lr**	L^2 **reg**	**Batch size**	**Val. Acc.**	**Val. IOU**	**Test Acc.**	**Test IOU**
Leaky ReLU	0.005	0.001	16	0.75082	0.05708	0.72607	0.0557
Leaky ReLU	0.0005	0	16	0.7352	0.04292	0.7089	0.0434
ReLU	0.001	0	16	0.74286	0.0642	0.71399	0.06003

The plots and segmentation masks look like the following for the best set of parameters from the table :

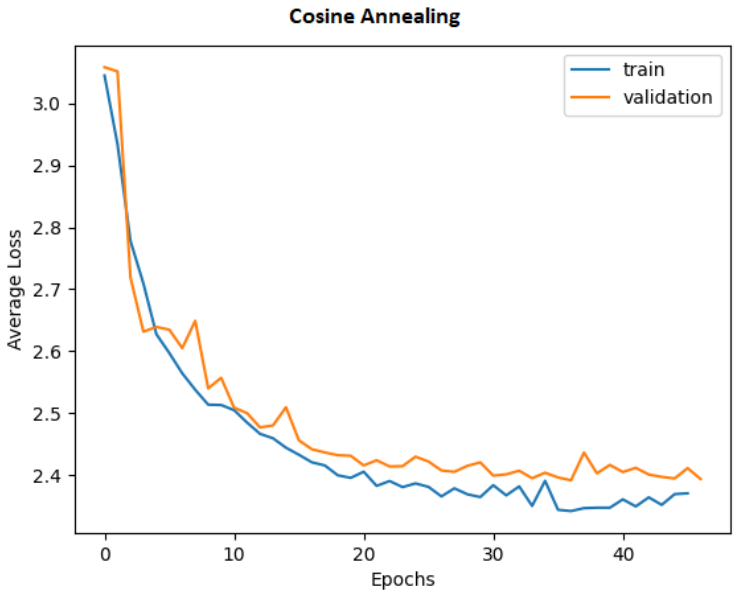


Figure 8: Loss plot for Improvement over baseline with Cosine Lr. Scheduler

(a) Ground Truth (b) Predicted

Figure 9: Segmentation mask of images predicted by baseline model with Cosine Lr Scheduler

4.2.2 Data Augmentation :

First, we try to incorporate data augmentation as an improvement over our baseline architecture. Keeping the basic setup the same as above unless mentioned otherwise, the best results of the experiments carried out can be found in the following:

Table 3: Improvements over baseline with Data Augmentation

Activation	lr	L^2 reg	Batch size	Val. Acc.	Val. IOU	Test Acc.	Test IOU
Leaky ReLU	0.0005	0.0005	16	0.73851	0.06463	0.70971	0.06159
Leaky ReLU	0.001	0.001	16	0.73061	0.04578	0.70575	0.04079

The loss plots and the segmentation masks for the best set of parameters above can be found in the following:

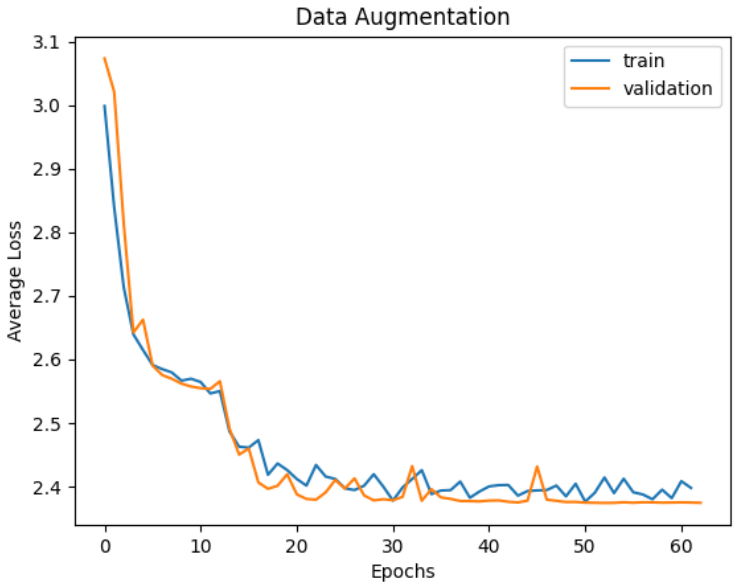


Figure 10: Loss plot for Improvement over baseline with data augmentation

(a) Ground Truth

(b) Predicted

Figure 11: Segmentation mask of images predicted by baseline model with data augmentation

‘

4.2.3 Addressing Class Imbalance Problem:

Now we try to address the class imbalance problem by introducing the weights as mentioned above. We keep the basic setup the same as above unless mentioned otherwise. Out of all the experiments conducted with different sets of parameters, the best ones are mentioned in the table below:

Table 4: Improvements over baseline to mitigate Class Imbalance Problem

Activation	lr	L^2 reg	Batch size	Val. Acc.	Val. IOU	Test Acc.	Test IOU
Leaky ReLU	0.0005	0.0005	16	0.70203	0.05035	0.67435	0.04670
Leaky ReLU	0.001	0.001	16	0.75080	0.05540	0.72890	0.05480

The loss plots and the segmentation masks for the best set of parameters above can be found in the following:

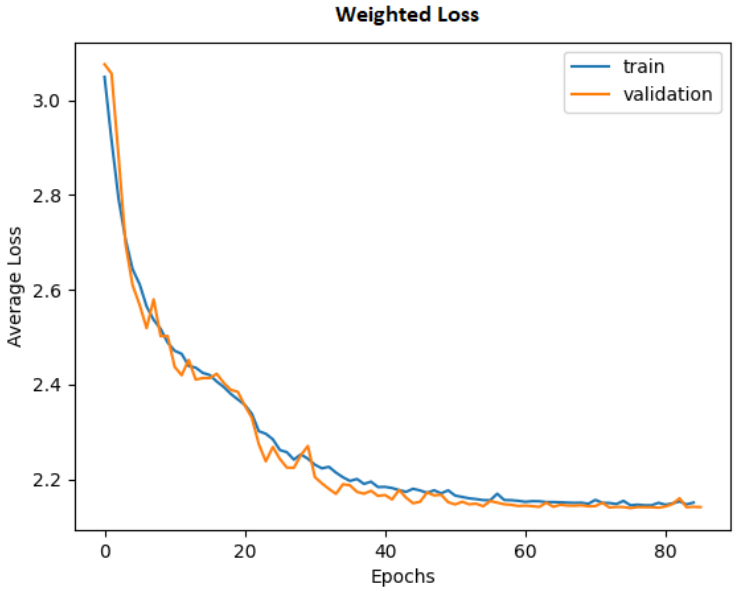
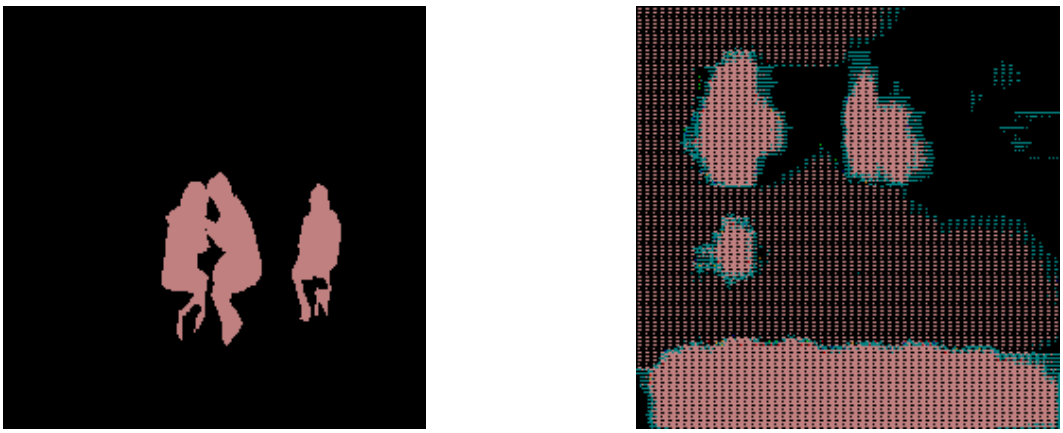


Figure 12: Loss plot for Improvement over baseline with Weighted Loss



(a) Ground Truth

(b) Predicted

Figure 13: Segmentation mask of images predicted by baseline with Weighted Loss

4.2.4 Experimenting with Focal Loss Function:

Keeping the basic setup the same as above (unless mentioned otherwise) Now we try to experiment with 'Focal loss function' instead of 'Cross-Entropy Loss'. Out of all the experiments conducted with different sets of parameters, the best ones are mentioned in the table below:

Table 5: Improvements over baseline using Focal Loss

Activation	lr	L^2 reg	Batch size	Val. Acc.	Val. IOU	Test Acc.	Test IOU
Leaky ReLU	0.0005	0	16	0.7378	0.0456	0.71082	0.0414
ReLU	0.001	0.001	16	0.7449	0.0624	0.7158	0.0571

The loss plots and the segmentation masks for the best set of parameters above can be found in the following:

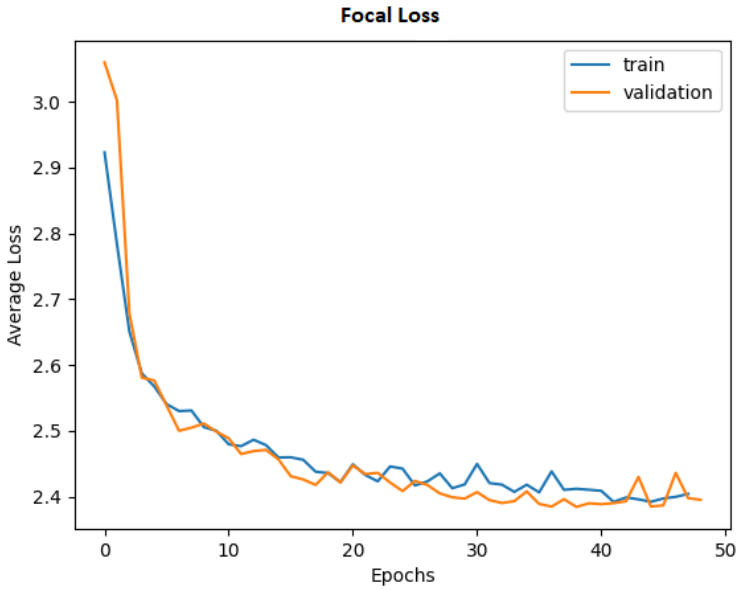


Figure 14: Loss plot for Improvement over baseline with Focal Loss

(a) Ground Truth (b) Predicted

Figure 15: Segmentation mask of images by baseline model with Focal loss

4.3 Experimentation:

Below we provide the best outcomes of experimentations involving Custom Architecture, Transfer Learning and UNet, in the table:

Table 6: Results on Transfer Learning, Unet and Custom Architecture

Model	Val. Acc.	Val. IOU	Test Acc.	Test IOU
Resnet34 Frozen	0.757	0.0786	0.726	0.0744
UNet	0.750	0.0667	0.729	0.0634
ResEncoder	0.751	0.0514	0.729	0.0467

In the experiments above, we kept learning rate=0.0005, batch-size=16, Xavier Initialization, and Adam optimizer.

The loss plots and segmentation masks for each of the tasks are mentioned below sequentially:

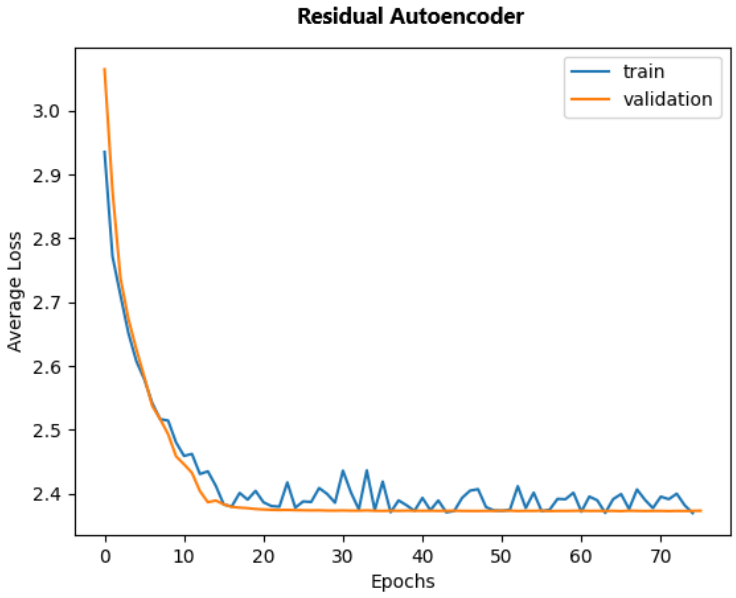


Figure 16: Loss plot for Residual Autoencoder model

(a) Ground Truth (b) Predicted

Figure 17: Segmentation masks of images predicted by Residual Autoencoder

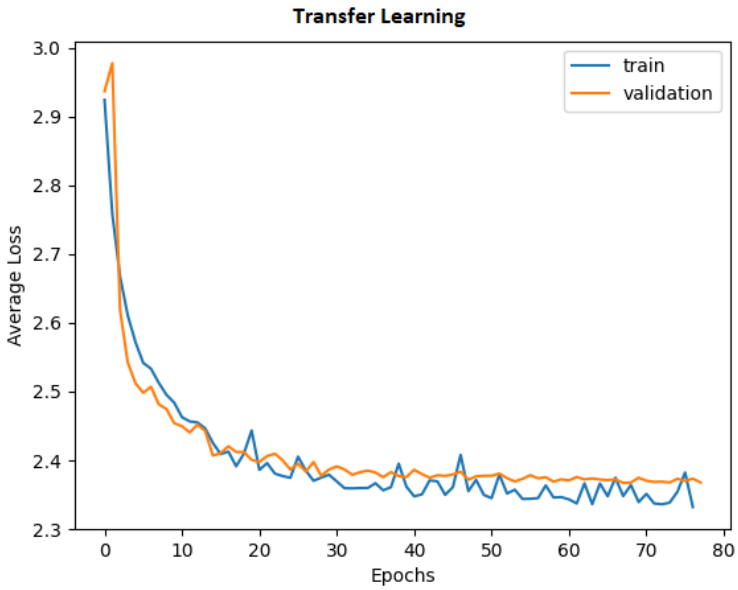


Figure 18: Loss plot for Transfer Learning

(a) Ground Truth (b) Predicted

Figure 19: Segmentation masks of images predicted by Transfer Learning

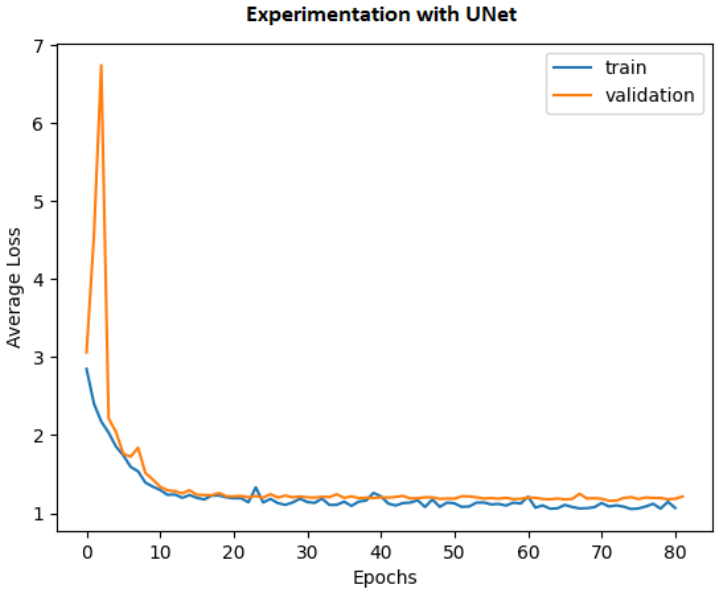


Figure 20: Loss plot for UNet model

(a) Ground Truth

(b) Predicted

Figure 21: Segmentation masks of images predicted by UNet model

5 Discussion :

5.1 Baseline Model :

Upon running the experiments with multiple values of the hyperparameters, we observed that Leaky ReLU serves as a 'better' activation function, giving us more Validation and Test IOU. This is expected because, for ReLU(x), the slope is zero when $x < 0$. However, for Leaky ReLU, the slope is negative (nonzero) when $x < 0$, which is why its slope/gradient does not become zero when $x < 0$ promoting more effective backpropagation. It also speeds up training because its mean is closer to 0. We also noticed that lower learning rates promote higher IOU because smaller learning rates can make a model run more optimal values, including more data points. The problem of semantic segmentation is complex enough and we do not have too much data for training our model. Hence

our model is not likely to undergo overfitting. That is why we can see a great performance even without regularizing. Among the batch sizes $\{16, 32, 64\}$, the smallest batch size is giving us the best result because smaller batches are noisy, thus offering a lower generalization error and a more regularizing effect. Also, since we are using GPUs, smaller batches make it easier to fit one batch worth of training data in memory.

5.2 Improving Baseline Model:

By addressing the class imbalance problem, we saw an increase in validation accuracy to 0.7508, but the IOU is lower compared to the baseline model. We believe it's because we are forcing the model to prioritize the minority classes over the majority classes. With the cosine annealing learning rate scheduler, we can see a similar increase in accuracy to 0.7509, along with better IOU scores of 0.0642. We experimented further by augmenting the data with multiple transforms. The data augmentation helped in making the model more robust as with this model, we saw an increase in the validation IOU to 0.0653. When we replaced Cross Entropy loss with Focal loss, we did not observe significant improvements in the performance.

5.3 Experimentation:

5.3.1 Custom Architecture:

It is observed that the Residual Autoencoder does not perform greatly and generates results similar to the 'baseline' model. This is due to the fact that the boundary features and the segmentation features are being combined in stead of being concatenated. This leads to loss of information. We used de-convolution layers while combining boundary and segmentation information which increases the total number of trainable parameters in the model.

5.3.2 Transfer Learning :

It can be observed that the Transfer Learning model performs significantly better than the baseline architecture. It is due to the fact that the PASCAL VOC dataset is not very big to train a large number of parameters. The ResNet-34 module used for Transfer Learning here is already pre-trained on the huge ImageNet dataset and hence is capable to capture better image segmentation features. Also, since we freeze the encoder module and only train the decoder module in our neural network, which significantly decreases the number of trainable parameters. It can also be observed that if the encoder module is trained simultaneously with the decoder module, the architecture performs sub-optimally.

5.3.3 Unet :

It can be observed that the Unet architecture generates results better than baseline architecture. As Unet architecture uses information from both the deeper and shallower layers, here the shallower layers provide boundary information and the deeper layers provide segmentation information. However, a conventional autoencoder only uses information from the deeper layers giving us suboptimal results.

6 Authors' Contributions and References

The project is a result of a team effort where both of us contributed as a group. Both ran the codes and provided necessary data for plotting and making tables in this report and both actively participated in hyperparameter tuning for the momentum method and finding optimal early stop epoch. However, roughly the contribution of each team member is listed below:

- Nishanth implemented validation and test loops, implemented UNet model, Transfer learning. Implemented focal loss, pixel accuracy, and Intersection over union. Implemented code for generating plots and segmentation tasks. Implemented code for data augmentation, data loader. Experimented with baseline model, cosine annealing, transfer learning, UNet, custom model and data augmentation. Reported the custom model, Transfer learning and UNet models results, and methodologies.

- Soumya primarily worked on implementing the codes for mitigating class Imbalance, enabling the Learning rate scheduler and training loop. He also experimented with Baseline Models, Weights Imbalance, Cosine Annealing, and Data Augmentation. For the Project report, he imported some of the plots, and images in the report along with writing the relevant theory behind, Transfer Learning, Cosine Annealing, UNet, Dataset details, etc. He also wrote the discussions of the results of Baseline architecture and typed the discussion section of Transfer Learning's and UNet's results.
- Pranav worked on implementing the code for the training loop part along with Soumya and Nishanth. He also ran multiple experiments like running baseline models,L_2 regularization, different learning schedulers,etc and different combinations of hyperparameters to improve the results. Pranav also worked on analysing of results to help improve the results. In the report, he worked on writing the results and abstract sections.
- Daamini worked on calculating IOU and pixel accuracy metrics along with Nishanth, implementing Baseline Models, code for mitigating weights imbalance by applying weighted loss, cosine annealing, and implementing transforms for data augmentation. Experimented with the improvement of baseline models; generated loss plots and segmentation masks for them. For the project report, wrote the theory for baseline architecture, class imbalance, and data augmentation in Methods. She wrote the discussions of the Improvement over the Baseline Model.

References

[1] Stevo Bozinovski. Reminder of the first paper on transfer learning in neural networks, 1976. *Informatica*, 44, 09 2020.

[2] Gary Cottrell. Convolutional networks, part ii. *CSE 251B Lecture Slides*, Winter 2023(2):1–64, 2023.

[3] Gary Cottrell. Introduction to convolutional networks. *CSE 251B Lecture Slides*, Winter 2023(1):1–71, 2023.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[7] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2016.

[8] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).