

CSE344: Computer Vision

Assignment-1

Divyajeet Singh (2021529)

February 21, 2024

Theory

Question 1.

- (a) The given problem of classifying papaya images into the classes ‘sweet’ and ‘not sweet’ is a binary classification problem. The Mean Squared Error function is non-convex for binary classification. Thus if a binary classifier is trained with MSE Loss, it is not guaranteed to converge to the minimum of the loss. Secondly, using MSE assumes an underlying Gaussian distribution¹, which is not valid in case of binary classification, since it is modeled with a Bernoulli distribution.
- (b) For a single training example, the Binary Cross Entropy loss function is given by (assuming \hat{y} is a valid probability in $[0, 1]$)

$$\mathcal{L}(\hat{y}) = -y \log_2 (\hat{y}) - (1 - y) \log_2 (1 - \hat{y})$$

- (c) On a negative example (where $y = 0$), if the predicted value is $\hat{y} = 0.9$, the value of loss is given by

$$\mathcal{L}(0.9) = -0 \log_2 (0.9) - (1 - 0) \log_2 (1 - 0.9) = -\log_2 (0.1) \approx 3.322$$

- (d) Now, we calculate the mean loss for 3 examples for the given ground truth and predicted values

$$\begin{aligned}\mathcal{L}(\hat{Y}) &= -\frac{1}{3} \sum_{i=1}^3 y_i \log_2 \hat{y}_i + (1 - y_i) \log_2 (1 - \hat{y}_i) \\ &= -\frac{1}{3} (\log_2 (0.1) + \log_2 (0.8) + \log_2 (0.3)) \\ &\approx \frac{1}{3} (3.322 + 0.322 + 1.737) = \frac{5.381}{3} \approx 1.793\end{aligned}$$

- (e) For a learning rate α and a trainable weight W , the update formula while using L_2 -regularization (hyperparameter λ) is given by

$$W_{\text{new}} \leftarrow W_{\text{old}} - \alpha \left(\frac{\partial L_{BCE}}{\partial W} + 2\lambda W_{\text{old}} \right)$$

The L_2 -regularization applied in model A penalizes very large weights by adding some proportion of the weights themselves to the gradient update. Hence, we expect model A ’s weights to be relatively smaller than model B ’s final trained weights. However, this technique (usually) reduces overfitting, as model A will be encouraged to learn simpler patterns than model B .

- (f) KL-divergence is a measure of ‘distance’ or difference between two probability distributions. KL-divergence measures the relative entropy of two distributions. On the other hand, cross entropy can be thought of as the total entropy between them. The formula for KL-divergence and cross-entropy are as follows

$$\begin{aligned}D_{\text{KL}}(P||Q) &= \mathbb{E}_{X \sim P} \left[\log \frac{P(X)}{Q(X)} \right] = \mathbb{E}_{X \sim P} [\log P(X) - \log Q(X)] \\ H(P, Q) &= -\mathbb{E}_{X \sim P} [\log Q(X)]\end{aligned}$$

¹In fact, mean squared error is the KL divergence between the empirical distribution of the data and a Gaussian model.

These quantities are closely related - minimizing the cross-entropy $H(P, Q)$ with respect to Q is equivalent to minimize the KL-divergence $D_{\text{KL}}(P||Q)$. Their relationship is given in the following equations

$$\begin{aligned} D_{\text{KL}}(P||Q) &= \mathbb{E}_{X \sim P}[\log P(X)] - \mathbb{E}_{X \sim P}[\log Q(X)] \\ &= H(P, Q) + \mathbb{E}_{X \sim P}[\log P(X)] \\ &= H(P, Q) - H(P) \\ \implies H(P, Q) &= H(P) + D_{\text{KL}}(P||Q) \end{aligned}$$

where Shannon entropy of a distribution P is defined as in class,

$$H(P) = -\mathbb{E}_{X \sim P}[\log P(X)]$$

Question 2.

We are given the following information about the 2-layer neural network for K -class classification problem.

$$\begin{aligned} z^{[i]} &= W^{[i]}a^{[i-1]} + b^{[i]}, \quad i = 1, 2 \quad a^{[0]} = x \\ a^{[1]} &= \text{LEAKYRELU}\left(z^{[1]}, \alpha = 10^{-2}\right) \\ \hat{y} &= \text{SOFTMAX}\left(z^{[2]}\right) = \sigma\left(z^{[2]}\right) \quad (\text{say}) \\ \mathcal{L}(\hat{y}) &= -\sum_{i=1}^K y_i \log(\hat{y}_i) \end{aligned}$$

- (a) Let us assume $z^{[2]}$ is of size $K \times 1$ (since it is a K -class classification problem). Since $z^{[1]}$ is of size $D_a \times 1$, $a^{[1]}$ must be of the same size, and hence, the $W^{[2]}$ must be of size $K \times D_a$. Following this, $b^{[2]}$ must be of size $K \times 1$.

$$\underbrace{z^{[2]}}_{K \times 1} = \underbrace{W^{[2]}}_{K \times D_a} \underbrace{a^{[1]}}_{D_a \times 1} + \underbrace{b^{[2]}}_{K \times 1}$$

By the same logic, $W^{[1]}$ must be of size $D_a \times D_x$, since the output $z^{[1]}$ must be of dimension $D_a \times 1$ while the input is of size $D_x \times 1$.

Now, if we want to vectorize our input to a batch size of m , the input X will be of size $D_x \times m$. This means that the output $z^{[1]}$ of the hidden layer will be $D_a \times m$, since

$$\underbrace{z^{[1]}}_{D_a \times m} = \underbrace{W^{[1]}}_{D_a \times D_x} \underbrace{X}_{D_x \times m} + \underbrace{b^{[1]}}_{D_a \times 1^*}$$

***Note:** Bias is added component-wise to each column of $W^{[1]}X$ in case of vectorization of operations.

- (b) Let us denote the sum of exponentials (normalizing term) of the entries in $z^{[2]}$ by

$$\mathbf{z} = \sum_{j=1}^K \exp z_j^{[2]}, \text{ which means } \hat{y} = \sigma\left(z^{[2]}\right) = \frac{\exp z^{[2]}}{\mathbf{z}}$$

We are required to find the partial derivative of the k^{th} entry in \hat{y} with respect to the k^{th} entry in $z^{[2]}$.

$$\begin{aligned} \frac{\partial \hat{y}_k}{\partial z_k^{[2]}} &= \frac{\partial}{\partial z_k^{[2]}} \left(\sigma\left(z_k^{[2]}\right) \right) = \frac{\partial}{\partial z_k^{[2]}} \left(\frac{\exp z_k^{[2]}}{\mathbf{z}} \right) \\ &= \frac{1}{\mathbf{z}^2} \left(\mathbf{z} \frac{\partial}{\partial z_k^{[2]}} (\exp z_k^{[2]}) - \exp z_k^{[2]} \frac{\partial}{\partial z_k^{[2]}} (\mathbf{z}) \right) \\ &= \frac{1}{\mathbf{z}^2} \left(\mathbf{z} \exp z_k^{[2]} - (\exp z_k^{[2]})^2 \right) = \frac{\exp z_k^{[2]}}{\mathbf{z}} - \left(\frac{\exp z_k^{[2]}}{\mathbf{z}} \right)^2 \\ &= \hat{y}_k - \hat{y}_k^2 = \hat{y}_k (1 - \hat{y}_k) \end{aligned}$$

(c) Next, we find the partial derivative of the k^{th} entry in \hat{y} with respect to the i^{th} entry in $z^{[2]}$, given $i \neq k$.

$$\begin{aligned}\frac{\partial \hat{y}_k}{\partial z_i^{[2]}} &= \frac{\partial}{\partial z_i^{[2]}} \left(\sigma(z_k^{[2]}) \right) = \frac{\partial}{\partial z_i^{[2]}} \left(\frac{\exp z_k^{[2]}}{\mathbf{z}} \right) \\ &= \frac{1}{\mathbf{z}^2} \left(\mathbf{z} \frac{\partial}{\partial z_i^{[2]}} (\exp z_k^{[2]}) - \exp z_k^{[2]} \frac{\partial}{\partial z_i^{[2]}} (\mathbf{z}) \right) \\ &= \frac{1}{\mathbf{z}^2} \left(0 - \exp z_k^{[2]} \exp z_i^{[2]} \right) = -\frac{\exp z_k^{[2]}}{\mathbf{z}} \cdot \frac{\exp z_i^{[2]}}{\mathbf{z}} \\ &= -\hat{y}_k \hat{y}_i\end{aligned}$$

(d) We are interested in the partial derivative of the loss with respect to entries in $z^{[2]}$. Using the chain rule, we have

$$\frac{\partial \mathcal{L}}{\partial z_i^{[2]}} = \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i^{[2]}}$$

With the above (given) loss function, we can find

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_i} = \frac{\partial}{\partial \hat{y}_i} \left(- \sum_{j=1}^K y_j \log(\hat{y}_j) \right) = - \sum_{j=1}^K y_j \frac{\partial}{\partial \hat{y}_i} (\log(\hat{y}_j)) = -\frac{y_i}{\hat{y}_i}$$

Note that as a result from **Question 2.** (b),

$$\frac{\partial \hat{y}_i}{\partial z_i^{[2]}} = \hat{y}_i(1 - \hat{y}_i)$$

So, we have

$$\frac{\partial \mathcal{L}}{\partial z_i^{[2]}} = -\frac{y_i}{\hat{y}_i} \cdot \hat{y}_i(1 - \hat{y}_i) = -y_i(1 - \hat{y}_i)$$

Now, we are given that $y_k = 1$ and $y_i = 0, i \neq k$. So, we get (where \mathbb{I} is an indicator variable)

$$\frac{\partial \mathcal{L}}{\partial z_i^{[2]}} = \mathbb{I}_{\{i=k\}}(\hat{y}_i - 1) = \begin{cases} \hat{y}_k - 1 & \text{if } i = k \\ 0 & \text{if } i \neq k \end{cases}$$

(e) While implementing the softmax function, we run into the problem of numerical overflow and underflow. Overflow occurs while exponentiating large numbers, when the result exceeds the computer's largest representable float value. In such cases, the result of e^z is estimated to be infinity (`torch.inf` in PyTorch). Conversely, underflow occurs while exponentiating small numbers, when extremely small values are represented by zeros. Rarely, it is also possible that the denominator is extremely small, which makes division unstable.

A common modification to resolve the issue is normalizing the input by max-shifting. In this technique, we subtract the maximum of the vector from all its entries, making all of them non-positive, and at least one entry 0. The output probabilities do not change because the relative difference between the entries in z does not change, but the numerical operations become more stable. So, for a vector z of size m , we get

$$\begin{aligned}z^* &= z - \max_{k=1,2,\dots,m} z_k \\ \sigma(z) &= \sigma(z^*) = \frac{\exp z^*}{\sum_{j=1}^m \exp z_j^*}\end{aligned}$$

Image Classification

The solutions to the questions in this section are given in `classification.ipynb`.

Question 1.

(a) The dataset was downloaded to my system from the Google Drive (I know, I couldn't figure out how to connect it to Google Colab and use GPU :sad:). The dataset was loaded using a custom class, `WildlifeDataset`, which also handled the category to label mapping.

- (b) The splits of ratio 0.7:0.1:0.2 were generated and loaded using dataloaders with batch size 256.
- (c) To visualize the data distribution across class labels, we count the number of images in the entire dataset. Then, I compared the proportion of images in each class in the training and validation set. ‘Proportion’ refers to the fraction of images of the entire dataset belonging to each class. The plots are given in the Figure (1) (Blue represents the training set and Orange represents the validation set).

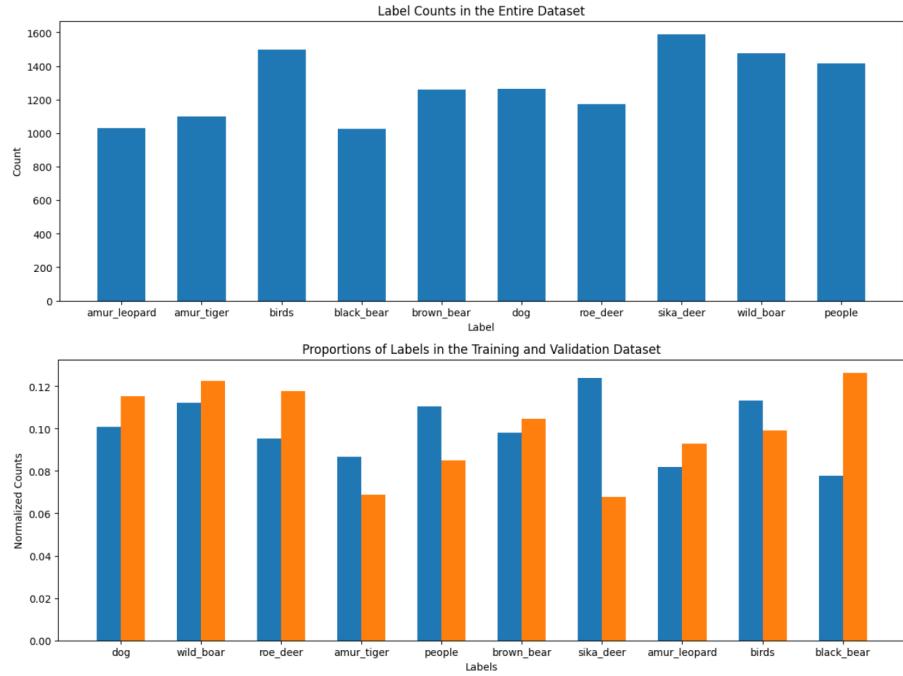


Figure 1: Class distribution in the entire dataset, training set, and validation set

Question 2.

- (a) The given CNN architecture was constructed in the class `ConvNet` using PyTorch and was trained for a total of 15 epochs.
- (b) The per epoch loss/accuracy curves generated using wandb are given in Figure (2).

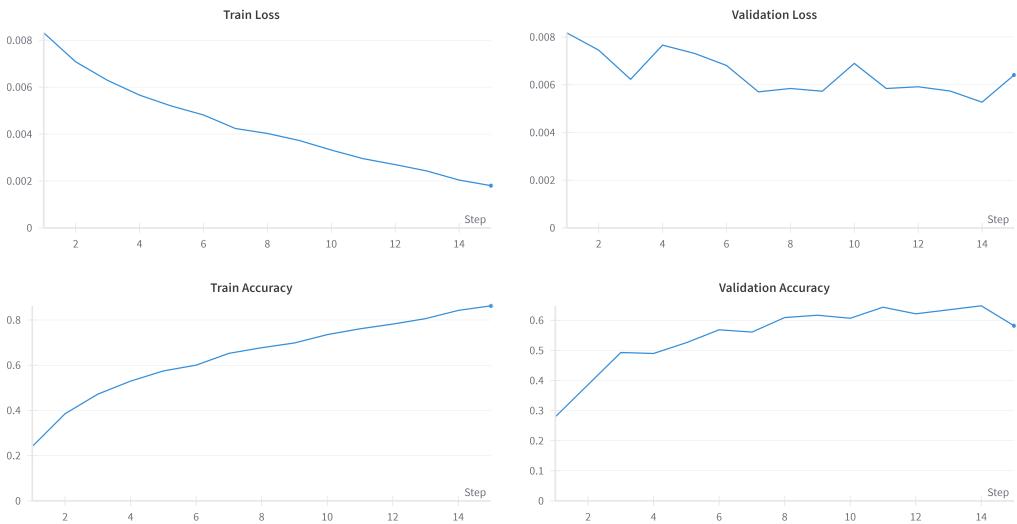


Figure 2: Loss and Accuracy curves (per epoch) for the given CNN architecture

- (c) Looking at the figures, it is clear that the model is overfitting after training for a few epochs, since the validation loss stops decreasing sharply while the training loss keeps decreasing. There is a large gap

between the training and validation losses, and the same goes for accuracies. The final training accuracy after 15 epochs is close to 0.86+, while the validation accuracy was around 0.58+ only.

- (d) The confusion matrix (logged using wandb) is given in Figure (3). However, it is still surprising that the accuracy on the test set was 0.917+. This is a good example of why accuracy is not the only indicator of a model's performance. The precision, recall, and F1-score for the classification all turned out to be 0.588+.

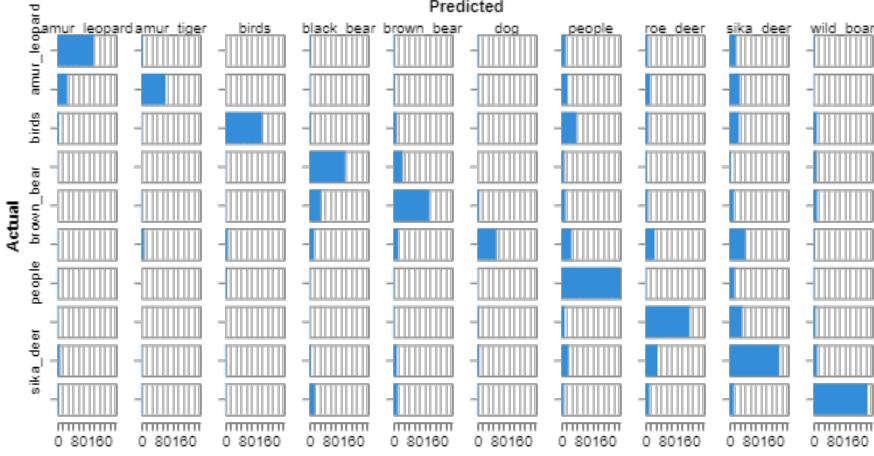


Figure 3: Confusion Matrix for the given CNN architecture

- (e) Finally, we visualize 3 misclassified images from each class. These are also given in the same notebook. A few examples are also given in Figure (4). There were quite a few examples where the ground truth class was not present or did not cover a majority of the image. For example, some images of class `amur_leopard` were not clear. In some cases, the ground truth class was also not present, for example, in case of `black_bear`. There was also a case where the image looks like the predicted class - `brown_bear` being predicted as `black_bear` and `wild_bear`.

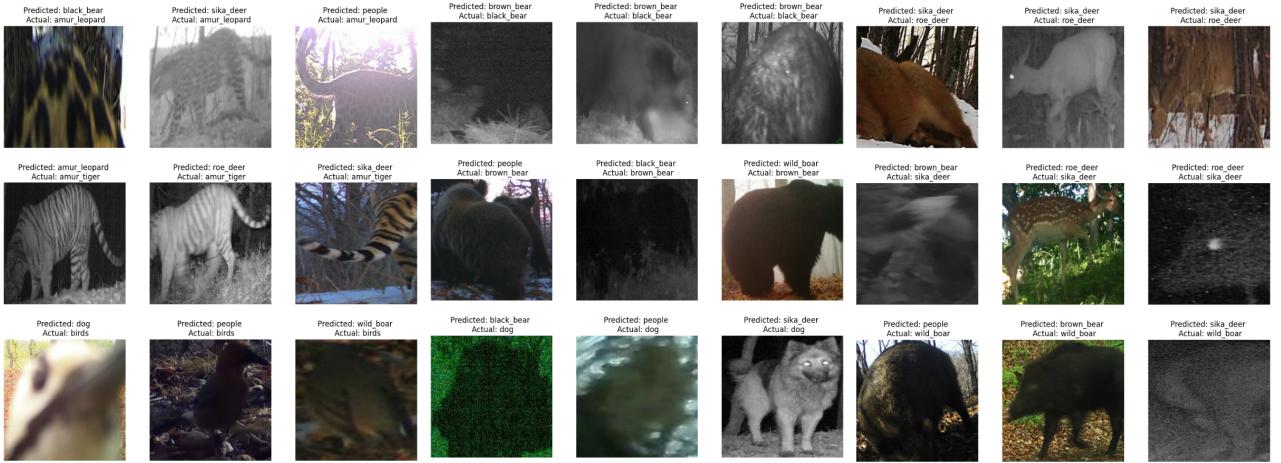


Figure 4: Classwise examples of misclassified images

Question 3.

- (a) Next, we finetune the Resnet-18 model available in PyTorch on the Russian Wildlife dataset. The model was finetuned for a total of 10 epochs due to computational limitations. The per epoch loss/accuracy curves generated using wandb are given in Figure (5).
- (b) This model is not overfitting as much as the previous model. This is inferred because the difference between the training loss and validation loss is lower, and the same goes for the accuracy.
- (c) The Resnet-18 model performed better than the previous model. It achieved an accuracy of 0.969+. The precision, recall, and F1-score for the classification all turned out to be 0.846+. The confusion matrix (logged using wandb) is given in Figure (6).

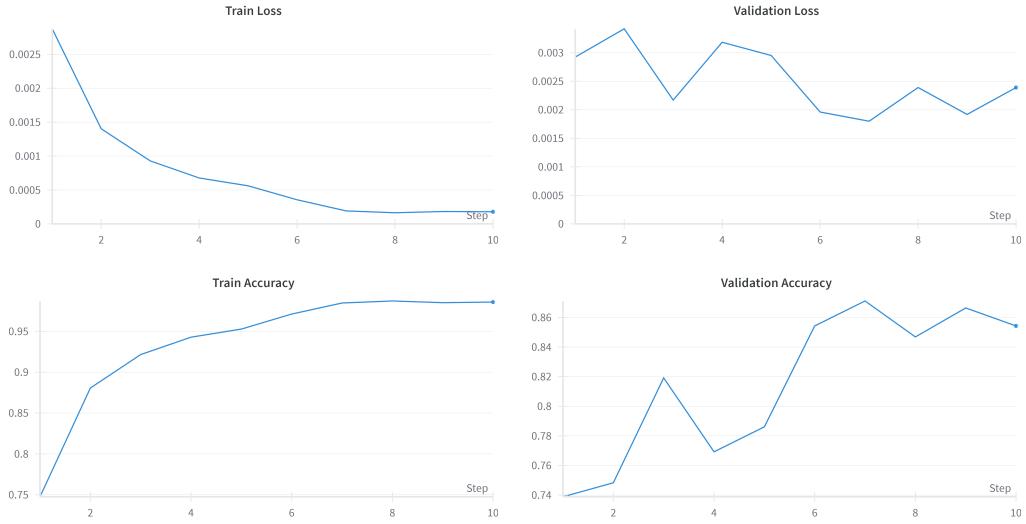


Figure 5: Loss and Accuracy curves (per epoch) for Resnet-18

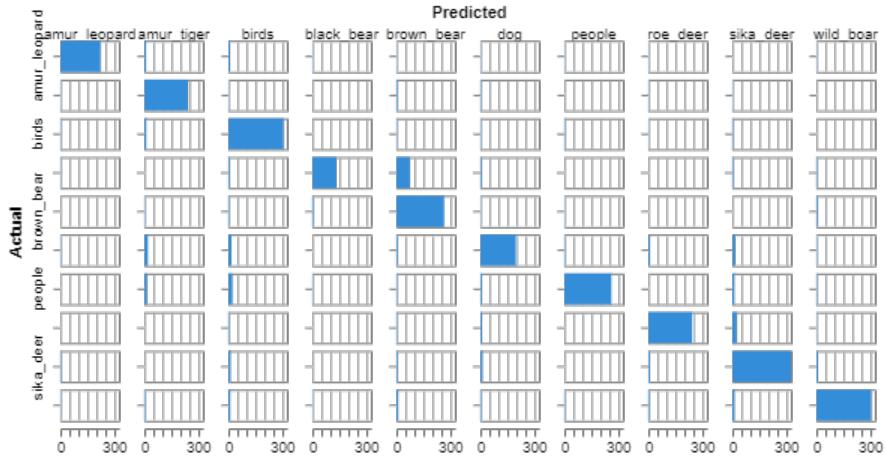


Figure 6: Confusion Matrix for the Resnet-18 architecture

- (d) The 2D and 3D representations of the feature space generated by input samples from training and validation sets by the classifier were generated and are given in Figure (7) and Figure (8) respectively.

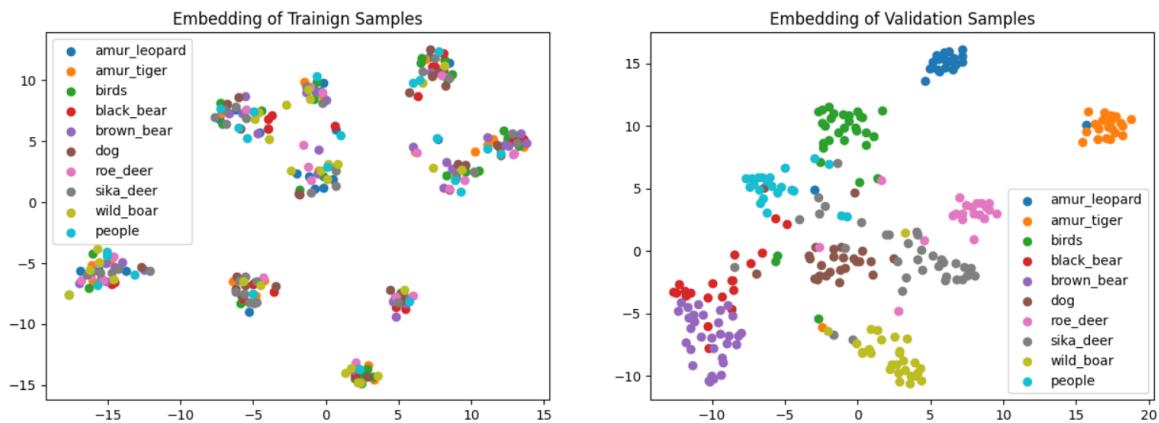


Figure 7: 2D t-SNE visualizations of the training and validation set

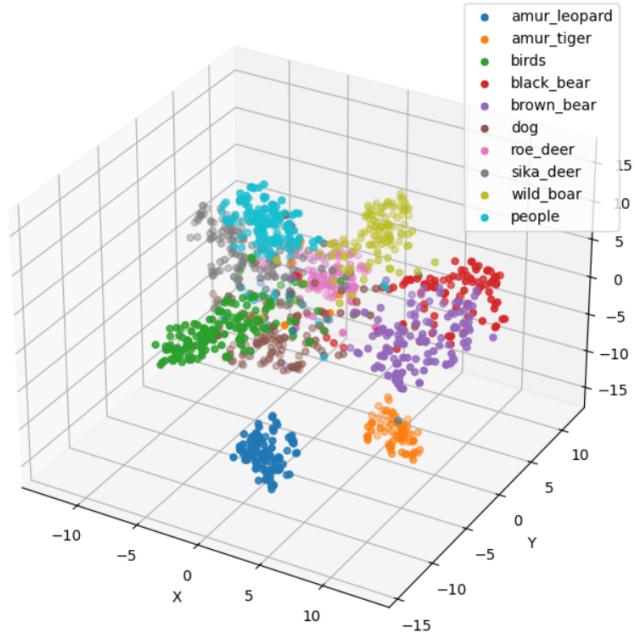


Figure 8: 3D t -SNE visualizations of the validation set

Question 4.

- (a) To perform data augmentation to increase robustness of the model, I used different transforms to add randomness in the data, which are available in `torchvision.transforms`. Specifically, I applied random affine transforms, random horizontal flipping with probability 0.5 for each image, and added random color noise or jitter in the images, altering their color schemes (brightness, contrast, saturation, and hue). A total of 2000 augmented images were added to the dataset.
- (b) As asked in the Question, the same steps as **Question 2.3 (a)** were followed to train the Resnet-18 model on the augmented dataset.
- (c) Looking at the loss curves, it is clear that this model performs even better than before, and is not overfitting. The per epoch loss/accuracy curves generated using wandb are given in Figure (9).



Figure 9: Loss and Accuracy curves (per epoch) for Resnet-18 architecture (on augmented dataset)

- (d) The Resnet-18 model finetuned on the augmented dataset performed marginally better than the previous model. It achieved an accuracy of 0.976+. The precision, recall, and F1-score for the classification all turned out to be 0.883+. The confusion matrix (logged using wandb) is given in Figure (10).

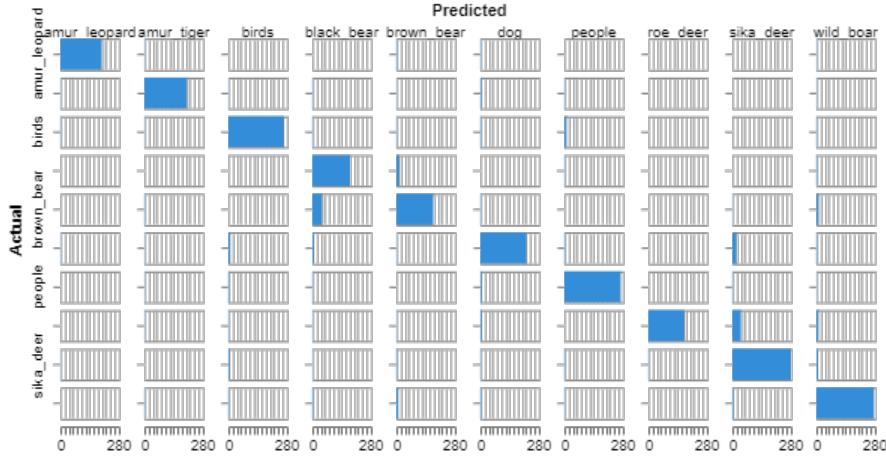


Figure 10: Confusion Matrix for the Resnet-18 architecture (on augmented dataset)

Question 5.

The analysis for this question is also given in the notebook. The euclidean distances between the feature space representations of the misclassified images were calculated from the mean of both the ground truth and predicted classes. It is interesting to note that in some cases, the distance to the mean of the ground truth class was lower than the distance to the mean of the predicted class. However, such cases were rare. A few examples are given in Figure (11).

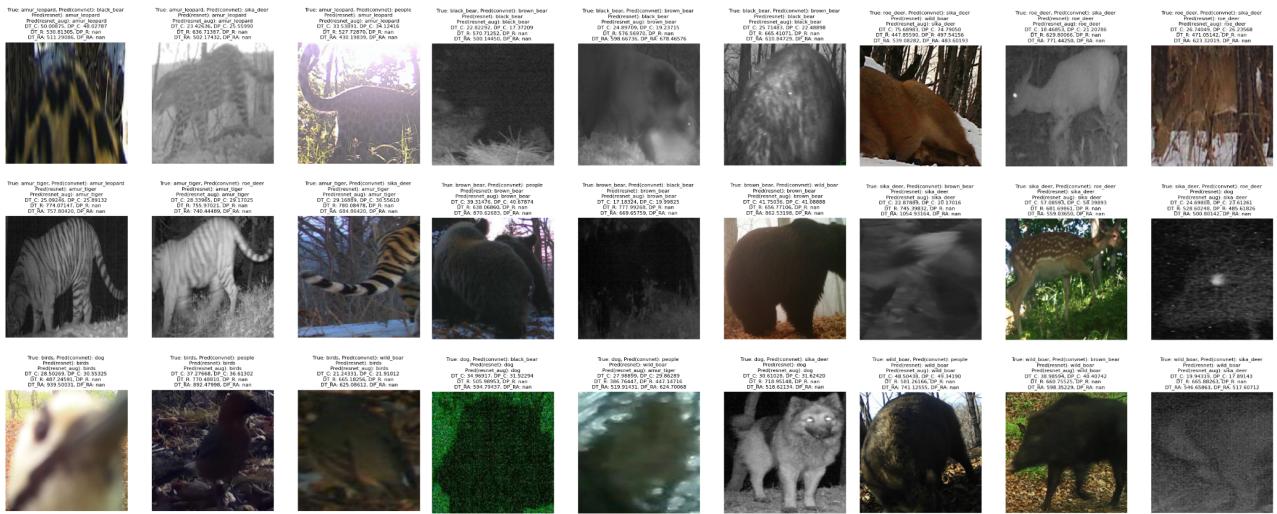


Figure 11: Euclidean distances of misclassified images to centroids of predicted and true classes

Question 6.

On comparing all the three models, it is pretty clear that the vanilla CNN architecture performs well, but not as well as we would like. Its recall and F1-scores are low. This is due to its comparitively lower capacity and higher tendency to overfit the data. On the flip side, the Resnet-18 architectures performed extremely well. Both models did not overfit, achieving high scores for all common evaluation metrics. The model trained with the augmented dataset outperformed the other one by a small margin, since adding some noise to the dataset increases its ability to generalize.

Image Segmentation

The solutions to the questions in this section are given in two notebooks, namely `segmentation-1.ipynb` and `segmentation-2.ipynb` (By this time, I figured out connecting Google Drive to Google Colab, and hence the second notebook performed inferences using the GPU).

Question 1.

- (a) Similar to the image classification problem, I created a custom dataset class, IDD, which also handles label mappings and mask color mappings for both the IDD and Cityscapes dataset. Since the images were larger in size, I used a batch size of 8 to create the dataloader. The dataset was imported from Google Drive :)
- (b) To visualize the data distribution across the dataset, I used pixel counts in the masks - the frequency of pixels belonging to each of the classes. Due to computational limitations, the plot was generated only for a fraction of the dataset. The distribution is given in Figure (12).

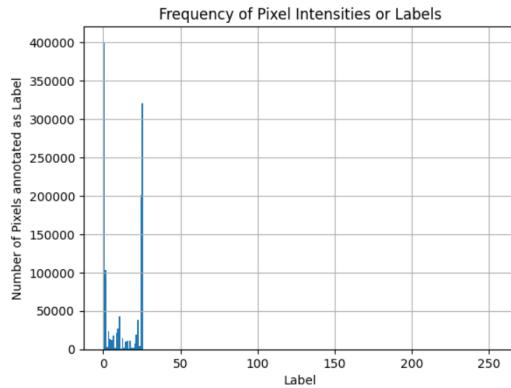


Figure 12: Pixel Counts of each class (representing data distribution)

- (c) For each class, I visualize the images and their masks (and show which class is being highlighted in each case). The masks were color coded according to the conventions of the IDD Dataset. The results are given in `segmentation-1.ipynb`, and a few examples are given in Figure (13).

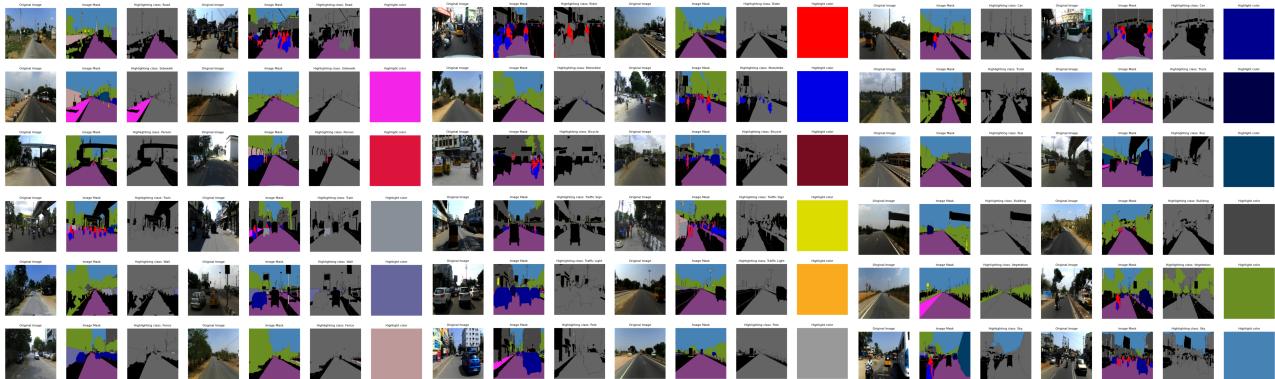


Figure 13: Visualization of three images of every class

Question 2.

- (a) The DeepLabv3Plus-PyTorch Resnet-101 model was used for performing inferences on 30% IDD dataset (using it as test set). The inference was performed in the second notebook, which is GPU enabled.
- (b) The classwise performances in terms of the given metrics were evaluated. The results are given in Figure (14) (Forgive me, I did not have enough time to present these in a better manner).
- (c) I visualize three images per class where the IoU between the predicted and true masks is at most 0.5. It was easy to notice the faults - in some cases, the IoU was close to 0.48+, for example, in case of **Road**. In other cases, like **Sidewalk** or **Traffic-Light**, the predicted masks did not contain these classes (hence the IoUs were 0). This is probably because the model was trained on a different and less complex dataset, resulting in classes that occur on only a few pixels per image to be *ignored*. In such cases, the surrounding objects and environment were overpowering the ground truth class. For example, most pixels of **Sidewalk** were being predicted as **Road**. These *bad IoU* examples are given in Figure (15).

Pixelwise Accuracy: 0.7160231471061707	Dice Coefficient for class Road: 0.8473682999610901
Pixelwise Accuracy for class Road: 0.9836636185646057	Dice Coefficient for class Sidewalk: 0.17147855460643768
Pixelwise Accuracy for class Sidewalk: 0.020291363820433617	Dice Coefficient for class Person: 0.39689794182777495
Pixelwise Accuracy for class Person: 0.314232259887848	Dice Coefficient for class Rider: 0.31636640429496765
Pixelwise Accuracy for class Rider: 0.2590262293815613	Dice Coefficient for class Motorbike: 0.31352084488836365
Pixelwise Accuracy for class Motorbike: 0.08950617164373398	Dice Coefficient for class Bicycle: 0.05926137790883885
Pixelwise Accuracy for class Bicycle: 0.00280378289767265	Dice Coefficient for class Car: 0.621828377468567
Pixelwise Accuracy for class Car: 0.6851978898048401	Dice Coefficient for class Truck: 0.380579948425293
Pixelwise Accuracy for class Bus: 0.0202464796602726	Dice Coefficient for class Bus: 0.5637191534042358
Pixelwise Accuracy for class Train: 0.0	Dice Coefficient for class Train: 0.0023486672434955835
Pixelwise Accuracy for class Wall: 0.001045203388286694	Dice Coefficient for class Wall: 0.2748376429080963
Pixelwise Accuracy for class Fence: 0.7999508635299683	Dice Coefficient for class Fence: 0.14599819447994232
Pixelwise Accuracy for class Traffic Sign: 0.3325228095054625	Dice Coefficient for class Traffic Sign: 0.119481422945766449
Pixelwise Accuracy for class Traffic Light: 0.0	Dice Coefficient for class Traffic Light: 0.0025529515628159046
Pixelwise Accuracy for class Pole: 0.0	Dice Coefficient for class Pole: 0.2409321367740631
Pixelwise Accuracy for class Building: 0.710885928440094	Dice Coefficient for class Building: 0.5946892499923706
Pixelwise Accuracy for class Vegetation: 0.7507008910179138	Dice Coefficient for class Vegetation: 0.8268332481384277
Pixelwise Accuracy for class Sky: 0.8298686146736145	Dice Coefficient for class Sky: 0.9192619323730469
Pixelwise Accuracy for class Unlabeled: nan	Dice Coefficient for class Unlabeled: 0.0010300553403794765

Intersection over Union for class Road: 0.7351597547531128	Intersection over Union for class Road: 0.7351597547531128
Intersection over Union for class Sidewalk: 0.09377989917993546	Intersection over Union for class Sidewalk: 0.09377989917993546
Intersection over Union for class Person: 0.24758119881153107	Intersection over Union for class Person: 0.24758119881153107
Intersection over Union for class Rider: 0.18796920998816	Intersection over Union for class Rider: 0.18796920998816
Intersection over Union for class Motorbike: 0.185990202927589417	Intersection over Union for class Motorbike: 0.185990202927589417
Intersection over Union for class Bicycle: 0.03083347628223896	Intersection over Union for class Bicycle: 0.03083347628223896
Intersection over Union for class Car: 0.45119795203208923	Intersection over Union for class Car: 0.45119795203208923
Intersection over Union for class Truck: 0.20341143012646814	Intersection over Union for class Truck: 0.20341143012646814
Intersection over Union for class Bus: 0.3924853205608847	Intersection over Union for class Bus: 0.3924853205608847
Intersection over Union for class Train: 0.0011757143074646592	Intersection over Union for class Train: 0.0011757143074646592
Intersection over Union for class Wall: 0.15931117534631745	Intersection over Union for class Wall: 0.15931117534631745
Intersection over Union for class Fence: 0.07822419703080674	Intersection over Union for class Fence: 0.07822419703080674
Intersection over Union for class Traffic Sign: 0.0012776560615748167	Intersection over Union for class Traffic Sign: 0.0012776560615748167
Intersection over Union for class Traffic Light: 0.1369657963514328	Intersection over Union for class Traffic Light: 0.1369657963514328
Intersection over Union for class Pole: 0.4231727719306946	Intersection over Union for class Pole: 0.4231727719306946
Intersection over Union for class Building: 0.7047874927520752	Intersection over Union for class Building: 0.7047874927520752
Intersection over Union for class Vegetation: 0.859587610383666	Intersection over Union for class Vegetation: 0.859587610383666
Intersection over Union for class Sky: 0.8298686146736145	Intersection over Union for class Sky: 0.8298686146736145
Intersection over Union for class Unlabeled: 0.0005152930389158428	Intersection over Union for class Unlabeled: 0.0005152930389158428
Mean Intersection over Union (mIoU): 0.2748618620583632	Mean Intersection over Union (mIoU): 0.2748618620583632

Figure 14: Evaluation Metrics for the Image Segmentation

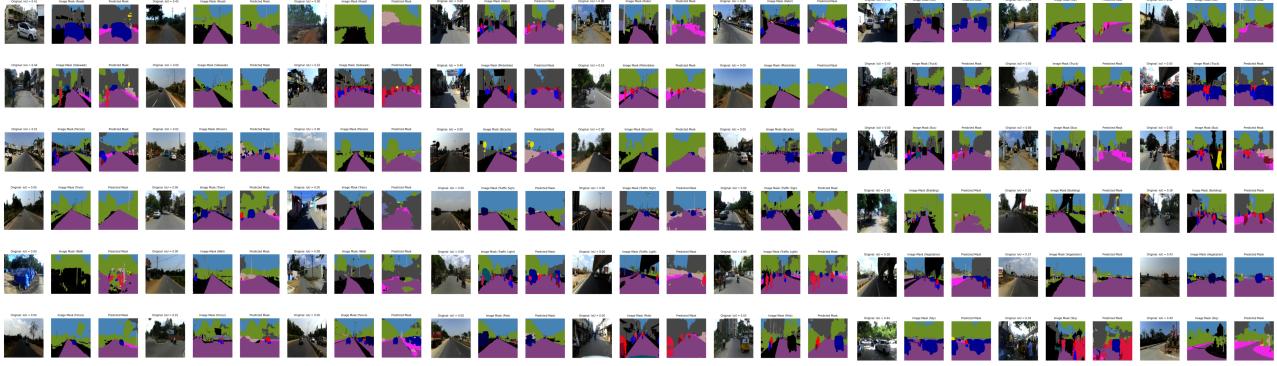


Figure 15: Examples of three images from each class with IoU ≤ 0.5

Question 3.

- (a) The confusion matrix for the final segmentation was generated and is given in Figure (16). We can infer that classes like **Road** and **Sky** are being predicted well, but others were not - this is seen due to the low count of true positives on the diagonals corresponding to these classes. It is also noticeable that most mistakes are being made in less frequent classes, which are being predicted as classes which are more prevalent.
- (b) The classwise precision, recall, and F1-scores were calculated and are given in Figure (17) (again, owing to the lack of time to present the results better). Again, it is clear that the model needs improvements in classes which occur on a small fraction of an image, for example classes like **Sidewalk**, **Train**, **Wall**, **Pole**, etc.

Question 4.

- (a) The GPU enabled notebook was used to perform inference on the Cityscapes dataset. Like before, the generated masks were stored in Google Drive.
- (b) Figure (18) shows the confusion matrix for these inferences, which are arguably better than the previous case. This is because the model was trained on Cityscapes dataset, and is hence able to recognize classes better. Now, more classes have a higher count of true positives (diagonal entries in the confusion matrix).
- (c) Similar to before, the worst performing class performs bad because it is being dominated by their environments containing classes like **Road**, **Vegetation**, and **Sky** - which span a larger area. Exactly these

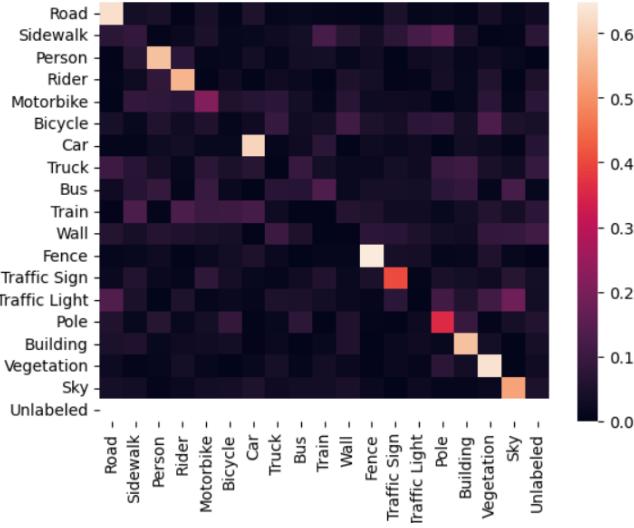


Figure 16: Confusion Matrix for Image Segmentation on Indian Driving Dataset

[Class Road]: precision = 0.4492692506282059, recall = 0.21632923777479, f1 score = 0.4473605305371149
[Class Sidewalk]: precision = 0.1629103477828865, recall = 0.5138562321663981, f1 score = 0.171478562774787
[Class Person]: precision = 0.3279780388179779, recall = 0.5024866461753845, f1 score = 0.396897932843315
[Class Rider]: precision = 0.5766628165817261, recall = 0.21798427932654817, f1 score = 0.31636639748266646
[Class Motorbike]: precision = 0.3809447765358342, recall = 0.214691624045372, f1 score = 0.3135260387111426
[Class Bicycle]: precision = 0.0346524643664875, recall = 0.20446501672267918, f1 score = 0.059261388427443826
[Class Car]: precision = 0.4855184853076935, recall = 0.8645526766777039, f1 score = 0.62128284159658408
[Class Truck]: precision = 0.7103180289268494, recall = 0.2218118906211182, f1 score = 0.3380579930416765
[Class Bus]: precision = 0.7143606543546654663, recall = 0.46554654836654663, f1 score = 0.5617191402113282
[Class Train]: precision = 0.00957831647247076, recall = 0.001338429516181349, f1 score = 0.002348667175256638
[Class Wall]: precision = 0.3322374224652781, recall = 0.2343496680259791, f1 score = 0.27483764091106333
[Class Fence]: precision = 0.001229241395662508, recall = 0.39894165152047, f1 score = 0.0015456459182212
[Class Traffic Sign]: precision = 0.40939575750536687, recall = 0.1131529695381165, f1 score = 0.11308142640182211
[Class Traffic Light]: precision = 0.0975240031957626, recall = 0.001229241395662508, f1 score = 0.00255205141292277993
[Class Pole]: precision = 0.2997402548789978, recall = 0.0141598175300598, f1 score = 0.2499331235191913
[Class Building]: precision = 0.4532054662704466, recall = 0.864686596502264, f1 score = 0.5946891765151598
[Class Vegetation]: precision = 0.7878096103668213, recall = 0.8699245452880859, f1 score = 0.8268333191231034
[Class Sky]: precision = 0.98029625415802, recall = 0.8653822541236877, f1 score = 0.9192619173119756
[Class Unlabeled]: precision = 0.00053609501954065, recall = 0.012639614753425121, f1 score = 0.0010300553899124083

Figure 17: Confusion Matrix for Image Segmentation on Indian Driving Dataset

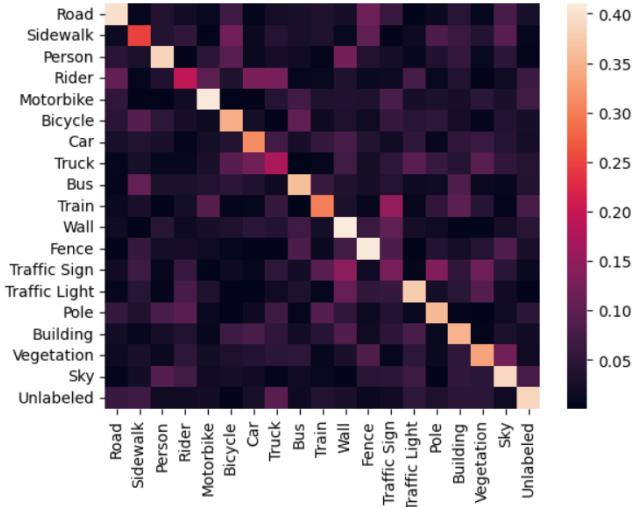


Figure 18: Confusion Matrix for Image Segmentation on Cityscapes Dataset

dominating classes perform well, since they were being predicted *more often* in a sense, and covering most of their true pixels. The best performing classes are clearly **Road**, followed by **Sky** and **Vegetation**. The worst performing classes are **Traffic.Sign** and **Truck**.