Assignment 1
Computer Vision
Theory

Aditya Aggarwal
(2022028)

Answer 1

(a) $H(p,q) = E_p[-\log q_c(x)]$

$\Rightarrow H(y,q) = -\sum_{i=1}^{K} y_i \log q_i$

let y' be smoothened label such that

$$y' = 1 - \varepsilon + \frac{\varepsilon}{K} \quad \text{for } i = c$$

$$y' = \frac{\varepsilon}{K} \quad \text{for } i \neq c$$

$\Rightarrow H(y',q) = -\sum_{i=1}^{K} y'_i \log q_i$

$$\boxed{\text{Cross Entropy} = -\left((1-\varepsilon+\frac{\varepsilon}{K})\log q_c + \sum_{j\neq c}^{K} \frac{\varepsilon}{K} \log q_j\right)}$$

(b) Effects of label smoothing

→ Since label smoothing prevents overconfidence in the model, it induces regularization

→ This technique reduces overfitting by curbing the model's confidence in wrong predictions. This leads to a better performance on unseen data (generalizable model)

→ Effect: The model tends to distribute mass probability over classes instead of overconfidence over 1 prediction.

Answer 2

(a) Gaussian Distribution: $p(x) = N(\mu_p, \sigma_p^2)$

$$= \frac{1}{\sqrt{2\pi\sigma_p}} \exp\left(\frac{x-\mu_p}{2\sigma_p^2}\right)$$

$$\boxed{H(p,q) = \underset{x\sim p}{E}\left[-\log\left(q(x)\right)\right]}$$

$$= \int_{-\infty}^{\infty} -p(x) \log q(x)\, dx$$

(b) $H(p_q) = \int_{-\infty}^{\infty} p(x) \log\left[\frac{1}{\sqrt{2\pi\sigma_q^2}} \exp\left(\frac{x-\mu_q}{2\sigma_q^2}\right)\right] dx$

$$= -\int_{-\infty}^{\infty} p(x)\left[\log\left(\frac{1}{\sqrt{2\pi\sigma_q^2}}\right) - \frac{(x-\mu_q)}{2\sigma_q^2}\right] dx$$

$$= \int_{-\infty}^{\infty} \left(p(x)\left(\log\sqrt{2\pi\sigma_q^2}\right) + p(x)\,\frac{(x-\mu_q)}{2\sigma_q^2}\right) dx$$

$$= \log\sqrt{2\pi\sigma_q^2} + \frac{1}{2\sigma_q^2}\int_{-\infty}^{\infty} p(x)\,(x-\mu_q)\, dx$$

$$\Rightarrow \boxed{H(p,q) = \log\left(\sqrt{2\pi\sigma_q^2}\right) + \frac{\sigma_p^2 + (\mu_p-\mu_q)^2}{2\sigma_q^2}}$$

(c) if $\sigma_q = \sigma_p = \sigma$

$$\boxed{H(p,q) = \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2} + \frac{(\mu_p-\mu_q)^2}{2\sigma^2}}$$

Date...../...../.....

Page.....

Interpretation : When variance of distribution is equal the first half of the equation represents entropy of the Gaussian Distribution with variance σ^2. The second half is a penalizing term that increases with increase in difference of means of distributions p & q.

Answer 3

(a) let the receptive field be denoted by R_F for 1D conv, kernel size K & dilation factor r,

$$R_F = (K-1)r + 1$$

for L layers of stacked convolutions,

$$R_F \approx L(K-1)r + 1$$

as the dilation factor increases, $r \in \{1, 2, 4 \ldots\}$

$$R_F = \sum_{l=1}^{K} \left((K-1) \prod_{i}^{l} r_i \right)$$

$$\prod_{i=1}^{l} r_i = 1 \cdot 2 \cdot 4 \cdots 2^{l-1} = 2^{\sum_{i=1}^{l-1}} = O(2^l)$$

$$\Rightarrow R_F = O(2^l)$$ where l is the l^{th} layer

Thus receptive field increases exponentially with increasing layers.

(b) for 2 Dimensional conv,

$$R_{dim} = 1 + (K-1) \sum_{l=1}^{1} \prod_{i=1}^{l} r_i$$

Since there are 2 dimensions, $R_F = (R_{dim})^2$

$$\Rightarrow R_F = O(2^l) \cdot O(2^l) = O(2^{2l})$$

Date...../...../.....

Page.....

Thus, we generalized the exponential relation of ~~as~~ Rf to layers for 2D convolutions.

Answer 3 (c)

Comparing Standard Conv & Dilated Conv.

let us assume a K×K kernel is used for a feature map with spatial size N×N.

complexity in standard convolution = $O(N^2K^2)$

In dilate convolution, elements are sparsely distributed. However, number of elements remain K×K. Thus, complexity of conv. with dilation is $O(N^2K^2)$, same as standard convolution.

We can thus conclude that dilation does not introduce any performance overheads.

CSE 344: Computer Vision

Assignment 1: Report

Aditya Aggarwal 2022028

---

1. Classification

1.1.a Dataset Class and WandB

```python
import os
import pandas as pd
from torchvision.io import read_image
from torch.utils.data import Dataset
from sklearn.model_selection import train_test_split

class CustomImageDataset(Dataset):
    def __init__(self, img_dir, images, labels, transform=None, target_transform=None):
        self.img_dir = img_dir
        self.images = images
        self.labels = labels
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        image_path = os.path.join(self.img_dir, self.images[idx])
        image = read_image(image_path)
        label = self.labels[idx]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

Input Parameters

img_dir: root directory path to all images

images: list of paths of images

labels: list of paths of labels

transform: transformations applied to the images

target_transform: transformations applied on the labels.

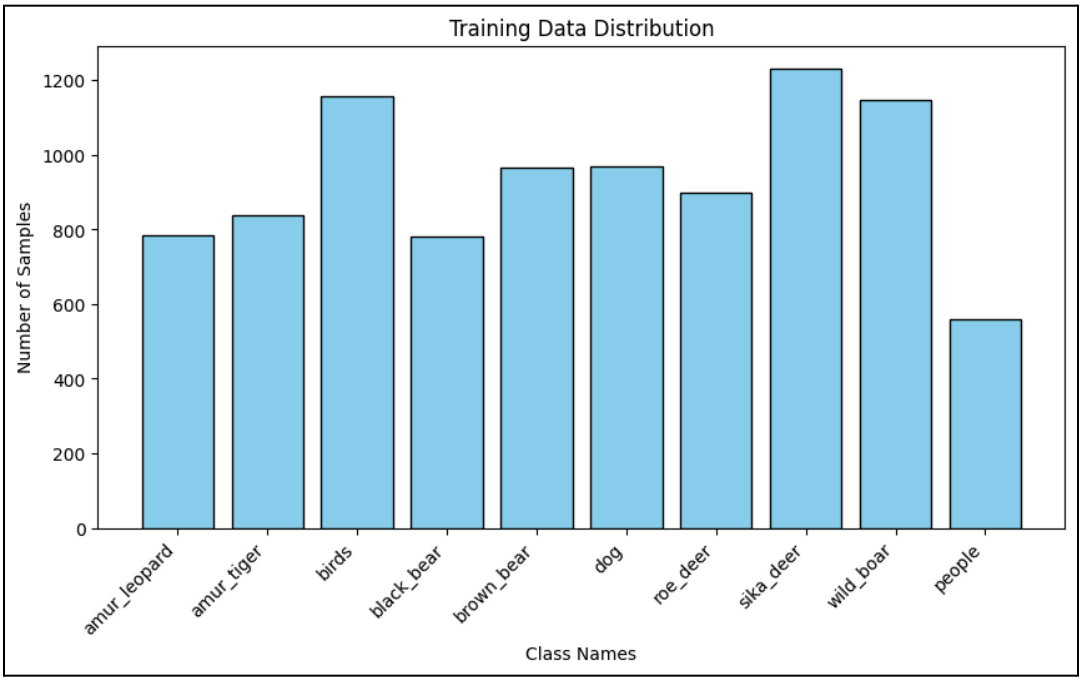
```python
def model_pipeline(model_name, config, is_train=True, is_test=True):
    wandb.login()
    with wandb.init(project=model_name, config=config):
        config = wandb.config
```

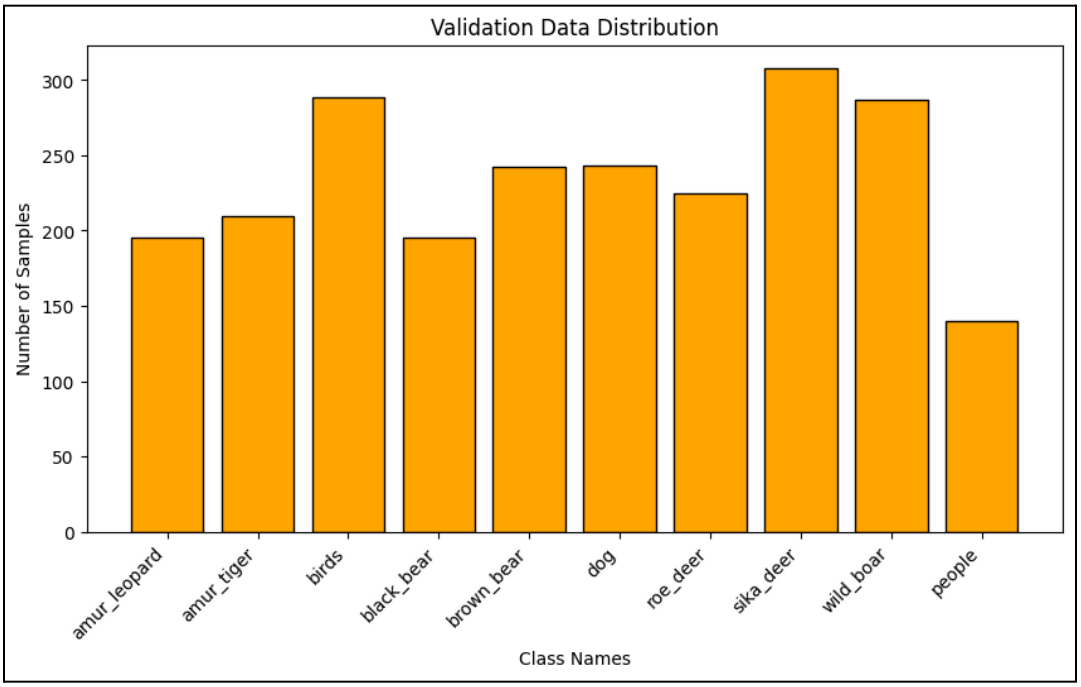
```
Tracking run with wandb version 0.19.5

Run data is saved locally in c:\Users\Aditya\Desktop\CV\2022028_HW1\Classification\wandb\run-20250221_182848-jaqmxead

Syncing run expert-blaze-4 to Weights & Biases (docs)

View project at https://wandb.ai/aditya22028-indraprastha-institute-of-information-techno/convnet

View run at https://wandb.ai/aditya22028-indraprastha-institute-of-information-techno/convnet/runs/jaqmxead
```

1.1.b Dataloaders

```python
if model_name == "convnet":
    train_dataset = CustomImageDataset(img_dir, train_images, train_labels, transform=transform)
    model_class = ConvNet
    weight_dir = "weights/convnet.pth"
elif model_name == "resnet":
    train_dataset = CustomImageDataset(img_dir, train_images, train_labels, transform=transform)
    model_class = ResNet18
    weight_dir = "weights/resnet.pth"

elif model_name == "resnet_aug":
    train_dataset = CustomImageDataset(img_dir, train_images, train_labels, transform=transform_aug)
    model_class = ResNet18
    weight_dir = "weights/resnet_aug.pth"
else:
    raise ValueError(f"Unknown model name: {model_name}")
train_dataloader = DataLoader(train_dataset, config.batch_size, shuffle=True)

val_dataset = CustomImageDataset(img_dir, val_images, val_labels, transform=transform)
val_dataloader = DataLoader(val_dataset, config.batch_size, shuffle=False)
```

1.1.c Visualizing the Data Distribution





```
Training Data Distribution:
  amur_leopard: 783
  amur_tiger: 839
  birds: 1157
  black_bear: 780
  brown_bear: 967
  dog: 970
  roe_deer: 899
  sika_deer: 1231
  wild_boar: 1148
  people: 560
```

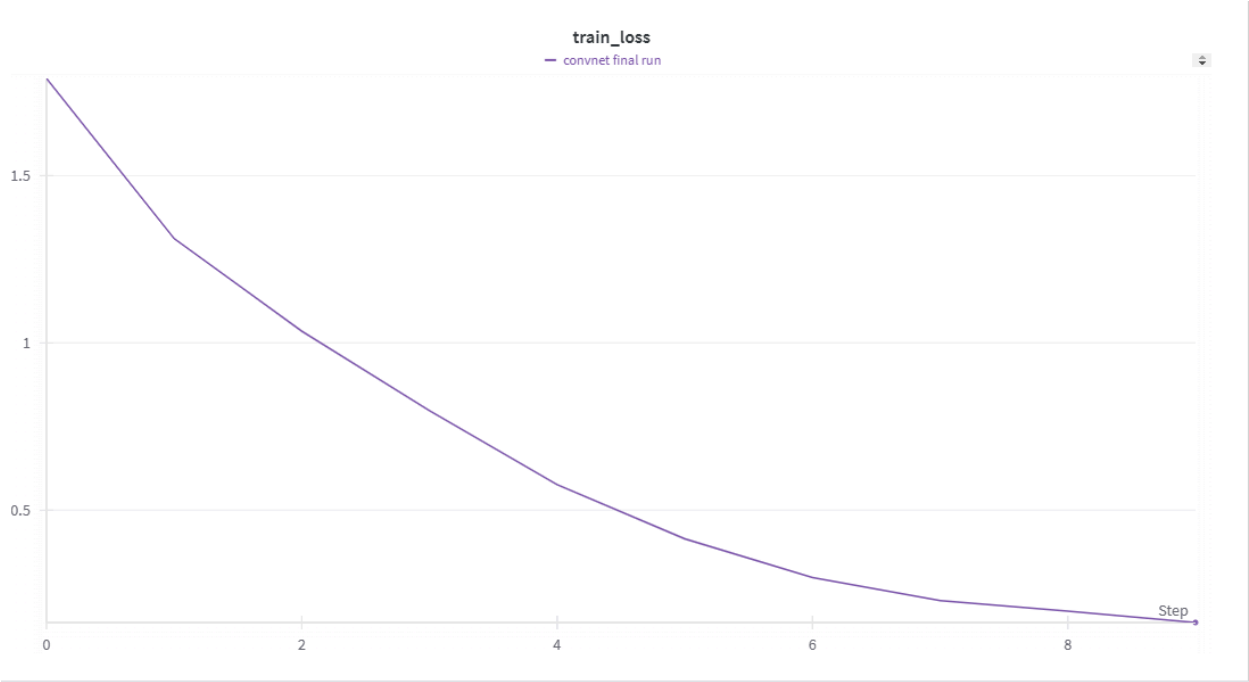
```
Validation Data Distribution:
  amur_leopard: 195
  amur_tiger: 210
  birds: 289
  black_bear: 195
  brown_bear: 242
  dog: 243
  roe_deer: 225
  sika_deer: 308
  wild_boar: 287
  people: 140
```

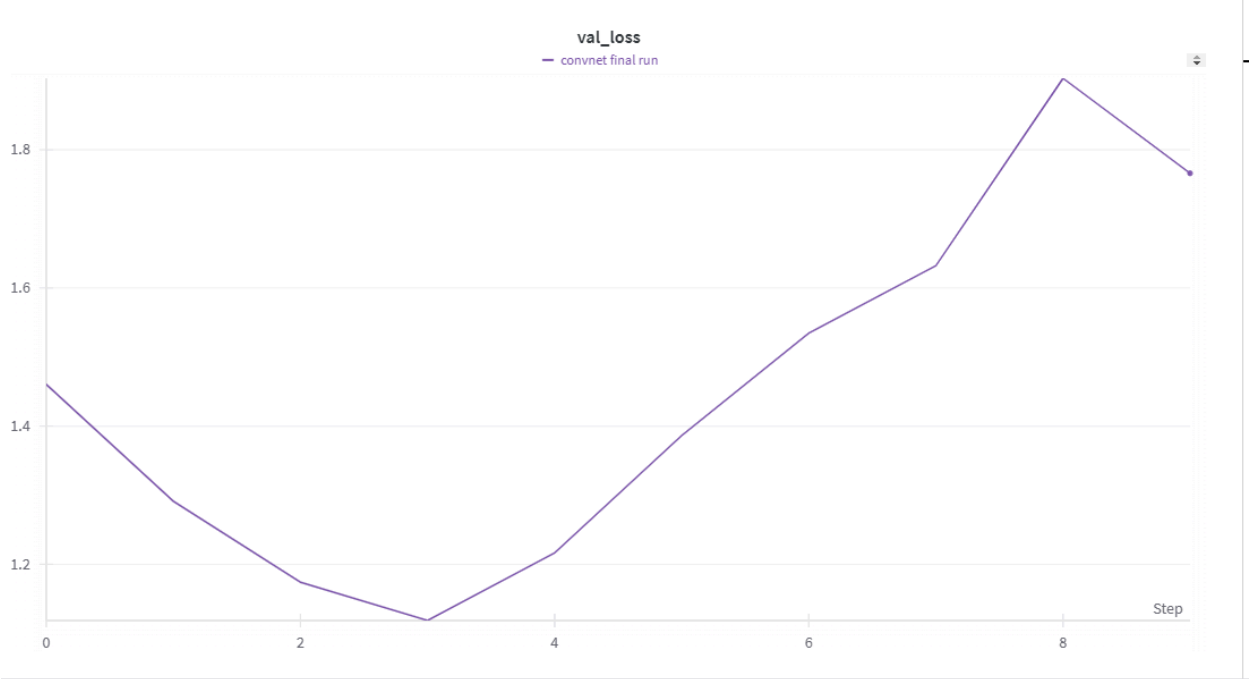
1.2.a Model Class: ConvNet

```python
class ConvNet(nn.Module):
    def __init__(self, num_classes: int=10):
        super(ConvNet, self).__init__()

        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=4, stride=4))
        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer3 = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(14*14*128, num_classes)

    def forward(self, x):
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = torch.flatten(x, start_dim=1)
        x = self.fc(x)
        return x
```

```python
class ResNet18(nn.Module):
    def __init__(self, num_classes: int=10):
        super(ResNet18, self).__init__()
        self.model = models.resnet18(weights='DEFAULT')
        in_features = self.model.fc.in_features
        self.model.fc = torch.nn.Linear(in_features, num_classes)
    def forward(self, x):
        return self.model(x)
```

1.2.b Wandb Plots for ConvNet





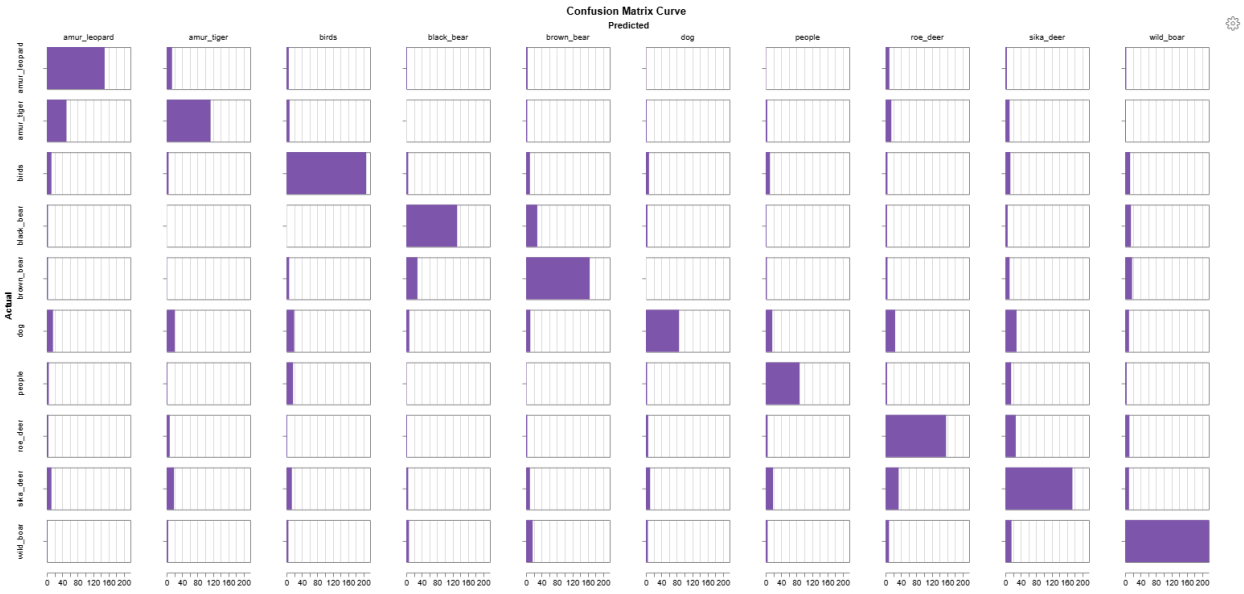
1.2.c Overfitting

The ConvNet model is indeed overfitting. The train loss continuously decreases, indicating that the model is learning, but the val loss increases after the third epoc,h suggesting that the model is not able to generalize well on unseen data. Thus, it is overfitting.

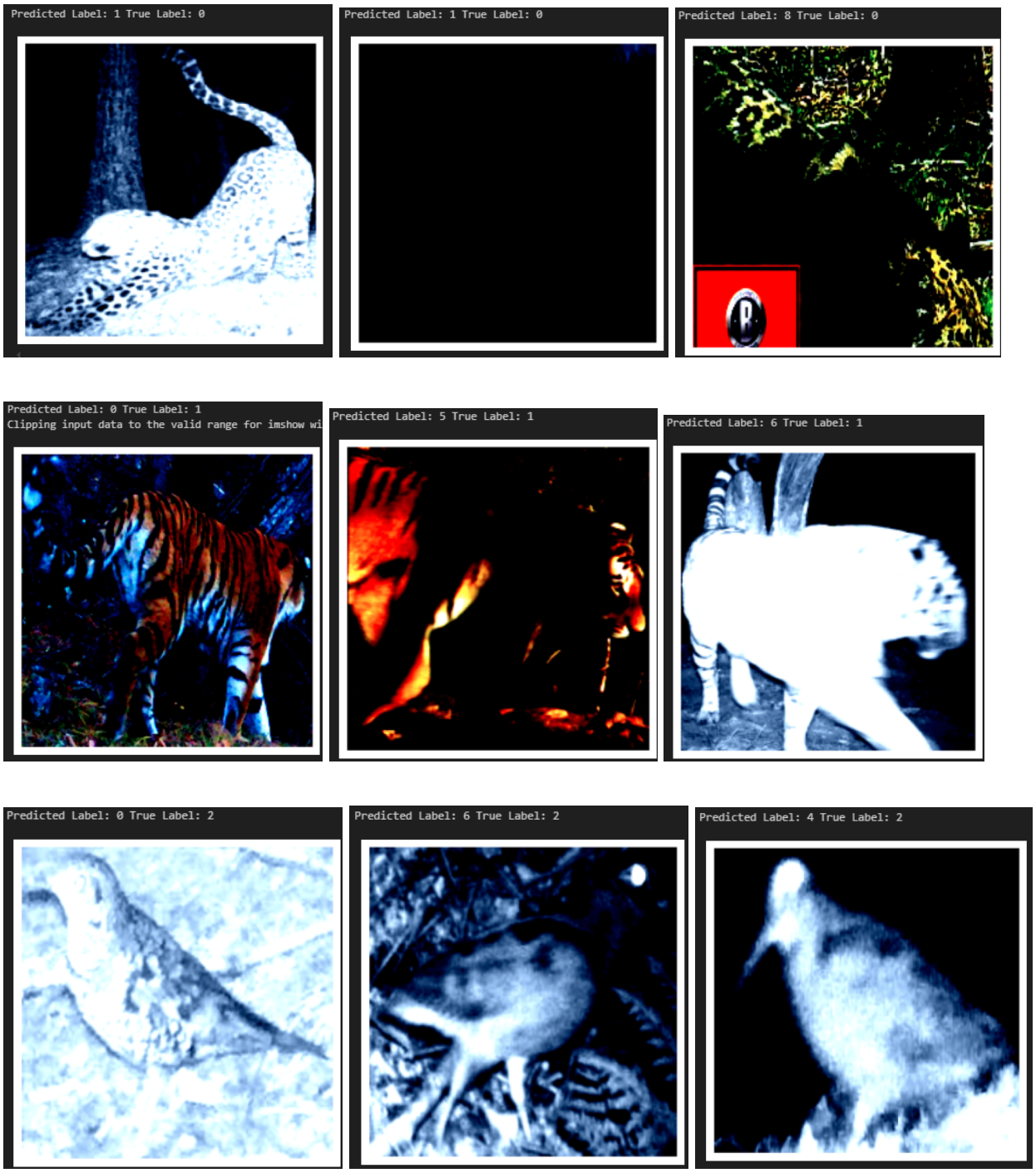
1.2.d Evaluating Convnet

```
Training convnet:
EPOCH: 1 | TRAIN LOSS: 1.790787265416716 | VAL LOSS: 1.4603627828215573
EPOCH: 2 | TRAIN LOSS: 1.3120989144320965 | VAL LOSS: 1.2914336889139484
EPOCH: 3 | TRAIN LOSS: 1.0352926837164251 | VAL LOSS: 1.1739790844120526
EPOCH: 4 | TRAIN LOSS: 0.7977687742665669 | VAL LOSS: 1.1186996357197831
EPOCH: 5 | TRAIN LOSS: 0.5763698423958601 | VAL LOSS: 1.2164395466357496
EPOCH: 6 | TRAIN LOSS: 0.41437104426105653 | VAL LOSS: 1.3862760489377184
EPOCH: 7 | TRAIN LOSS: 0.298618042522665 | VAL LOSS: 1.5345128968728334
EPOCH: 8 | TRAIN LOSS: 0.2300046780187567 | VAL LOSS: 1.6322748613030664
EPOCH: 9 | TRAIN LOSS: 0.19801280400227858 | VAL LOSS: 1.903397056008366
EPOCH: 10 | TRAIN LOSS: 0.1642746450892347 | VAL LOSS: 1.765878727115314
weights saved at weights/convnet.pth
Evaluation of convnet on VAL SET:
confusion matrix saved to wandb
ACCURACY: 0.6384
F1 SCORE: 0.6332
```

1.2.d Confusion Matrix

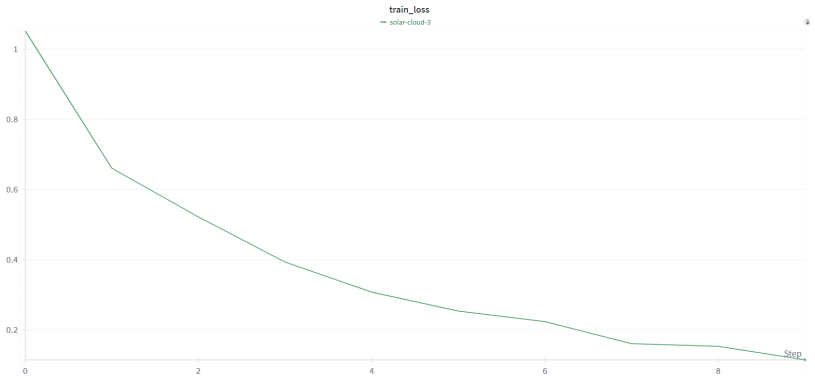


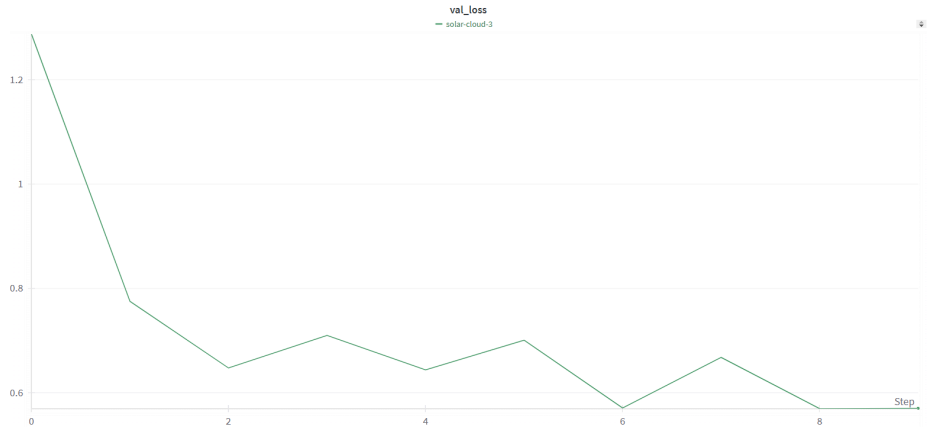
1.2.e Visualizing Misclassified Images



These images were misclassified because they are either blurry, occluded (very low brightness and visibility),cropped or high contrast brightness. Top right image contains a portion which is unrelated to the rest of the image.

1.3.a Wandb Plots for ResNet



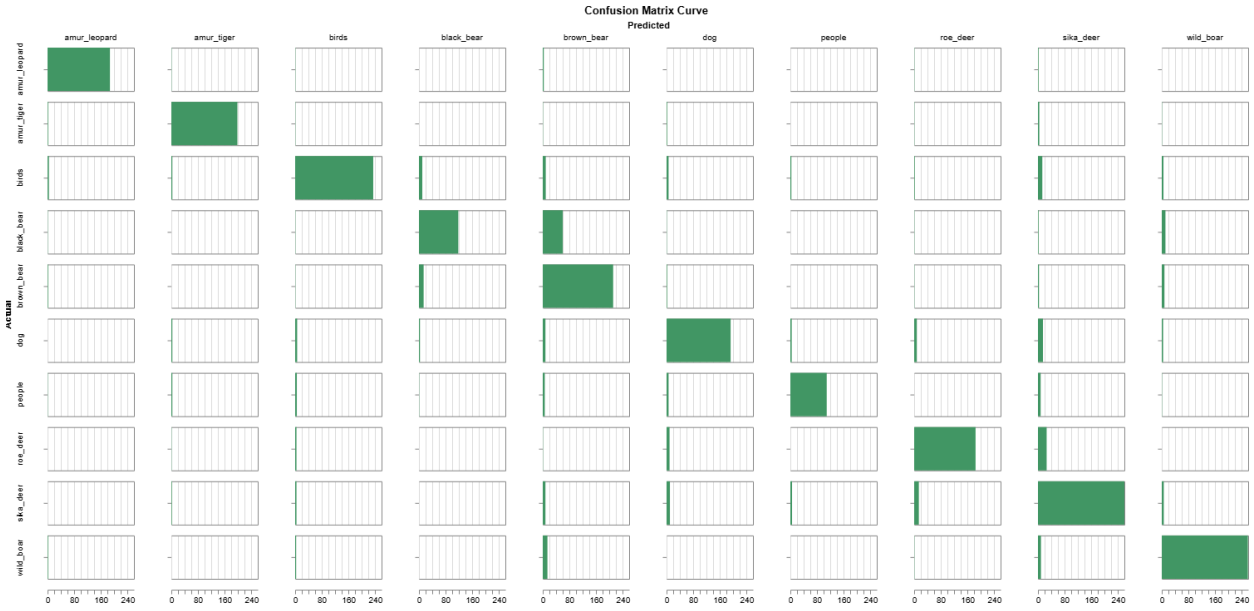


1.3.b Overfitting

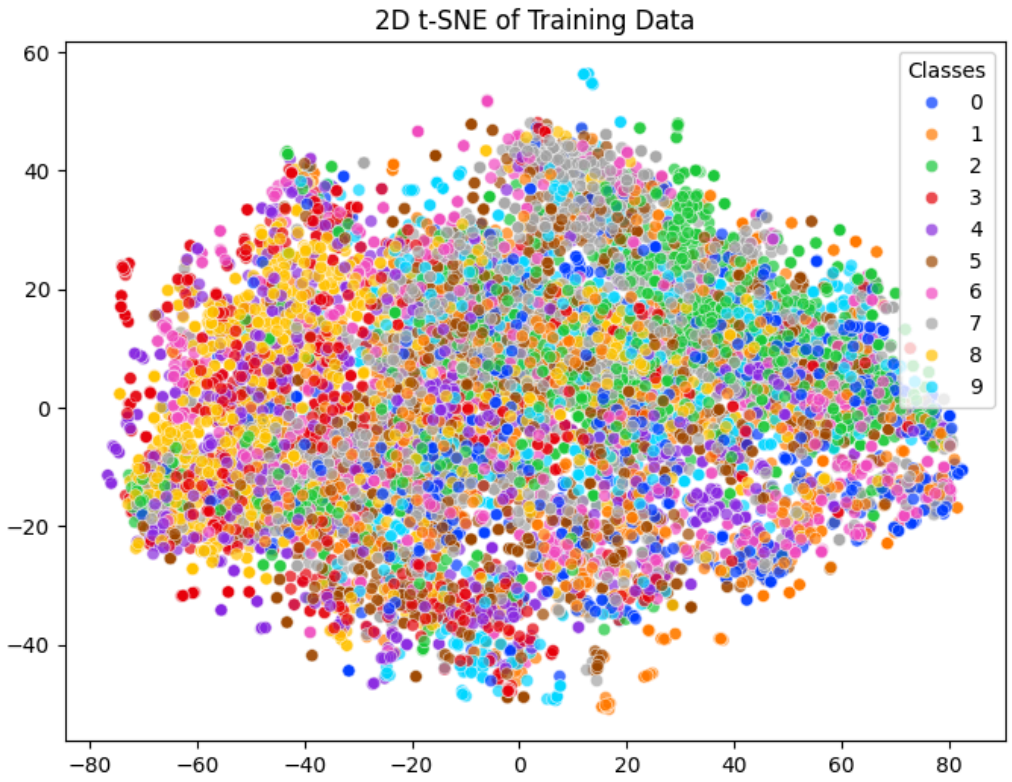
The val loss fluctuates after the initial drop. Overall, the val loss shows a decreasing trend, indicating better performance than the ConvNet model. However, the constant fluctuation means that the model is struggling to move towards a stable convergence, which may indicate overfitting. This may also be happening due to the large learning rate (1e-3).

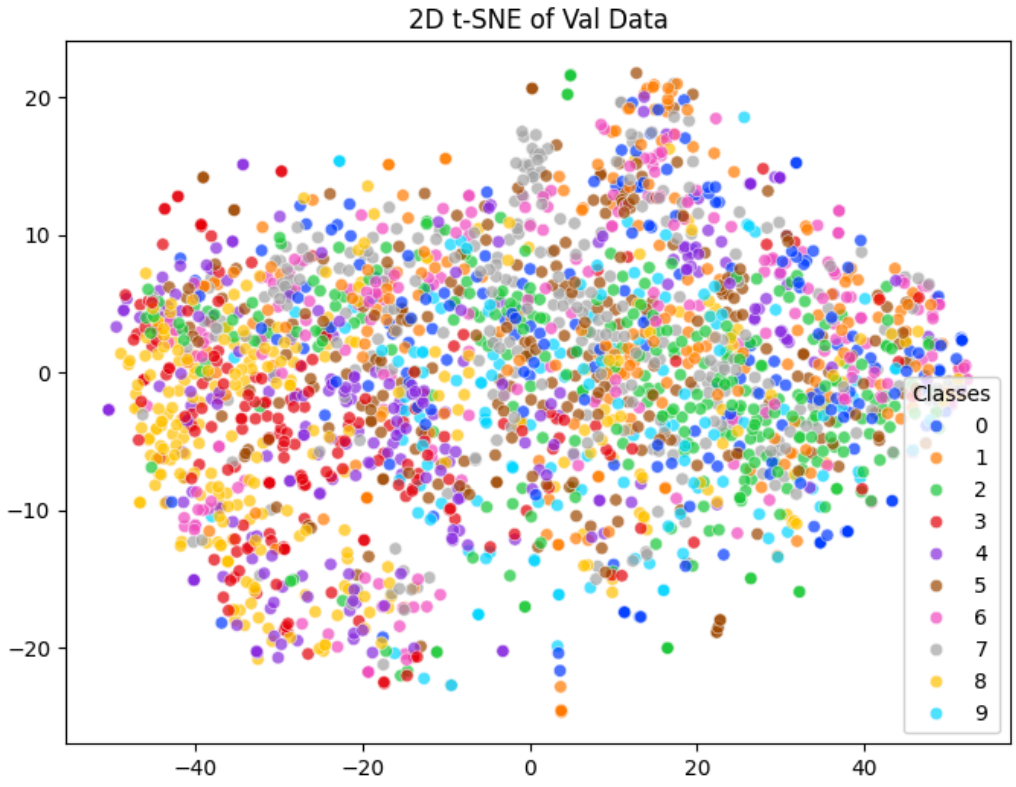
1.3.c Evaluating ResNet and Confusion Matrix

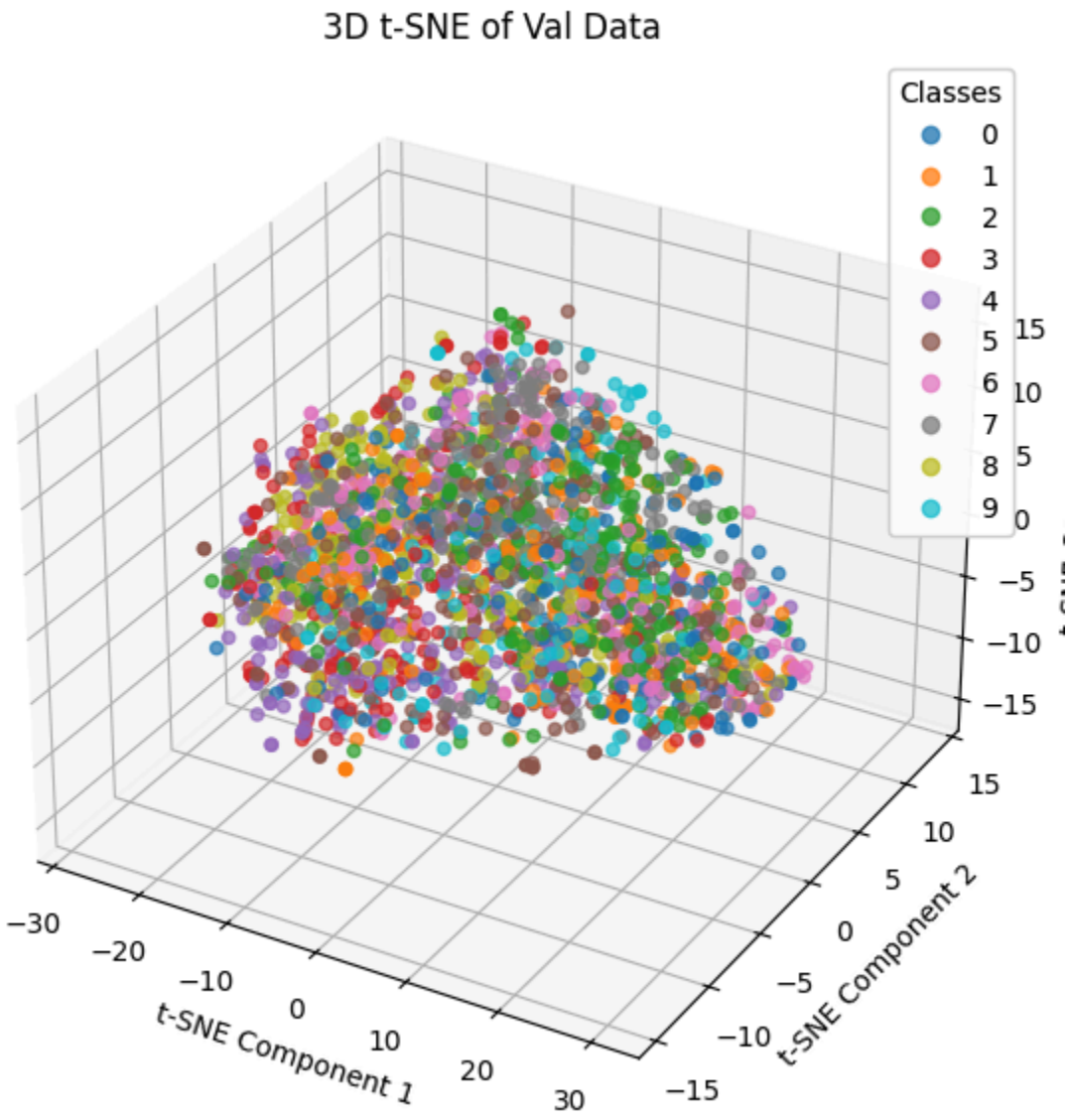
```
Training resnet:
EPOCH: 1 | TRAIN LOSS: 1.0525247297553657 | VAL LOSS: 1.2877112165310491
EPOCH: 2 | TRAIN LOSS: 0.660957632175676 | VAL LOSS: 0.775107585117157
EPOCH: 3 | TRAIN LOSS: 0.522104720327822 | VAL LOSS: 0.6473099642976697
EPOCH: 4 | TRAIN LOSS: 0.3935075347597757 | VAL LOSS: 0.7096586214137466
EPOCH: 5 | TRAIN LOSS: 0.3077198980271957 | VAL LOSS: 0.6436032163044685
EPOCH: 6 | TRAIN LOSS: 0.25372602057668975 | VAL LOSS: 0.7006744481399719
EPOCH: 7 | TRAIN LOSS: 0.22389102553961132 | VAL LOSS: 0.5705081117183812
EPOCH: 8 | TRAIN LOSS: 0.16077161069304063 | VAL LOSS: 0.6675999082861135
EPOCH: 9 | TRAIN LOSS: 0.1534755456580425 | VAL LOSS: 0.5692948704105008
EPOCH: 10 | TRAIN LOSS: 0.11493255886785449 | VAL LOSS: 0.5699010936490674
weights saved at weights/resnet.pth
Evaluation of resnet on VAL SET:
confusion matrix saved to wandb
ACCURACY: 0.8355
F1 SCORE: 0.8355
```



1.3.d TSNE Visualization



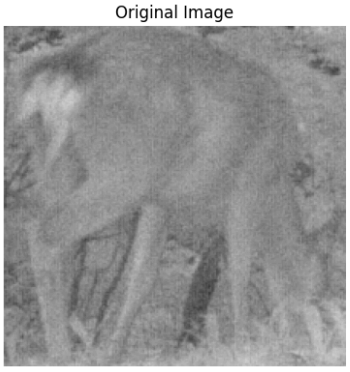




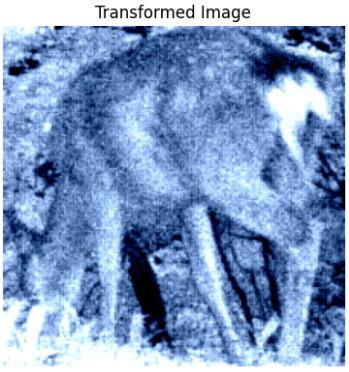
1.4.a Data Augmentation

```
transform_aug = v2.Compose([
    v2.ConvertImageDtype(torch.float32),
    v2.RandomResizedCrop(size=(224, 224), scale=(0.8, 1.0), ratio=(0.9, 1.1), antialias=True),
    v2.RandomHorizontalFlip(p=0.5),
    v2.RandomRotation(degrees=5),
    v2.ColorJitter(brightness=0.05, contrast=0.05, saturation=0.05, hue=0.02),
    v2.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

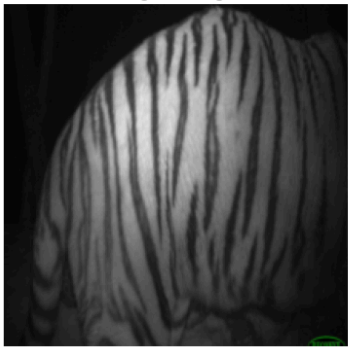
Original Image



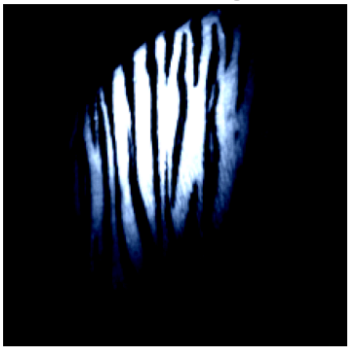
Transformed Image



Original Image

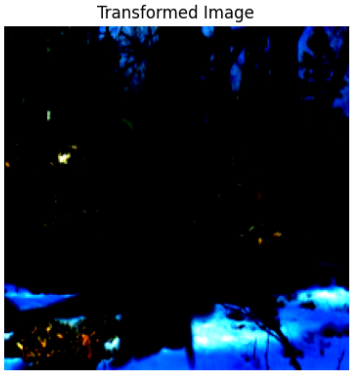


Transformed Image



Original Image

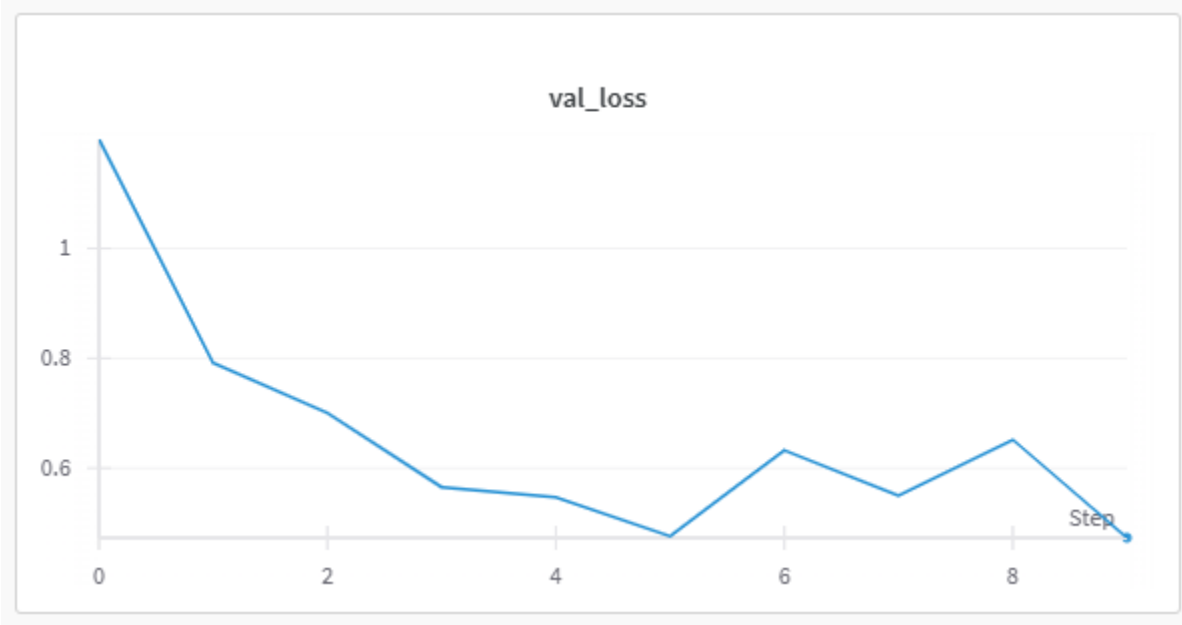
Transformed Image

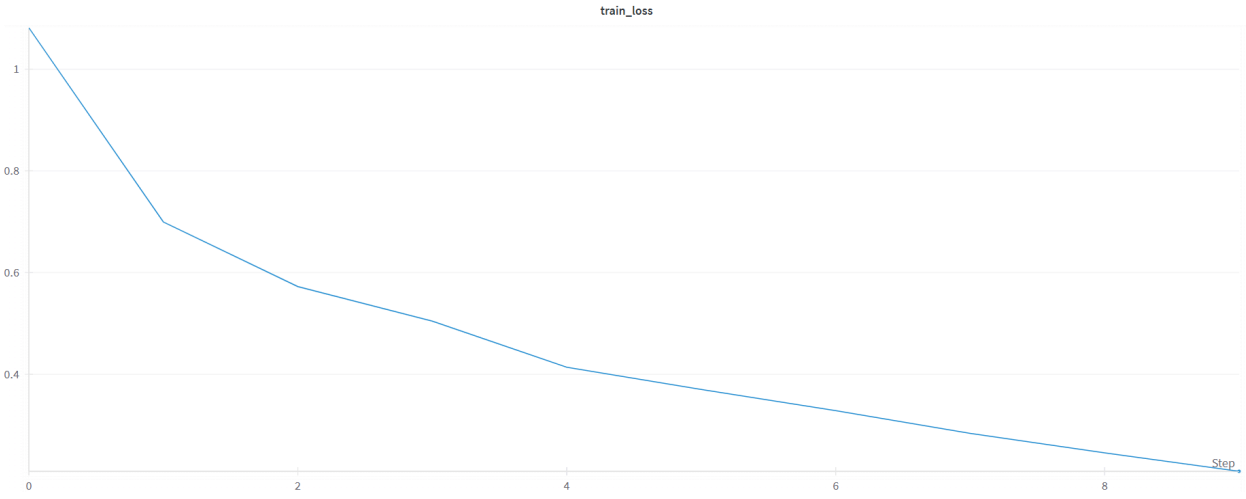


Original Image

Transformed Image

1.4.b Model Training



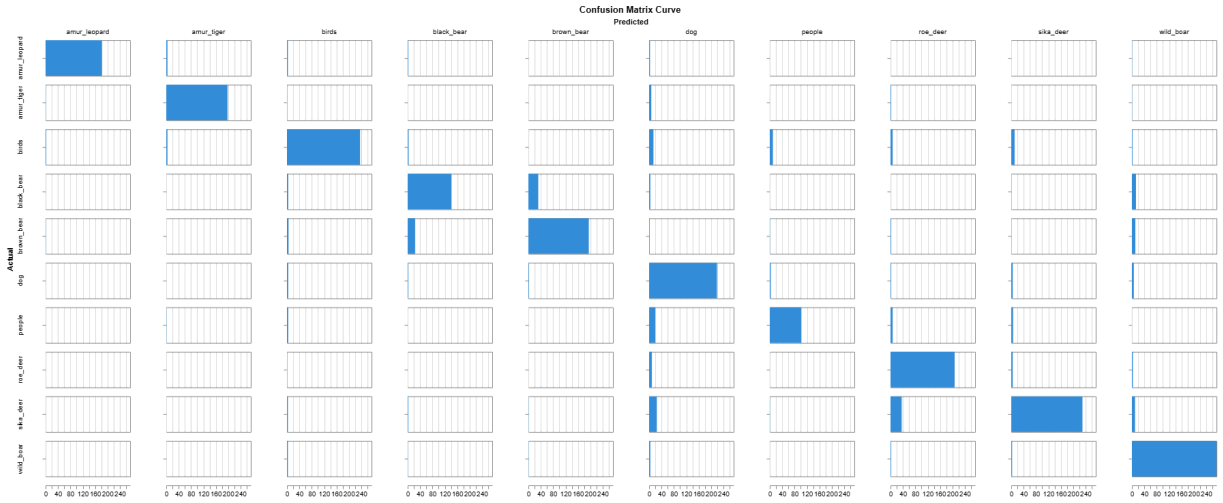


1.4.c Overfitting

The problem of overfitting is not completely solved (a lot of other transformations were tried, and this yielded the best results). However, it can be seen that earlier, the model overfitted at the 3rd epoch, whereas now it overfits towards the end (8th epoch). It is thus partially solved.

1.4.d Evaluation Metric and Confusion Matrix

```
Evaluation of resnet_aug on VAL SET:
confusion matrix saved to wandb
ACCURACY: 0.8496
F1 SCORE: 0.8492
```



1.5 Comparison of Models

- ResNet Aug achieves the highest accuracy (84.96%) and F1-score (84.92%), outperforming both ConvNet and vanilla ResNet. ResNet Aug's validation loss decreases consistently and only slightly increases at the 8th epoch, suggesting better generalization.
- ResNet (83.55% accuracy, 83.55% F1-score) is a significant improvement over ConvNet (63.84% accuracy, 63.32% F1-score), showing the benefits of deeper architectures. ResNet's validation loss fluctuates, suggesting that while it learns well, it may overfit or be sensitive to noise.
- ConvNet underperforms significantly, suggesting it lacks the capacity to learn complex features compared to ResNet-based models.ConvNet's validation loss increases after the 3rd epoch, indicating early overfitting. This suggests it fails to generalize.

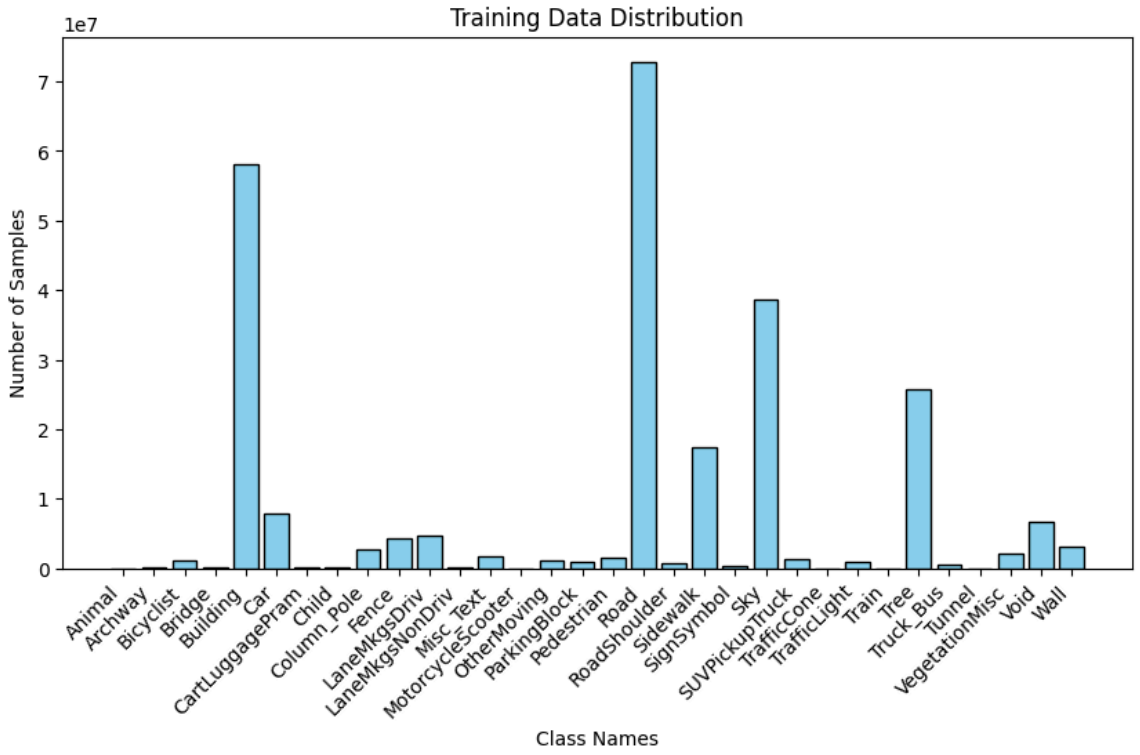
2. Segmentation

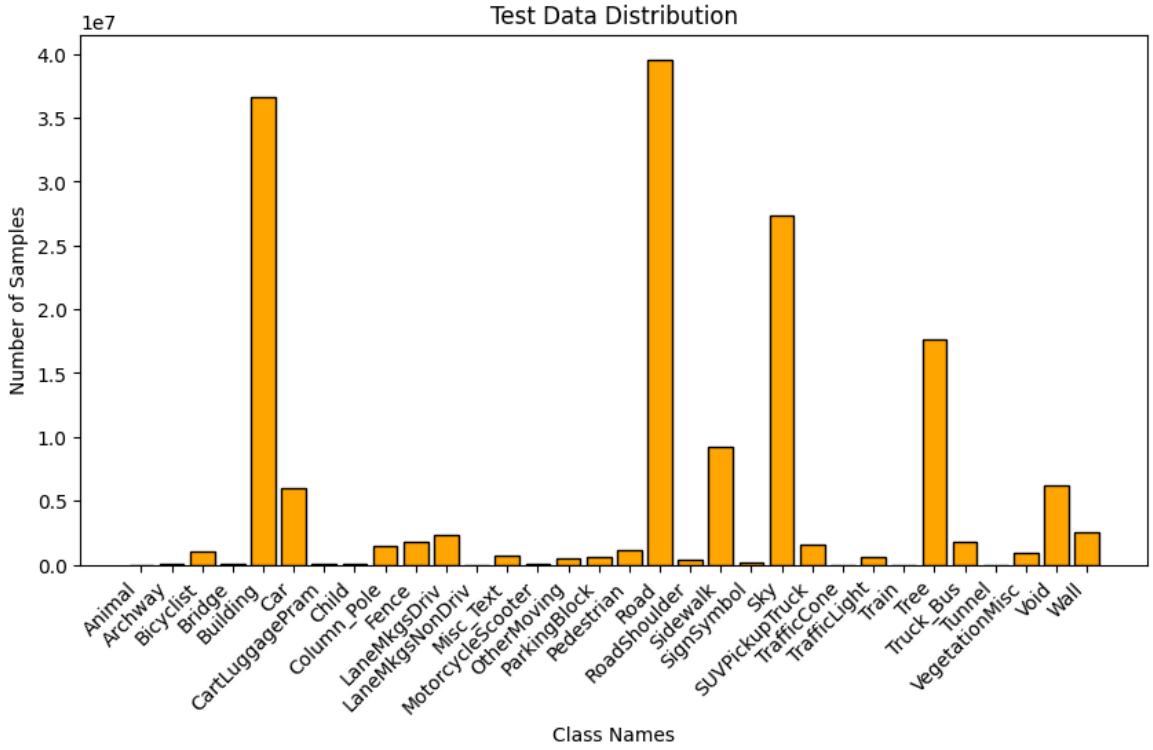
2.1.a Dataset Class

```
class CamVidDataset(Dataset):
    def __init__(self, img_dir, label_dir, class_dict_dir, transform=None, target_transform=None):
        self.img_dir = img_dir
        self.label_dir = label_dir
        self.images = []
        self.labels = []
        self.transform = transform
        self.target_transform = target_transform

        df = pd.read_csv(class_dict_dir)
        self.class_dict = {(item['r'], item['g'], item['b']): index for index, item in df.iterrows()}
        try:
            for img in os.listdir(img_dir):
                self.images.append(img)
            for label in os.listdir(label_dir):
                self.labels.append(label)

            self.images.sort()
            self.labels.sort()

            if len(self.images) != len(self.labels):
                raise ValueError("Number of images does not match the number of labels.")

        except FileNotFoundError:
            print(f"Directory for label not found")

        except ValueError as e:
            print(f'Error: {e}')
```

```
def label_encode(self, label):
    (_, height, width) = label.shape
    encoded_label = torch.zeros((height, width), dtype=torch.long)

    for (rgb, label_num) in self.class_dict.items():
        mask = torch.all(label == torch.tensor(rgb).view(-1, 1, 1), dim=0)
        encoded_label[mask] = label_num

    return encoded_label


def __len__(self):
    return len(self.images)

def __getitem__(self, idx):
    img_path = os.path.join(self.img_dir, self.images[idx])
    label_path = os.path.join(self.label_dir, self.labels[idx])

    image = read_image(img_path)
    label = read_image(label_path)

    if self.transform:
        image = self.transform(image)
        label = self.target_transform(label)

    label = self.label_encode(label)
    return image, label
```

2.1.b Classwise Pixel Distribution





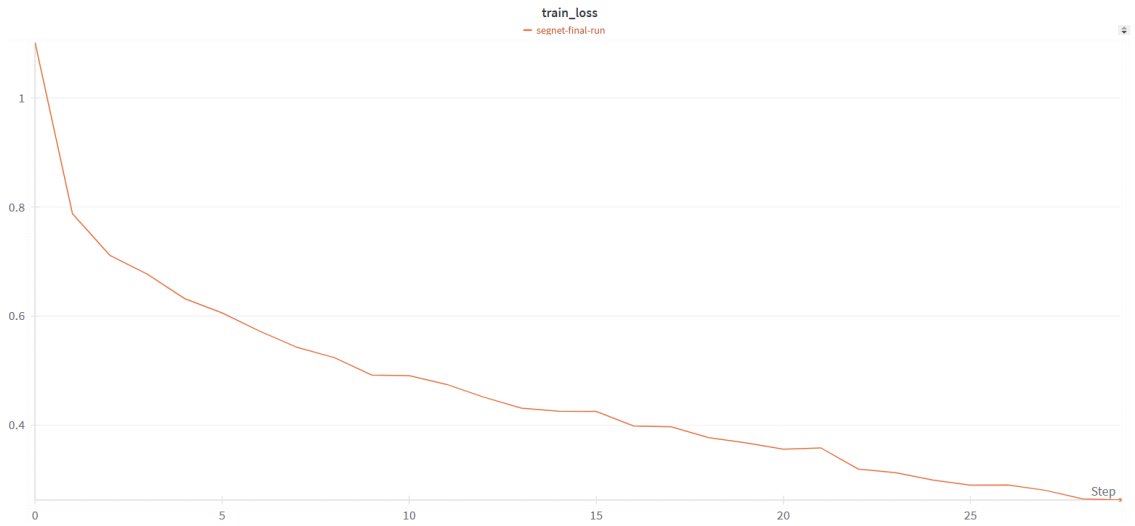
2.1.c Visualize 2 Images and Labels for Each Class

2.2.a Model Class and Wandb plots

```
class SegNet_Decoder(nn.Module):
    def __init__(self, in_chn=3, out_chn=32, BN_momentum=0.5):
        super(SegNet_Decoder, self).__init__()
        self.in_chn = in_chn
        self.out_chn = out_chn

        #implement the architecture.
        self.MaxDec = nn.MaxUnpool2d(kernel_size=2,stride=2)

        # stage 5:
        # Max Unpooling: Upsample using ind5 to size4
        # Channels: 512 → 512 → 512 (3 convolutions)
        # Batch Norm: Applied after each convolution
        # Activation: ReLU after each batch norm
        self.ConvDec51 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
        self.BNDec51 = nn.BatchNorm2d(512, momentum=BN_momentum)
        self.ConvDec52 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
        self.BNDec52 = nn.BatchNorm2d(512, momentum=BN_momentum)
        self.ConvDec53 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
        self.BNDec53 = nn.BatchNorm2d(512, momentum=BN_momentum)

        #stage 4:
        # Max Unpooling: Upsample using ind4 to size3
        # Channels: 512 → 512 → 256 (3 convolutions)
        # Batch Norm: Applied after each convolution
        # Activation: ReLU after each batch norm
        self.ConvDec41 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
        self.BNDec41 = nn.BatchNorm2d(512, momentum=BN_momentum)
        self.ConvDec42 = nn.Conv2d(512, 512, kernel_size=3, padding=1)
        self.BNDec42 = nn.BatchNorm2d(512, momentum=BN_momentum)
        self.ConvDec43 = nn.Conv2d(512, 256, kernel_size=3, padding=1)
        self.BNDec43 = nn.BatchNorm2d(256, momentum=BN_momentum)
```

```
# Stage 3:
# Max Unpooling: Upsample using ind3 to size2
# Channels: 256 → 256 → 128 (3 convolutions)
# Batch Norm: Applied after each convolution
# Activation: ReLU after each batch norm
self.ConvDec31 = nn.Conv2d(256, 256, kernel_size=3, padding=1)
self.BNDec31 = nn.BatchNorm2d(256, momentum=BN_momentum)
self.ConvDec32 = nn.Conv2d(256, 256, kernel_size=3, padding=1)
self.BNDec32 = nn.BatchNorm2d(256, momentum=BN_momentum)
self.ConvDec33 = nn.Conv2d(256, 128, kernel_size=3, padding=1)
self.BNDec33 = nn.BatchNorm2d(128, momentum=BN_momentum)

# Stage 2:
# Max Unpooling: Upsample using ind2 to size1
# Channels:  128 → 64 (2 convolutions)
# Batch Norm: Applied after each convolution
# Activation: ReLU after each batch norm
self.ConvDec21 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
self.BNDec21 = nn.BatchNorm2d(128, momentum=BN_momentum)
self.ConvDec22 = nn.Conv2d(128, 64, kernel_size=3, padding=1)
self.BNDec22 = nn.BatchNorm2d(64, momentum=BN_momentum)

# Stage 1:
# Max Unpooling: Upsample using ind1
# Channels: 64 → out_chn (2 convolutions)
# Batch Norm: Applied after each convolution
# Activation: ReLU after the first convolution, no activation after the last one
self.ConvDec11 = nn.Conv2d(64, 64, kernel_size=3, padding=1)
self.BNDec11 = nn.BatchNorm2d(64, momentum=BN_momentum)
self.ConvDec12 = nn.Conv2d(64, self.out_chn, kernel_size=3, padding=1)
self.BNDec12 = nn.BatchNorm2d(self.out_chn, momentum=BN_momentum)

#For convolution use kernel size = 3, padding =1
#for max unpooling use kernel size=2 ,stride=2
```

```python
def forward(self,x,indexes,sizes):
    ind1,ind2,ind3,ind4,ind5=indexes[0],indexes[1],indexes[2],indexes[3],indexes[4]
    size1,size2,size3,size4,size5=sizes[0],sizes[1],sizes[2],sizes[3],sizes[4]

    #stage5
    x = self.MaxDec(x, ind5, size4)
    x = F.relu(self.BNDec51(self.ConvDec51(x)))
    x = F.relu(self.BNDec52(self.ConvDec52(x)))
    x = F.relu(self.BNDec53(self.ConvDec53(x)))

    #stage4
    x = self.MaxDec(x, ind4, size3)
    x = F.relu(self.BNDec41(self.ConvDec41(x)))
    x = F.relu(self.BNDec42(self.ConvDec42(x)))
    x = F.relu(self.BNDec43(self.ConvDec43(x)))

    #stage3
    x = self.MaxDec(x, ind3, size2)
    x = F.relu(self.BNDec31(self.ConvDec31(x)))
    x = F.relu(self.BNDec32(self.ConvDec32(x)))
    x = F.relu(self.BNDec33(self.ConvDec33(x)))

    #stage2
    x = self.MaxDec(x, ind2, size1)
    x = F.relu(self.BNDec21(self.ConvDec21(x)))
    x = F.relu(self.BNDec22(self.ConvDec22(x)))

    #stage1
    x = self.MaxDec(x, ind1, torch.Size([size1[0], 32, 360, 480]))
    x = F.relu(self.BNDec11(self.ConvDec11(x)))
    x = self.ConvDec12(x)
    return x
```



2.2.b Evaluation Metrics

```
miou: 0.234375
class: 0 | pixel wise accuracy: 0.999941555914751
class: 1 | pixel wise accuracy: 0.9997877254948915
class: 2 | pixel wise accuracy: 0.9950717642480843
class: 3 | pixel wise accuracy: 0.9994346414032567
class: 4 | pixel wise accuracy: 0.9205309107598978
class: 5 | pixel wise accuracy: 0.9811573076309068
class: 6 | pixel wise accuracy: 0.9996670957056194
class: 7 | pixel wise accuracy: 0.999737812300447
class: 8 | pixel wise accuracy: 0.9904796505826947
class: 9 | pixel wise accuracy: 0.9890616269556194
class: 10 | pixel wise accuracy: 0.9919834570561942
class: 11 | pixel wise accuracy: 0.9999726861829502
class: 12 | pixel wise accuracy: 0.9949067588601532
class: 13 | pixel wise accuracy: 0.9998107988106641
class: 14 | pixel wise accuracy: 0.9957336566091954
class: 15 | pixel wise accuracy: 0.9947935125319285
class: 16 | pixel wise accuracy: 0.9915673890485313
class: 17 | pixel wise accuracy: 0.9710235322876756
class: 18 | pixel wise accuracy: 0.9979791766443167
class: 19 | pixel wise accuracy: 0.975482344548212
class: 20 | pixel wise accuracy: 0.9990353857359515
class: 21 | pixel wise accuracy: 0.9802480693247126
class: 22 | pixel wise accuracy: 0.9894644246886973
class: 23 | pixel wise accuracy: 0.9999899225734356
class: 24 | pixel wise accuracy: 0.9971128671775223
class: 25 | pixel wise accuracy: 1.0
class: 26 | pixel wise accuracy: 0.9543556184147509
class: 27 | pixel wise accuracy: 0.9870752264926564
class: 28 | pixel wise accuracy: 1.0
```

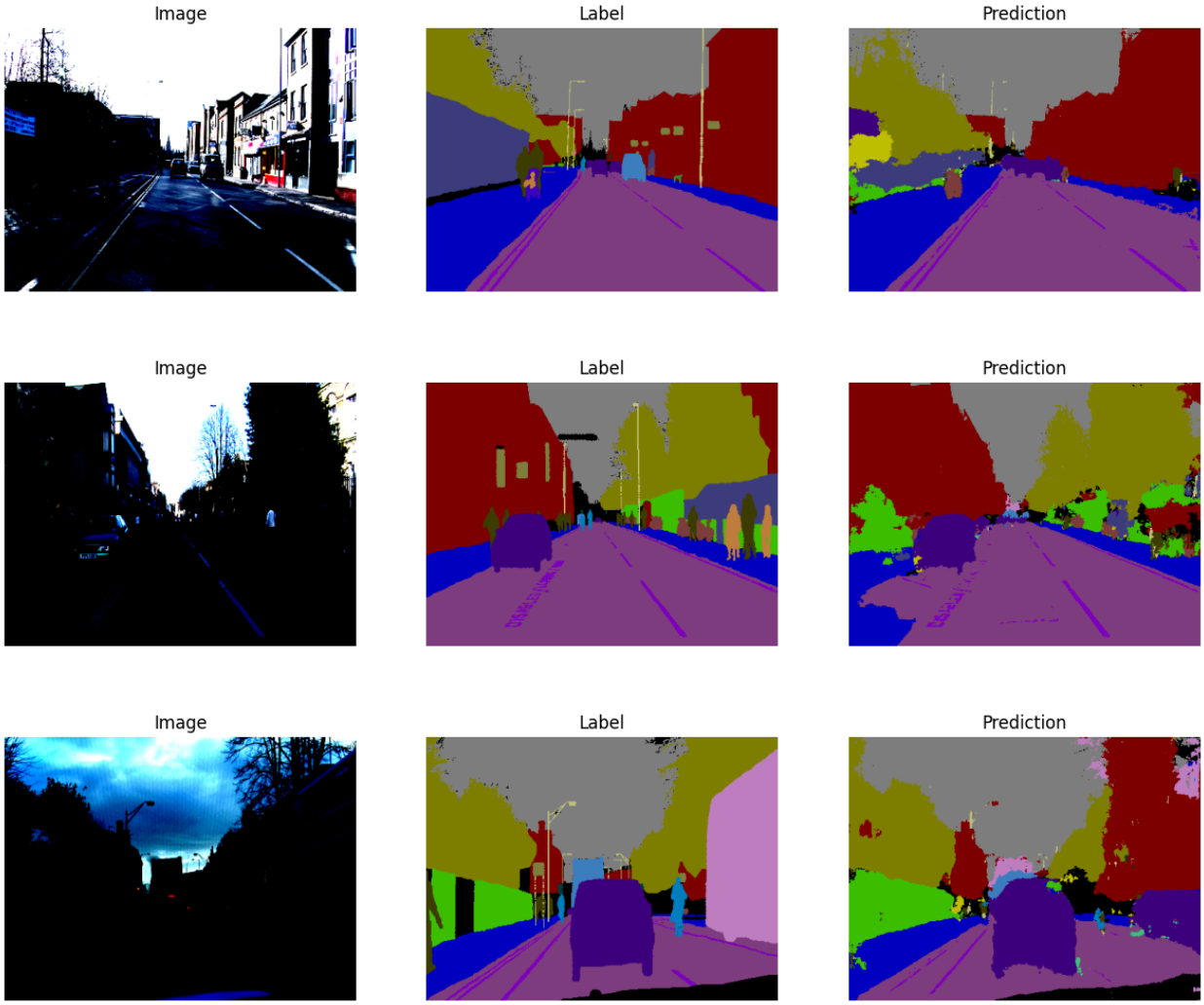
```
class: 29 | pixel wise accuracy: 0.9878914731002554
class: 30 | pixel wise accuracy: 0.96055233776341
class: 31 | pixel wise accuracy: 0.9817069763729247
-------------------------------------------
class: 0 | IoU: 0.0
class: 1 | IoU: 0.0
class: 2 | IoU: 0.3
class: 3 | IoU: 0.0
class: 4 | IoU: 0.7
class: 5 | IoU: 0.6
class: 6 | IoU: 0.0
class: 7 | IoU: 0.0
class: 8 | IoU: 0.1
class: 9 | IoU: 0.3
class: 10 | IoU: 0.5
class: 11 | IoU: 0.0
class: 12 | IoU: 0.0
class: 13 | IoU: 0.0
class: 14 | IoU: 0.2
class: 15 | IoU: 0.1
class: 16 | IoU: 0.1
class: 17 | IoU: 0.9
class: 18 | IoU: 0.1
class: 19 | IoU: 0.7
class: 20 | IoU: 0.2
class: 21 | IoU: 0.9
class: 22 | IoU: 0.0
class: 23 | IoU: 0.0
class: 24 | IoU: 0.3
class: 25 | IoU: 0.0
```

```
class: 26 | IoU: 0.6
class: 27 | IoU: 0.1
class: 28 | IoU: 0.0
class: 29 | IoU: 0.2
class: 30 | IoU: 0.3
class: 31 | IoU: 0.3
-----------------------------------------
class: 0 | Dice Coefficient: 0.0
class: 1 | Dice Coefficient: 0.0
class: 2 | Dice Coefficient: 0.47521654479243186
class: 3 | Dice Coefficient: 0.0
class: 4 | Dice Coefficient: 0.8397072886129567
class: 5 | Dice Coefficient: 0.7608178716920719
class: 6 | Dice Coefficient: 0.0
class: 7 | Dice Coefficient: 0.0
class: 8 | Dice Coefficient: 0.23481578456708274
class: 9 | Dice Coefficient: 0.4338140262128032
class: 10 | Dice Coefficient: 0.6726564553027039
class: 11 | Dice Coefficient: 0.0
class: 12 | Dice Coefficient: 0.08534389306480042
class: 13 | Dice Coefficient: 0.0
class: 14 | Dice Coefficient: 0.3474000702064974
class: 15 | Dice Coefficient: 0.23769210541693364
class: 16 | Dice Coefficient: 0.2477626021904464
class: 17 | Dice Coefficient: 0.9413260542563846
class: 18 | Dice Coefficient: 0.25687501146599406
class: 19 | Dice Coefficient: 0.8004633442598722
class: 20 | Dice Coefficient: 0.30097069828816725
class: 21 | Dice Coefficient: 0.9421956199319688
class: 22 | Dice Coefficient: 0.05123804688921474
```

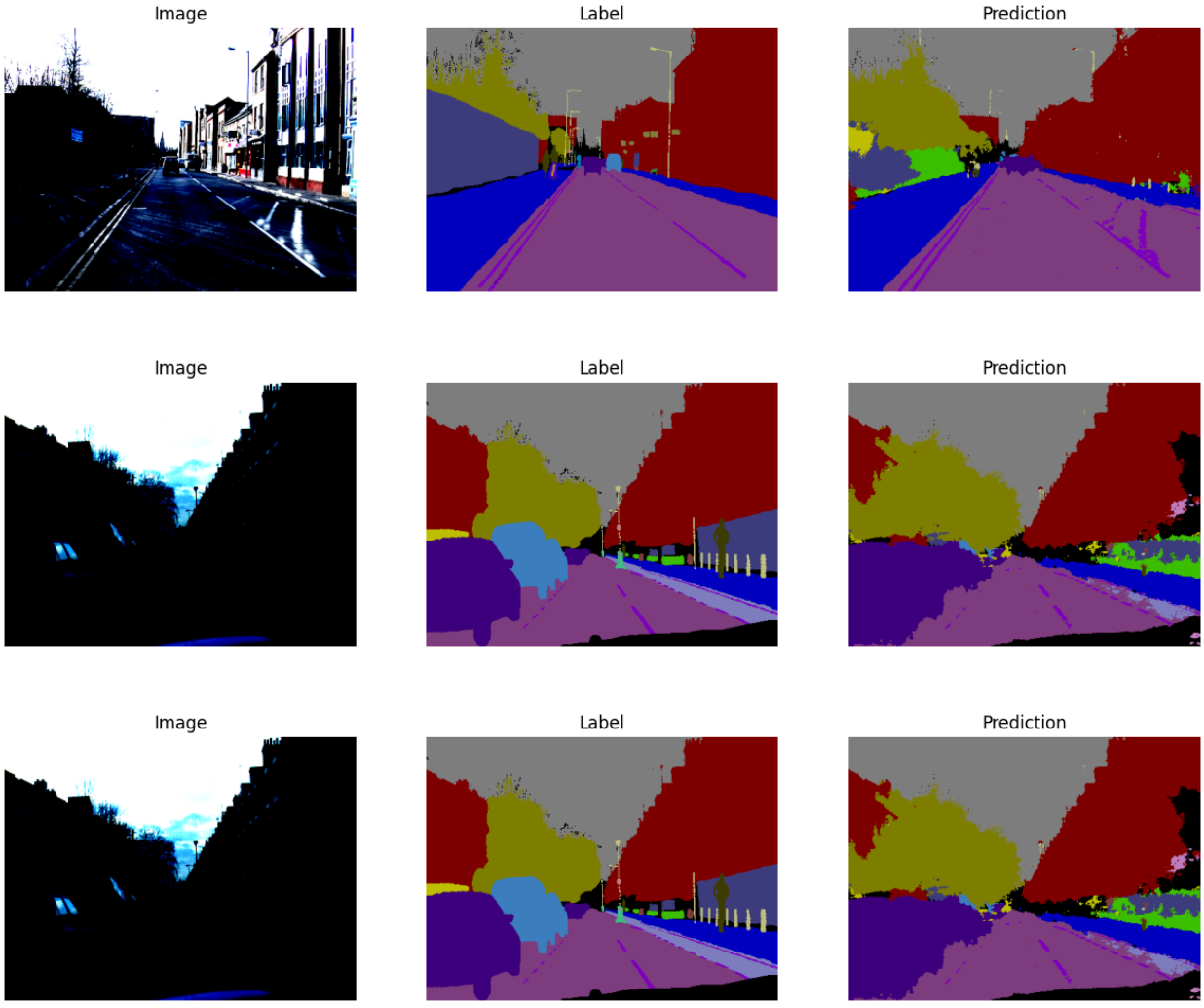
```
class: 23 | Dice Coefficient: 0.0
class: 24 | Dice Coefficient: 0.45888733052828 21
class: 25 | Dice Coefficient: 0.0
class: 26 | Dice Coefficient: 0.78186826803858 98
class: 27 | Dice Coefficient: 0.13872225131110 7596
class: 28 | Dice Coefficient: 0.0
class: 29 | Dice Coefficient: 0.29334440476086 49
class: 30 | Dice Coefficient: 0.45464411226439577
class: 31 | Dice Coefficient: 0.48181373671253 83
-----------------------------------------------
Overall Accuracy: 0.81277785261015 32
```

2.2.c Visualizing Misclassifications

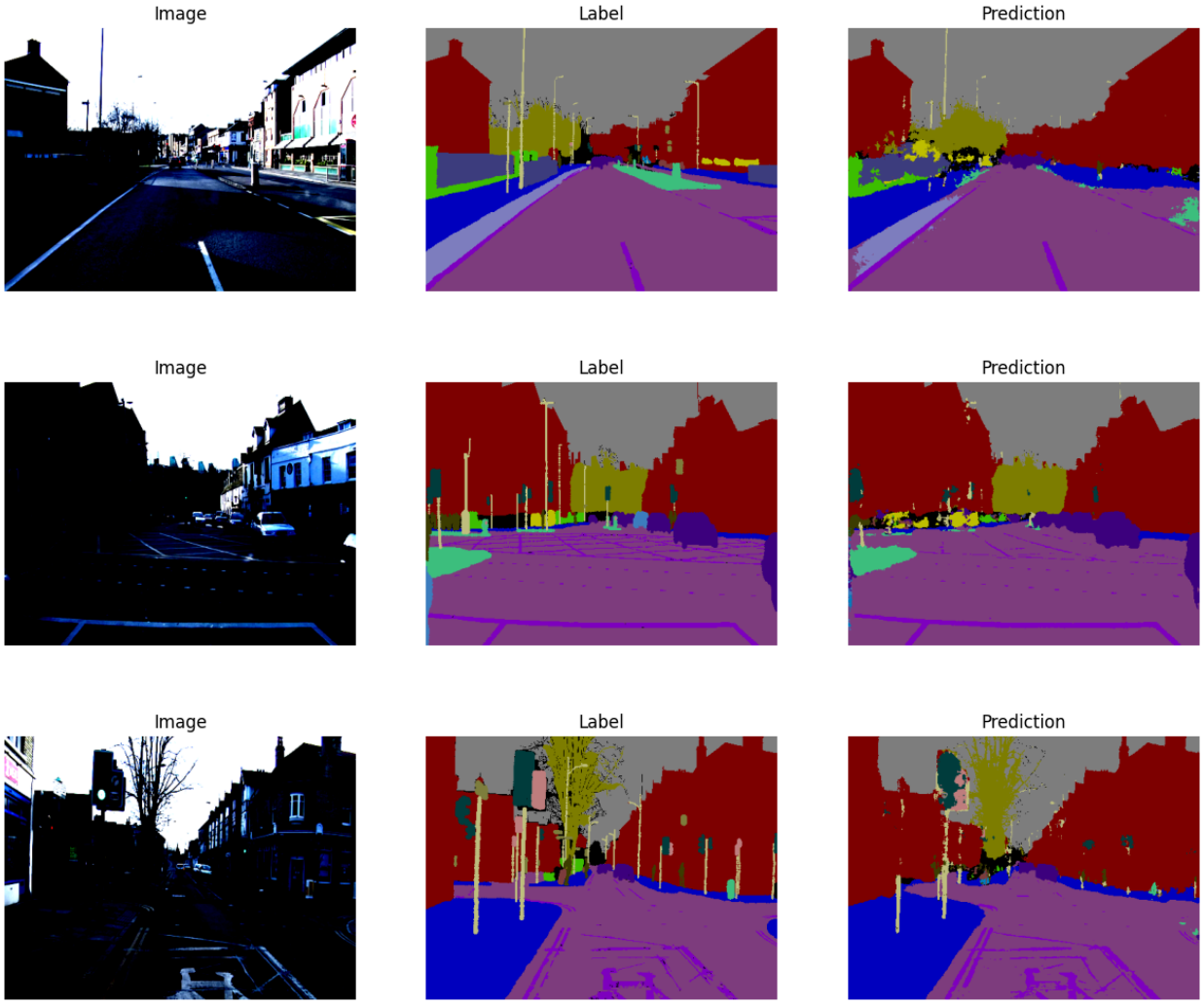
Class 2 (Bicyclist)



Class 9 (Fence)



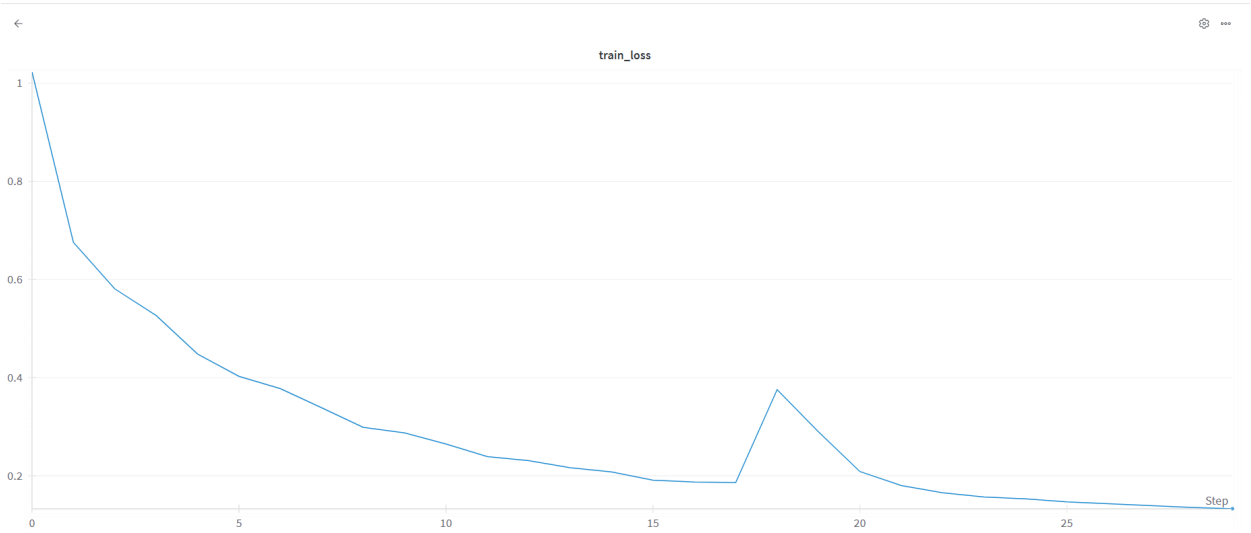
Class 31 (Wall)



Note: Common Reasoning for low IoU is given after DeepLabV3 visualizations.

2.3.a Model Class and Wandb Plots

```python
class DeepLabV3(nn.Module):
    def __init__(self, num_classes=32):
        super(DeepLabV3, self).__init__()
        # TODO: Initialize DeepLabV3 model here using pretrained=True
        self.model = deeplabv3_resnet50(weights=DeepLabV3_ResNet50_Weights.DEFAULT)
        #  should be a Conv2D layer with input channels as 256 and output channel as num_classes using a stride o
        self.model.classifier[4] = nn.Conv2d(in_channels=256, out_channels=num_classes, kernel_size=1, stride=1)

    def forward(self, x):
        return self.model(x)['out']
```



2.3.b Evaluation Metrics

```
miou: 0.36875
class: 0 | pixel wise accuracy: 0.999941555914751
class: 1 | pixel wise accuracy: 0.9997797683189655
class: 2 | pixel wise accuracy: 0.9962508481002554
class: 3 | pixel wise accuracy: 0.9991082974137931
class: 4 | pixel wise accuracy: 0.9575265405491699
class: 5 | pixel wise accuracy: 0.9908039242097701
class: 6 | pixel wise accuracy: 0.999647813896871
class: 7 | pixel wise accuracy: 0.999775303320562
class: 8 | pixel wise accuracy: 0.9904750858077905
class: 9 | pixel wise accuracy: 0.9930608935983397
class: 10 | pixel wise accuracy: 0.990082190892401
class: 11 | pixel wise accuracy: 0.9999726861829502
class: 12 | pixel wise accuracy: 0.9959261254789272
class: 13 | pixel wise accuracy: 0.9998120709610473
class: 14 | pixel wise accuracy: 0.9978514876676245
class: 15 | pixel wise accuracy: 0.997810928520115
class: 16 | pixel wise accuracy: 0.9929638858955939
class: 17 | pixel wise accuracy: 0.9776191331417624
class: 18 | pixel wise accuracy: 0.9986395224696679
class: 19 | pixel wise accuracy: 0.9864047034642401
class: 20 | pixel wise accuracy: 0.9992605064655172
class: 21 | pixel wise accuracy: 0.9848354934945721
class: 22 | pixel wise accuracy: 0.9895954811222861
class: 23 | pixel wise accuracy: 0.9999899225734356
class: 24 | pixel wise accuracy: 0.9981275941890166
class: 25 | pixel wise accuracy: 1.0
class: 26 | pixel wise accuracy: 0.9689755697238186
class: 27 | pixel wise accuracy: 0.9897876756066412
class: 28 | pixel wise accuracy: 1.0
```

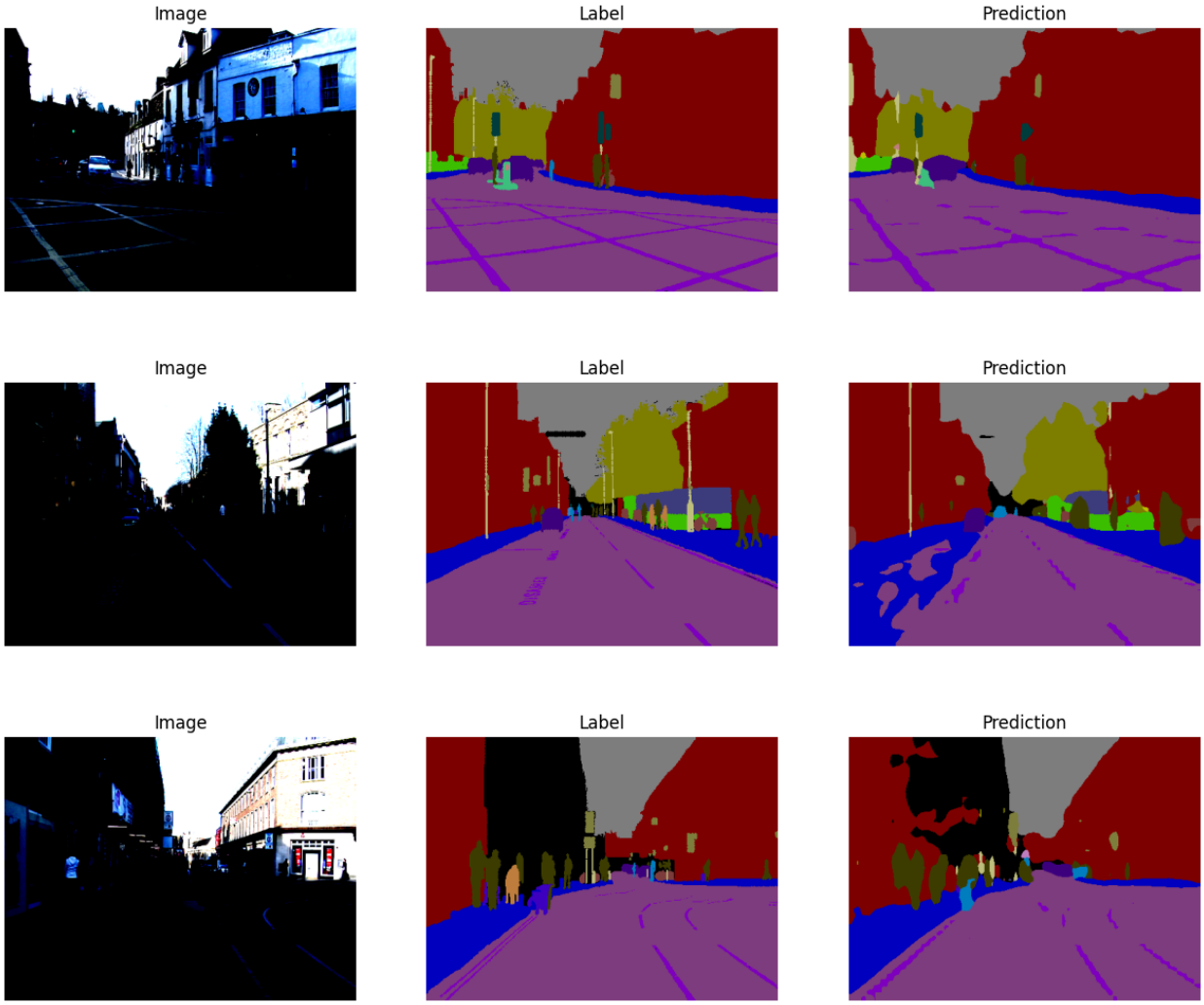
```
class: 29 | pixel wise accuracy: 0.9945293293023627
class: 30 | pixel wise accuracy: 0.9749394855523628
class: 31 | pixel wise accuracy: 0.9916170777458493
-----------------------------------------------
class: 0 | IoU: 0.0
class: 1 | IoU: 0.0
class: 2 | IoU: 0.4
class: 3 | IoU: 0.3
class: 4 | IoU: 0.8
class: 5 | IoU: 0.8
class: 6 | IoU: 0.0
class: 7 | IoU: 0.2
class: 8 | IoU: 0.2
class: 9 | IoU: 0.5
class: 10 | IoU: 0.4
class: 11 | IoU: 0.0
class: 12 | IoU: 0.3
class: 13 | IoU: 0.0
class: 14 | IoU: 0.4
class: 15 | IoU: 0.4
class: 16 | IoU: 0.4
class: 17 | IoU: 0.9
class: 18 | IoU: 0.4
class: 19 | IoU: 0.8
class: 20 | IoU: 0.3
class: 21 | IoU: 0.9
class: 22 | IoU: 0.3
class: 23 | IoU: 0.0
class: 24 | IoU: 0.5
class: 25 | IoU: 0.0
```

```
class: 26 | IoU: 0.8
class: 27 | IoU: 0.2
class: 28 | IoU: 0.0
class: 29 | IoU: 0.5
class: 30 | IoU: 0.5
class: 31 | IoU: 0.6
-----------------------------------------
class: 0 | Dice Coefficient: 0.0
class: 1 | Dice Coefficient: 0.08193823437661621
class: 2 | Dice Coefficient: 0.6174567703905793
class: 3 | Dice Coefficient: 0.45095991399169943
class: 4 | Dice Coefficient: 0.9099895534123423
class: 5 | Dice Coefficient: 0.8831286660290525
class: 6 | Dice Coefficient: 0.024324511091146132
class: 7 | Dice Coefficient: 0.3907750574867854
class: 8 | Dice Coefficient: 0.3979920983014179
class: 9 | Dice Coefficient: 0.6913557509242033
class: 10 | Dice Coefficient: 0.6179811662114024
class: 11 | Dice Coefficient: 0.0
class: 12 | Dice Coefficient: 0.49308155017971056
class: 13 | Dice Coefficient: 0.013357778941852624
class: 14 | Dice Coefficient: 0.5930077067376043
class: 15 | Dice Coefficient: 0.5902424675379222
class: 16 | Dice Coefficient: 0.545067471779909
class: 17 | Dice Coefficient: 0.9547654664757074
class: 18 | Dice Coefficient: 0.5985529327768779
class: 19 | Dice Coefficient: 0.8872928318464387
class: 20 | Dice Coefficient: 0.5065250682468788
class: 21 | Dice Coefficient: 0.9557841804215542
class: 22 | Dice Coefficient: 0.4268922203490195
```

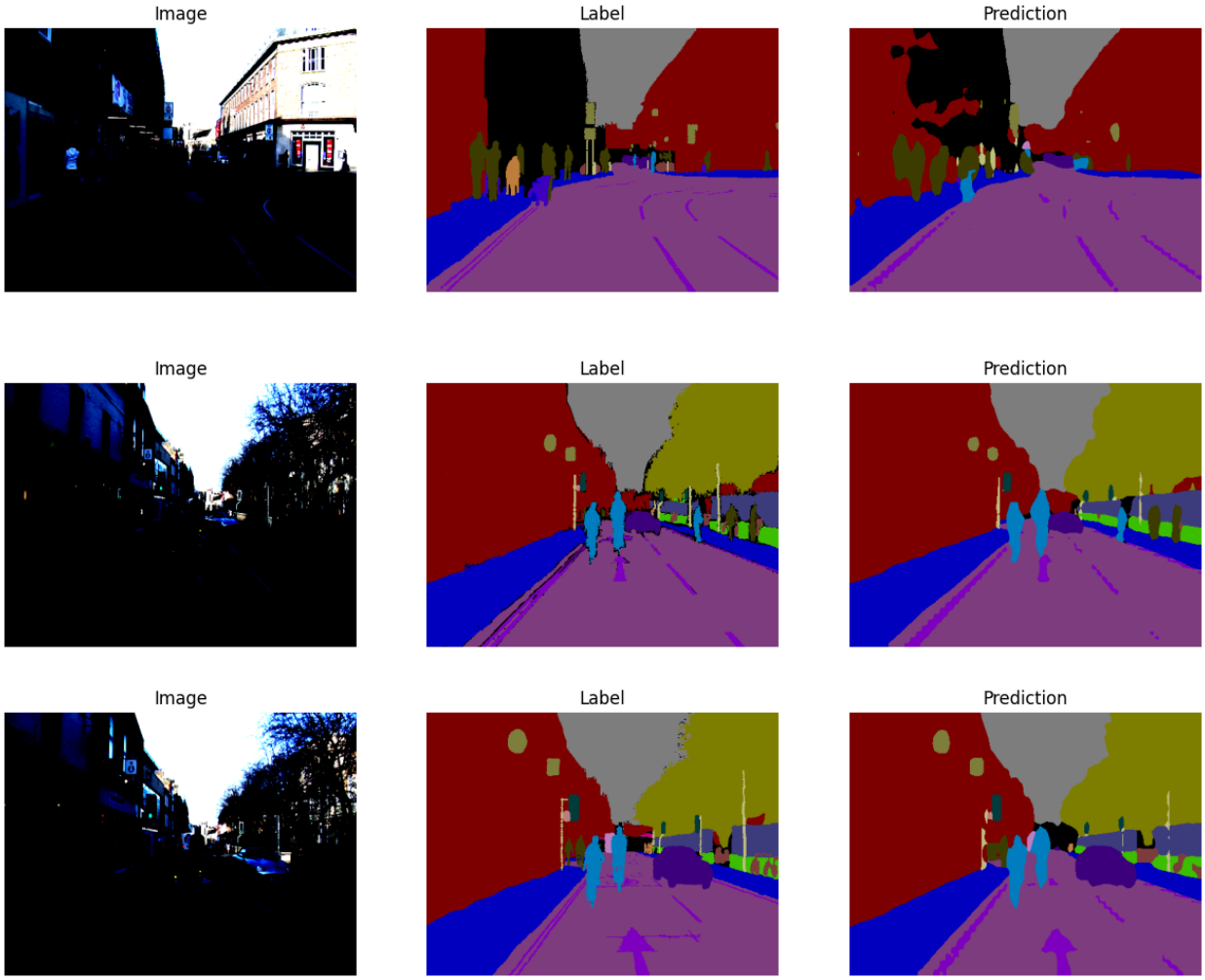
```
class: 23 | Dice Coefficient: 0.0
class: 24 | Dice Coefficient: 0.70736651696606 51
class: 25 | Dice Coefficient: 0.0
class: 26 | Dice Coefficient: 0.8611120820027618
class: 27 | Dice Coefficient: 0.3633954894326332
class: 28 | Dice Coefficient: 0.0
class: 29 | Dice Coefficient: 0.6254109826520475
class: 30 | Dice Coefficient: 0.6254061313624759
class: 31 | Dice Coefficient: 0.7171925596300006
-----------------------------------------
Overall Accuracy: 0.8775554507902299
```

2.3.c Visualizing Misclassifications

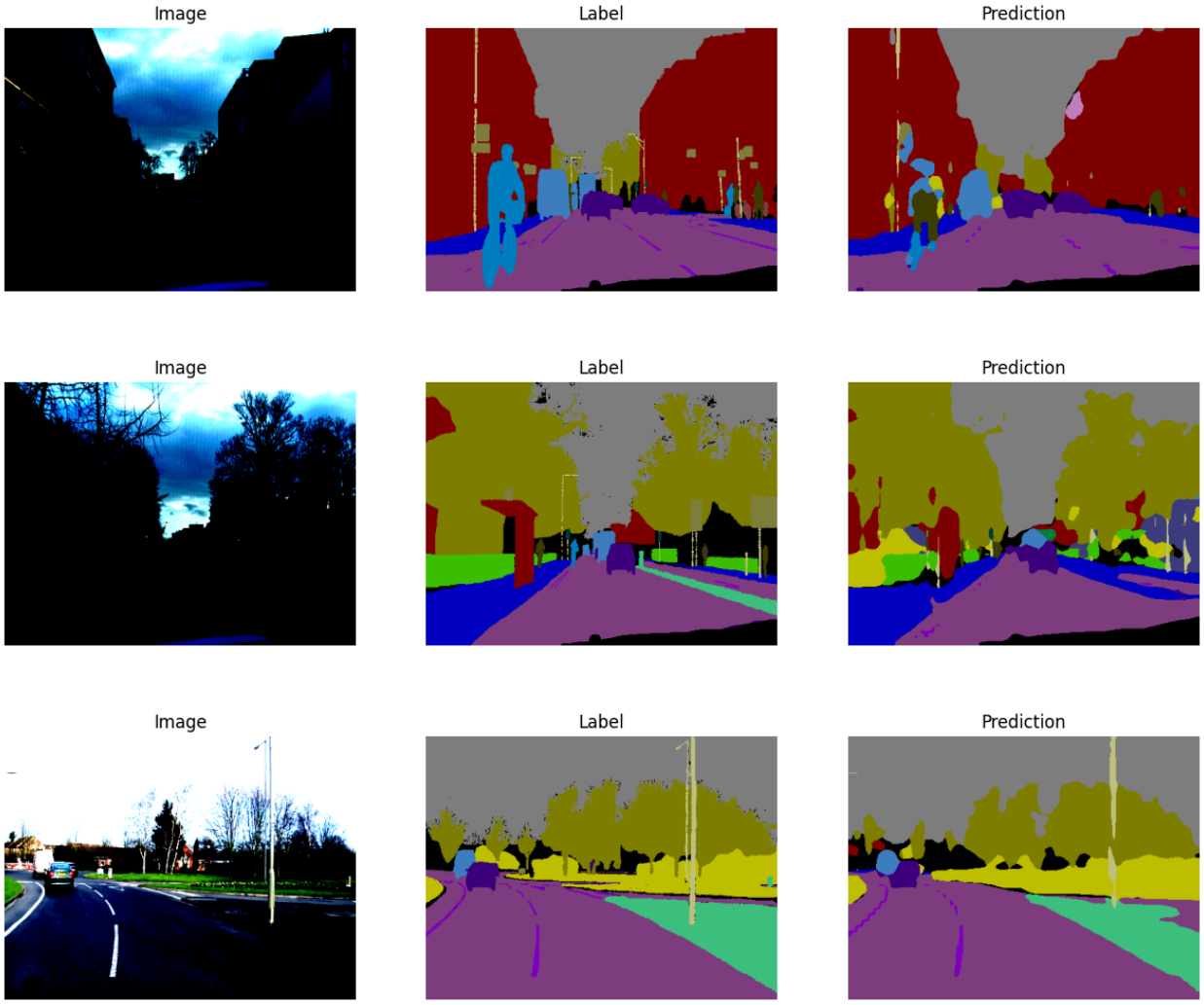
Class 2 (Bicylcist)



Class 7 (Child)



Class 10 (LaneMkgsDriv)



Reasoning for Low IoU: After visualizing masks for predictions with low IoU (in both color masks and focused masks - code in Python notebook), these are the possible inferences:

- The brightness of the environment of the target object is low, leading to a lack of visibility for the object.
- The objects are far away from the camera and thus are extremely small in the image (for example, child class).
- The class is not present in the image, leading to high (biased) pixel accuracy but near zero IoU.

3. Detection

3.3.a Handling the Dataset and Prediction

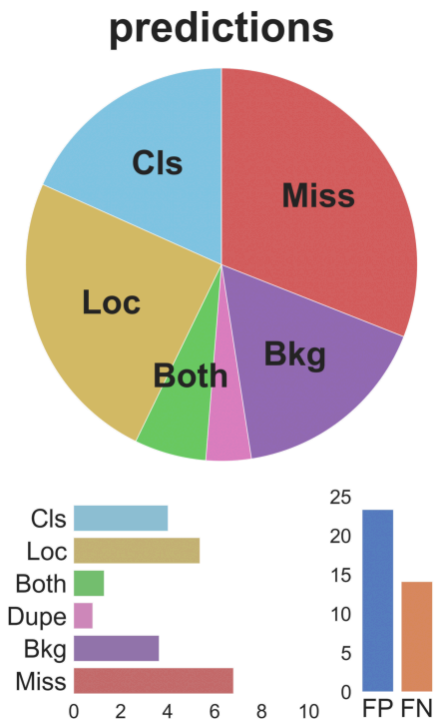
```python
from ultralytics import YOLO

if __name__ == "__main__":
    model = YOLO("yolov8n.pt")
    results = model.val(data="coco.yaml", imgsz=640, workers=0)
```

3.3.a Average Precision and Average Recall

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.374
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.526
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.405
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.186
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.411
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.535
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.320
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.533
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.589
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.369
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.655
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.769
```

3.3.c TIDE ToolBox

```
-- predictions --

bbox AP @ 50: 52.20

                        Main Errors
============================================================
  Type      Cls      Loc      Both      Dupe      Bkg      Miss
------------------------------------------------------------
   dAP     4.02     5.38      1.30      0.82     3.64      6.80
============================================================

        Special Error
============================
  Type   FalsePos   FalseNeg
----------------------------
   dAP      23.34      14.13
============================
```



Classification Error (Cls) - dAP: 4.02

The model predicts the correct object but assigns it the wrong class.

Localization Error (Loc) - dAP: 5.38

The model classifies objects correctly but places bounding boxes inaccurately.

Both Classification and Localization Error (Both) - dAP: 1.30

The model makes mistakes in both classification and localization, leading to a combined error.

Duplicate Error (Dupe) - dAP: 0.82

The model detects the same object multiple times due to poor Non-Maximum Suppression (NMS) settings, where overlapping detections are not merged properly.

Background Error (Bkg) - dAP: 3.64

False positives occur when the model detects objects that do not actually exist due to excessive sensitivity to certain patterns in the background.

Missed Ground Truth (Miss) - dAP: 6.80

The model fails to detect objects that are present in the ground truth.

False Positives (FalsePos) - dAP: 23.34

The model predicts too many objects that are incorrect, significantly lowering precision.

False Negatives (FalseNeg) - dAP: 14.13

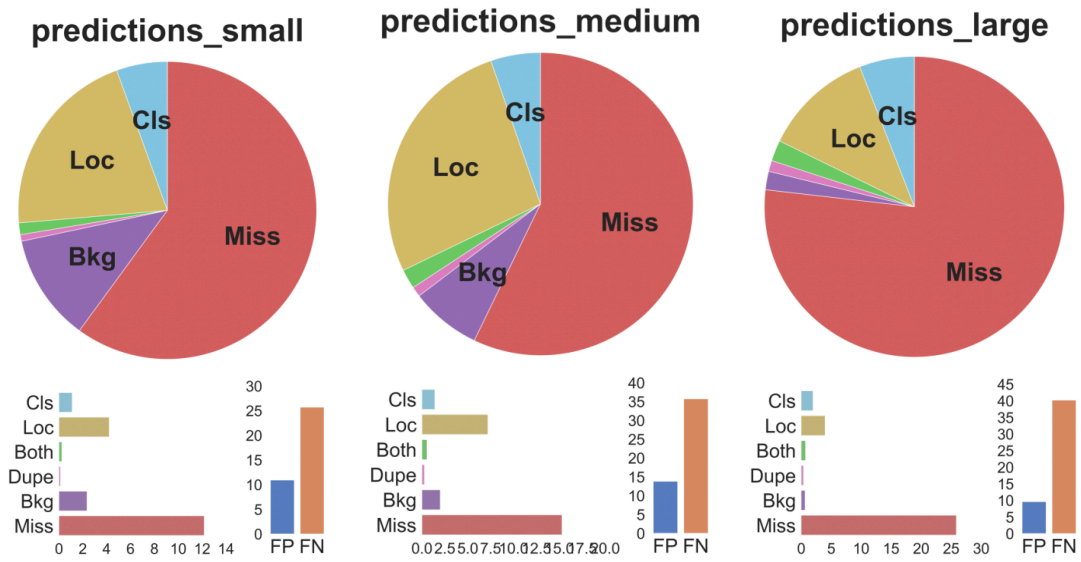
The model fails to detect objects that exist, lowering recall.

3.3.d Expected Calibration Error (ECE)

```
bin_sizes = df_pred["bin"].value_counts().sort_index().to_list()
bin_acc = df_pred.groupby("bin", observed=True)["acc"].sum().to_list()
bin_conf = df_pred.groupby("bin", observed=True)["score"].sum().to_list()

ECE = 0
for i in range(num_bins-1):
    ECE += abs(bin_acc[i] - bin_conf[i]) / sum(bin_sizes)
print(ECE)
```

```
0.2706045977515154
```

3.3.e Size-based Analysis



3.3.f Analysis

- **Observation:** Miss increases as object size grows. Loc remains relatively stable across all sizes. Cls are slightly reduced for larger objects, whereas Bkg and Dupe are negligible.
- **Inference:** The model struggles more with larger objects, as missed detections increase with size. Constant Classification errors remain relatively constant, indicating consistent confusion across sizes. Localization errors are significant for small and medium objects but decrease for large objects. Thus, the model locates bigger objects quickly once detected. False negatives dominate across all sizes, meaning the model fails to detect a large portion of objects, resulting in lower recall.
- The original TIDE prediction across all object sizes shows a more balanced distribution. Missed detections (Miss) are still there but are lower than in large-object predictions. Classification errors (Cls) are more prominent in the original plot, indicating challenges in handling diverse sizes. Localization errors (Loc) remain consistent.