Theory

Q.1)

a) $y_{smooth,i} = \begin{cases} 1 - \epsilon + \frac{\epsilon}{K} & \text{if } i == K \\ \frac{\epsilon}{K} & \text{otherwise} \end{cases}$

Cross Entropy Loss : $H(y, q) = E_{y_{smooth}}[-\log(q(x))] = \sum_{i=1}^{K} y_{smooth,i}(-\log q_i)$

Substituting $y_{smooth,i}$: $H(y,q) = \left(1 - \epsilon + \frac{\epsilon}{K}\right)(-\log q_K) + \sum_{i \neq K} \frac{\epsilon}{K}(-\log q_i)$

$$H(y,q) = (1-\epsilon)(-\log q_K) + \frac{\epsilon}{K}\sum_{i=1}^{K}(-\log q_i)$$

b)

1) Prevents Overfitting : Discourage logits from becoming excessive large
2) Promoting Entropy: Smoother predicted distribution, improving robustness to noise
3) Balance Coefficient: Prior belief that prediction should not be too peaked, enhancing generalisation

Q.2)

a) $H(p, q) = E_{x \sim p(x)}[-\log(q(x))]$

b) $H(p, x) = \int -p(x) \log \frac{1}{\sqrt{2\pi\sigma_q^2}} e^{-\frac{1}{2}\left[\frac{x-\mu_q}{\sigma_q}\right]} dx$

$= \int -p(x) \log \frac{1}{\sqrt{2\pi\sigma_q^2}} dx + \int p(x) \frac{1}{2}\left(\frac{x-\mu_q}{\sigma_q}\right)^2 dx$

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

$= -\log\left(\frac{1}{\sqrt{2\pi\sigma_q^2}}\right) + \frac{E_{x \sim p(x)}[x^2] - 2\mu_q E_{x \sim p(x)}[x] + \mu_q^2}{2\sigma_q^2}$

$= -\log \frac{1}{\sqrt{2\pi\sigma_q^2}} + \frac{\mu_p^2 + \sigma_p^2 - 2\mu_q\mu_p + \mu_q^2}{2\sigma_q^2}$

Teacher's Signature:.....

$$= -\log \frac{1}{\sqrt{2\pi\sigma_q^2}} + \frac{(\mu_p - \mu_q)^2}{2\sigma_q^2} + \frac{\sigma_p^2}{2\sigma_q^2}$$

iii) $\sigma_p = \sigma_q = \sigma$

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

Q.3)

1) $$w_{out} = \frac{w_{in} - 2P - D(K-1) - 1}{S} + 1 = w_{in} - D(K-1)$$

after L layers, $w_{out} = w_{in} - (K-1)\sum_{i=0}^{L-1} e^i$

$$= w_{in} - (K-1)(2^L - 1)$$

$$w_{in} = w_{out} + (K-1)(2^L - 1)$$

if $w_{out} = 1 \Rightarrow \boxed{w_{in} = 1 + K(2^L - 1)}$

∴ Receptive field grows exponentially in L

2) It follows the same pattern as 1D

So R_L = Receptive field = $w_{in} \times w_{in}$

$$R_L = (1 + (K-1)(2^L - 1))^2$$

3) Standard $K \times K$ convolution requires multiply-divide operations

$$O(N^2 K^2)$$

Dilated $K \times K$ does act on wider area but does not increase the number of parameter

$$O(N^2 K^2)$$

Does not reduce no. of operation but increases receptive field significantly

M^{th} Add operation for both are related & std is better

M^{th} Add position $= K^2$

Total M^{th} Add operation $= (H-K+1) \times (W-K+1) \times K^2$

Teacher's Signature:.....

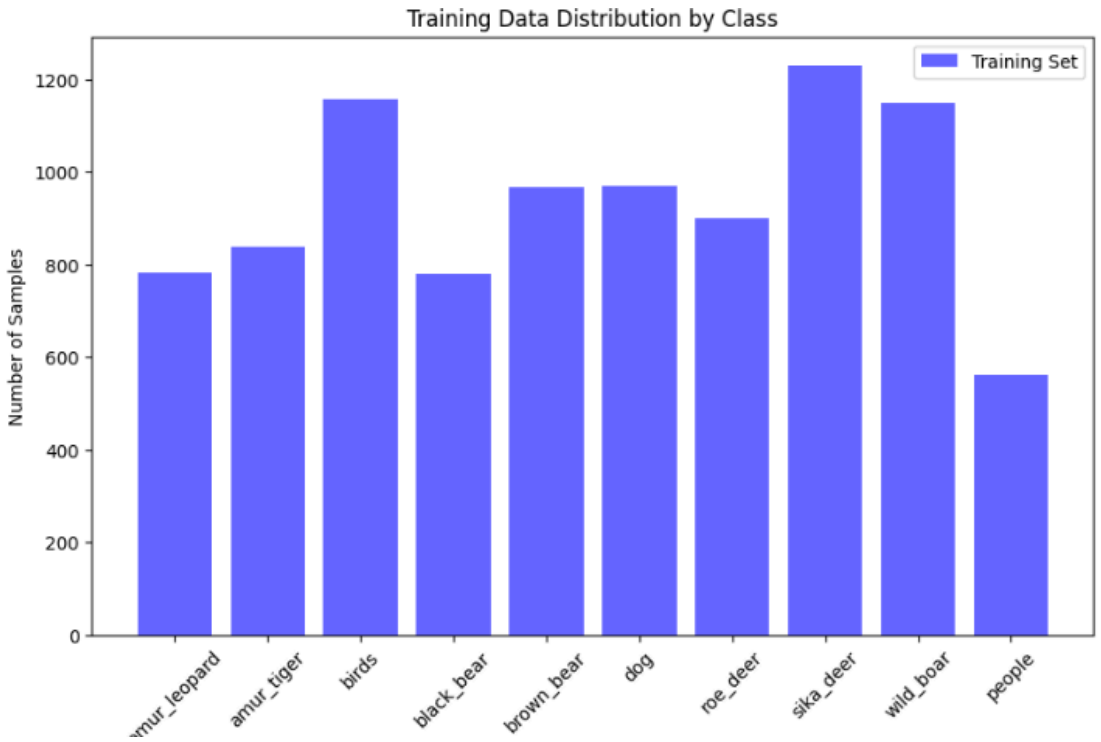
Image Classification

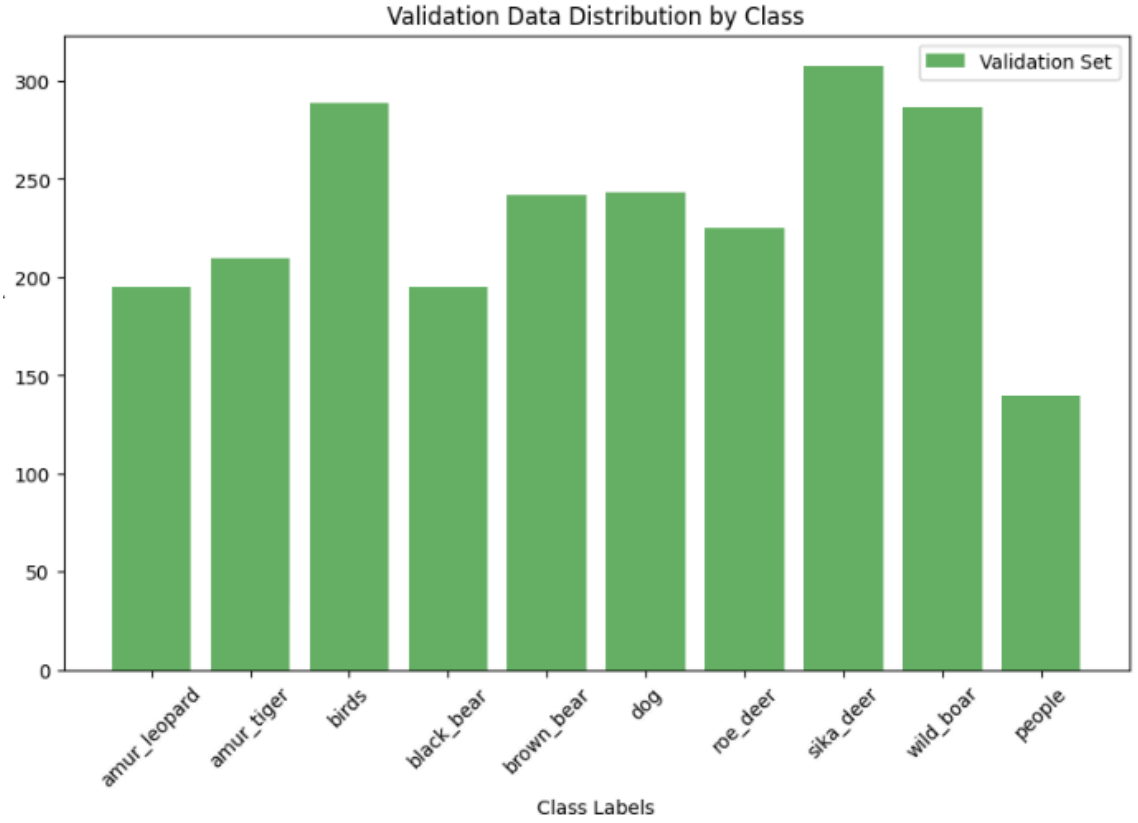
1)

a)

```python
# Step 3: Dataset class
class Dataset(Dataset):
    def __init__(self, image_paths, labels, transform=None):
        self.image_paths = image_paths
        self.labels = labels
        self.transform = transform

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

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        label = self.labels[idx]
        img = Image.open(img_path).convert('RGB')
        if self.transform:
            img = self.transform(img)
        return img, label
```

c)





2)

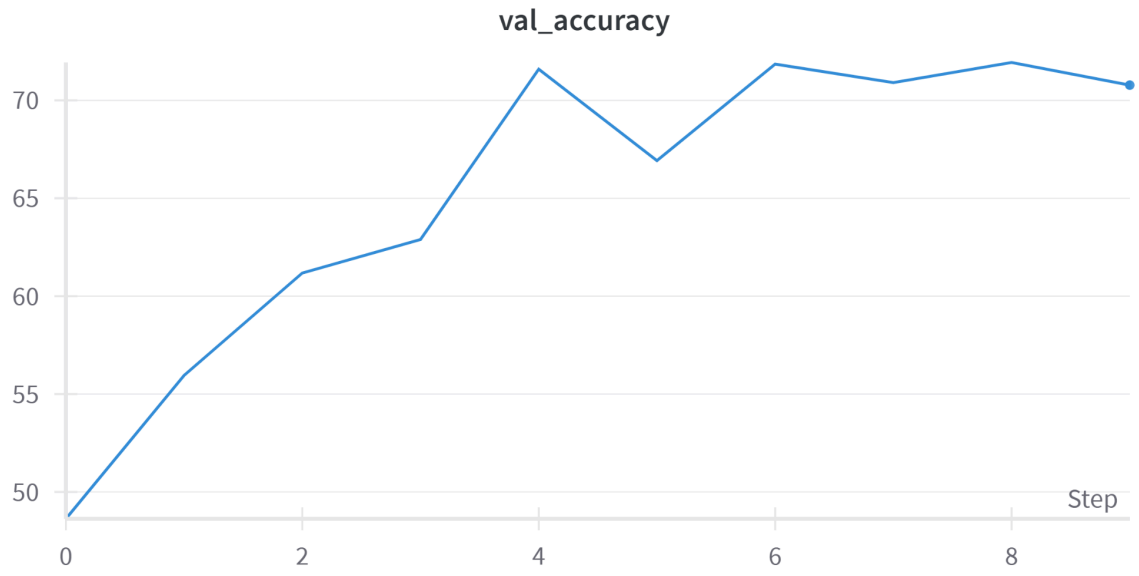
a)

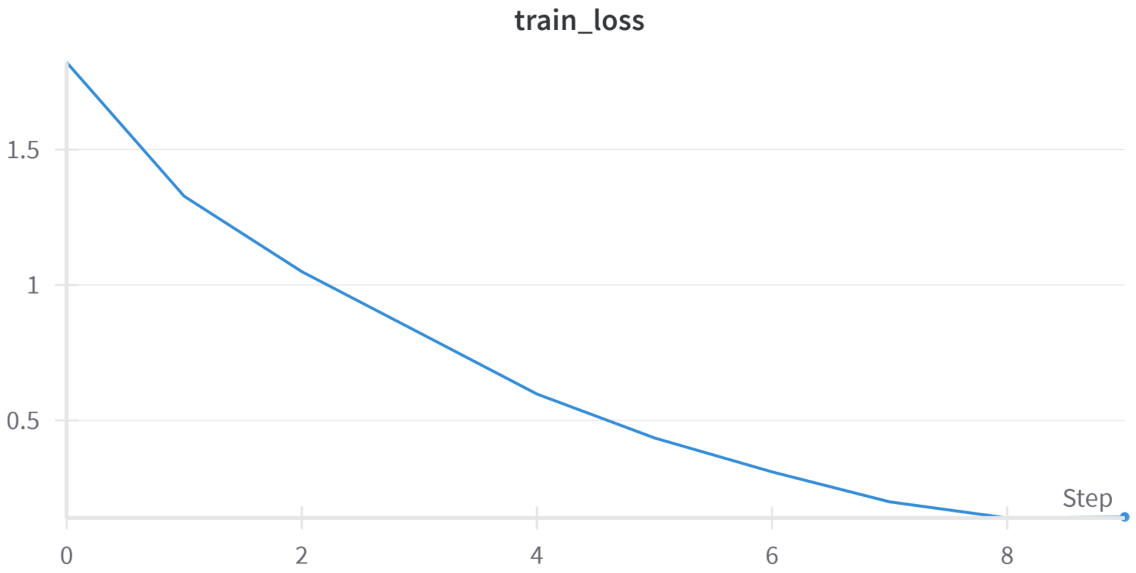
```python
class CNN_Model(nn.Module):
    def __init__(self, num_classes=10):
        super(CNN_Model, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=30, kernel_size=3, padding=1, stride=1)
        self.pool1 = nn.MaxPool2d(kernel_size=4, stride=4)
        self.conv2 = nn.Conv2d(in_channels=30, out_channels=64, kernel_size=3, padding=1, stride=1)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1, stride=1)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        # self.dropout = nn.Dropout(0.3)

        self.fc_input_size = 128 // 4 // 2 // 2
        self.fc_input_size = self.fc_input_size * self.fc_input_size * 128
        self.fc1 = nn.Linear(self.fc_input_size, 512)
        self.fc2 = nn.Linear(512, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool1(x)
        x = self.relu(self.conv2(x))
        x = self.pool2(x)
        x = self.relu(self.conv3(x))
        x = self.pool3(x)
        x = x.view(-1, self.fc_input_size)
        x = self.relu(self.fc1(x))

        # x = self.dropout(x)

        x = self.fc2(x)
        return x
```

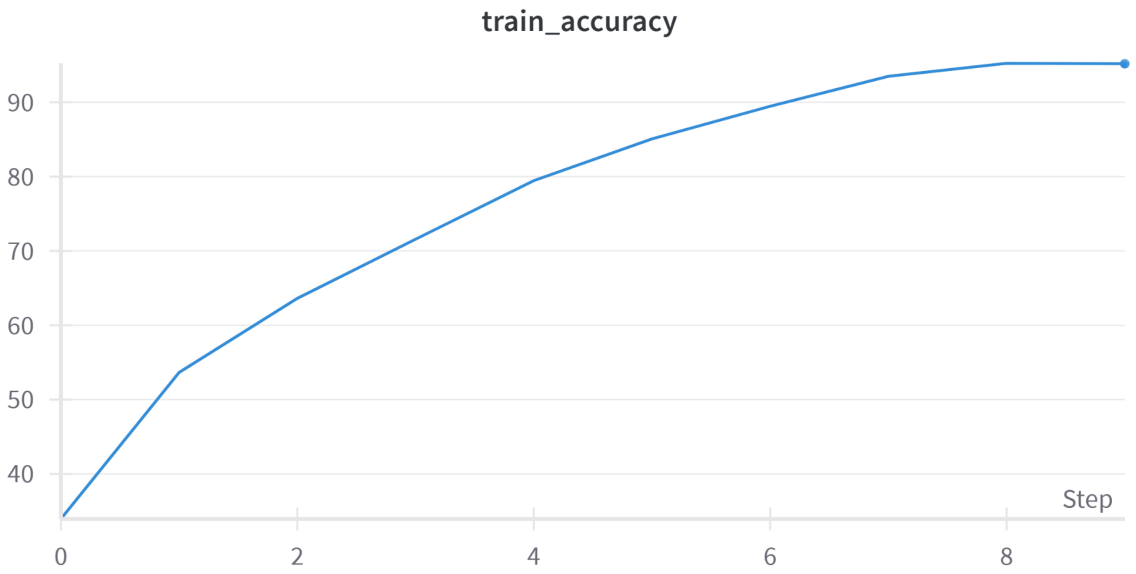
Dropout was initially used to prevent overfitting but later part of qs made me realise we have to not do so.

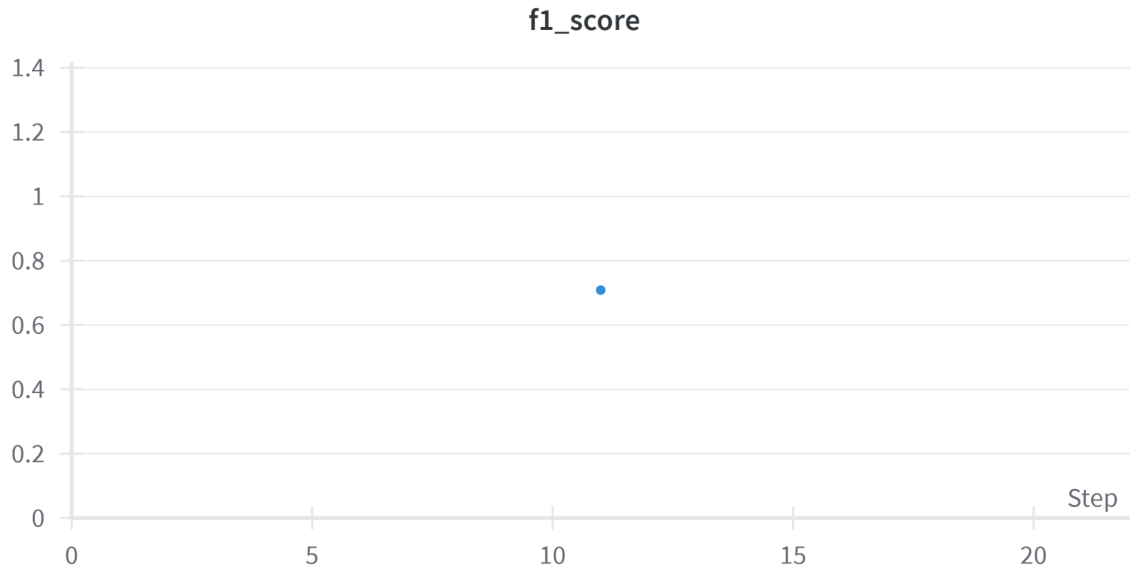
b)


val_loss
1.4
1.3
1.2
1.1
1
Step
0
2
4
6
8


val_accuracy
70
65
60
55
50
Step
0
2
4
6
8


train_loss
1.5
1
0.5
Step
0
2
4
6
8


train_accuracy
90
80
70
60
50
40
Step
0
2
4
6
8

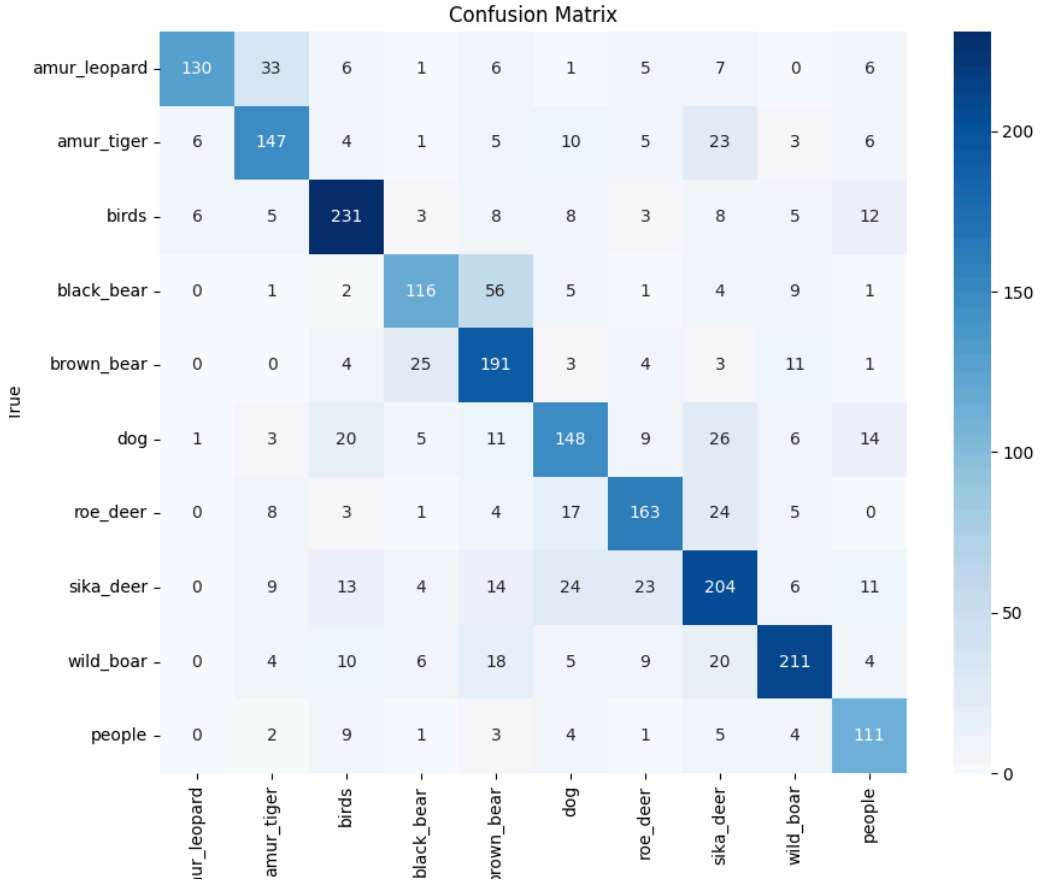


(c)
Yes, The model is overfitting:

The model shows signs of overfitting. While the training loss decreases steadily and the training accuracy reaches 95.59%, the validation loss fluctuates and slightly increases after epoch 4, with validation accuracy stagnating at 71.57%. This growing gap between training and validation accuracy indicates the model is fitting too closely to the training data, causing it to perform less well on the validation set. To mitigate overfitting, techniques like regularization, early stopping, or data augmentation could be helpful.

(d)

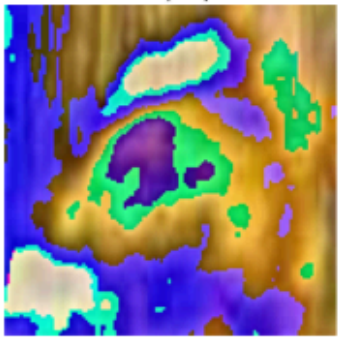
```
wandb: Run summary:
wandb:       f1_score 0.7168
wandb: train_accuracy 96.63595
wandb:     train_loss 0.10707
wandb:   val_accuracy 71.55099
wandb:       val_loss 1.28039
```


Confusion Matrix
amur_leopard
amur_tiger
birds
black_bear
brown_bear
dog
roe_deer
sika_deer
wild_boar
people
True
130 33 6 1 6 1 5 7 0 6
6 147 4 1 5 10 5 23 3 6
6 5 231 3 8 8 3 8 5 12
0 1 2 116 56 5 1 4 9 1
0 0 4 25 191 3 4 3 11 1
1 3 20 5 11 148 9 26 6 14
0 8 3 1 4 17 163 24 5 0
0 9 13 4 14 24 23 204 6 11
0 4 10 6 18 5 9 20 211 4
0 2 9 1 3 4 1 5 4 111
200
150
100
50
0

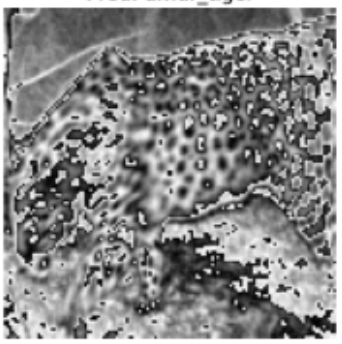
e)

Misclassified Images for amur_leopard

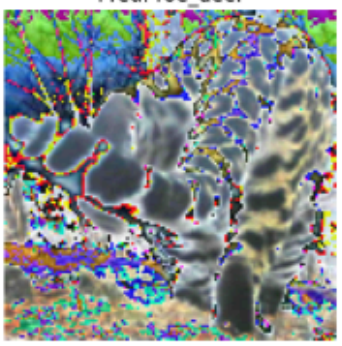
True: amur_leopard
Pred: people



True: amur_leopard
Pred: amur_tiger

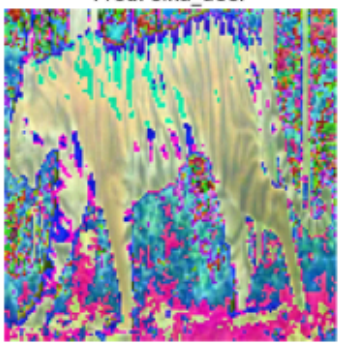


True: amur_leopard
Pred: roe_deer



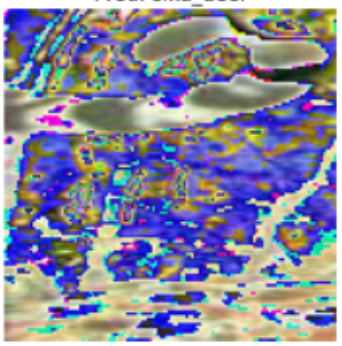
Misclassified Images for amur_tiger

True: amur_tiger
Pred: sika_deer



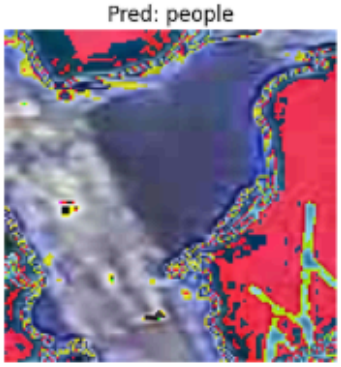
True: amur_tiger
Pred: roe_deer

True: amur_tiger
Pred: sika_deer

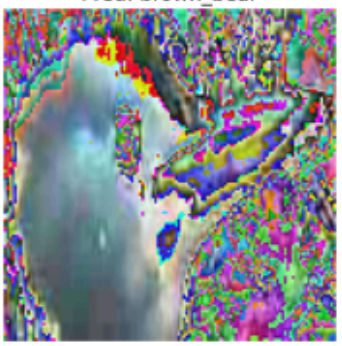


Misclassified Images for birds

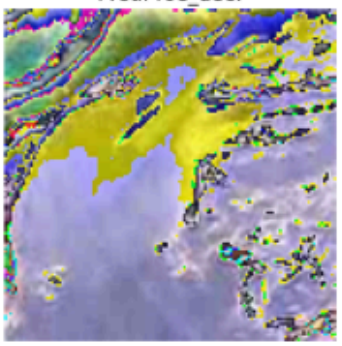
True: birds
Pred: people



True: birds
Pred: brown_bear



True: birds
Pred: roe_deer



Misclassified Images for black_bear

True: black_bear
Pred: wild_boar

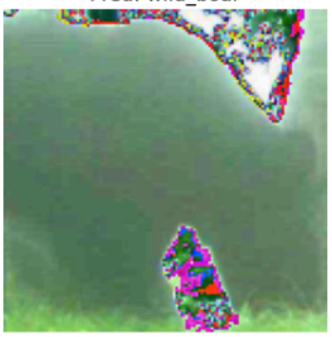
True: black_bear
Pred: wild_boar

True: black_bear
Pred: wild_boar

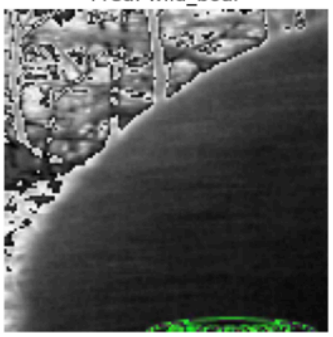
Misclassified Images for black_bear

True: black_bear
Pred: wild_boar

True: black_bear
Pred: wild_boar

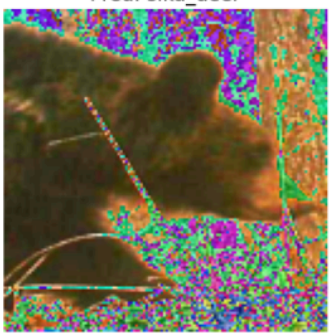


True: black_bear
Pred: wild_boar

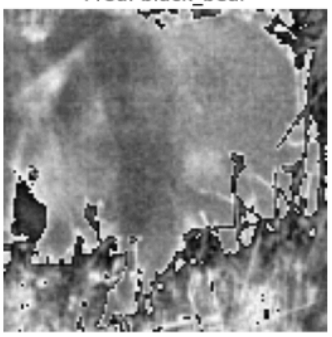


Misclassified Images for brown_bear

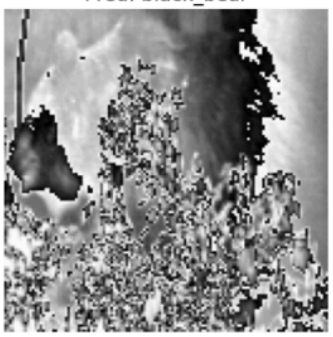
True: brown_bear
Pred: sika_deer



True: brown_bear
Pred: black_bear



True: brown_bear
Pred: black_bear

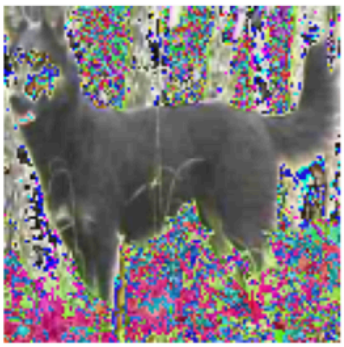


Misclassified Images for dog

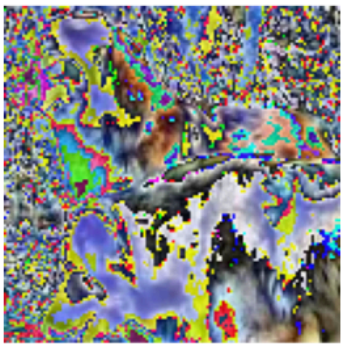
True: dog

Misclassified Images for dog

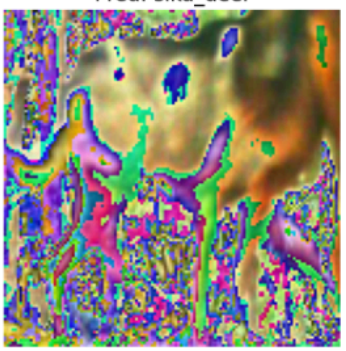
True: dog
Pred: birds



True: dog
Pred: birds

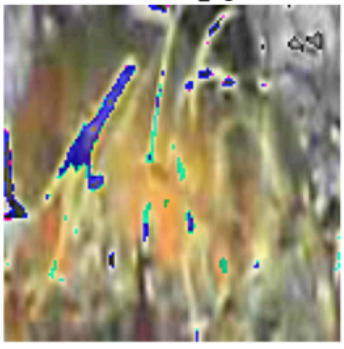


True: dog
Pred: sika_deer

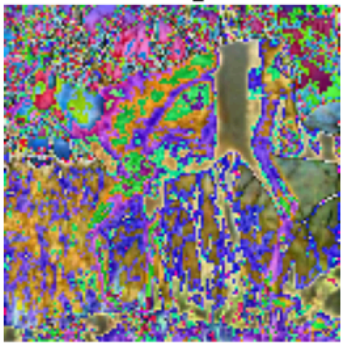


Misclassified Images for roe_deer

True: roe_deer
Pred: amur_tiger



True: roe_deer
Pred: sika_deer



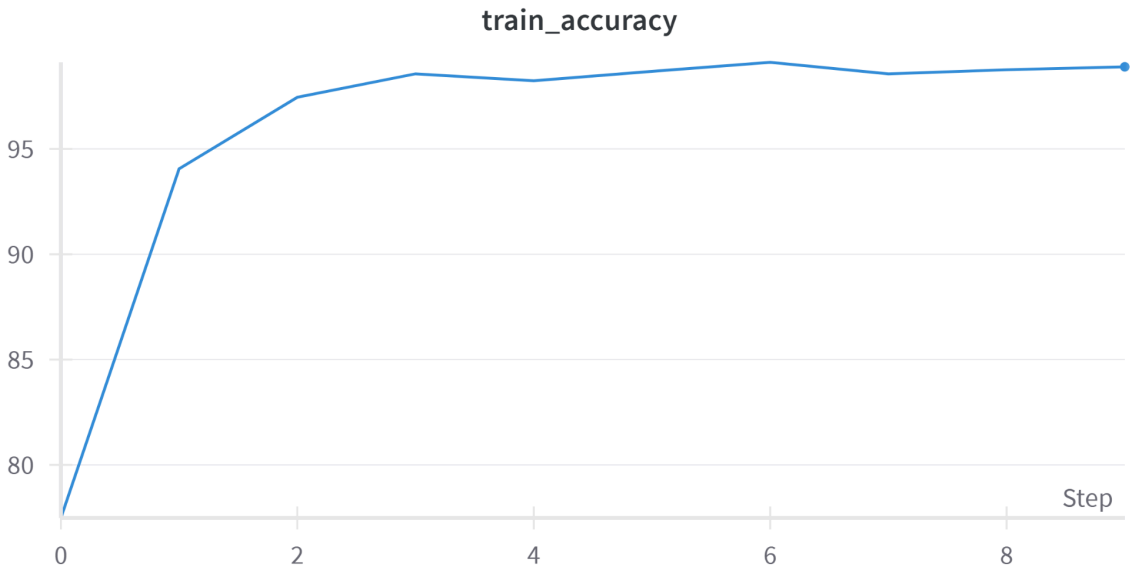
True: roe_deer
Pred: sika_deer

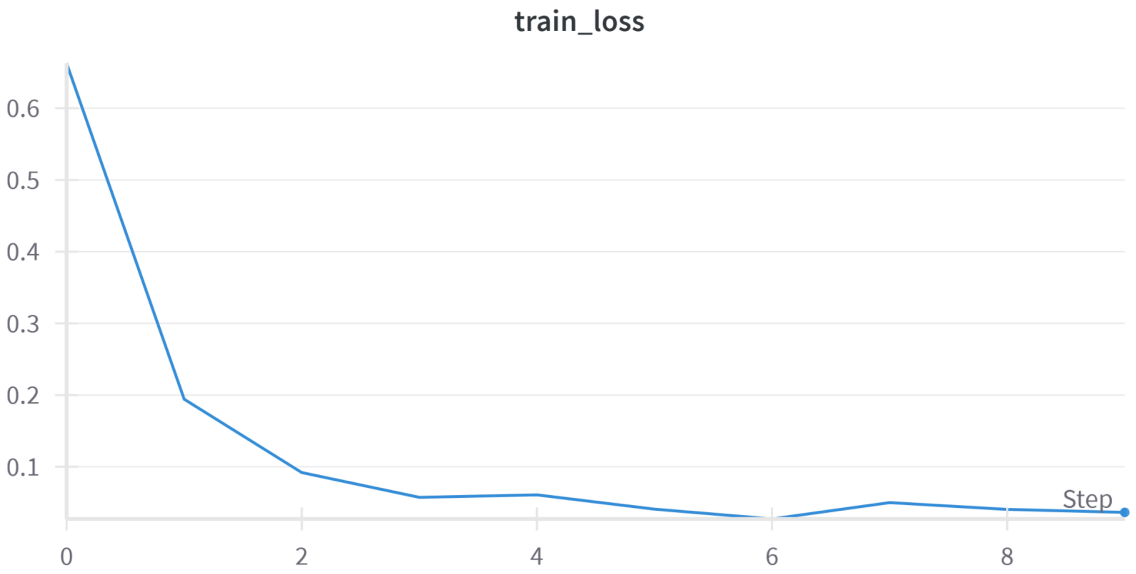
Misclassified Images for sika_deer

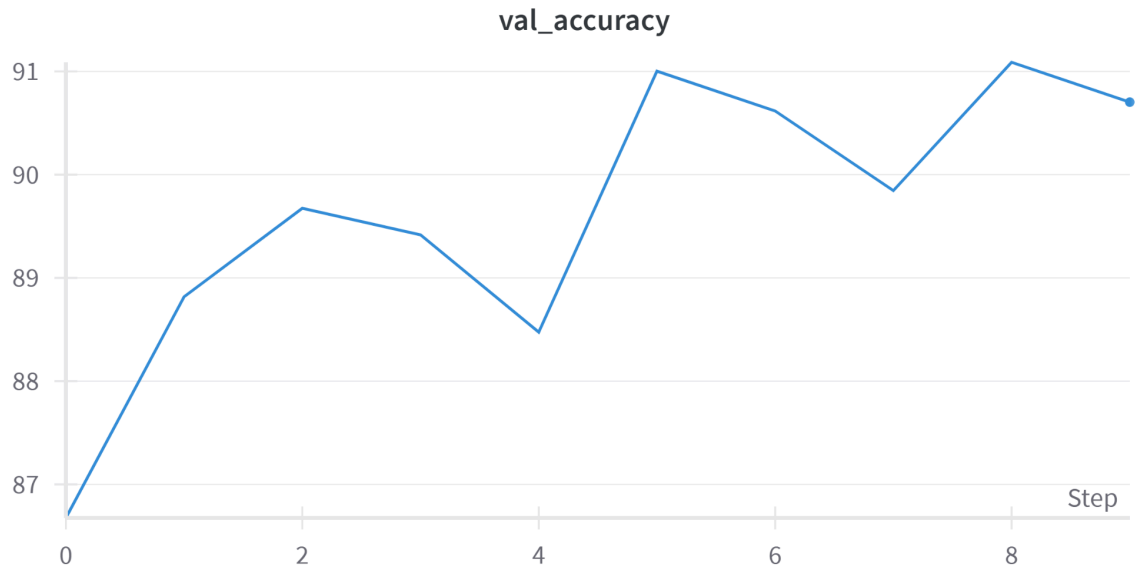
The model's failure in these cases seems to be primarily due to incorrect ground truth labels and the inherent similarities between certain classes. In several instances, we observed that the misclassified images actually contained animals that looked more similar to the predicted class rather than the ground truth. This suggests that the model was likely making an error based on visual features that were more aligned with the predicted class. In other cases, the misclassification could be attributed to mislabeling during the dataset preparation process, where images that did not contain the ground truth class at all were incorrectly labeled. These labeling errors can significantly confuse the model, leading to predictions that seem accurate but are actually based on faulty data.

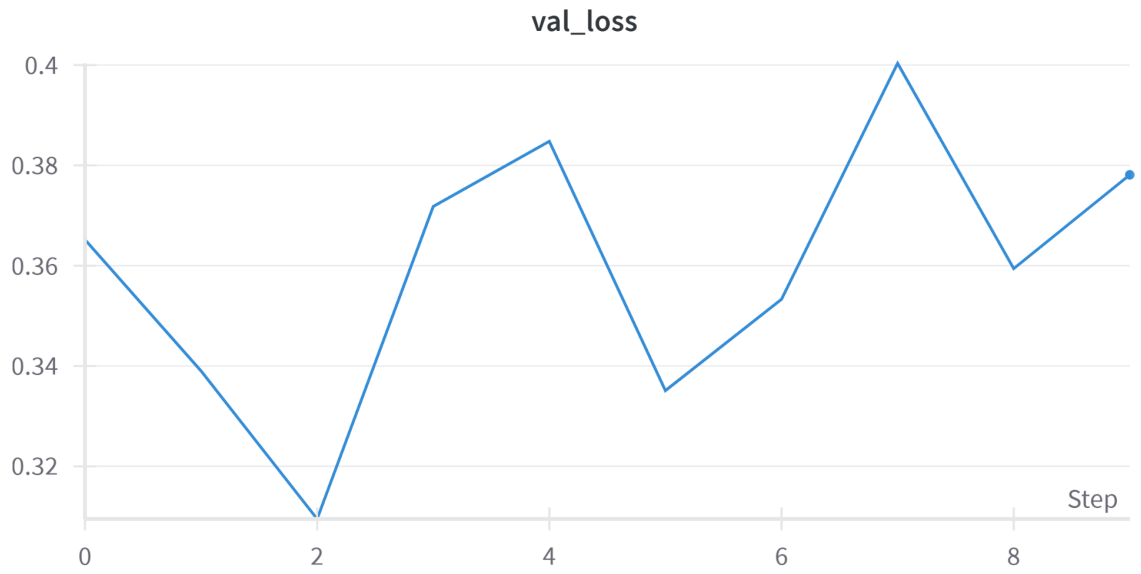
One possible workaround for these issues would be to carefully review and correct the labels in the dataset. This would ensure that the model has accurate information during training and can learn the correct patterns. Additionally, using data augmentation techniques can help the model generalize better and avoid overfitting to subtle features that might be misleading. By applying transformations like random rotations, cropping, and color adjustments, we can expose the model to a wider variety of data, reducing the chances of it learning incorrect or overly specific features. Another solution could be to implement a robust error-checking mechanism that flags and reviews potentially misclassified or mislabeled samples during training, improving the model's overall accuracy.

3)
a)


train_accuracy
95
90
85
80
Step
0
2
4
6
8


train_loss
0.6
0.5
0.4
0.3
0.2
0.1
Step
0
2
4
6
8





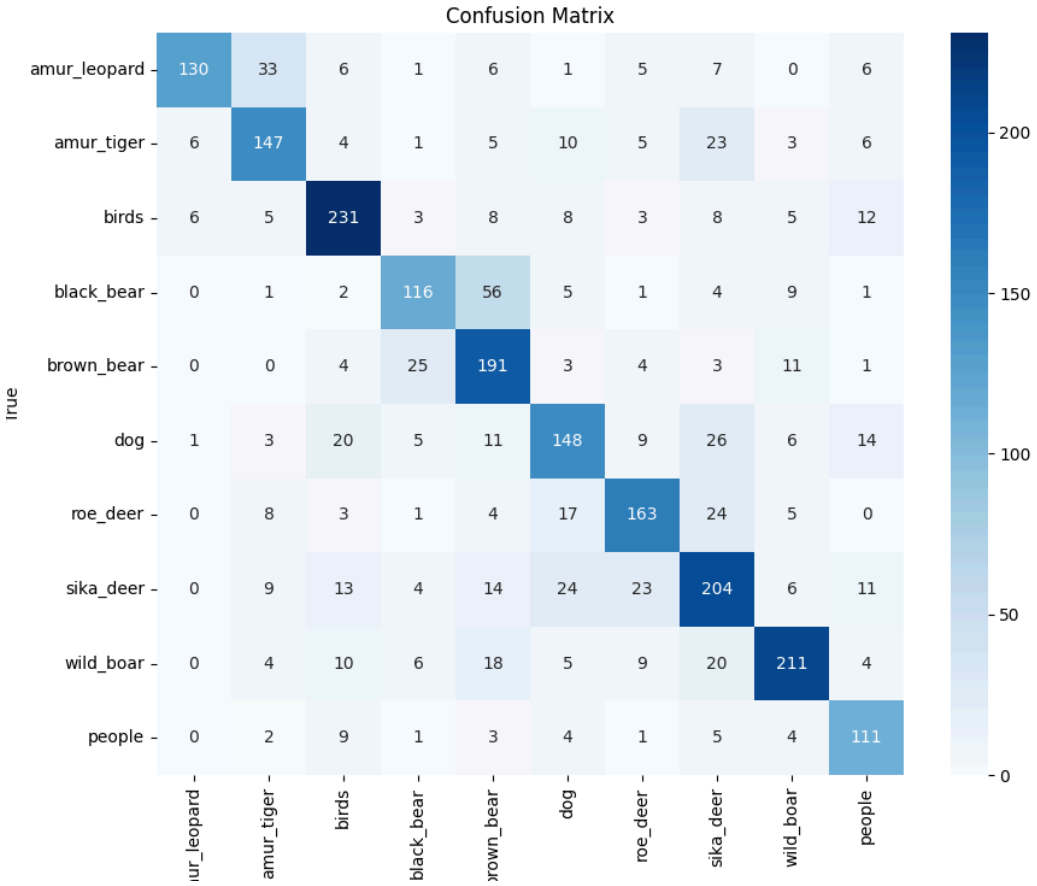
b)

The model shows signs of overfitting. While the training accuracy reaches nearly 99%, the validation accuracy only increases to around 91% and fluctuates slightly. The training loss steadily decreases, but the validation loss stagnates or fluctuates, especially in later epochs. This discrepancy between training and validation performance suggests the model is memorizing the training data but struggling to generalize to the validation set. To address overfitting, techniques like early stopping, dropout, data augmentation, or regularization can be applied.

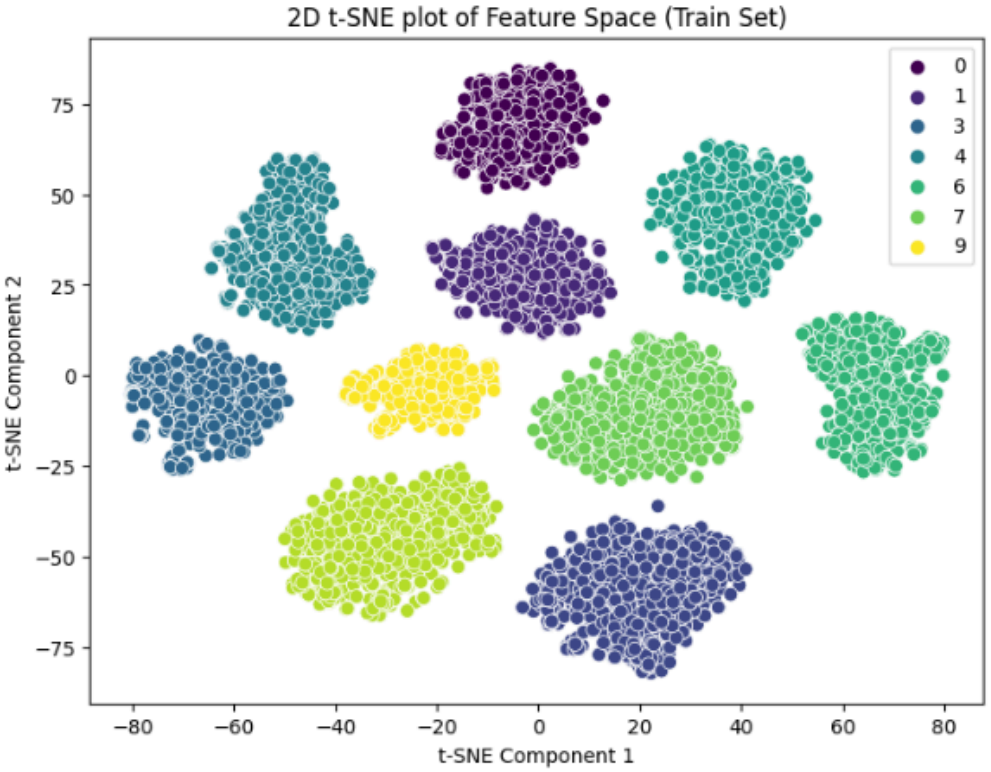
c)

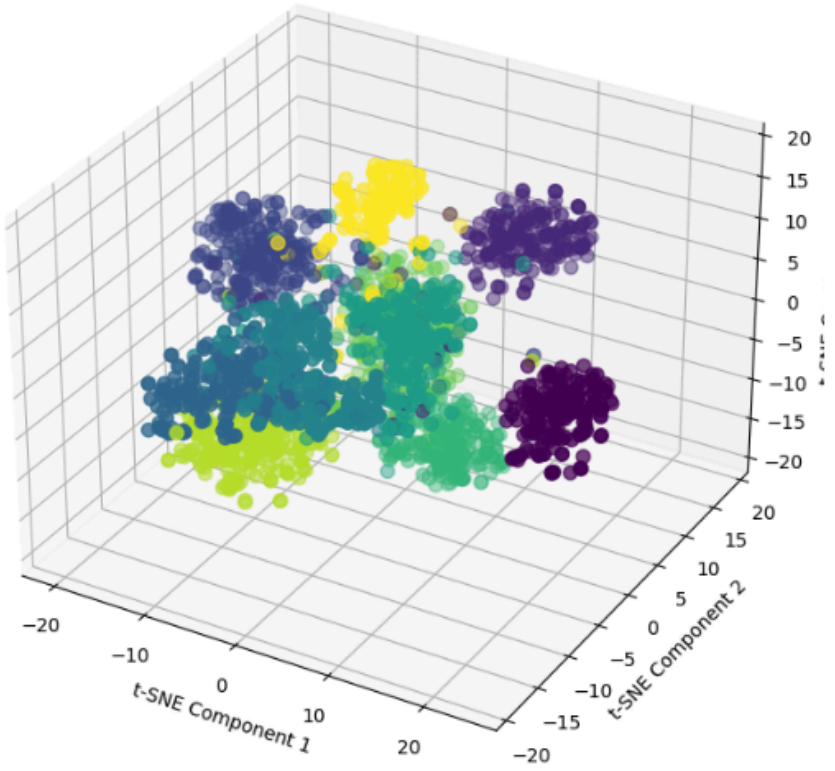
```
Training Loss: 0.0256, Training Accuracy: 99.15%
```

```
Validation Loss: 0.4280, Validation Accuracy: 90.15%
Validation Accuracy: 0.9015
Validation F1-Score: 0.9013
```



d)


2D t-SNE plot of Feature Space (Train Set)
0
1
3
4
6
7
9
t-SNE Component 2
t-SNE Component 1
75
50
25
0
−25
−50
−75
−80
−60
−40
−20
0
20
40
60
80

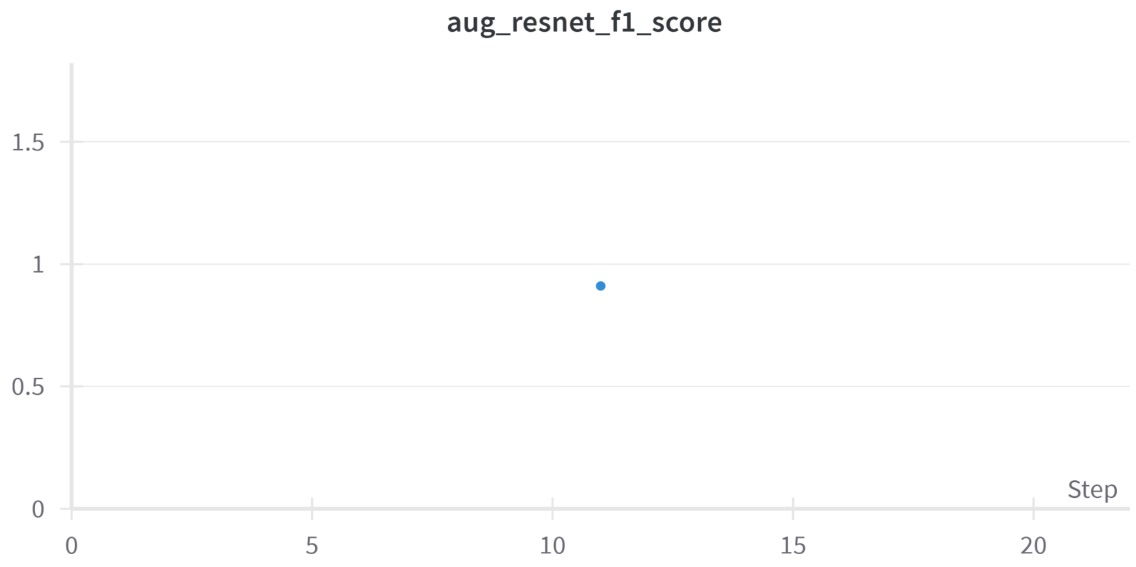

t-SNE Component 1
t-SNE Component 2
−20
−10
0
10
20
−20
−15
−10
−5
0
5
10
15
20

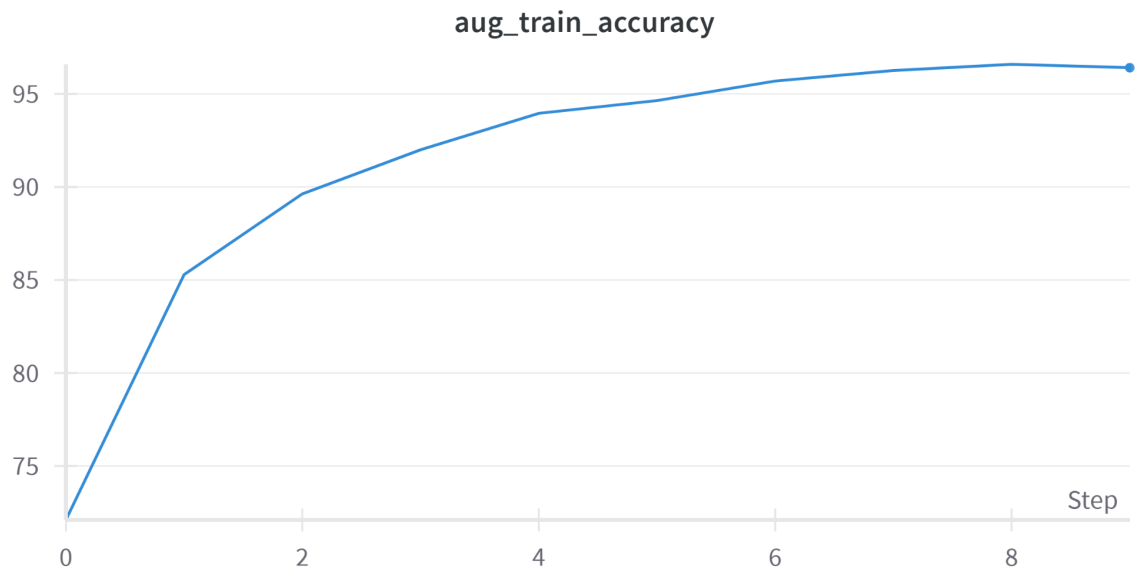
4)

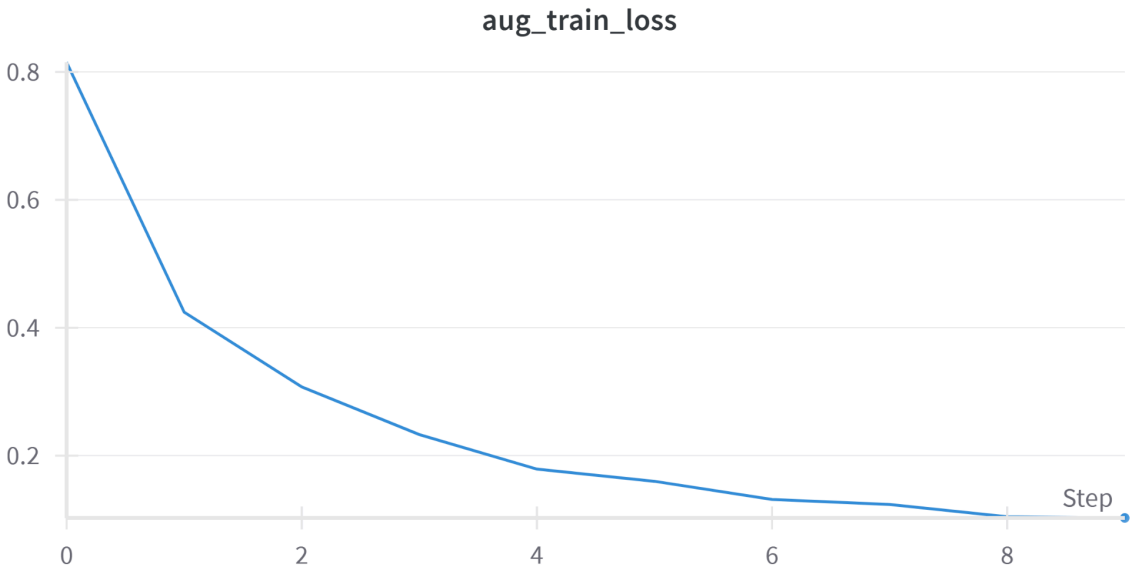
a)

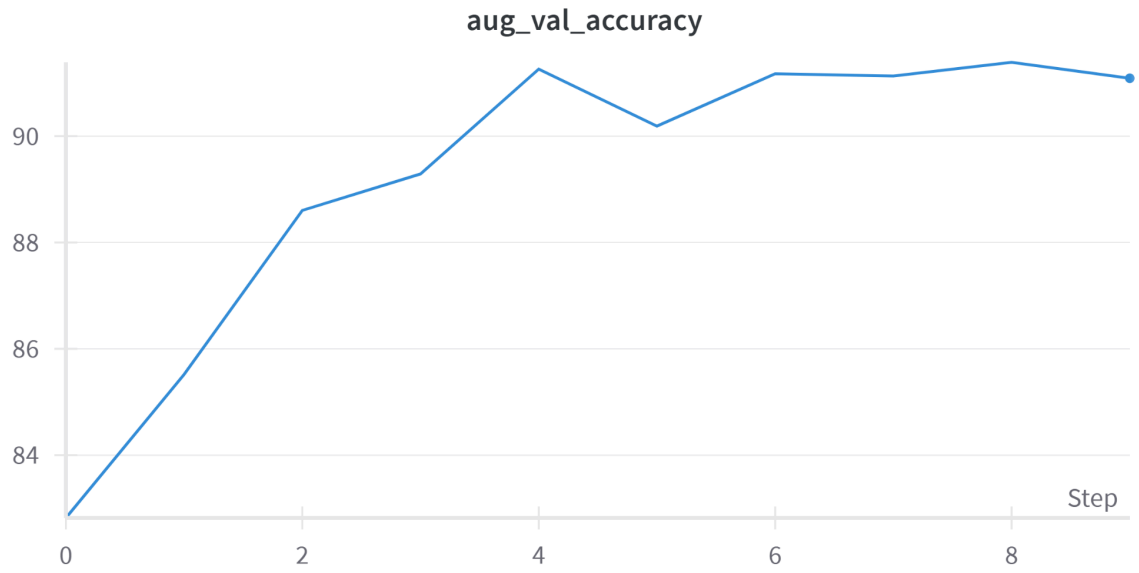
```
augmentation_transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],  std=[0.229, 0.224, 0.225])
])
```

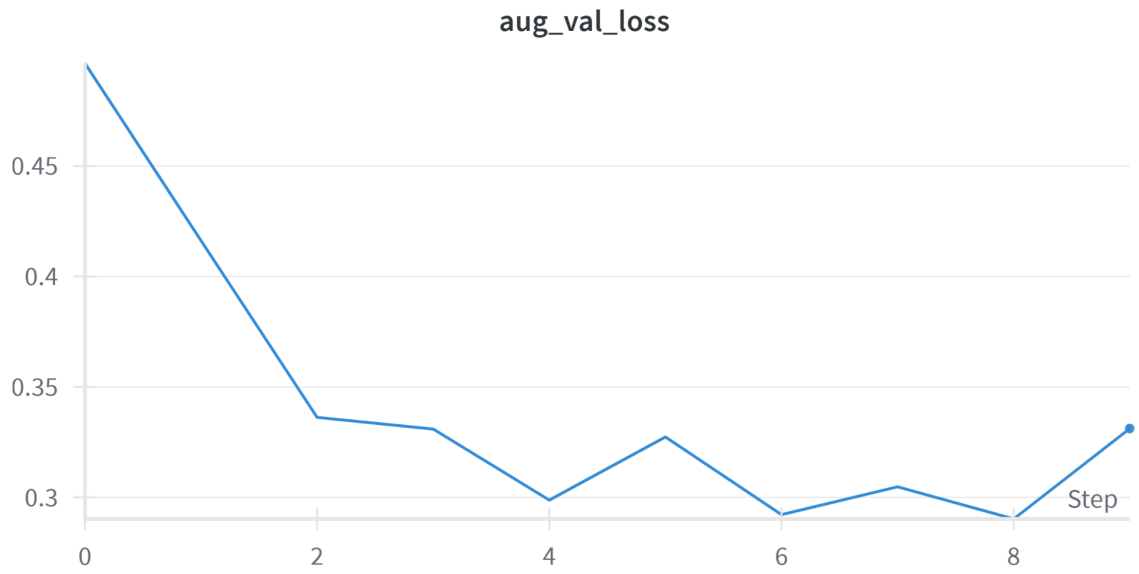
b)


aug_resnet_f1_score
1.5
1
0.5
0
Step
0
5
10
15
20


aug_train_accuracy
95
90
85
80
75
Step
0
2
4
6
8


aug_train_loss
0.8
0.6
0.4
0.2
Step
0
2
4
6
8





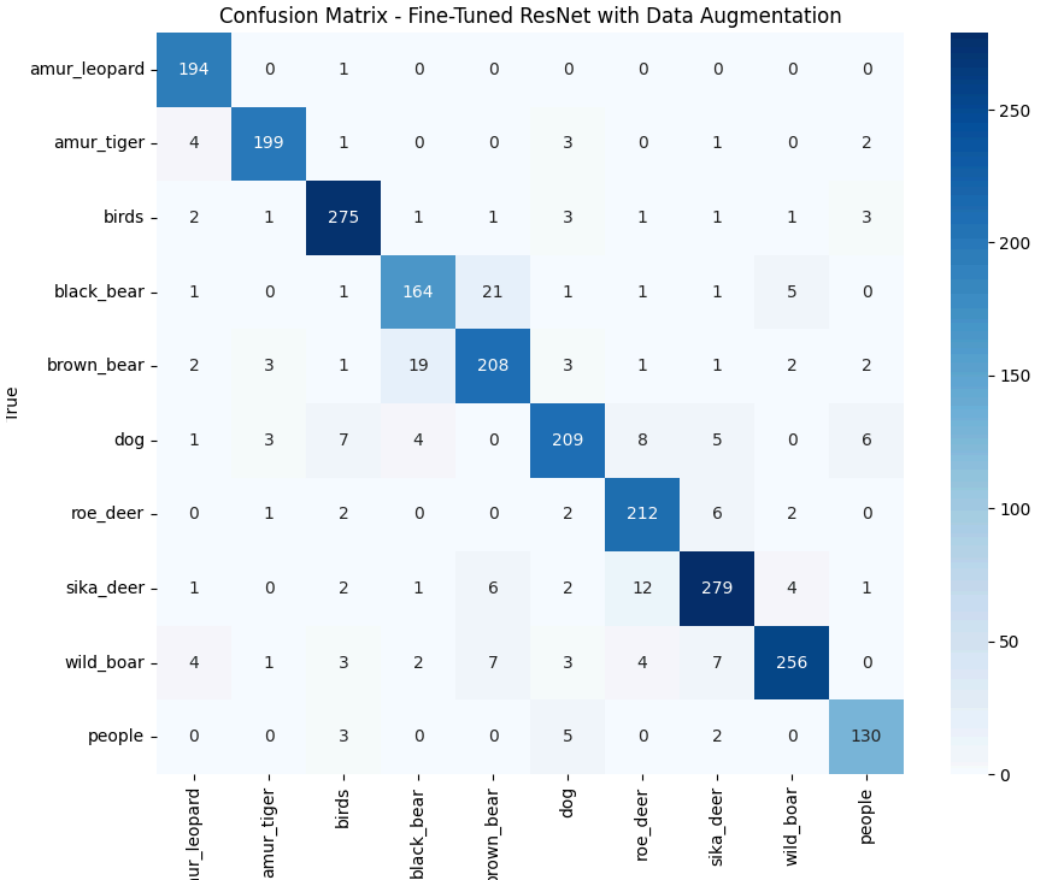
c)

The model doesn't appear to be overfitting in this case. While the training accuracy consistently increases, the validation accuracy also improves and stabilizes at around 91%. Both the training and validation loss show a steady decrease, indicating that the model is learning general patterns rather than memorizing the training data. The performance gap between training and validation metrics remains relatively small, which suggests the use of data augmentation has helped in regularizing the model and preventing overfitting.

d)

Run summary:

aug_resnet_f1_score	0.91061
aug_train_accuracy	96.41097
aug_train_loss	0.10263
aug_val_accuracy	91.08826
aug_val_loss	0.33117



5)

Among the three models, each has its strengths and weaknesses, but when considering the overall performance—particularly in terms of generalization and validation accuracy—**ResNet-18** with data augmentation stands out as the best.

CNN (Convolutional Neural Network):

The CNN starts with a low validation accuracy of 48.63% but steadily improves as the epochs progress, reaching 71.94% by the end. However, while the training accuracy reaches an impressive 95.19%, the validation accuracy stagnates around the 71-72% range, indicating that the model is overfitting. The performance on unseen data does not improve significantly despite the drastic improvement in training accuracy. This suggests that CNN, being a simpler model without residual connections, is not as effective for image classification tasks, especially when dealing with complex features in images. The gap between training and validation performance is a clear indicator of this overfitting.

ResNet-18:

The ResNet-18 performs significantly better than the CNN, with training accuracy climbing steadily to 98.90% by epoch 10. The validation accuracy, however, remains at a high 90.70%, and its validation F1-score of 0.9070 further emphasizes its robustness. The ResNet-18, by leveraging convolutional layers, is better equipped to extract spatial hierarchies in image data, leading to better generalization. Despite some fluctuations in validation loss, the model exhibits stable and consistent performance across epochs. This consistency in both training and validation results demonstrates that the ResNet-18 is a solid choice, showing good generalization on unseen data.

ResNet-18 (Pre-trained + Fine-tuned):

The ResNet-18 (Pre-trained + Fine-tuned) model, when fine-tuned with data augmentation, achieves the highest validation accuracy at 91.09%. Its training accuracy, although slightly lower than that of the ResNet-18(96.41%), is still quite high. The key strength of ResNet-18 (Pre-trained + Fine-tuned) lies in its architecture—residual connections help mitigate the vanishing gradient problem, allowing the model to learn more complex features without suffering from diminishing returns during training. Additionally, the use of data augmentation helps prevent overfitting by artificially increasing the diversity of the training data. The model's validation loss remains relatively stable, and its accuracy continues to improve steadily throughout the epochs. While it does not exceed Resnet-18 performance in training accuracy, it achieves slightly better validation performance and is more resistant to overfitting.

Conclusion:

While the CNN shows strong improvement during training, it struggles to generalize effectively to unseen data, as seen in the plateauing of its validation accuracy. The ResNet-18, on the other hand, provides solid and reliable performance, achieving a high validation accuracy of 90.70%. However, **ResNet-18 with data augmentation** emerges as the best model overall. Its use of a pre-trained model with fine-tuning, combined with data augmentation, results in the highest validation accuracy of 91.09%, showing it excels at generalizing to new data. The ResNet-18's stability and relatively higher validation performance, without overfitting, make it the optimal choice for this task, especially when balancing both training efficiency and validation performance

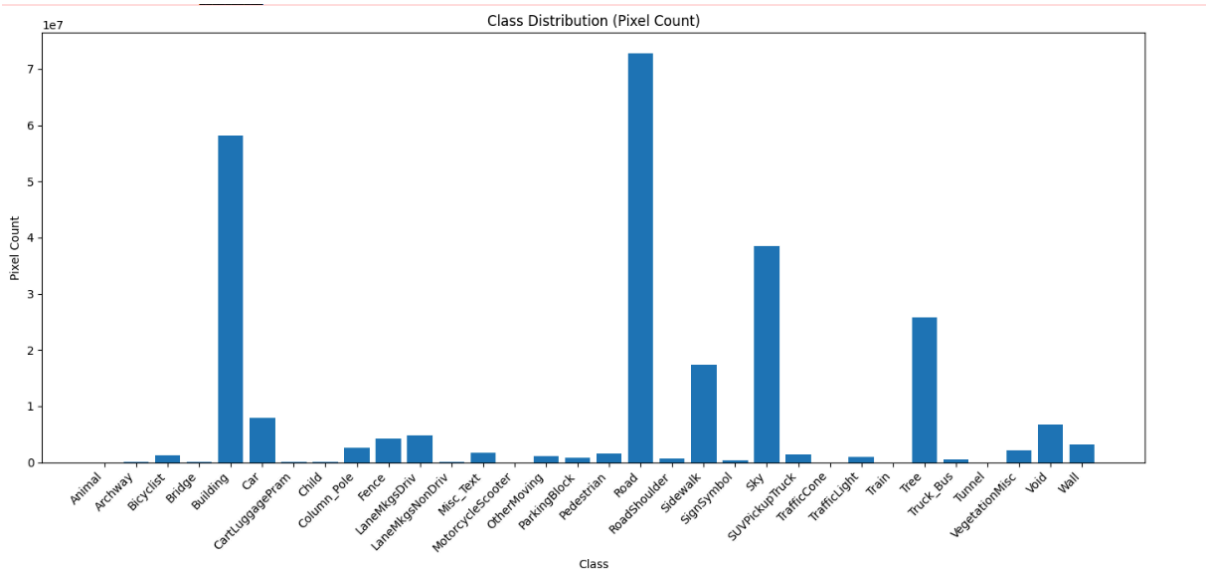
Image Segmentation

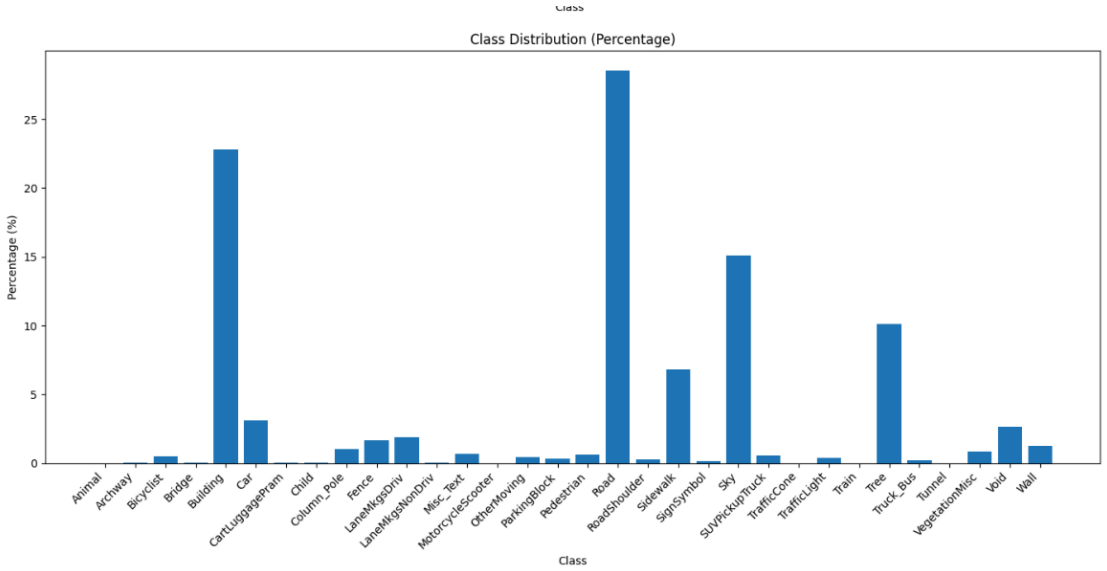
1.

(a)

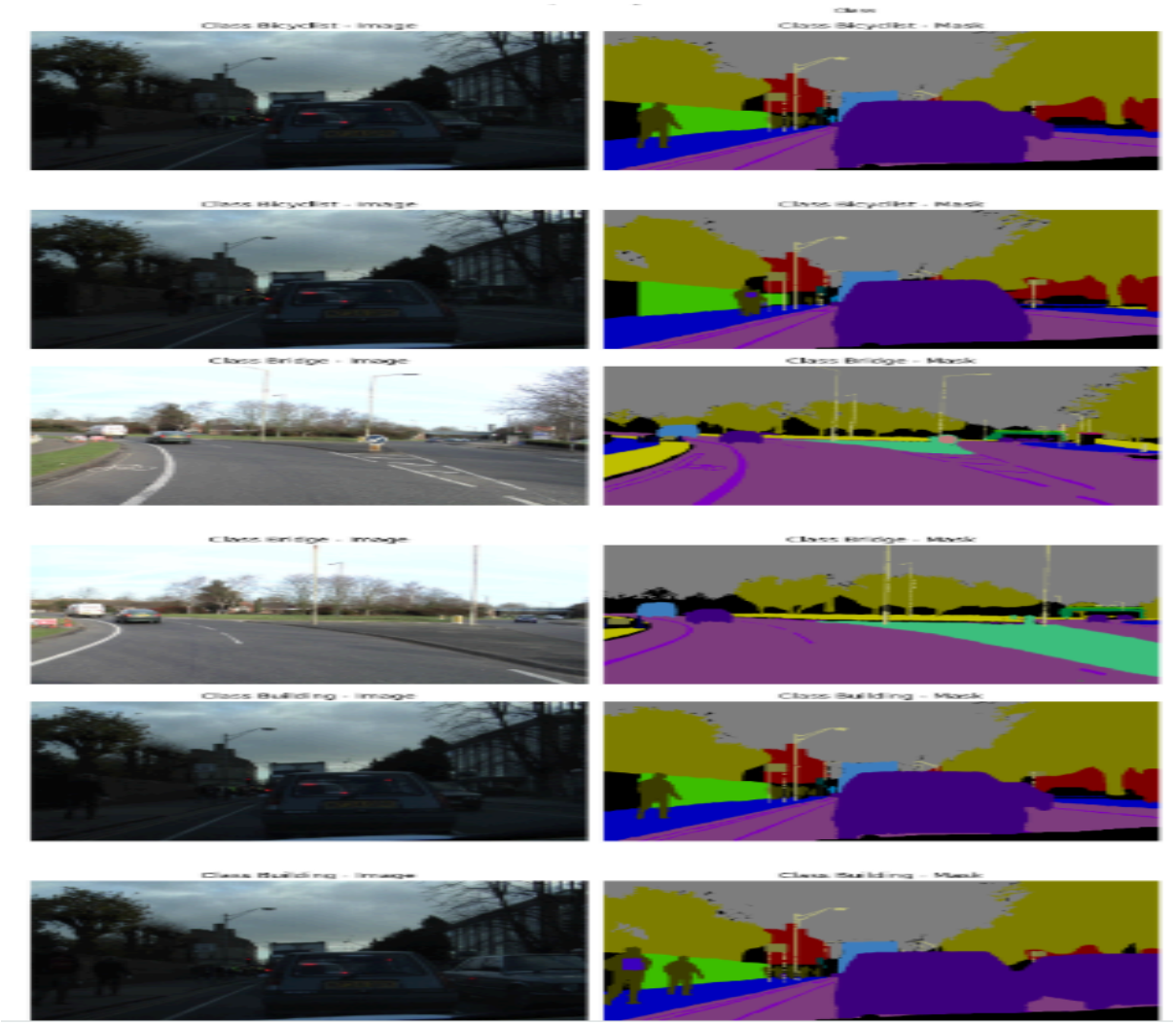
```python
# Dataset class for CamVid.
class CamVidDataset(Dataset):
    def __init__(self, image_dir, label_dir, transform=None, label_transform=None):
        self.image_dir = image_dir
        self.label_dir = label_dir
        self.transform = transform
        self.label_transform = label_transform
        self.images = sorted([f for f in os.listdir(image_dir) if f.endswith(".png")])
        self.labels = sorted([f for f in os.listdir(label_dir) if f.endswith(".png")])
        assert len(self.images) == len(self.labels), "Mismatch between image and label count"

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

    def __getitem__(self, idx):
        img_path = os.path.join(self.image_dir, self.images[idx])
        label_path = os.path.join(self.label_dir, self.labels[idx])
        image = Image.open(img_path).convert("RGB")
        label = Image.open(label_path).convert("RGB")  # keep as RGB for conversion
        if self.transform:
            image = self.transform(image)
        if self.label_transform:
            label = self.label_transform(label)
        else:
            label = torch.as_tensor(np.array(label), dtype=torch.long)
        return image, label

def get_dataloader(image_dir, label_dir, batch_size=11, shuffle=True):
    dataset = CamVidDataset(image_dir, label_dir, transform=img_transform, label_transform=label_transform)
    return DataLoader(dataset, batch_size=batch_size, shuffle=shuffle, num_workers=4, pin_memory=True)

# ----------------------------
```

(b)




Class Distribution (Percentage)
Percentage (%)
0
5
10
15
20
25
Animal
Archway
Bicyclist
Bridge
Building
Car
CartLuggagePram
Child
Column_Pole
Fence
LaneMkgsDriv
LaneMkgsNonDriv
Misc_Text
MotorcycleScooter
OtherMoving
ParkingBlock
Pedestrian
Road
RoadShoulder
Sidewalk
SignSymbol
Sky
SUVPickupTruck
TrafficCone
TrafficLight
Train
Tree
Truck_Bus
Tunnel
VegetationMisc
Void
Wall
Class

(c)



Class Car - Image

Class Car - Mask

Class Car - Image

Class Car - Mask

Class CartLuggagePram - Image

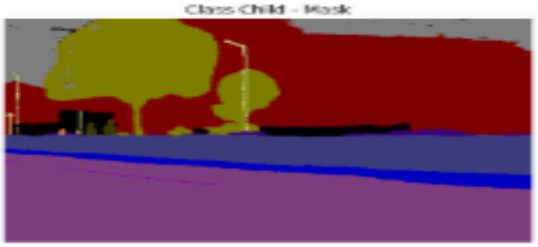
Class CartLuggagePram - Mask

Class CartLuggagePram - Image

Class CartLuggagePram - Mask

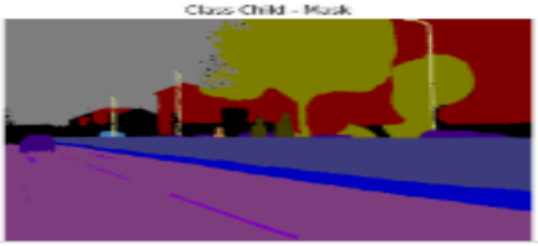
Class Child - Image

Class Child - Mask



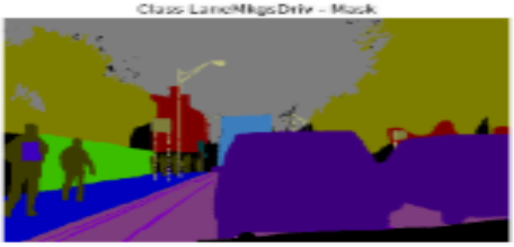
Class Child - Image

Class Child - Mask



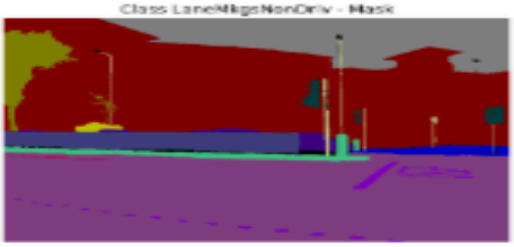
Class LaneMkgsDriv - Image

Class LaneMkgsDriv - Mask



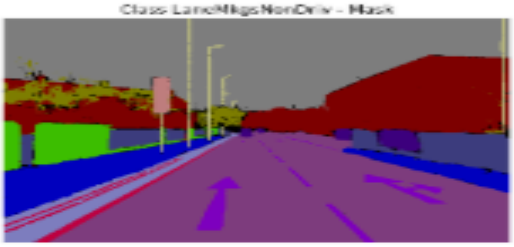
Class LaneMkgsNonDriv - Image

Class LaneMkgsNonDriv - Mask



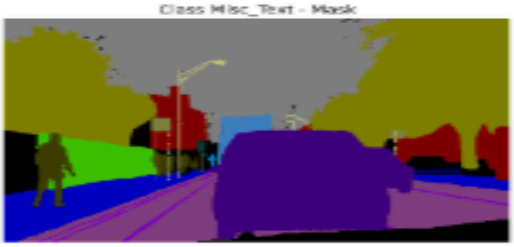
Class LaneMkgsNonDriv - Image

Class LaneMkgsNonDriv - Mask



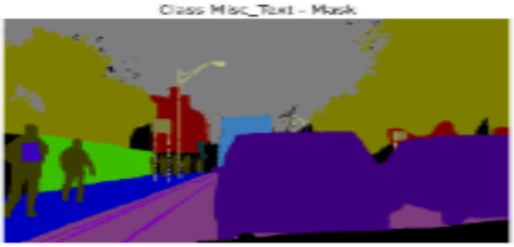
Class Misc_Text - Image

Class Misc_Text - Mask



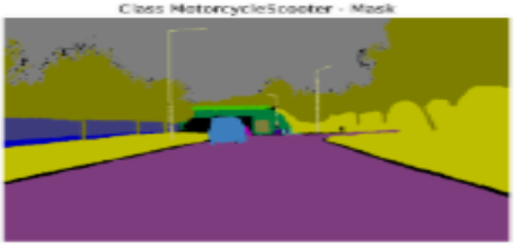
Class Misc_Text - Image

Class Misc_Text - Mask



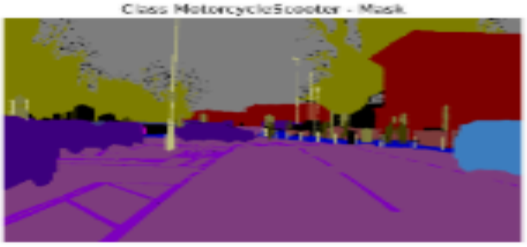
Class MotorcycleScooter - Image

Class MotorcycleScooter - Mask



Class MotorcycleScooter - Image

Class MotorcycleScooter - Mask



Class OtherMoving - Image

Class OtherMoving - Mask

Class OtherMoving - Image

Class OtherMoving - Mask

Class ParkingBlock - Image

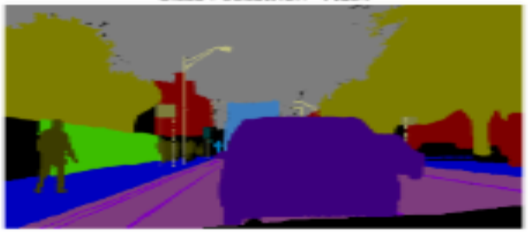
Class ParkingBlock - Mask

Class ParkingBlock - Image

Class ParkingBlock - Mask

Class Pedestrian - Image

Class Pedestrian - Mask

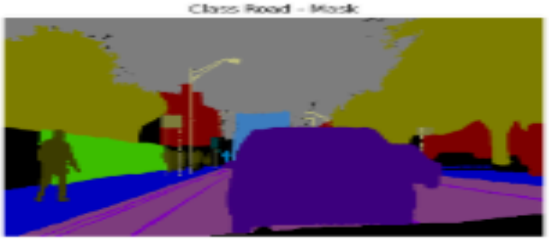


Class Pedestrian - Image

Class Pedestrian - Mask

Class Road - Image

Class Road - Mask



Class Road - Image

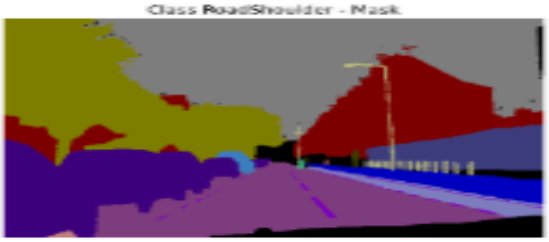
Class Road - Mask

Class RoadShoulder - Image

Class RoadShoulder - Mask

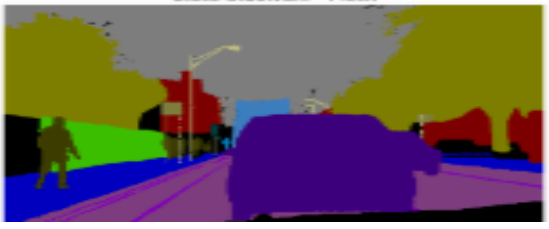
Class RoadShoulder - Image

Class RoadShoulder - Mask



Class Sidewalk - Image

Class Sidewalk - Mask

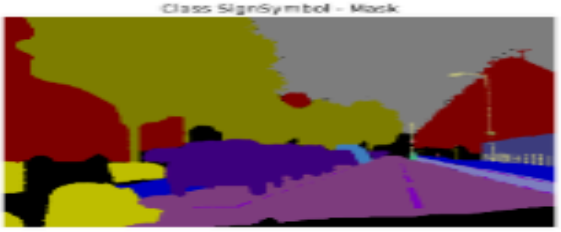


Class Sidewalk - Image

Class Sidewalk - Mask

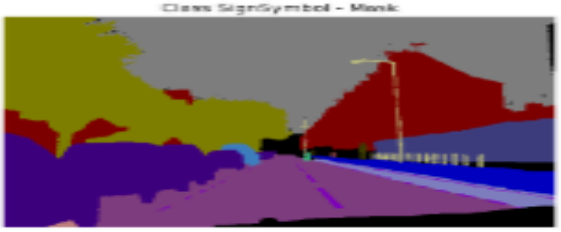
Class SignSymbol - Image

Class SignSymbol - Mask



Class SignSymbol - Image

Class SignSymbol - Mask



Class Sky - Image

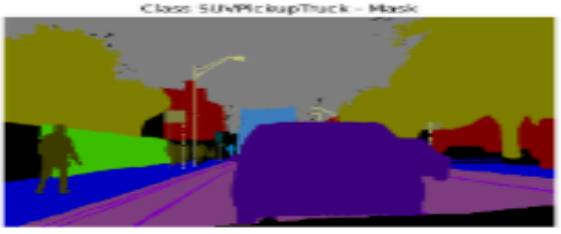
Class Sky - Mask

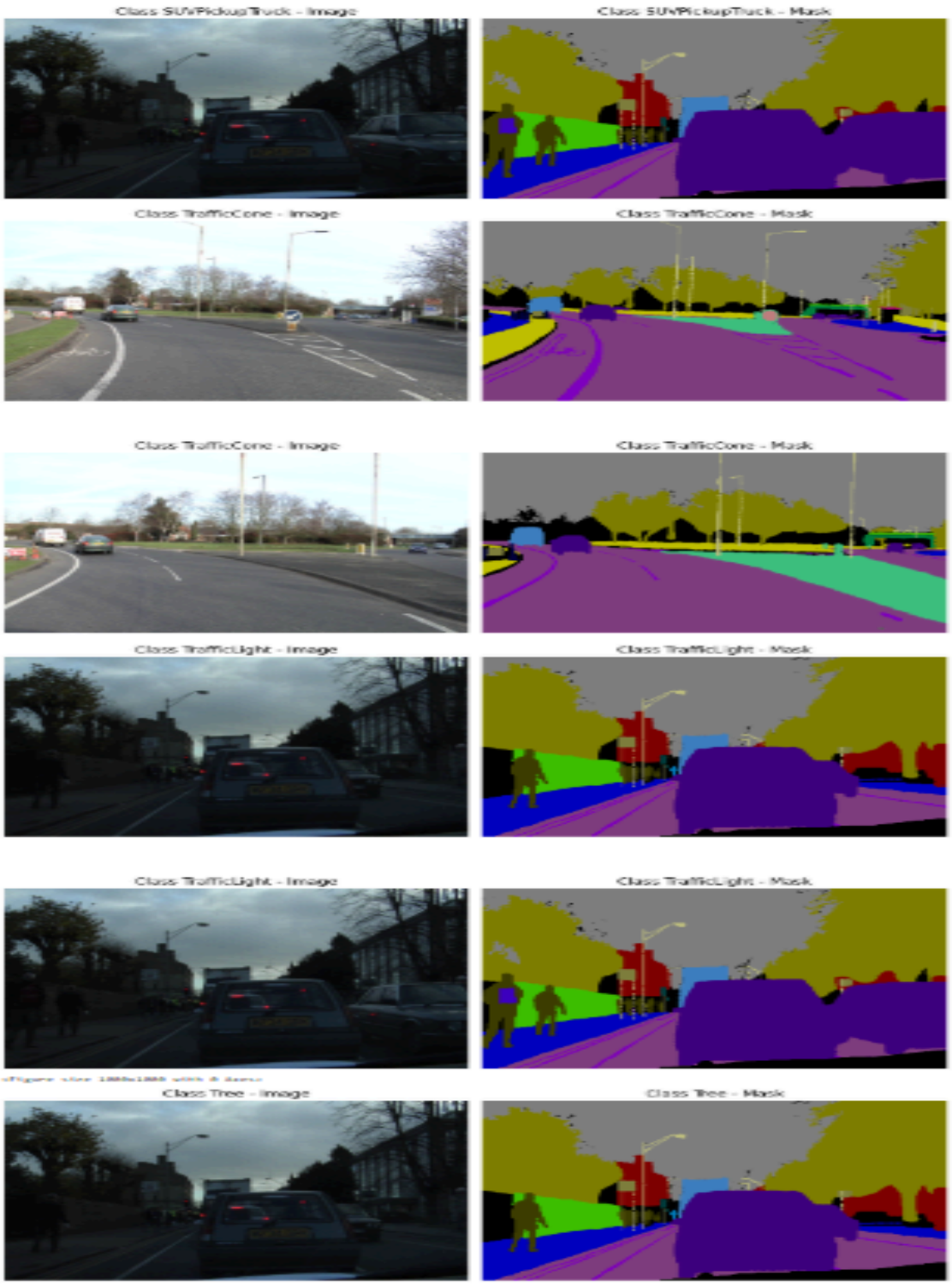
Class Sky - Image

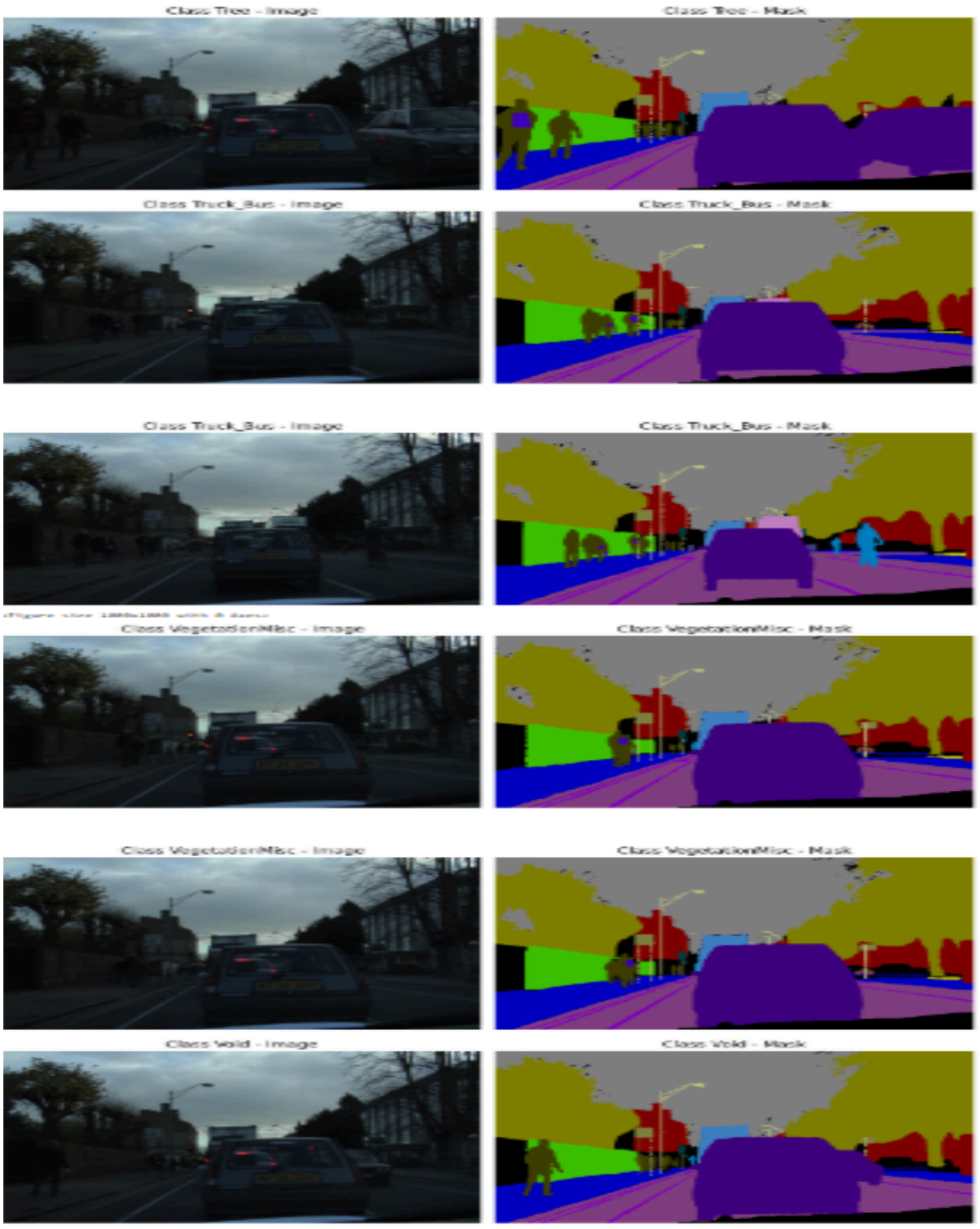
Class Sky - Mask

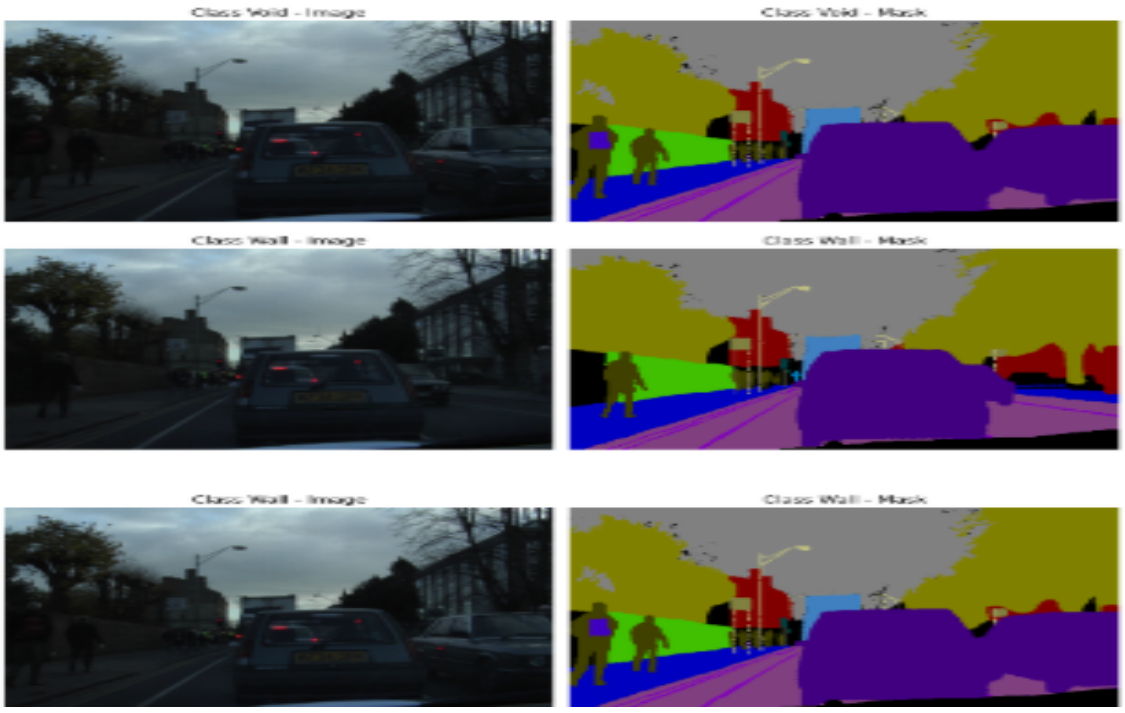
Class SUVPickupTruck - Image

Class SUVPickupTruck - Mask



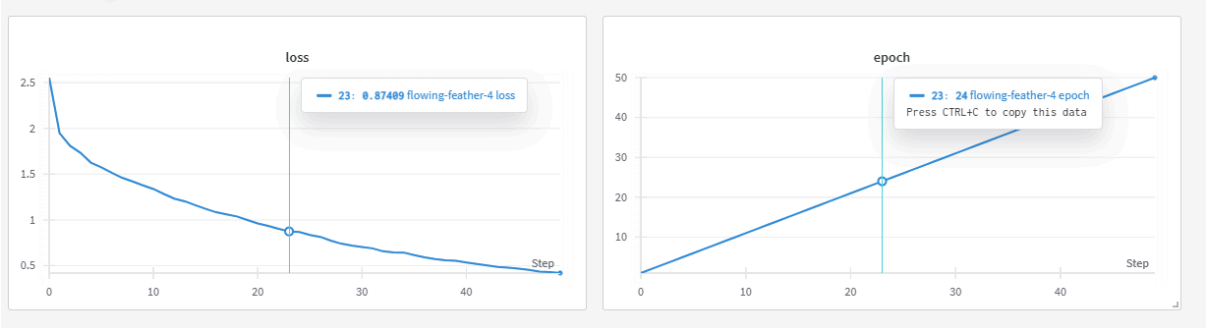

Class SUVPickupTruck - Image
Class SUVPickupTruck - Mask
Class TrafficCone - Image
Class TrafficCone - Mask
Class TrafficCone - Image
Class TrafficCone - Mask
Class TrafficLight - Image
Class TrafficLight - Mask
Class TrafficLight - Image
Class TrafficLight - Mask
Class Tree - Image
Class Tree - Mask


Class Tree - Image
Class Tree - Mask
Class Truck_Bus - Image
Class Truck_Bus - Mask
Class Truck_Bus - Image
Class Truck_Bus - Mask
Class VegetationMisc - Image
Class VegetationMisc - Mask
Class VegetationMisc - Image
Class VegetationMisc - Mask
Class Void - Image
Class Void - Mask



2.

(a)



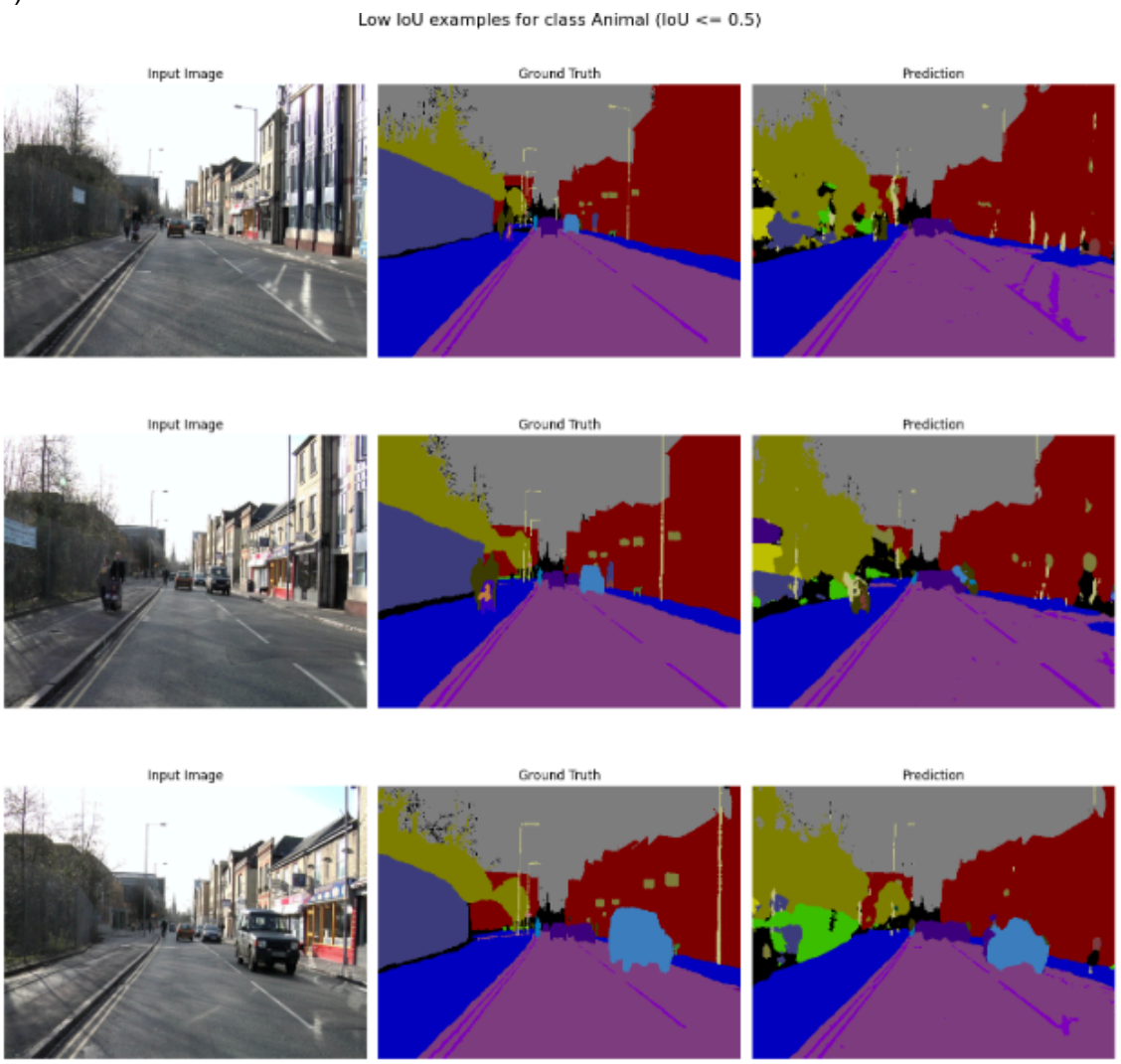
(b)
Class-wise Metrics:
Class Animal: {'pixel_accuracy': 0.9999414810823755, 'precision': 0.0, 'recall': 0.0, 'IoU': 0.0, 'dice': 0.0}
Class Archway: {'pixel_accuracy': 0.9997887232598979, 'precision': 0.13333333333244443, 'recall': 0.00023643456673365365, 'IoU': 0.00023607176581680553, 'dice': 0.00047203209818267087}
Class Bicyclist: {'pixel_accuracy': 0.9956088362068966, 'precision': 0.8096559238925436, 'recall': 0.3864968570822679, 'IoU': 0.35430390704088954, 'dice': 0.5232265892447023}
Class Bridge: {'pixel_accuracy': 0.999436711765645, 'precision': 0.0, 'recall': 0.0, 'IoU': 0.0, 'dice': 0.0}
Class Building: {'pixel_accuracy': 0.9327334520673691, 'precision': 0.8268329005318665, 'recall': 0.89200782706297, 'IoU': 0.7515967987018538, 'dice': 0.858184713809569}
Class Car: {'pixel_accuracy': 0.9830157197876757, 'precision': 0.7754127091250905, 'recall': 0.7733512028856018, 'IoU': 0.6318279344302007, 'dice': 0.7743805840054104}

Class CartLuggagePram: {'pixel_accuracy': 0.9996695152857599, 'precision': 0.0, 'recall': 0.0, 'IoU': 0.0, 'dice': 0.0}
Class Child: {'pixel_accuracy': 0.9997476901740102, 'precision': 0.7389100126741776, 'recall': 0.05556614563476882, 'IoU': 0.0544961675079449, 'dice': 0.10335963123836447}
Class Column_Pole: {'pixel_accuracy': 0.9881572278097063, 'precision': 0.31430630344740945, 'recall': 0.2623551063542123, 'IoU': 0.16685472620227948, 'dice': 0.28599057355723384}
Class Fence: {'pixel_accuracy': 0.9911188188457855, 'precision': 0.7298467722718934, 'recall': 0.31676877592544705, 'IoU': 0.28352487337429083, 'dice': 0.4417910073357989}
Class LaneMkgsDriv: {'pixel_accuracy': 0.9922547992496807, 'precision': 0.7458412179436934, 'recall': 0.6963092276278284, 'IoU': 0.5627742324308187, 'dice': 0.7202246117859917}
Class LaneMkgsNonDriv: {'pixel_accuracy': 0.999972810903576, 'precision': 0.0, 'recall': 0.0, 'IoU': 0.0, 'dice': 0.0}
Class Misc_Text: {'pixel_accuracy': 0.9921011683828225, 'precision': 0.1760692758126799, 'recall': 0.18440832856264436, 'IoU': 0.09898705363494092, 'dice': 0.18014234709596902}
Class MotorcycleScooter: {'pixel_accuracy': 0.9998120709610473, 'precision': 0.0, 'recall': 0.0, 'IoU': 0.0, 'dice': 0.0}
Class OtherMoving: {'pixel_accuracy': 0.9967918113425925, 'precision': 0.45457617244392606, 'recall': 0.32291545961694423, 'IoU': 0.23274015832582656, 'dice': 0.3775980797801037}
Class ParkingBlock: {'pixel_accuracy': 0.9962930785041507, 'precision': 0.4404120295840036, 'recall': 0.27279558937229464, 'IoU': 0.20257885191187036, 'dice': 0.3369073912946477}
Class Pedestrian: {'pixel_accuracy': 0.9912791846264368, 'precision': 0.31537837785029005, 'recall': 0.23561074942340807, 'IoU': 0.1558826298901194, 'dice': 0.2697205163554329}
Class Road: {'pixel_accuracy': 0.9737889876676246, 'precision': 0.9502828629664487, 'recall': 0.9432591533306692, 'IoU': 0.8988987954610203, 'dice': 0.946757981636175 5}
Class RoadShoulder: {'pixel_accuracy': 0.9978459750159643, 'precision': 0.49013561656138593, 'recall': 0.3850604449880046, 'IoU': 0.2749332482493406, 'dice': 0.4312904203053171}
Class Sidewalk: {'pixel_accuracy': 0.9780437569843551, 'precision': 0.7851803496401989, 'recall': 0.8503956567867917, 'IoU': 0.6898854655908815, 'dice': 0.8164878385407708}
Class SignSymbol: {'pixel_accuracy': 0.9991383301404854, 'precision': 0.7171111777045208, 'recall': 0.2863193663404238, 'IoU': 0.25726203530499414, 'dice': 0.40924171426616884}
Class Sky: {'pixel_accuracy': 0.984325161637931, 'precision': 0.9626395735075463, 'recall': 0.9447308345705426, 'IoU': 0.9113170476442352, 'dice': 0.9536011294070391}
Class SUVPickupTruck: {'pixel_accuracy': 0.985874067089719, 'precision': 0.18624110252915335, 'recall': 0.1409128907338491, 'IoU': 0.08721462512008148, 'dice': 0.16043681367963336}
Class TrafficCone: {'pixel_accuracy': 0.9999904214559387, 'precision': 0.0, 'recall': 0.0, 'IoU': 0.0, 'dice': 0.0}
Class TrafficLight: {'pixel_accuracy': 0.9973697667225415, 'precision': 0.8191033806194415, 'recall': 0.3592296660520383, 'IoU': 0.33282504571424937, 'dice': 0.4994279583572671}
Class Train: {'pixel_accuracy': 1.0, 'precision': 0.0, 'recall': 0.0, 'IoU': 0.0, 'dice': 0.0}
Class Tree: {'pixel_accuracy': 0.9553498662994891, 'precision': 0.8112286602739083, 'recall': 0.774860398338982, 'IoU': 0.656489712764953, 'dice': 0.7926275758986321}
Class Truck_Bus: {'pixel_accuracy': 0.9890550666507024, 'precision': 0.6672114536004957, 'recall': 0.08889469558400834, 'IoU': 0.08512057914687593, 'dice': 0.15688685807396244}
Class Tunnel: {'pixel_accuracy': 1.0, 'precision': 0.0, 'recall': 0.0, 'IoU': 0.0, 'dice': 0.0}
Class VegetationMisc: {'pixel_accuracy': 0.9895716345386335, 'precision': 0.2691637719918436, 'recall': 0.4417751056360233, 'IoU': 0.20085139244910086, 'dice': 0.334514984471926}

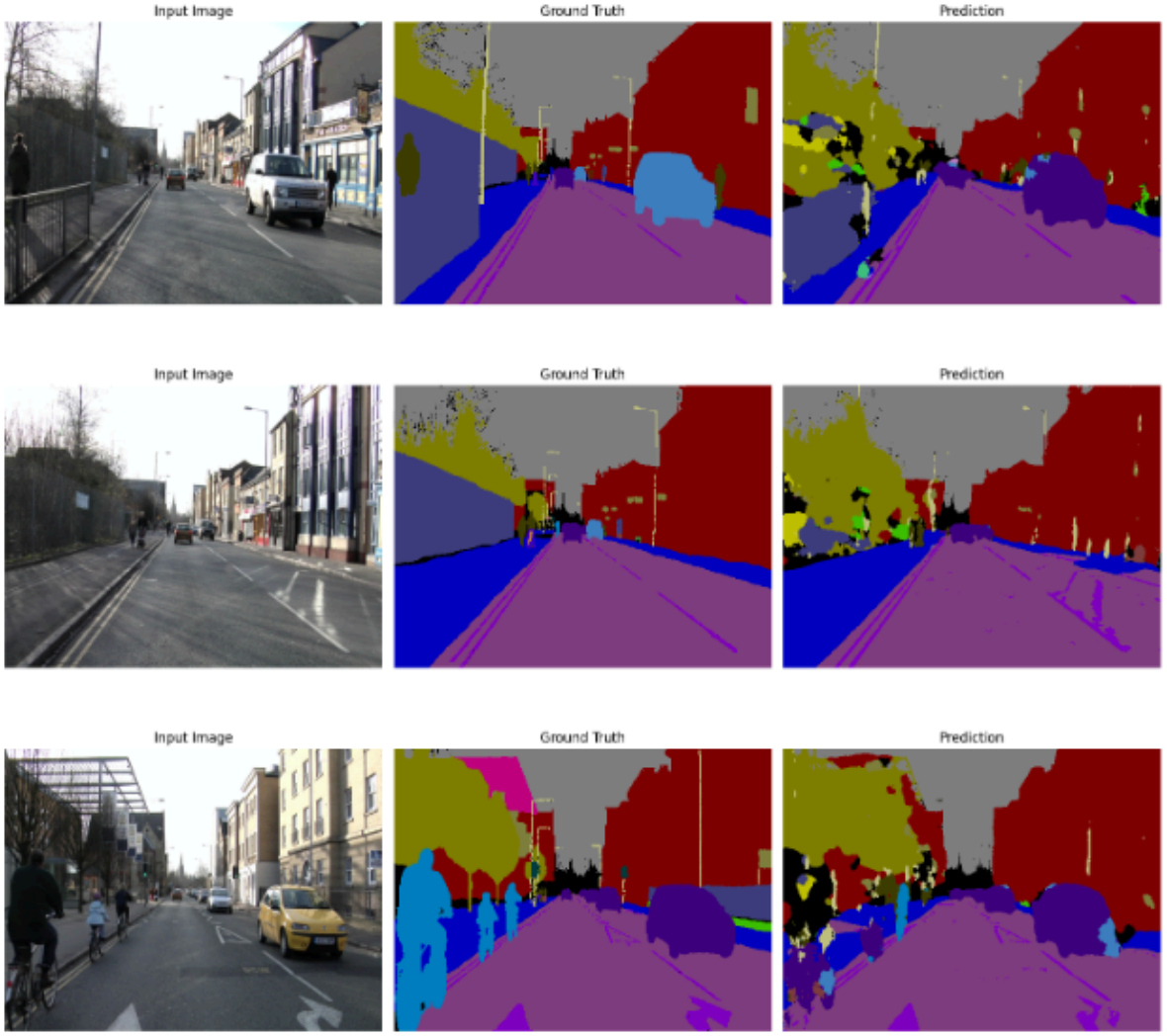
Class Void: {'pixel_accuracy': 0.947801873802682, 'precision': 0.3896129806414062, 'recall': 0.607047459401483, 'IoU': 0.3111419376756214, 'dice': 0.47461213578022005}
Class Wall: {'pixel_accuracy': 0.986559531649106, 'precision': 0.6237851105992743, 'recall': 0.4099127448466624, 'IoU': 0.3286598189900923, 'dice': 0.4947238025755983}

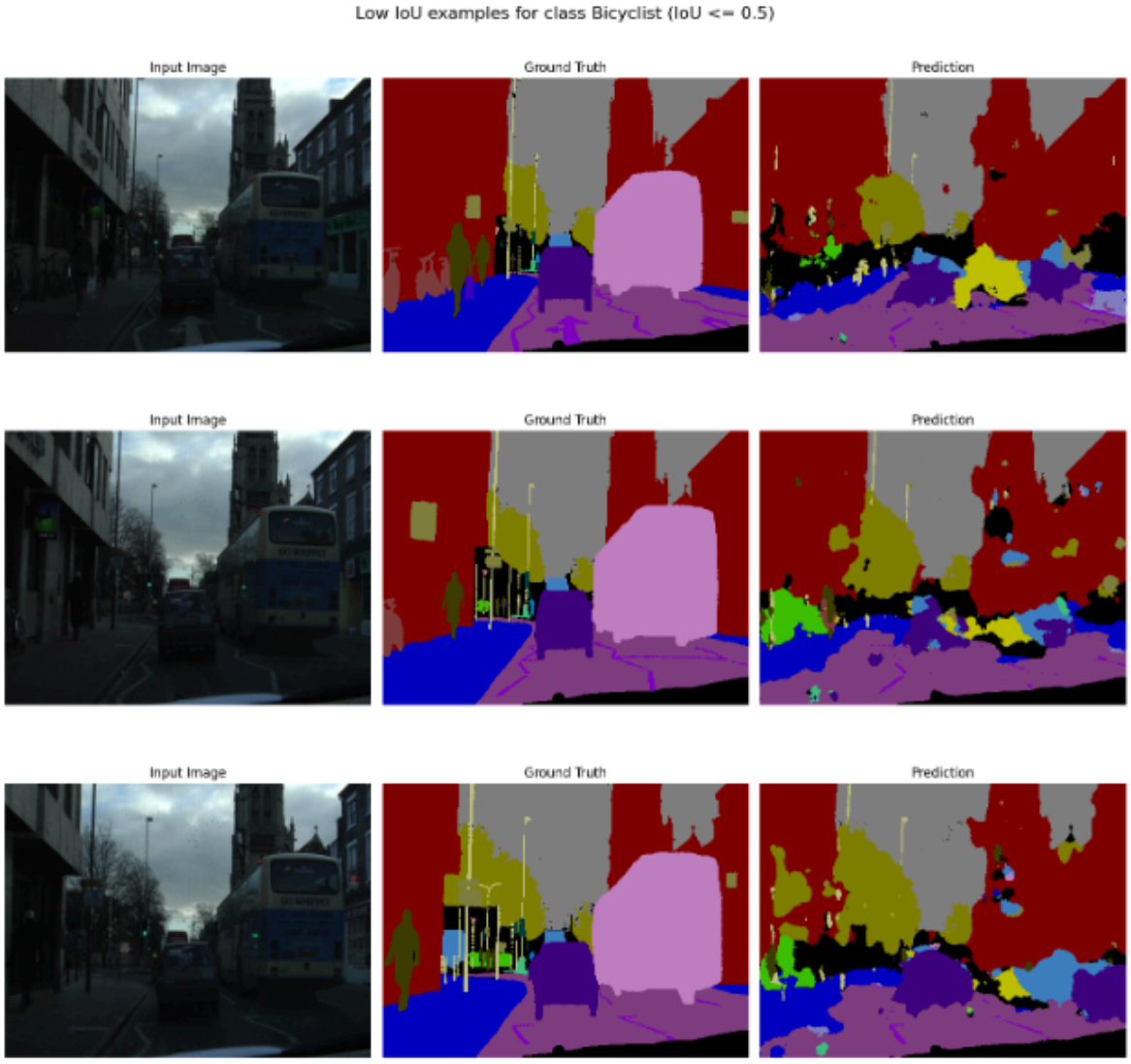
Overall Pixel Accuracy: 0.8212187699553001
Mean IoU: 0.2665750971665093

(c)



Low IoU examples for class Archway (IoU <= 0.5)

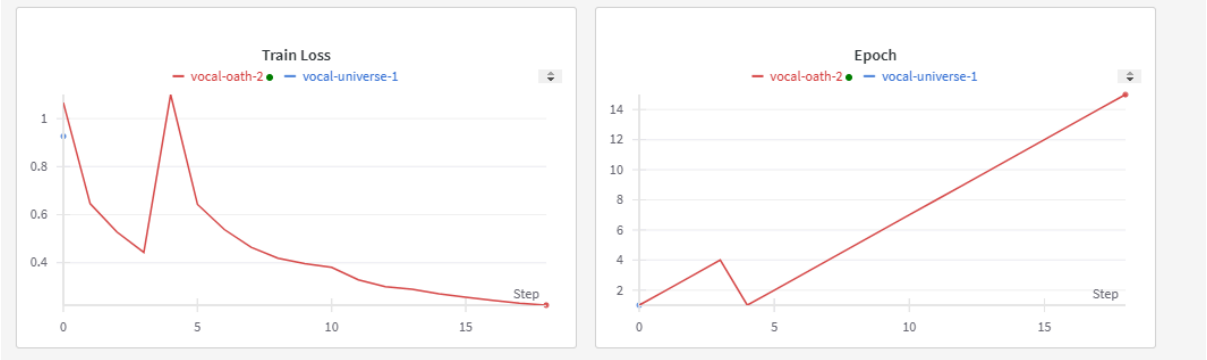




The model faces difficulties in several key scenarios. One challenge is **occlusion**, where objects are only partially visible, making it hard for the model to accurately segment them. **Misclassification** also occurs, particularly when the model confuses objects that look similar, such as **bicyclists** and **pedestrians**. **Environmental challenges** like low lighting or complex backgrounds further complicate segmentation, as these conditions can introduce noise and obscure object details. Additionally, **small object detection** is problematic, especially in datasets like **CamVid**, which contains small, thin objects like poles and signs that convolutional layers struggle to segment accurately. **Ambiguous boundaries** between objects, such as the unclear distinction between a **road** and a **sidewalk**, can also confuse the model. Finally, **SegNet** lacks advanced boundary refinement techniques, which limits its ability to precisely delineate object edges, further impacting its segmentation performance.

(3)

(a)



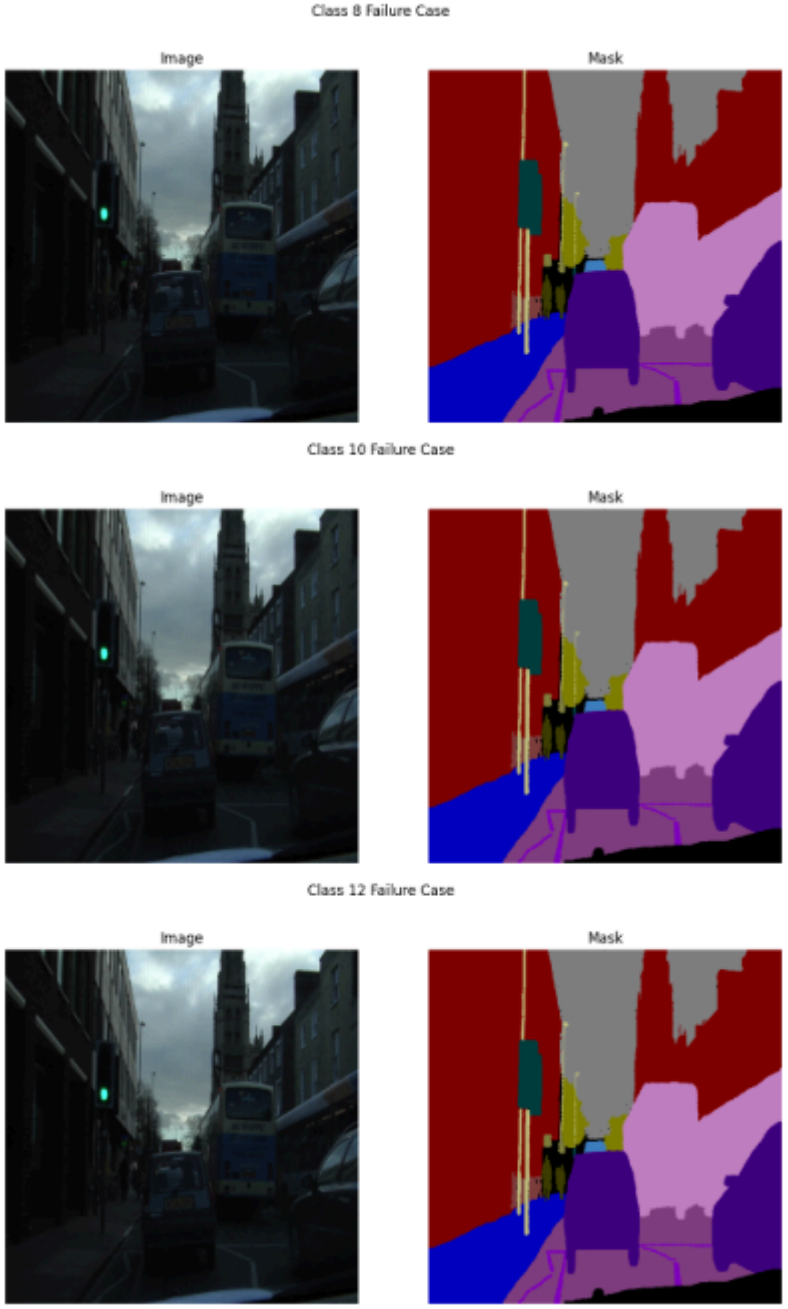
(b)

Test Set Performance:
Pixel Accuracy: 0.8807
Mean IoU: 0.6791
Per Class IoU: [0.95238096 0.8917749 0.5075472 0.95570566 0.77197657 0.68486181
0.60173162 0.87040384 0.18068587 0.45686966 0.41896198 0.99134199
0.32333073 0.98268398 0.50810361 0.79026584 0.28679955 0.91297917
0.79862877 0.74752261 0.51419151 0.91444969 0.54517388 0.98701299
0.44220817 1. 0.68364765 0.67296775 1. 0.47091816
0.35787251 0.50679621]
Per Class Dice: [0.95238096 0.8917749 0.54531433 0.95626029 0.84804075 0.77287543
0.60173162 0.88345276 0.28739446 0.49659677 0.57427278 0.99134199
0.40962782 0.98268398 0.53905827 0.81401228 0.38856463 0.95371973
0.8032288 0.82753995 0.5353462 0.95423515 0.60787743 0.98701299
0.50773314 1. 0.77813871 0.69540814 1. 0.5098495
0.48552763 0.59201035]
Per Class Precision: [0. 0. 0.64393734 0.07428707 0.86808458 0.81832954
0. 0.12092496 0.39418686 0.45300258 0.68352249 0.
0.493813 0. 0.54513741 0.41608639 0.52954257 0.9514756
0.13952143 0.8386974 0.43470724 0.96770363 0.41837636 0.
0.67360997 0. 0.78225926 0.34226105 0. 0.4171426
0.5704392 0.5744675]
Per Class Recall: [0. 0. 0.32118921 0.03673611 0.91913056 0.87182592
0. 0.04203407 0.32540682 0.38744366 0.57226899 0.
0.46996465 0. 0.20568244 0.24767466 0.44403736 0.95722581
0.11209875 0.91252844 0.20694002 0.94430012 0.29633039 0.
0.50830614 0. 0.90270146 0.18805993 0. 0.35885039
0.49808205 0.68501485]

IoU Bins Histogram:
{'0.0-0.1': 0, '0.1-0.2': 1, '0.2-0.3': 1, '0.3-0.4': 2, '0.4-0.5': 4, '0.5-0.6': 5, '0.6-0.7': 4, '0.7-0.8': 4, '0.8-0.9': 2, '0.9-1.0': 7}

Classes with IoU ≤ 0.5: [8, 9, 10, 12, 16, 24, 29, 30]

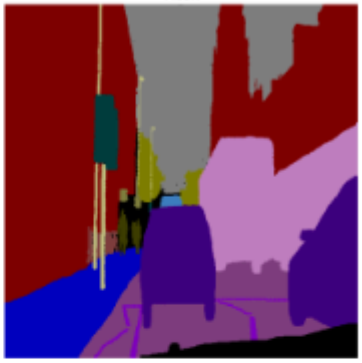
(c)



Class 16 Failure Case

Image

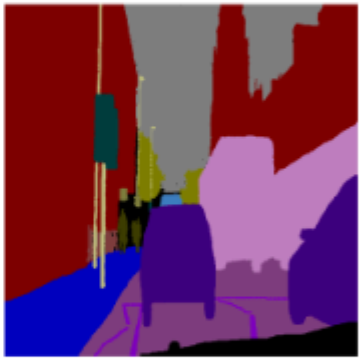
Mask



Class 24 Failure Case

Image

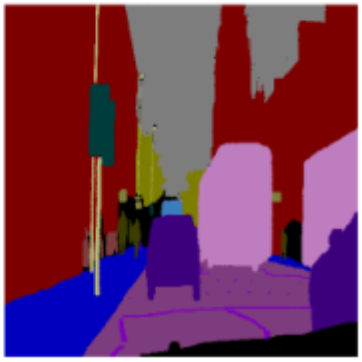
Mask



Class 10 Failure Case

Image

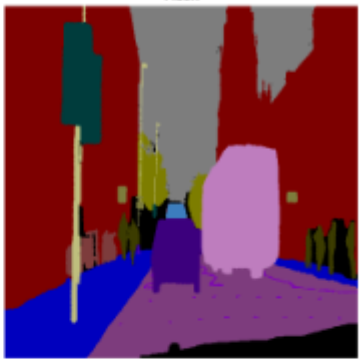
Mask



Class 12 Failure Case

Image

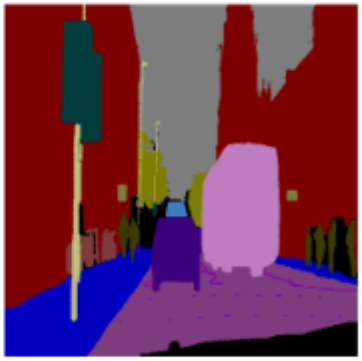
Mask



Class 16 Failure Case

Image

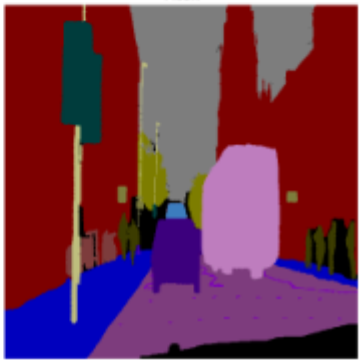
Mask



Class 24 Failure Case

Image

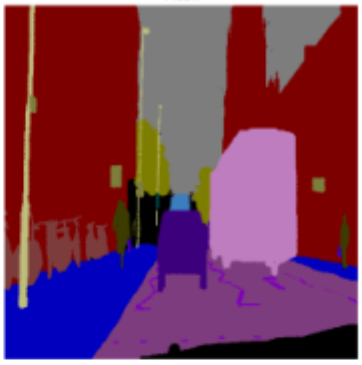
Mask



Class 8 Failure Case

Image

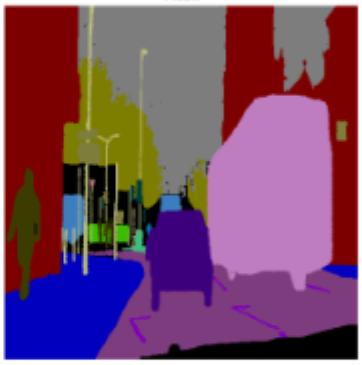
Mask



Class 8 Failure Case

Image

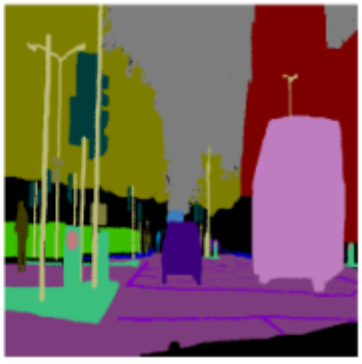
Mask



Class 30 Failure Case

Image

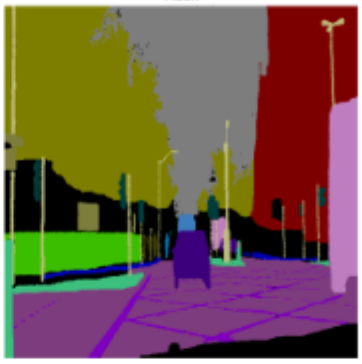
Mask



Class 30 Failure Case

Image

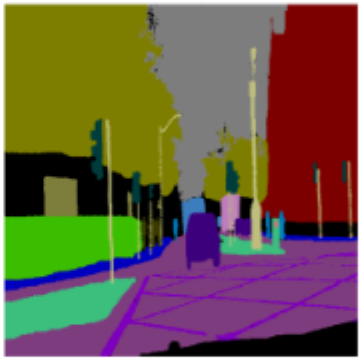
Mask



Class 30 Failure Case

Image

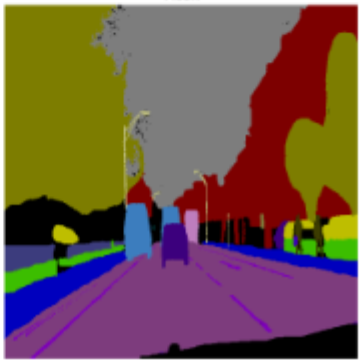
Mask



Class 9 Failure Case

Image

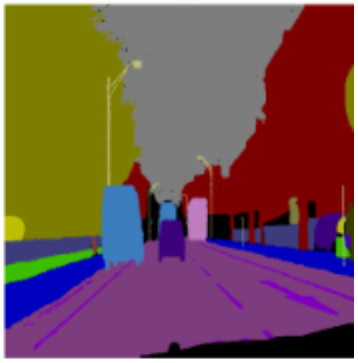
Mask



Class 9 Failure Case

Image

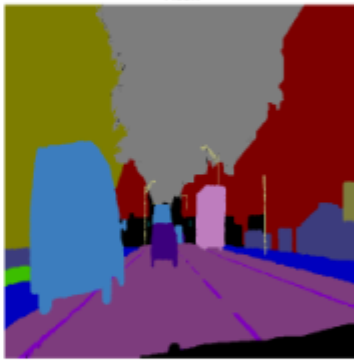
Mask



Class 9 Failure Case

Image

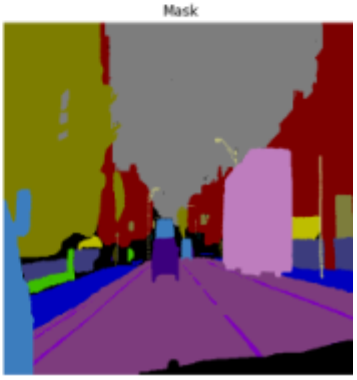
Mask

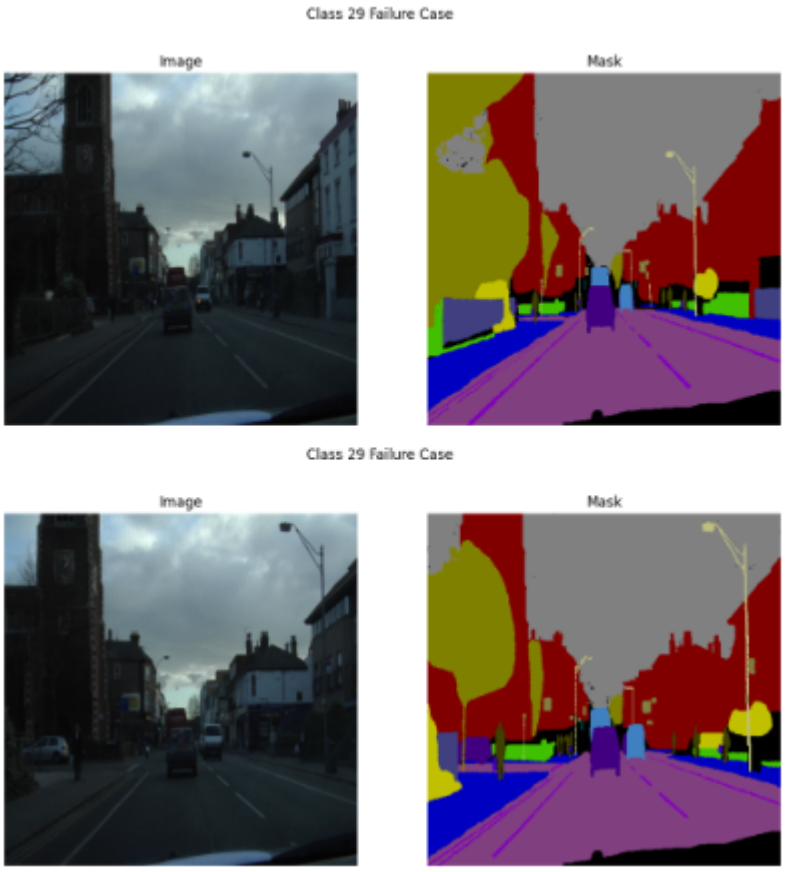


Class 29 Failure Case

Image

Mask





(c)(2)
Overall, DeepLabv3 outperforms SegNet in many cases, particularly in semantic segmentation tasks. However, it still faces challenges in certain situations. One such challenge is occlusion, where objects are partially visible, making it difficult for the model to correctly segment them. Additionally, misclassification is another issue, especially when the model confuses visually similar objects, like bicyclists and pedestrians, leading to errors in segmentation. Environmental challenges, such as low lighting or complex backgrounds, can also affect the model's performance, as these conditions introduce noise and ambiguity into the images. While DeepLabv3 is known for handling object boundaries well, it may still struggle with overlapping objects, where it has difficulty distinguishing between objects that are close together or partially obscured. Despite these limitations, DeepLabv3 remains a strong performer overall in many segmentation tasks.

Object Detection and Multi-Object Tracking

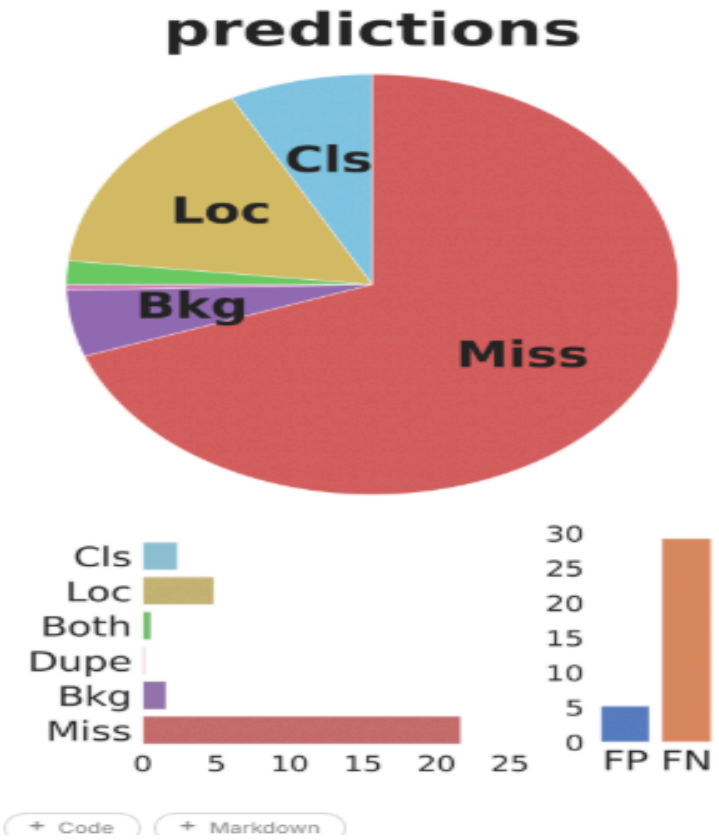
b)

```
loading annotations into memory...
Done (t=0.44s)
creating index...
index created!
Loading and preparing results...
DONE (t=1.27s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=11.07s).
Accumulating evaluation results...
DONE (t=2.17s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.313
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.422
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.345
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.113
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.343
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.480
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.260
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.362
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.366
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.127
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.398
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.567
```

```
loading annotations into memory...
Done (t=1.41s)
creating index...
index created!
Loading and preparing results...
DONE (t=0.15s)
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=12.50s).
Accumulating evaluation results...
DONE (t=2.32s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.480
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.616
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.525
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.284
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.534
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.667
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.363
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.539
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.546
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.320
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.599
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.742
```

c)

```
-- predictions --

bbox AP @ 50: 61.62

                    Main Errors
=================================================================
  Type       Cls       Loc      Both      Dupe       Bkg      Miss
-----------------------------------------------------------------
   dAP      2.37      4.86      0.57      0.13      1.59     21.65
=================================================================

        Special Error
==============================
  Type    FalsePos    FalseNeg
------------------------------
   dAP        5.19       29.26
==============================
```



The output of the TIDE analysis reveals insightful details about the performance of the object detection model on the COCO dataset. The Average Precision (AP) at an IoU threshold of 50 is 61.62%, which gives a general sense of how well the model detects objects. However, the error analysis highlights several key issues in the model's performance. The errors are categorized into different types, including **Type**, **Cls** (classification errors), **Loc** (location errors), **Both** (errors that affect both classification and localization), **Dupe** (duplicate detections), **Bkg** (background errors), and **Miss** (missed detections). The **Miss** error is particularly significant, with a high value of 21.65%, suggesting that the model fails to detect a large number of objects. The **FalsePos** (false positives) error is 5.19%, while the **FalseNeg** (false negatives) error is much higher at 29.26%, showing that the model tends to miss a substantial portion of the objects in the dataset. Furthermore, the **dAP** (differential AP) values for each error type suggest that misclassifications and missed detections are the most prominent issues. Understanding these specific error types is crucial for improving the

model’s performance in future iterations, especially by addressing the high missed detection rate and minimizing false negatives.

d)

```
Expected Calibration Error (ECE): 0.07381181388059795
```

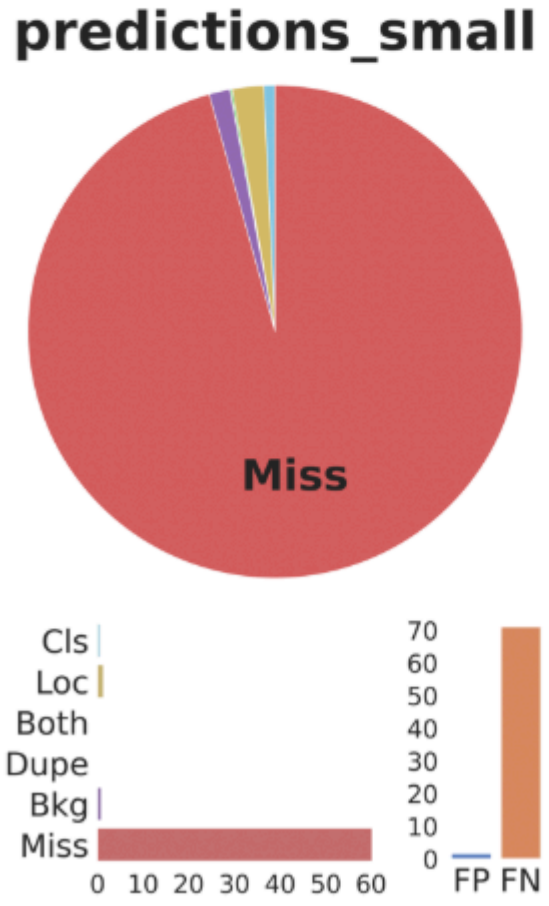
The **Expected Calibration Error (ECE)** value for my model is **0.0738**. This metric tells me how well the model's predicted probabilities align with the actual correctness of the predictions. In simple terms, if the model predicts an object with a confidence of, say, 80%, we expect that it should be correct 80% of the time if it's well-calibrated.

An ECE of **0.0738** suggests that the model's confidence is fairly decent, but it's not perfectly calibrated. While this isn't a very high value, it indicates that there’s still a bit of mismatch between the predicted probabilities and the actual likelihood of correctness. Essentially, there are areas where the model is overconfident or underconfident in its predictions.

This means that while the model's confidence scores are mostly reliable, there's still some room for improvement. If I were to apply this model in high-stakes situations (like medical diagnoses or autonomous driving), I'd consider working on improving the calibration further. Techniques like Platt scaling or temperature scaling could be helpful in reducing this calibration error, making the model’s confidence more trustworthy and accurate.

e)

```
Evaluating TIDE statistics for small objects using predictions_small.json ...
-- predictions_small --

bbox AP @ 50: 9.39

                        Main Errors
============================================================
  Type      Cls      Loc      Both     Dupe      Bkg      Miss
------------------------------------------------------------
   dAP     0.46     1.28     0.08     0.02     0.85     60.17
============================================================

       Special Error
============================
  Type   FalsePos   FalseNeg
----------------------------
   dAP       1.55      70.95
============================
```



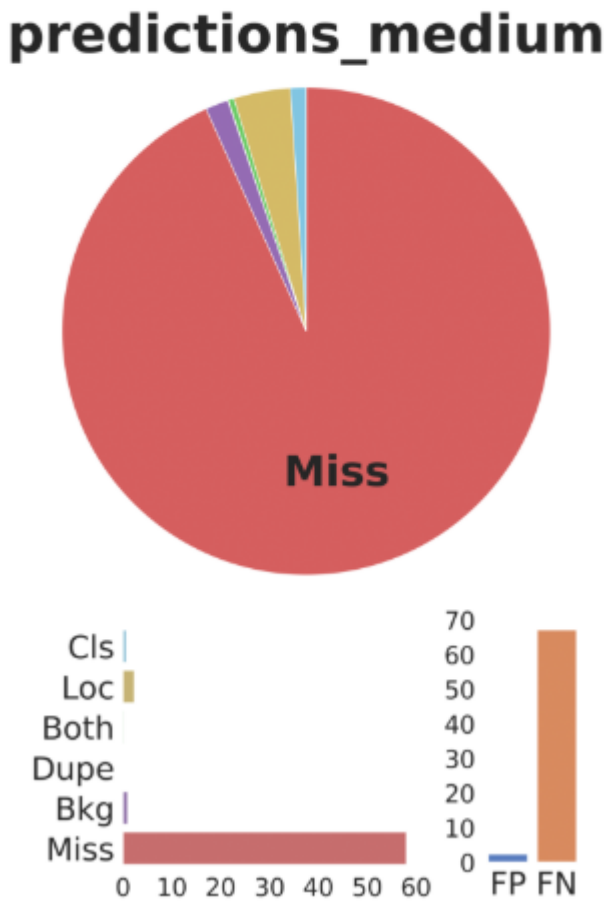
bbox AP @ 50: 22.19

Main Errors

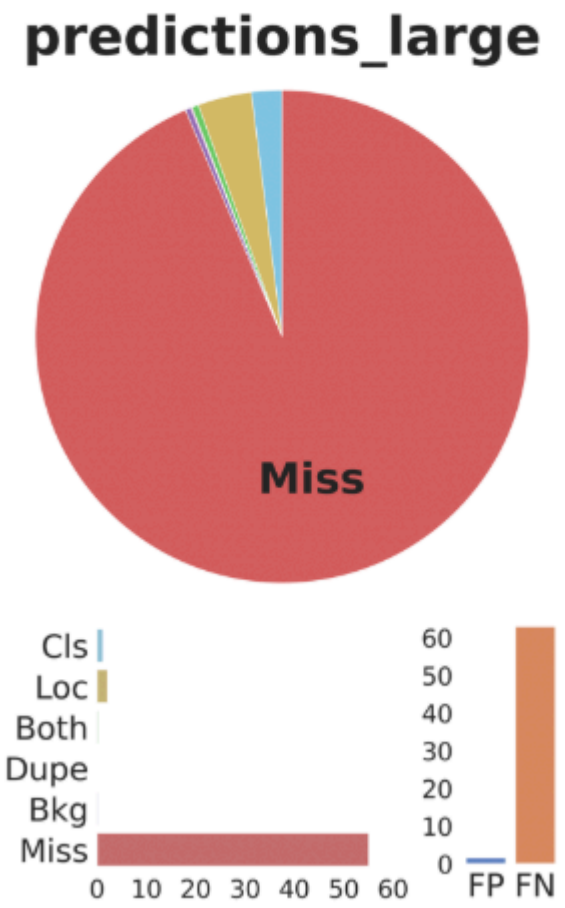
Type	Cls	Loc	Both	Dupe	Bkg	Miss
dAP	0.64	2.34	0.24	0.02	0.94	58.00

Special Error

Type	FalsePos	FalseNeg
dAP	2.27	67.04



```
bbox AP @ 50: 31.29

                    Main Errors
============================================================
  Type      Cls      Loc      Both     Dupe      Bkg      Miss
------------------------------------------------------------
   dAP     1.18     2.08     0.26     0.05     0.21     55.07
============================================================

        Special Error
============================
  Type   FalsePos   FalseNeg
----------------------------
   dAP       1.56      62.91
============================
```



Computing ECE per scale:

ECE for small objects: 0.0814

ECE for medium objects: 0.0413

ECE for large objects: 0.0317

For small scale is 32*32 as given in boilerplate

The TIDE evaluation results reveal that the model performs significantly better with larger objects than with smaller ones. **Large objects** have the highest **AP @ 50** (31.29%) and the lowest **Miss** error (55.07%), suggesting more reliable detection and confidence calibration. For **small objects**, the **AP @ 50** is much lower (9.39%) with a very high **Miss** error (60.17%), indicating that the model struggles with detecting smaller objects. The **ECE** for small objects (0.0814) is also the highest, showing poor confidence calibration. **Medium objects** show intermediate performance with an **AP @ 50** of 22.19%, and the **Miss** error (58.00%) is still relatively high, but better than for small objects. The **ECE** for medium objects (0.0413) is more reliable than for small objects, but not as well-calibrated as for large ones.

The model's calibration is best for large objects (ECE = 0.0317), while small objects exhibit the worst calibration (ECE = 0.0814), and medium objects fall in between. This suggests the need for improvements in detecting small and medium objects, possibly through higher-resolution inputs, multi-scale training, or calibration techniques like Platt scaling. By improving the model's detection and calibration for smaller objects, we can enhance overall performance.

f)

1. What can you infer from these observations?

From these observations, it's clear that the model's performance is heavily influenced by the size of the objects it is detecting. The model handles larger objects more effectively, achieving better detection results and more reliable confidence calibration. For small objects, the performance is quite poor, with a high **Miss** rate and low **AP @ 50**. This suggests that the model struggles to detect small objects accurately and tends to miss many of them. The confidence calibration, as indicated by the **ECE**, is also worse for small objects, implying that the predicted confidence is less reliable. Overall, the model's ability to detect and confidently classify objects varies considerably depending on their size.

2. Comment on your observations across each of the three scales.

The model’s performance shows a clear trend as the object size increases. For **small objects**, the **AP @ 50** is quite low (9.39%), and the **Miss** rate is alarmingly high (60.17%), with the model failing to detect a large portion of these objects. The **FalseNeg** rate (70.95%) further highlights the model’s struggle to detect small objects. Additionally, the **ECE** for small objects is also the highest (0.0814), pointing to poor confidence calibration. For **medium objects**, the performance improves somewhat, with an **AP @ 50** of 22.19% and a **Miss** rate of 58.00%. While better than small objects, medium objects still present a significant challenge, as indicated by the **FalseNeg** rate (67.04%) and the moderate **ECE** of 0.0413. The model performs the best on **large objects**, where it achieves the highest **AP @ 50** (31.29%) and the lowest **Miss** rate (55.07%). The **FalseNeg** rate (62.91%) is also better

than for the smaller objects, and the **ECE** is the lowest for large objects (0.0317), showing that the model's confidence calibration is more reliable for these objects.

3. Compare these statistics with the relevant metrics computed with all objects, as you computed in 4.(c) and 4.(d).

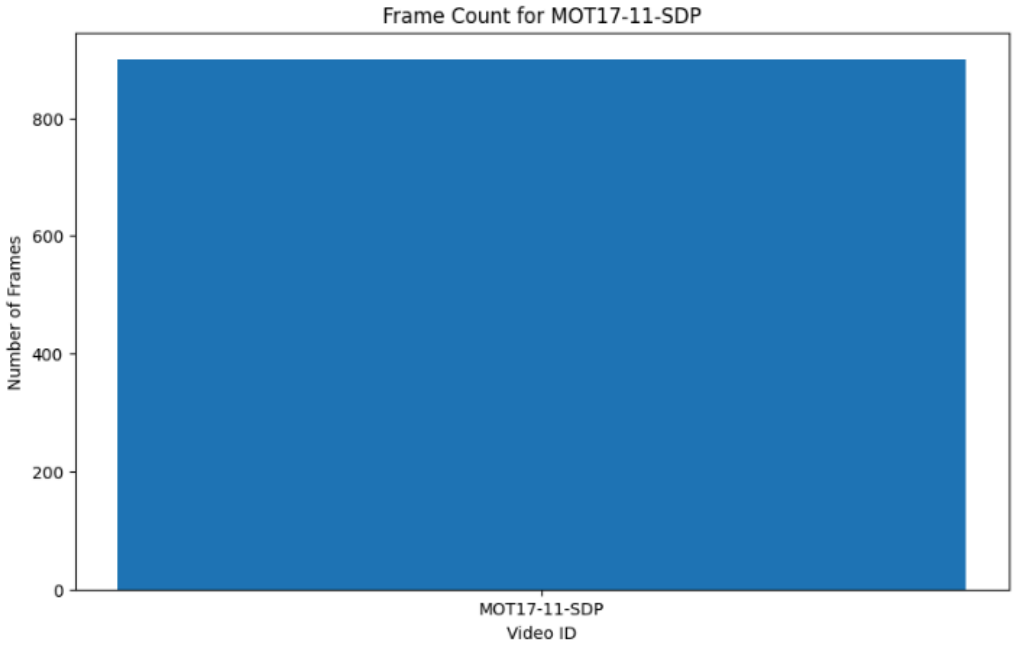
When comparing these results with the overall performance metrics computed in **4.(c)** and **4.(d)**, we see a significant difference. In **4.(c)**, the model achieves an overall **AP @ 50** of 61.62% when considering all objects, which is much higher than the individual scales for small (9.39%), medium (22.19%), and large (31.29%) objects. This indicates that the model performs better when it is evaluated across all object sizes together, likely due to the higher detection rates for large objects. However, the smaller object categories still show clear weaknesses. In **4.(d)**, the overall **ECE** of 0.0738 is also higher than the ECE for large (0.0317) and medium (0.0413) objects, but lower than the ECE for small objects (0.0814). This further highlights that while the model's confidence calibration is better for larger objects, there is still significant room for improvement, particularly for smaller objects, where the confidence predictions are less reliable.

P.T.O

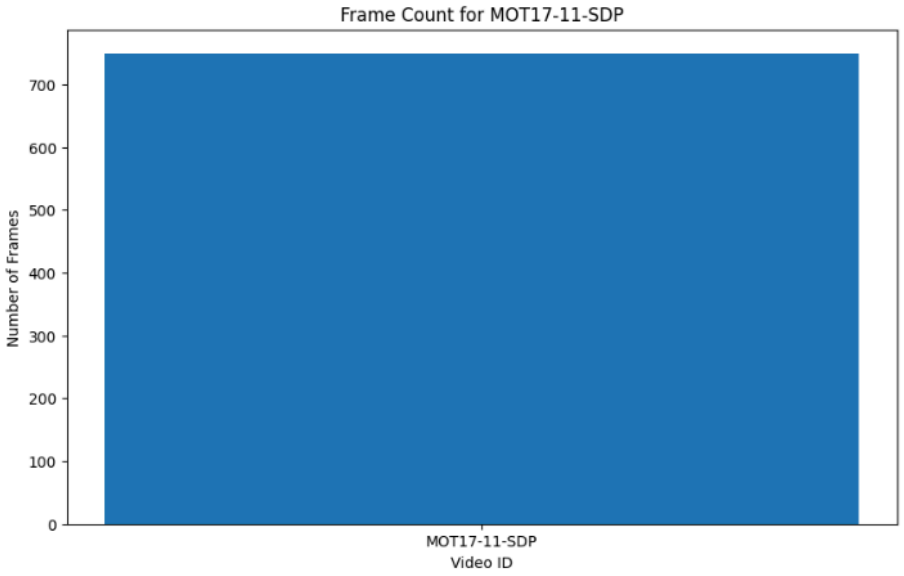
BONUS

a)


First frame of MOT17-11-SDP


Frame Count for MOT17-11-SDP
Number of Frames
0
200
400
600
800
MOT17-11-SDP
Video ID

b)



b)

Score saved for both visualisation done for 1, submitting pkl for only 1

Video link-https://drive.google.com/file/d/1yoOQD3z8xHwGuDKIf1tgu5AeToE_SVDB/view?usp=sharing

C)
Score saved for both visualisation done for 1, submitting pkl for only 1

Video link- IOU_Tracked.mp4

D)

Results

```
Loaded 900 frames from image folder.
Detections generated: 7190
Detections generated: 7190
Tracking results generated: 6940
Tracking results generated: 6940
Ground truth entries: 10617
Ground truth entries: 10617
After filtering conf==0: 9436
         IDF1   IDP   IDR  Rcll  Prcn GT MT PT ML  FP   FN IDs   FM  MOTA  MOTP IDt IDa IDm
Overall 61.1% 72.0% 53.0% 69.8% 94.9% 75 23 37 15 355 2848  86  195 65.1% 0.142  54  40  15
MOTA Score: 65.13
```

IOU MOTA17-11

```
Loaded 750 frames from image folder.
Detections generated: 7695
Tracking results generated: 7695
Ground truth entries: 20202
After filtering conf==0: 11642
         IDF1  IDP  IDR  Rcll  Prcn  GT MT PT ML  FP   FN  IDs   FM MOTA  MOTP IDt  IDa IDm
Overall 3.5% 4.5% 2.9% 57.6% 87.1% 110 47 25 38 993 4940 5161  375 4.7% 0.211  13 5148   0
MOTA Score: 4.71
```

IOU MOTA17-13

```
         IDF1   IDP   IDR  Rcll  Prcn  GT MT PT ML  FP   FN IDs   FM  MOTA  MOTP IDt IDa IDm
Overall 59.9% 79.3% 48.1% 55.7% 91.7% 110 43 20 47 586 5163 119  230 49.6% 0.203  55  56  20
MOTA Score: 49.60
```

BYTE MOTA17-13

```
Loaded 900 frames from image folder.
Detections generated: 7190
Tracking results generated: 7190
Ground truth entries: 10617
After filtering conf==0: 9436
         IDF1   IDP   IDR  Rcll  Prcn GT MT PT ML  FP   FN  IDs   FM  MOTA  MOTP IDt  IDa IDm
Overall 13.2% 15.2% 11.6% 71.3% 93.5% 75 26 36 13 464 2707 2181  245 43.3% 0.149   6 2176   1
MOTA Score: 43.26
```

BYTE MOTA17-11