CSE344: Computer Vision

Assignment-1

Himanshu Raj (2022216)

January 6, 2025

Image Classification

1. a) Custom dataset class

```python
class WildlifeDataset(Dataset):
    """
    Implements a custom dataset (similar to the one in PyTorch) to handle data preparation of
    Russian Wildlife Dataset for training Neural Networks for Image Classification Task. This
    dataset is compatible with PyTorch's DataLoader.
    """
    LABEL2INDEX: dict[str, int] = {
        'amur_leopard': 0,
        'amur_tiger': 1,
        'birds': 2,
        'black_bear': 3,
        'brown_bear': 4,
        'dog': 5,
        'roe_deer': 6,
        'sika_deer': 7,
        'wild_boar': 8,
        'people': 9
    } #a class dictionary to map string labels to numbers or indices

    #a class dictionary to map number labels to strings
    INDEX2LABEL: dict[int, str] = {index: label for label, index in LABEL2INDEX.items()}

    def __init__(self, root: str, transform: transforms.Compose =None, label_transform: transforms.Compose =None):
        """
        Initializes the Wildlife dataset.
        """
        self.root = root
        self.transform = transform
        self.label_transform = label_transform
        self.images = []
        self.labels = []
        #add image paths and labels in lists
        for label in WildlifeDataset.LABEL2INDEX:
            label_dir = os.path.join(self.root, label)
            for image in os.listdir(label_dir):
                self.images.append(os.path.join(label_dir, image)) #add the relative path to the image
                self.labels.append(WildlifeDataset.LABEL2INDEX[label]) #add the number mapped to label

    def __len__(self) -> int:
        """
        Returns the number of samples in the dataset.
        """
        return len(self.labels)

    def __getitem__(self, idx: int) -> tuple[torch.Tensor]:
        """
        Returns the image and label at the given index.
        """
        image = self.images[idx]
        label = self.labels[idx]
        image = Image.open(image)
        if self.transform:
            image = self.transform(image)
        if self.label_transform:
            label = self.label_transform(label)
        return image, label
```

Splitting the dataset into training and validation sets in the ratio of 0.8:0.2

```
train_dataset, val_dataset = torch.utils.data.random_split(dataset=dataset, lengths=[0.8, 0.2])
```

Weights & Biases initialization is done in the pipeline used for training models.

```
def pipeline(hyperparams, train_loader, val_loader):
    with wandb.init(entity="cv-himanshu", project="cv-wildlife-classification", config=hyperparams):
```

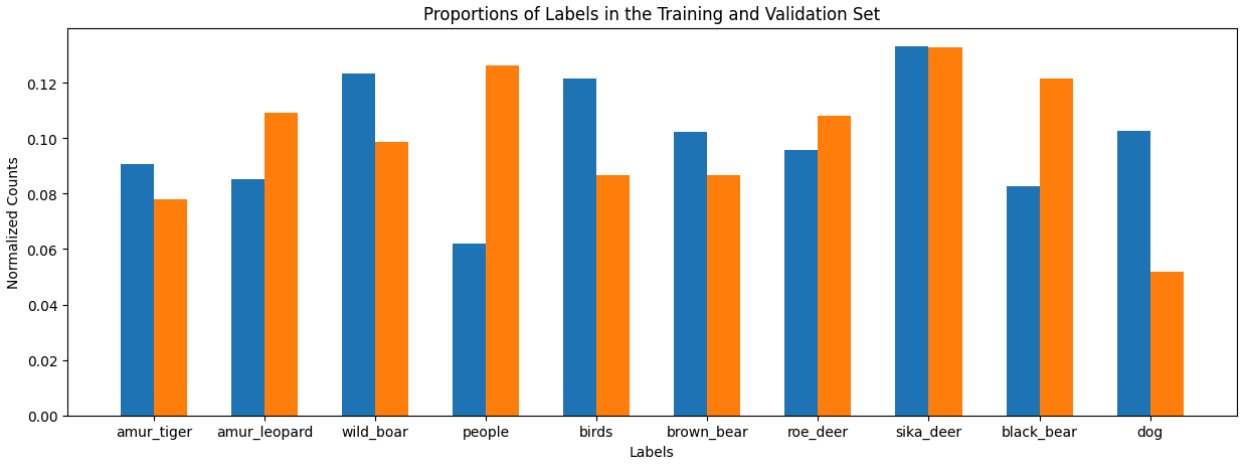
b) Data Loaders

```
batch_size = 256
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(dataset=val_dataset, batch_size=batch_size, shuffle=True)
```

c) Visualization of data distribution
Entire Dataset Stats: total samples: 11668
class wise distribution: {'amur_leopard': 978, 'amur_tiger': 1049, 'birds': 1446, 'black_bear': 975, 'brown_bear': 1209, 'dog': 1213, 'roe_deer': 1124, 'sika_deer': 1539, 'wild_boar': 1435, 'people': 700}

Training Set Stats: total samples: 9335
class wise distribution: {'amur_tiger': 847, 'amur_leopard': 796, 'wild_boar': 1151, 'people': 579, 'birds': 1136, 'brown_bear': 957, 'roe_deer': 894, 'sika_deer': 1244, 'black_bear': 773, 'dog': 958}

Validation Set Stats: total samples: 2333
class wise distribution: {'amur_leopard': 182, 'dog': 255, 'roe_deer': 230, 'sika_deer': 295, 'amur_tiger': 202, 'black_bear': 202, 'brown_bear': 252, 'birds': 310, 'wild_boar': 284, 'people': 121}



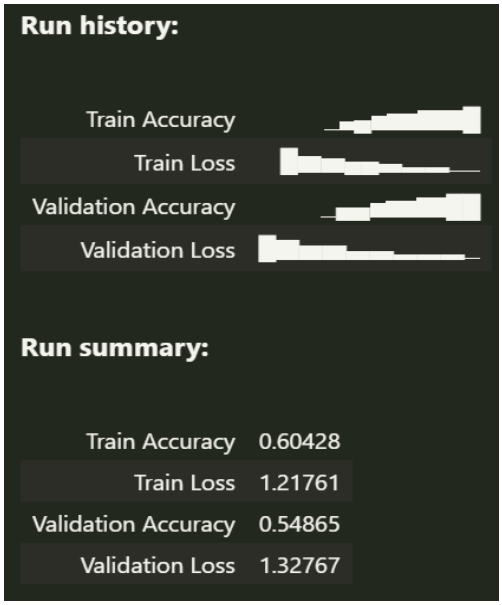
From the dictionaries and the plot, we can see that the data distribution is not balanced in the dataset, which leads to an imbalance in the training and validation sets as well.

2. Convolutional Neural Network

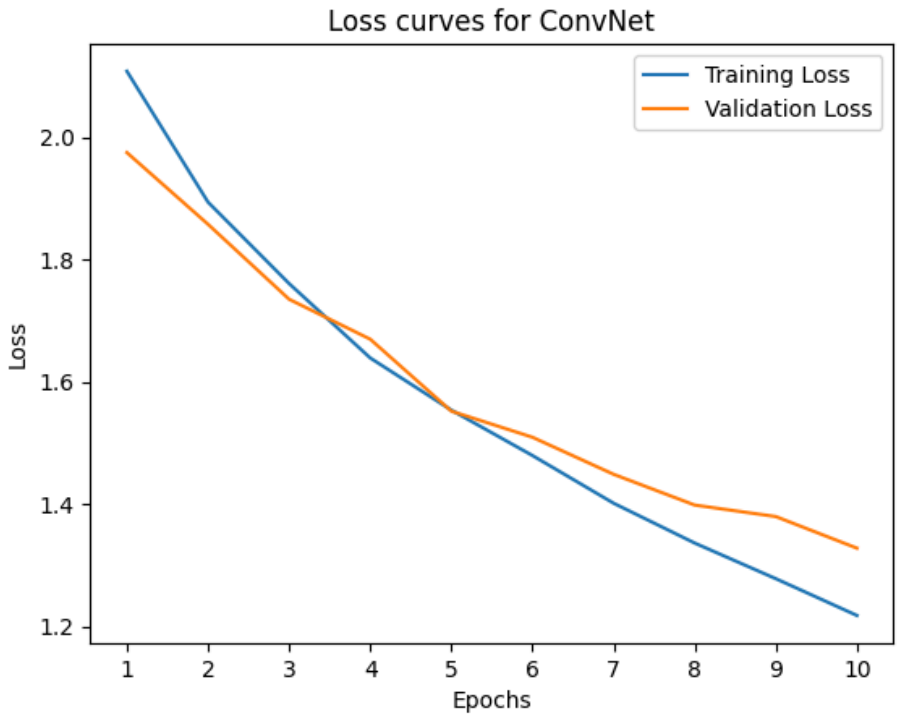
a) ConvNet class

```python
class ConvNet(nn.Module):
    """
    Implements a custom CNN architecture with 3 convolution layers using PyTorch
    to be used on Russian Wildlife Dataset for Image Classification Task.
    """

    def __init__(self, num_classes: int):
        """
        Initializes the CNN model.
        """
        super(ConvNet, self).__init__()

        #initially, image size is 256x256
        self.conv = nn.Sequential(
            #conv layer 1
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=4, stride=4),
            #after conv layer 1, image size is 63x63

            #conv layer2
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            #after conv layer 2, image size is 31x31

            #conv layer 3
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2),
            #after conv layer 3, image size is 16x16
        )
        self.fc = nn.Linear(128 * 16*16, num_classes) #fully connected layer

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """
        Defines the forward pass of the model.
        """
        out = self.conv(x)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out
```

b) WandB training plots for ConvNet



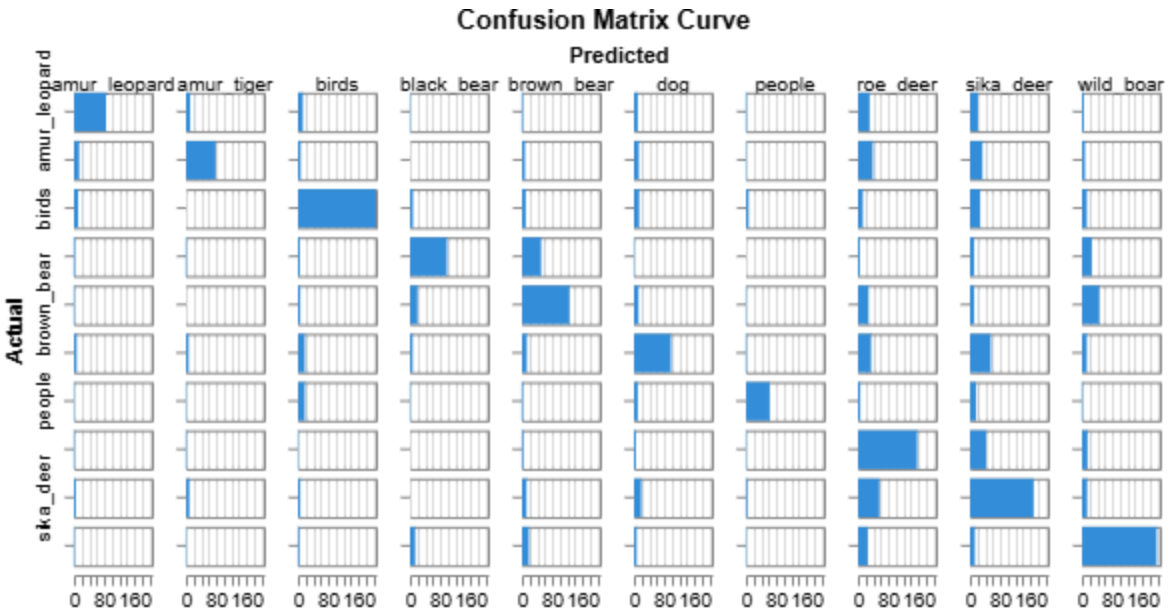
c) Loss curves for ConvNet



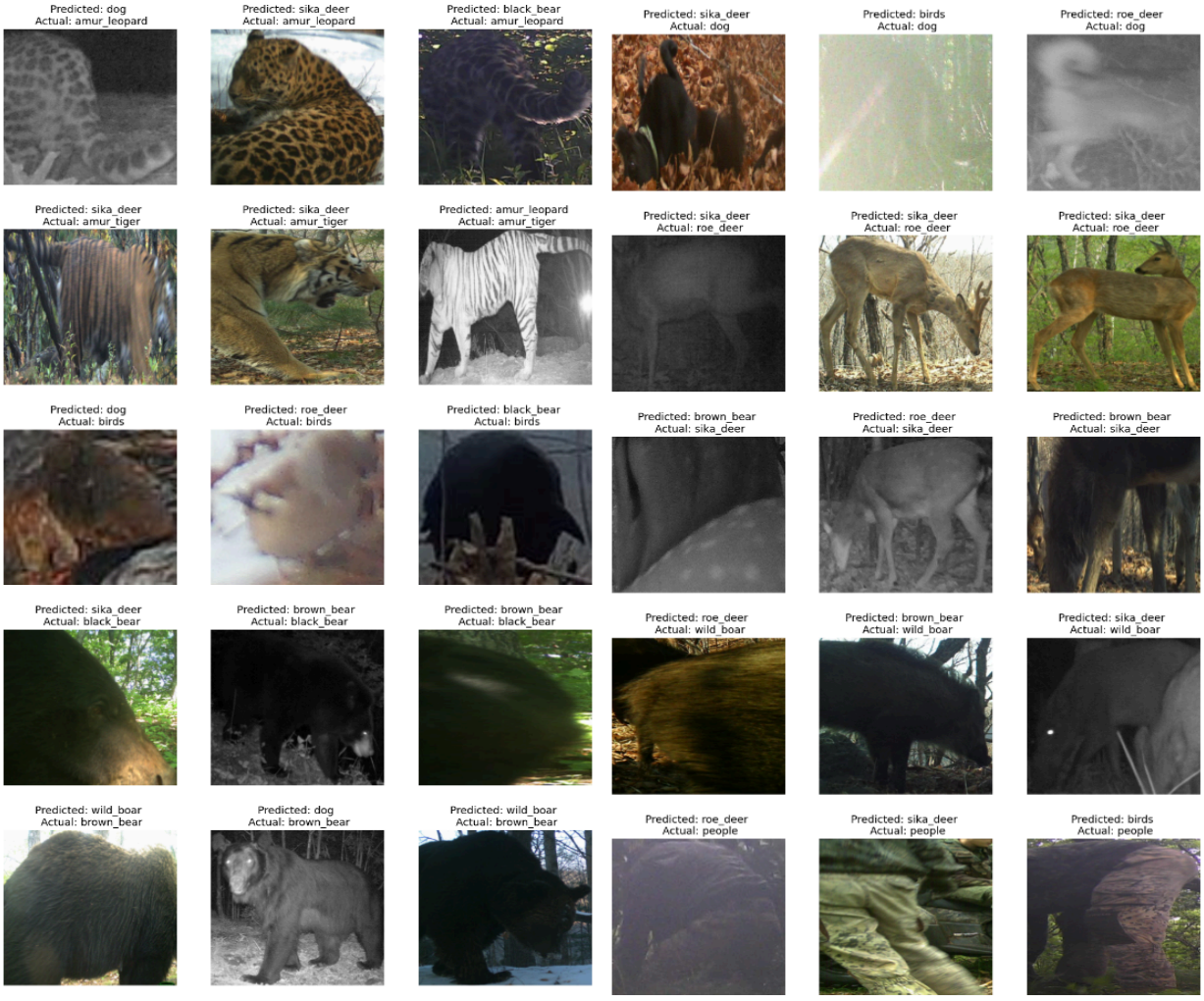
We see that the validation set loss is decreasing consistently over the epochs, so the model does not overfit the data.

d) Accuracy and F1-score and confusion matrices on the validation set for ConvNet

```
Validation Set: Accuracy: 0.5486498071153022 | F1-Score: 0.5496961395095264
```



e) Misclassifications by ConvNet

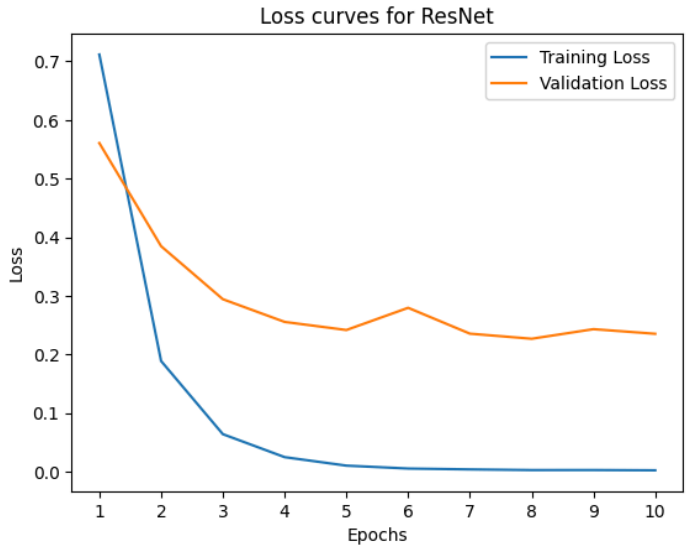


In some of the misclassifications, either the ground truth did not cover a majority of the image or the predicted class looks a lot similar to the ground truth in the image. For example, all misclassifications for ‘roe_deer’ are predicted as ‘sika_deer’ because the images look a lot similar to the predicted class as well. All misclassifications for ‘people’ only show a part of their body.
The misclassifications for the ‘people’ class can be worked around if we train the classifier on random and small crops of the entire image so that it classifies them even if they don’t cover the majority of the image.

3. Fine-tuning ResNet-18

a) WandB training plots for ResNet

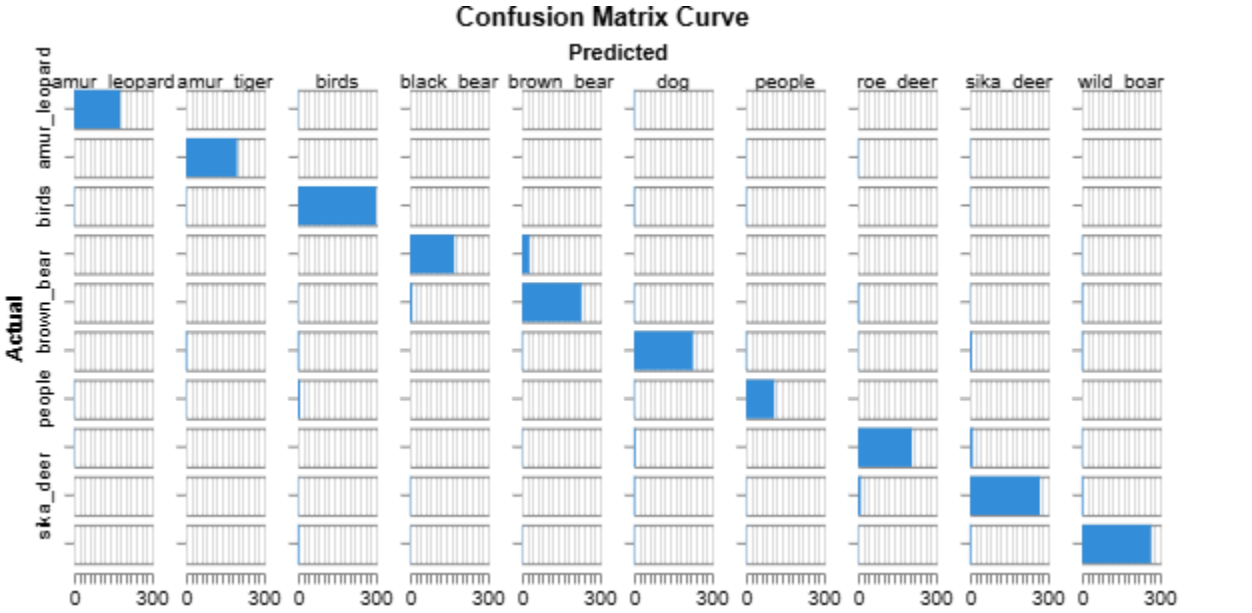
b) Loss curves for ResNet



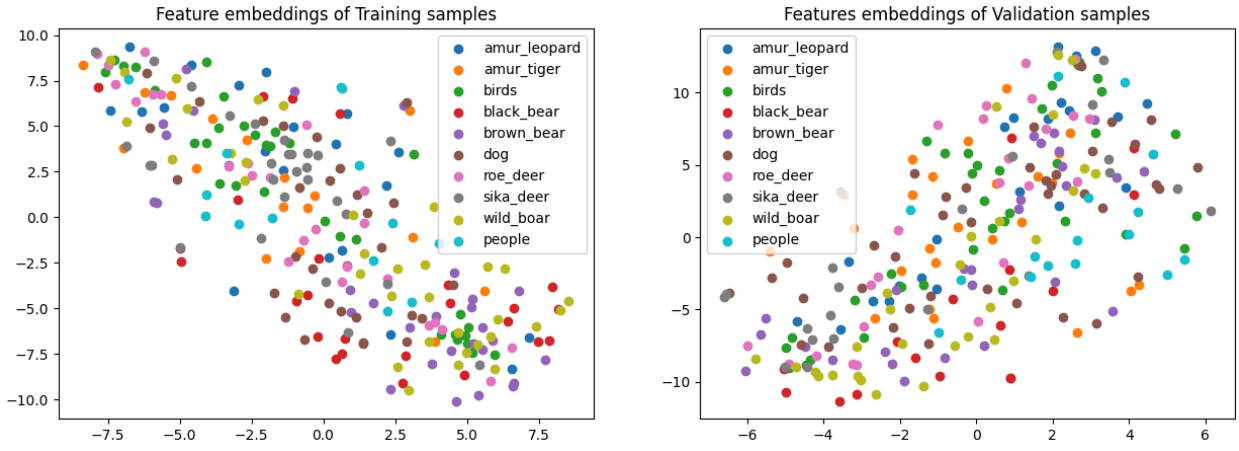
We see that the validation set loss is decreasing consistently over the epochs and almost converges after epoch 7, so the model does not overfit the data.

c) Accuracy and F1-score and confusion matrices on the validation set for ResNet

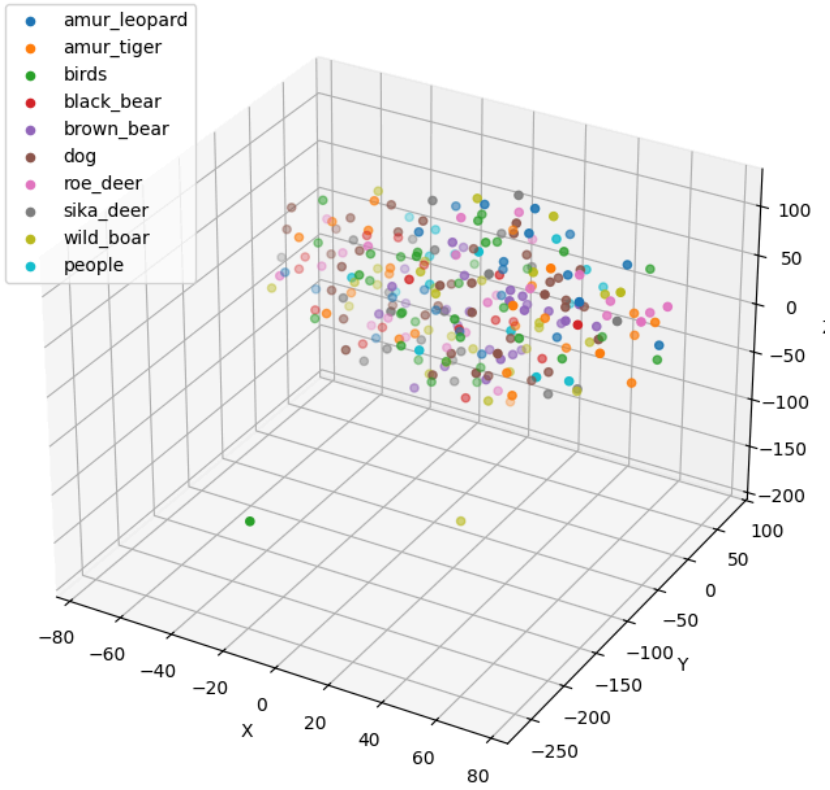
```
Validation Set: Accuracy: 0.9288469781397343 | F1-Score: 0.9286443411509563
```



d) t-SNE plots for feature space visualization in ResNet



2-D feature visualization for the first 256 samples in training and validation sets

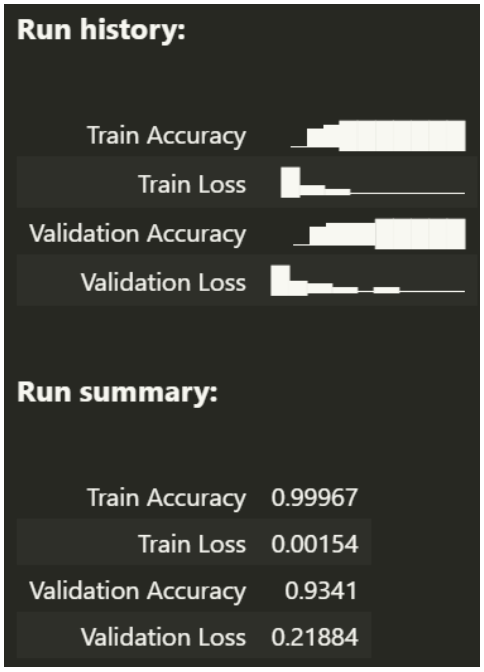


3-D feature visualization for the first 256 samples in the validation set

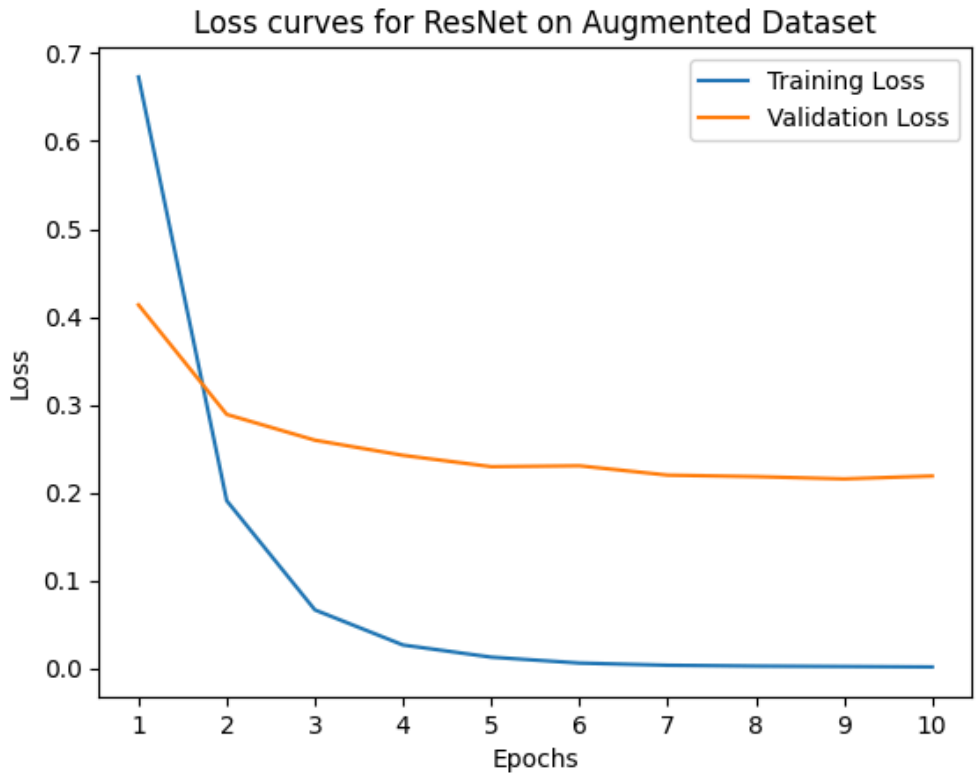
4. Augmenting dataset to handle class imbalances

a) Augmented images

b) WandB training plots for ResNet on the augmented dataset



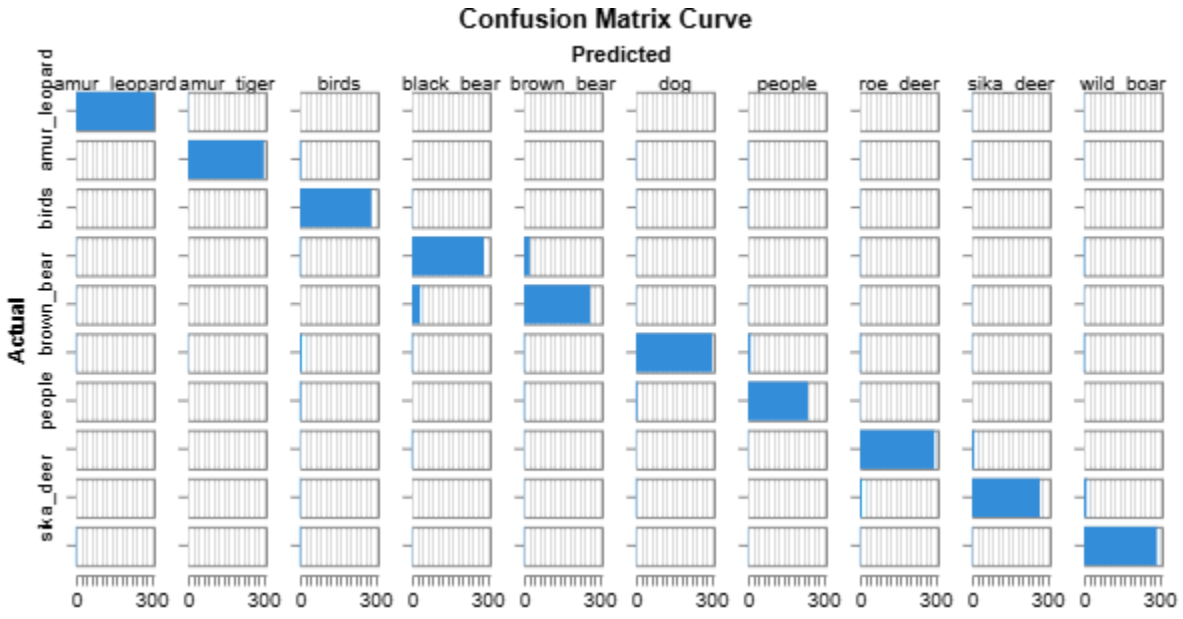
c) Loss curves for ResNet on the augmented dataset



We can see that the validation loss stops decreasing after 3 epochs, and the model converges. So, the problem of overfitting is not there.

d) Accuracy and F1-score and confusion matrices on the validation set of the augmented dataset for ResNet

```
Validation Set: Accuracy: 0.9340983606557377 | F1-Score: 0.9340459729176422
```



5. Comparison of models

When comparing all the architectures, it is pretty evident that vanilla CNN architecture doesn't perform that well, as it didn't converge in the 10 epochs. Its validation accuracy and F1 score are very low due to its comparatively lower capacity.
On the other hand, the ResNet-18 architectures performed extremely well. Both models converged within 10 epochs and achieved a high validation accuracy and F1 score.
The model trained on the augmented dataset performed slightly better than its other variant as we handled class imbalances in the dataset, and it could generalize better on unseen data.