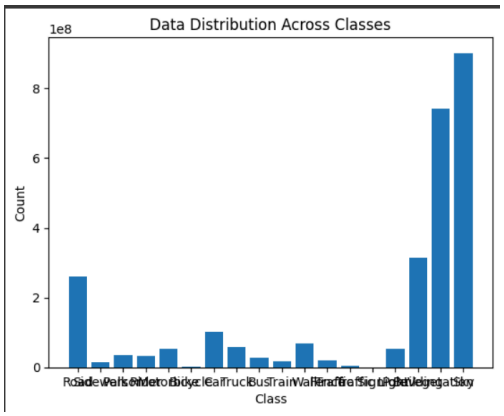
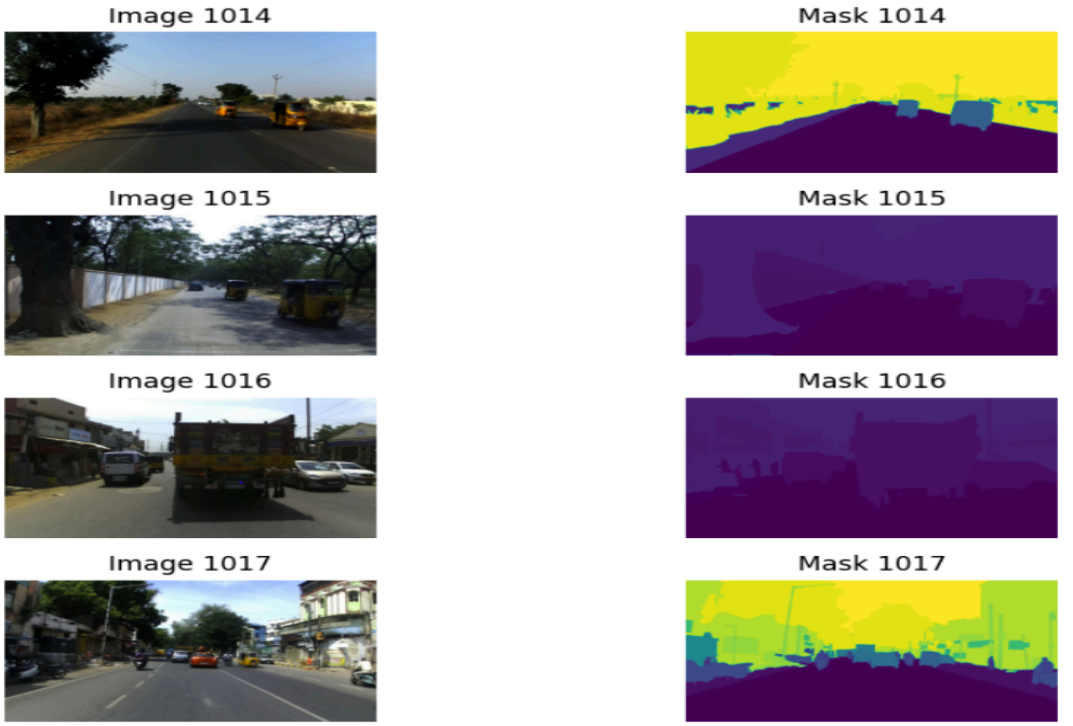
Q3.

1. a) I have initialized a CustomDataset class in which I have defined image directory(`/content/drive/MyDrive/IDD20K_II`), transformation, percentage of dataset(we have to select), img_path(list), mask_path(dictionary), and have selected 30% of image and their respective masks(ground truth)
2. Created two separate directories `'/content/drive/MyDrive/30_images_masks/images'` and `'/content/drive/MyDrive/30_images_masks/masks'` and stored images and masks in that directory.
3. Then visualized the distribution of the dataset and plotted the graph of the distribution.

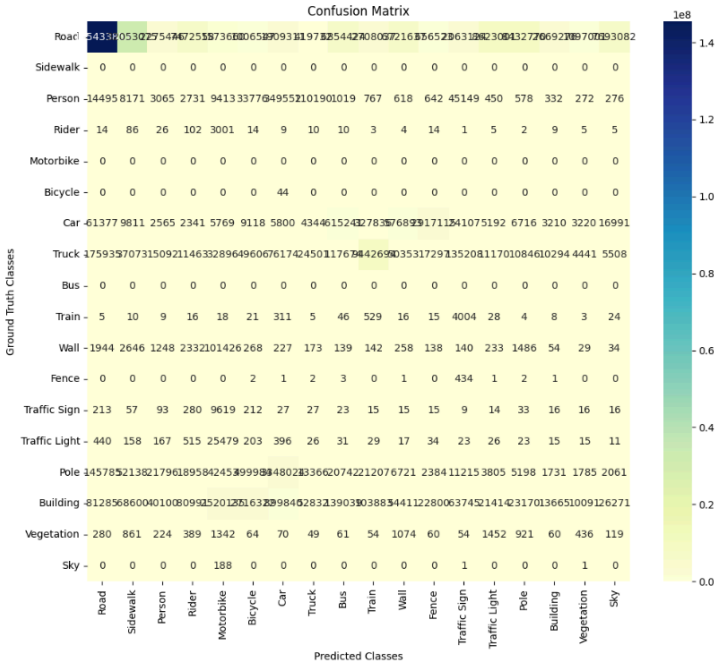


4. Then visualized the image with its respective masks



5. After that I put the relevant classes for which we have to segment into a list and a dictionary with its respective class id mapping that I got from the research paper figure and then defined the color coding and transformed the images to (512*512) dimension and imported the DeepLabV3 model from pytorch website and looped through each image and got the inference_image for each image by training the image with DeepLabV3 model(code for training I got through pytorch website) and then filtered the output from the classes which are not in the given question and then saved the images to output_mask directory.

6. Metrics evaluation part:
7. I have imported `accuracy_score, precision_score, recall_score, f1_score, confusion_matrix from sklearn` and does the same thing made a mapping of classes with respective field and then passed the images to the DeepLabV3 model after which I passed every output segmented masks to the metrics and got the result. Then calculated the Iou of every class using the formula $$IoU = \frac{TP}{(TP + FP + FN)}$$ which I got through reading https://learnopencv.com/intersection-over-union-iou-in-object-detection-and-segmentation/ this article and then took the mean over IoUs to calculate the MAP.
8. After which I have plotted the confusion matrix heatmap of every class:



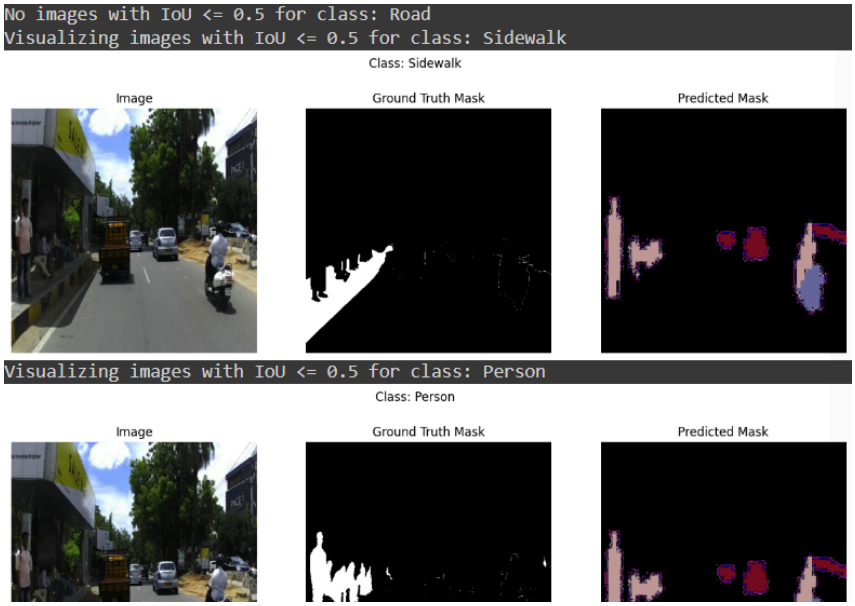
9. Inference :
    - The diagonal cells (from top left to bottom right) represent mostly the correctly predicted instances or some got misclassified for each class, which appear brighter.
    - The off-diagonal cells represent the misclassifications, and the darkness of these cells indicates the extent of confusion between different classes.
    - For example, there seems to be significant confusion between 'Car' and 'Road', 'Building' and 'Road'.
10. Now I have calculated the precision, recall, F1 score of every class using the data that I have obtained from confusion matrix, and compared it with the mean(threshold) f1 score for the information on which classes performed well

```
Class:  Road
Precision:  0.996698276852131
Recall:  0.6068263286819379
F1 Score:  0.7543666671012516
------------------------
Class:  Sidewalk
Precision:  0.0
Recall:  0.0
F1 Score:  0.0
------------------------
Class:  Person
Precision:  0.0012988218224101642
Recall:  0.005270887503955315
F1 Score:  0.002084093336103058
------------------------
Class:  Rider
Precision:  1.3433998764072114e-05
```

```
Recall:  0.03072289156626506
F1 Score:  2.6856254268696296e-05
----------------------
Class:  Motorbike
Precision:  0.0
Recall:  0.0
F1 Score:  0.0
----------------------
Class:  Bicycle
Precision:  0.0
Recall:  0.0
F1 Score:  0.0
----------------------
Class:  Car
Precision:  0.0008801499775561755
Recall:  0.0012615151318740069
F1 Score:  0.0010368778107433412
----------------------
Class:  Truck
Precision:  0.0391854869277753
Recall:  0.002395430292157241
F1 Score:  0.004514864446267105
----------------------
Class:  Bus
Precision:  0.0
Recall:  0.0
F1 Score:  0.0
----------------------
Class:  Train
Precision:  4.196682066665207e-05
Recall:  0.10429810725552051
F1 Score:  8.389988222296306e-05
----------------------
Class:  Wall
Precision:  3.4808334248513697e-05
Recall:  0.002284864103722203
F1 Score:  6.857202088788807e-05
----------------------
Class:  Fence
Precision:  0.0
Recall:  0.0
F1 Score:  0.0
----------------------
Class:  Traffic Sign
Precision:  3.834332958136753e-06
Recall:  0.0008411214953271028
F1 Score:  7.633866205468053e-06
```

```
-----------------------
Class:  Traffic Light
Precision:  2.9999569621558892e-06
Recall:  0.0009417560127499276
F1 Score:  5.980861931917319e-06
-----------------------
Class:  Pole
Precision:  0.0006128452987703362
Recall:  0.0012319424328801122
F1 Score:  0.0008185116535557309
-----------------------
Class:  Building
Precision:  0.005063603529292752
Recall:  0.0019694191647472094
F1 Score:  0.0028358668149106618
-----------------------
Class:  Vegetation
Precision:  0.0003902211999301898
Recall:  0.05759577278731836
F1 Score:  0.0007751903527916189
-----------------------
Class:  Sky
Precision:  0.0
Recall:  0.0
F1 Score:  0.0
-----------------------
Mean F1 Score: 0.0426
Classes where the model performs well:  [{'Class': 'Road', 'F1
Score': 0.7543666671012516}]
Classes where improvement is needed:  [{'Class': 'Sidewalk', 'F1
Score': 0.0}, {'Class': 'Person', 'F1 Score': 0.002084093336103058},
{'Class': 'Rider', 'F1 Score': 2.6856254268696296e-05}, {'Class':
'Motorbike', 'F1 Score': 0.0}, {'Class': 'Bicycle', 'F1 Score':
0.0}, {'Class': 'Car', 'F1 Score': 0.0010368778107433412}, {'Class':
'Truck', 'F1 Score': 0.004514864446267105}, {'Class': 'Bus', 'F1
Score': 0.0}, {'Class': 'Train', 'F1 Score': 8.389988222296306e-05},
{'Class': 'Wall', 'F1 Score': 6.857202088788807e-05}, {'Class':
'Fence', 'F1 Score': 0.0}, {'Class': 'Traffic Sign', 'F1 Score':
7.633866205468053e-06}, {'Class': 'Traffic Light', 'F1 Score':
5.980861931917319e-06}, {'Class': 'Pole', 'F1 Score':
0.0008185116535557309}, {'Class': 'Building', 'F1 Score':
0.0028358668149106618}, {'Class': 'Vegetation', 'F1 Score':
0.0007751903527916189}, {'Class': 'Sky', 'F1 Score': 0.0}]
```

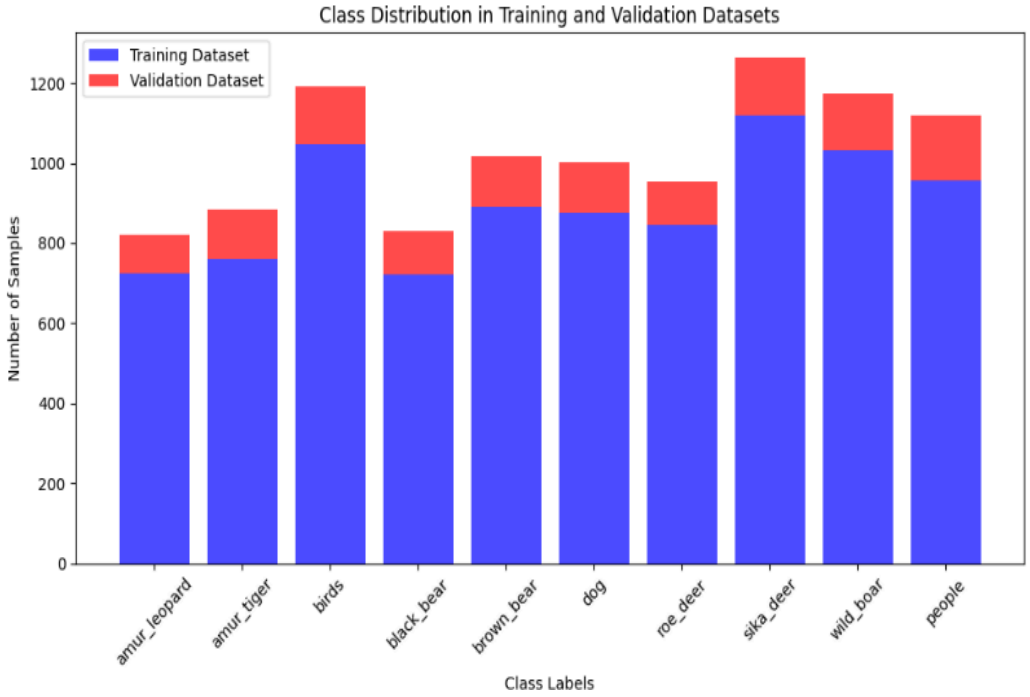
11. Then plotted the image, ground truth mask and predicted mask of every class having IoU less than or equal to 0.5



Similarly, for other classes.

Q2.

1. I have first installed Weight and Biases and then created my own Custom Dataset class in which I am defining image directory, transformation and classes and appending all the images to the directory and then splitted the dataset into 3 set train, test and validation set using random split (length = len(dataset)
   train_size = int(length*0.7)
   val_size = int(length*0.1)
   test_size = length - train_size - val_size
   sum_all = train_size + test_size + val_size) and then assigned the index to all the classes.
2. Configured wandb by declaring learning rate, epochs, batch_size and then again created 3 separate data loader for train, test and validate set.
3. Visualized the data distribution in both training and validation set :

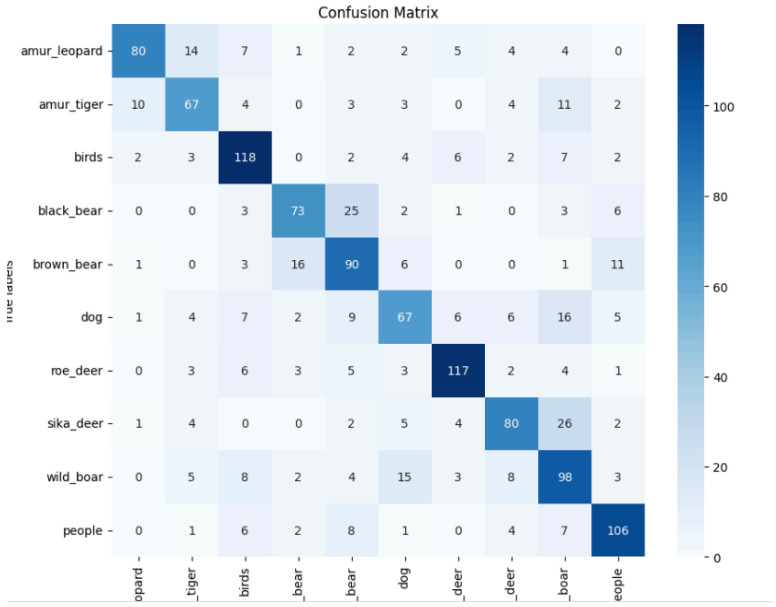


4. Then created a custom CNN class in which I have inherited the NN class and defined convolutional layer as given in the question.
5. After that I have declared the Adam optimizer and loss(i.e. CrossEntropyLoss).

In this case validation loss is less than training loss, so we can not say that model is overfitting.

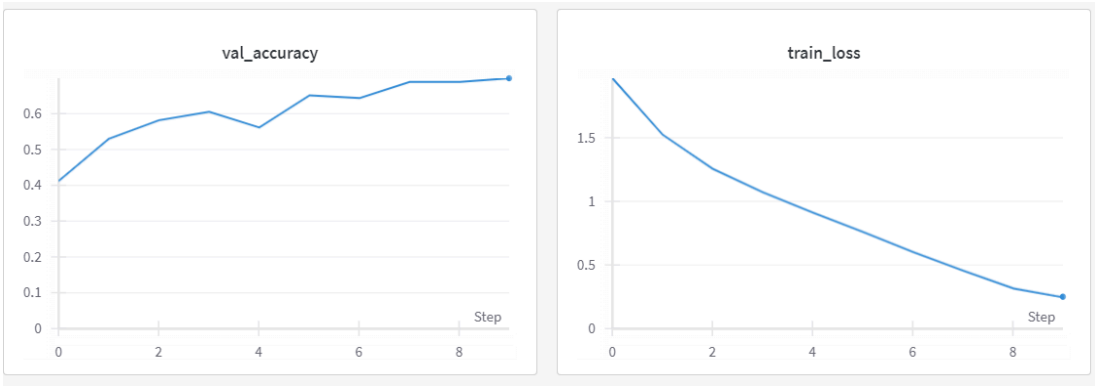
```
from sklearn.metrics import accuracy_score, f1_score
accuracy = accuracy_score(all_targets, all_preds)
print(f'Accuracy: {accuracy}')
f1 = f1_score(all_targets, all_preds, average='weighted')
print(f'F1 Score: {f1}')
```

```
Accuracy: 0.6989079563182528
F1 Score: 0.6996144936703907
```



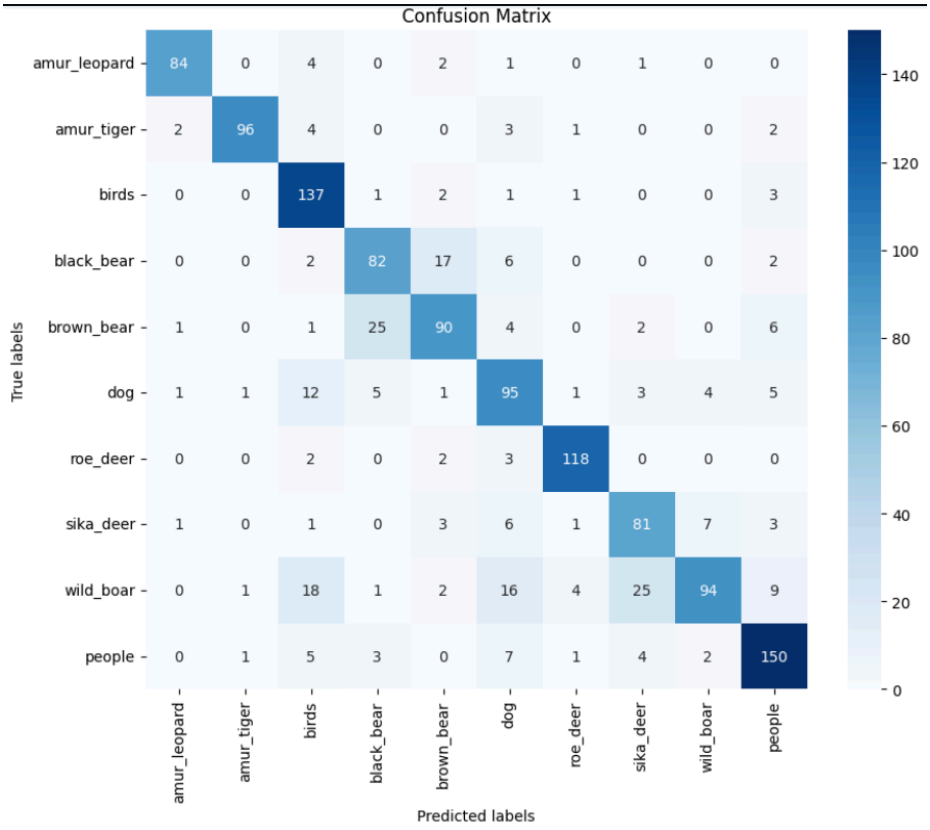
ResNet-18 model:

I did the same for data and data loader as done previously only instead of using customCNN class I have used the pretrained resnet model.

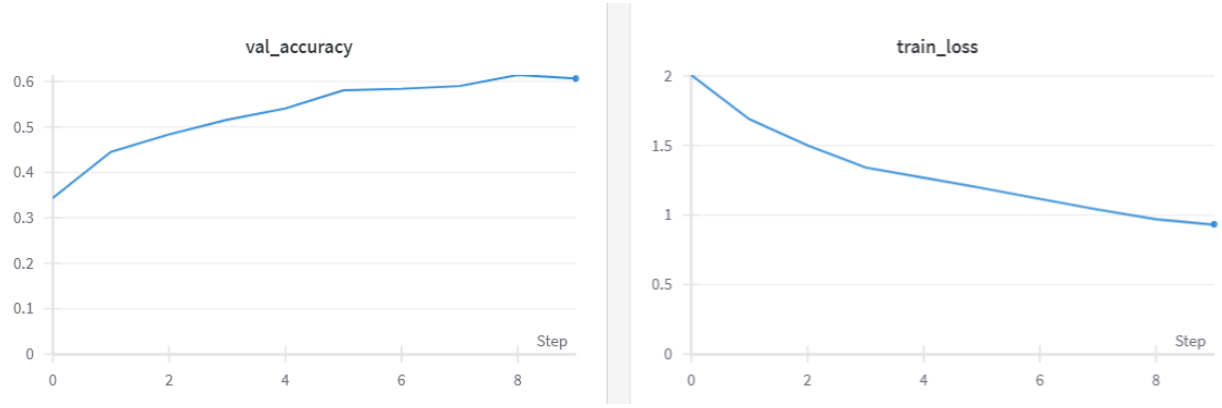



```
💡 Click here to ask Blackbox to help you code faster
from sklearn.metrics import accuracy_score, f1_score
accuracy = accuracy_score(all_targets, all_preds)
print(f'Accuracy: {accuracy}')
f1 = f1_score(all_targets, all_preds, average='weighted')
print(f'F1 Score: {f1}')
```

```
Accuracy: 0.8010920436817472
F1 Score: 0.7987961553744557
```



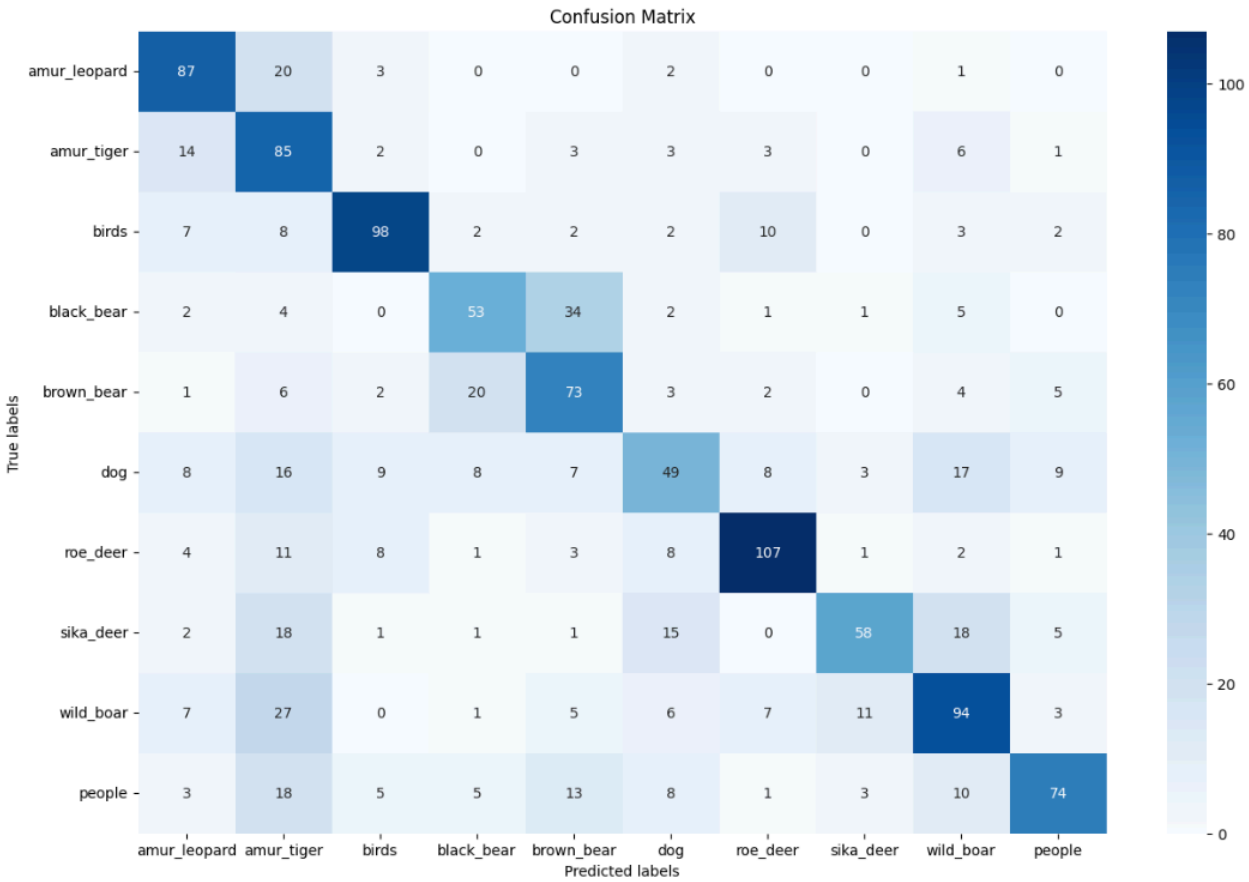
Data Augmentation Part:
For data augmentation I have used horizontal and vertical flip with probability 0.5 and random rotation of the image




```
💡 Click here to ask Blackbox to help you code faster
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

accuracy = accuracy_score(all_targets, all_preds)
print(f'Accuracy: {accuracy}')
f1 = f1_score(all_targets, all_preds, average='weighted')
print(f'F1 Score: {f1}')
```

```
Accuracy: 0.6068642745709828
F1 Score: 0.6076482654633393
```



Q2. 2 © The model is not overfitting

Q2 .3 © The model is not overfitting

Q2 .4 © Overfitting problem is getting resolved.

Q1.(Theory)

A. Since MSE is designed to calculate the average squared difference between expected and actual values, it presumes continuous outputs. In our case, determining whether something is sweet or not is a binary classification task with labels 0 and 1. These kinds of discrete outputs are not directly handled by MSE.
Inaccurate interpretations of the error map result from the squared difference calculation used by MSE, which greatly magnifies even small errors for incorrectly classified papayas (e.g., predicting 0.8 for a "not sweet" papaya).

B. **BCE(y, ŷ) = -(y * log(ŷ) + (1 - y) * log(1 - ŷ))**

C. **BCE(y, ŷ) = -(0* log(0.9) + (1 - 0) * log(1 - 0.9)) = -(log(0.1)) = 1**

D.

a. BCE(1, 0.1) = - (1 * log(0.1) + (1 - 1) * log(1 - 0.1)) = 1
b. BCE(0, 0.2) = -log(1 - 0.2) = 0.097
c. BCE(0, 0.7) = -log(1 - 0.7) = 0.522

Average individual losses = (1+0.097+0.522)/3 = 0.54 = L

E. W = W(old) - α (Lbce + λ W(old))

Weights in model A will be smaller as penalty term is being introduced in A while model B is without regularizer.

Q2(Theory)

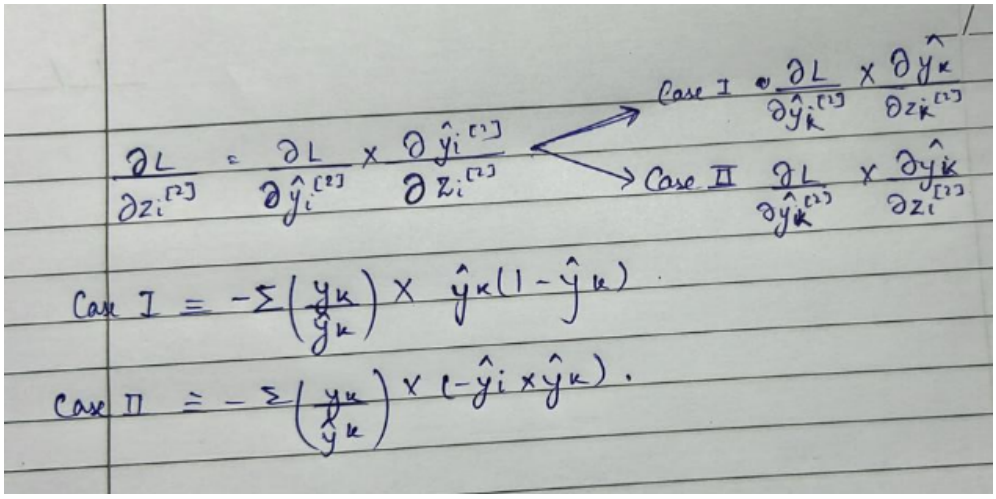
A. Shape (W2) = $D_a * K$

Shape(B2) = K * 1

After vectorizing shape of hidden output layer = $D_a * M$

B. $y_k^{\Lambda}(1 - y_k^{\Lambda})$

C. $- y_i^{\Lambda} * y_k^{\Lambda}$

D. 

E. There may be some problems with softmax functions of numerical instability when computing. There are two basic reasons why this occurs:

   1. Overflow: Exponentiating big input values (scores) in the softmax computation may result in overflow faults. This happens because, for big values, the exponential function grows quickly and exceeds the maximum number that the computer can represent.

   2. Underflow: On the other hand, exponentiating very small input numbers may lead to underflow issues. Here, due to numerical constraints, the computed values become incredibly small, losing precision and possibly approaching zero.

The Log-Sum-Exp Trick is a method of calculating the softmax that substitutes logarithms for exponentials. It takes advantage of the fact that exp(x - c) = exp(x) / exp(c) to deduct the highest score possible from each input prior to exponentiation. This prevents overflow and underflow by guaranteeing that all exponentiated numbers remain within a reasonable range.