COL775 Deep Learning

Assignment 1

Name: T Karthikeyan

Entry No. 2023AIZ8140

Model weights: https://drive.google.com/drive/folders/1yaTBrdYZf8YEecXYrpVgE2dmCD6obP7N

Part 1

ResNet Architecture

Resnet with inbuilt Batch Normalization

For comparison across all the models, hyperparameters are kept constant

Hyperparameters used are:

N = 2

Batch size = 32

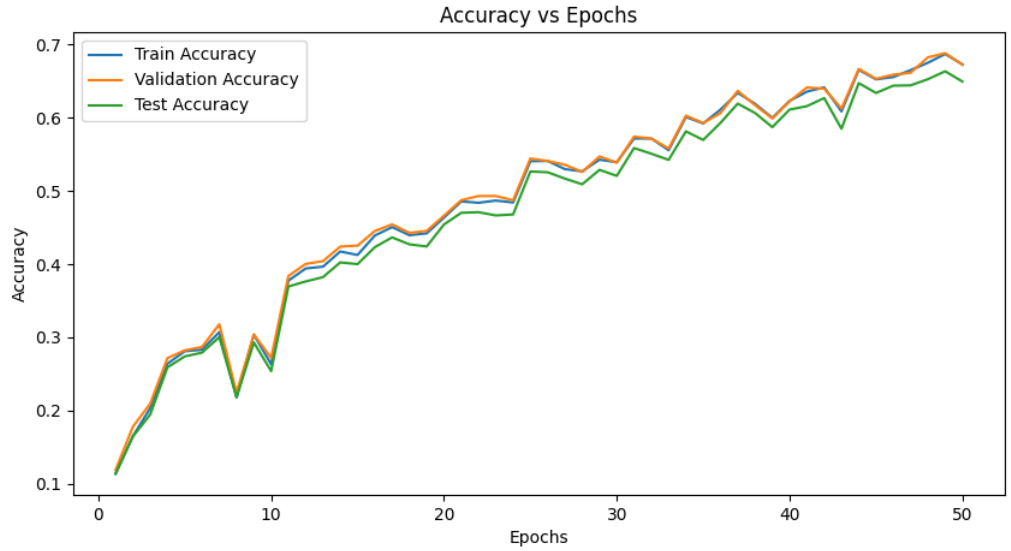
Epochs = 50

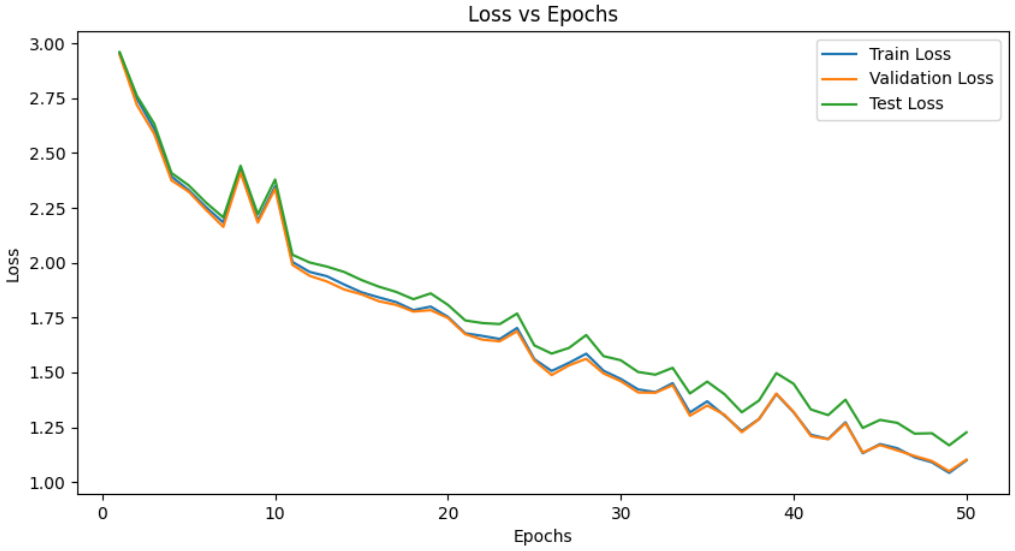
R = 25

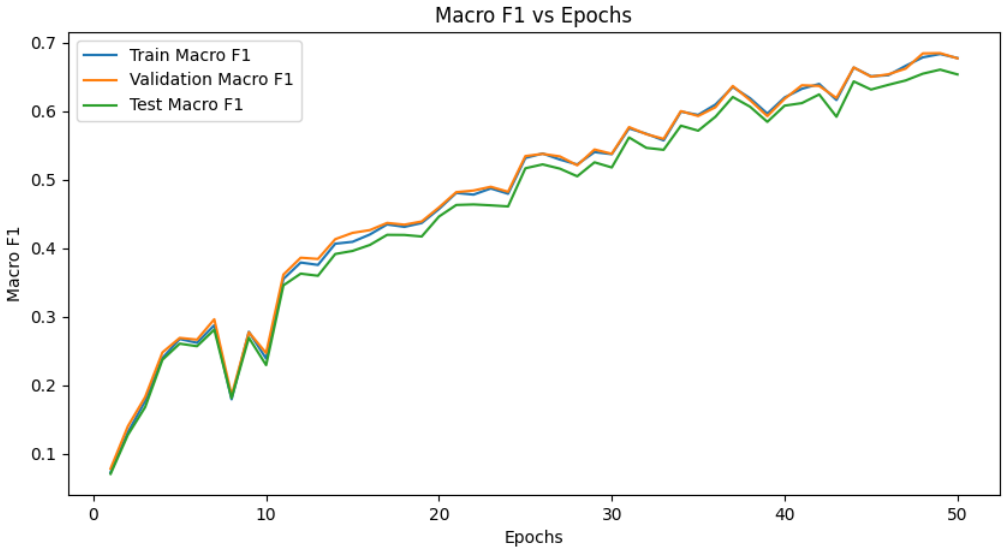
Optimizer = SGD

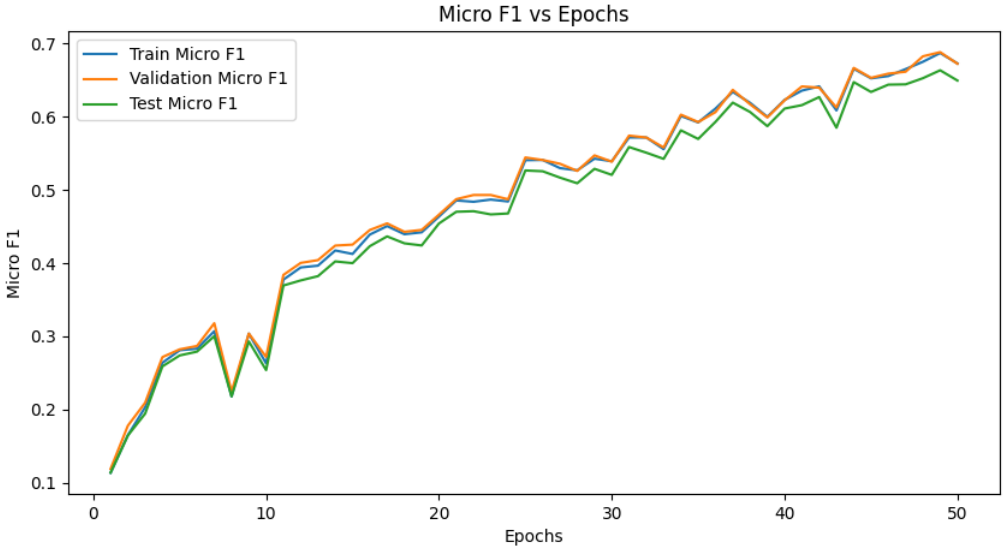
Learning Rate = 0.0001


```
{
    "train": {
        "train_loss": 1.0990780644442724,
        "train_accuracy": 0.672706258286025,
        "train_micro_f1": 0.672706258286025,
        "train_macro_f1": 0.6773762513777224
    },
    "val": {
        "val_loss": 1.1019920509577703,
        "val_accuracy": 0.6722928800665636,
        "val_micro_f1": 0.6722928800665636,
        "val_macro_f1": 0.67698776861159
    },
    "test": {
        "test_loss": 1.2264648625191221,
        "test_accuracy": 0.6493333333333333,
        "test_micro_f1": 0.6493333333333333,
        "test_macro_f1": 0.6535093058287208
    }
}
```


Accuracy vs Epochs
Train Accuracy
Validation Accuracy
Test Accuracy
Accuracy
Epochs
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
10
20
30
40
50


Loss vs Epochs
Train Loss
Validation Loss
Test Loss
Loss
Epochs
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
0
10
20
30
40
50

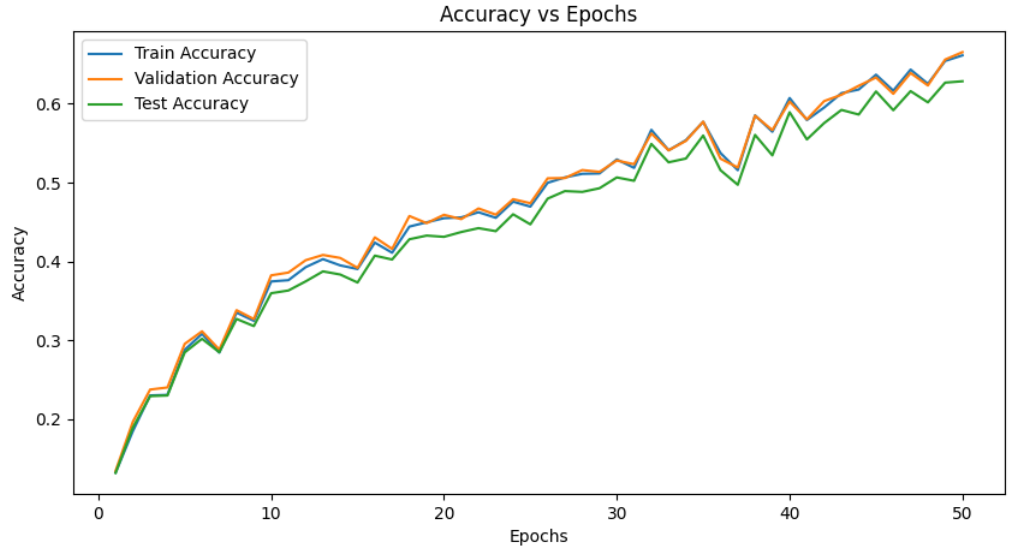


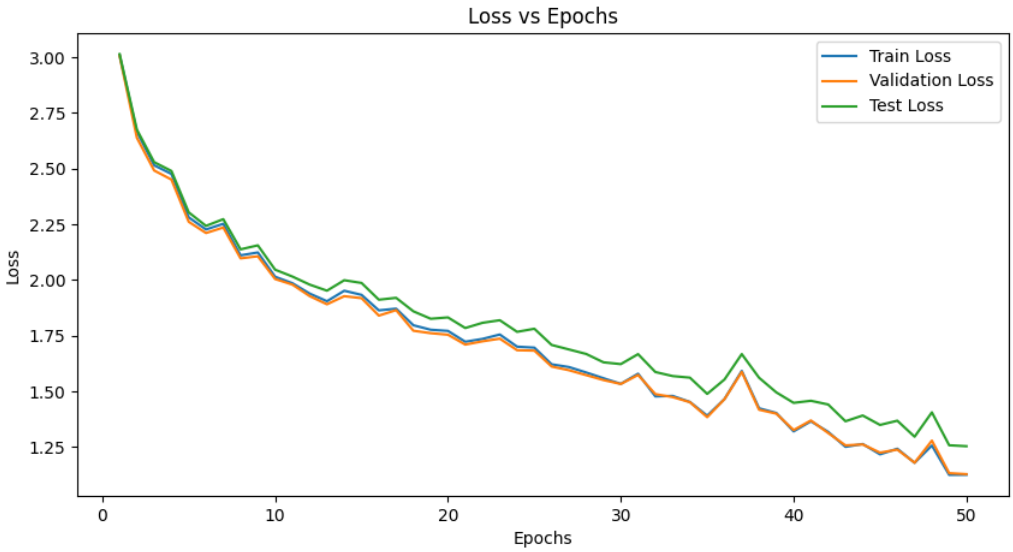


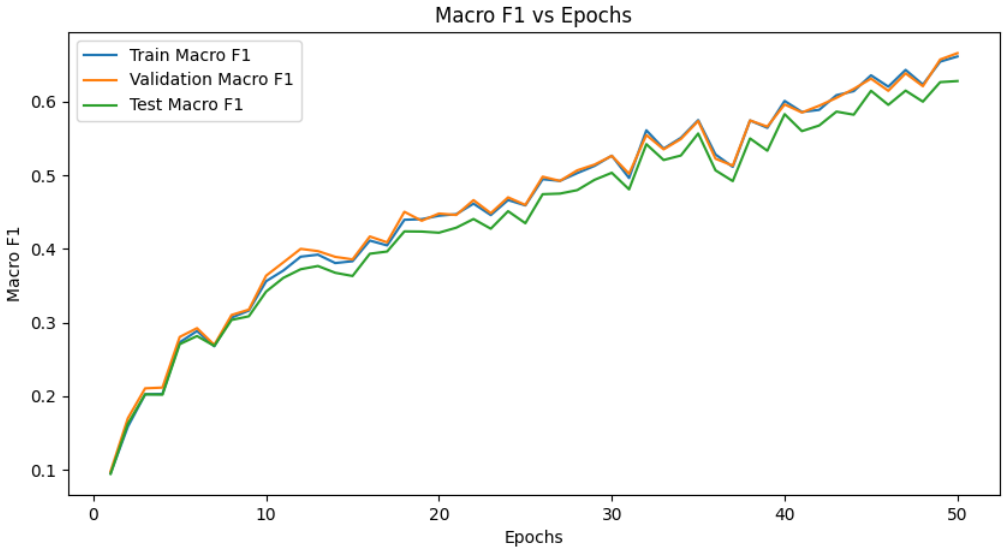
Impact of Normalization

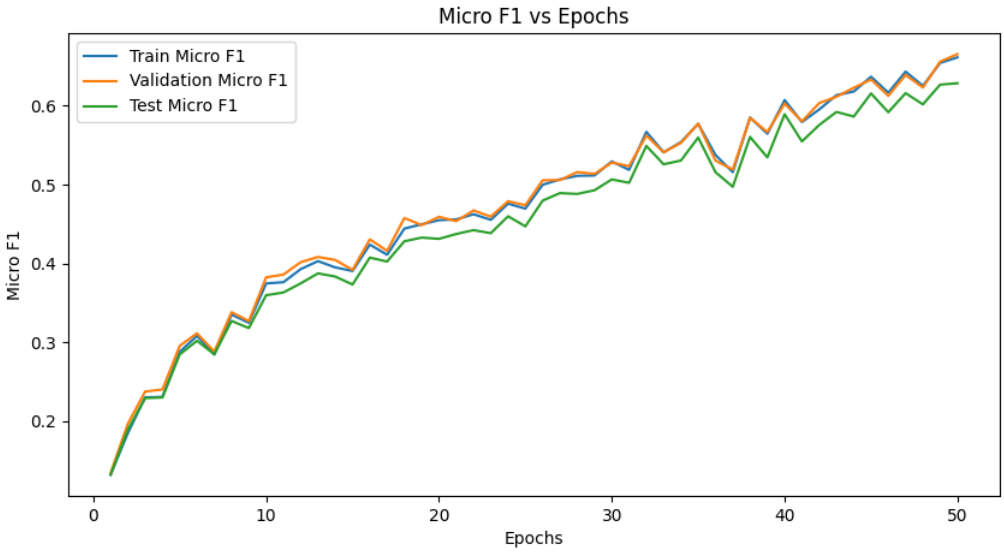
Resnet with custom Batch Normalization

```
{
    "train": {
        "train_loss": 1.1256951518680738,
        "train_accuracy": 0.661556242988748,
        "train_micro_f1": 0.661556242988748,
        "train_macro_f1": 0.6615537950801205
    },
    "val": {
        "val_loss": 1.128570377600057,
        "val_accuracy": 0.6656365149173897,
        "val_micro_f1": 0.6656365149173897,
        "val_macro_f1": 0.6660566801308103
    },
    "test": {
        "test_loss": 1.25388516492032,
        "test_accuracy": 0.6286666666666667,
        "test_micro_f1": 0.6286666666666667,
        "test_macro_f1": 0.6281564183355739
    }
}
```





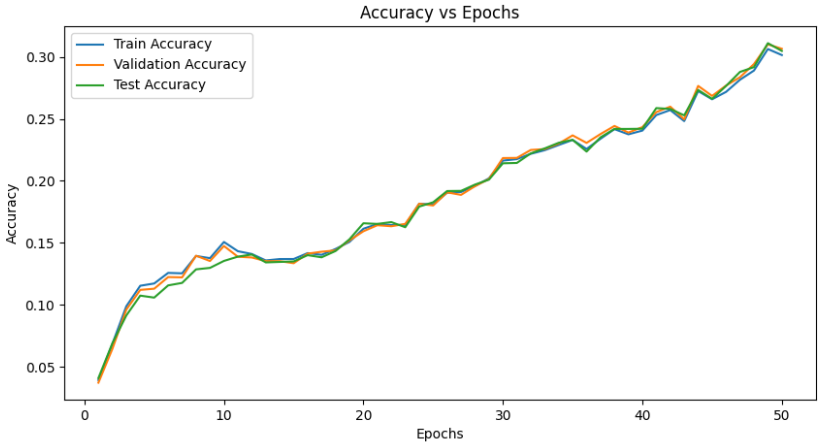


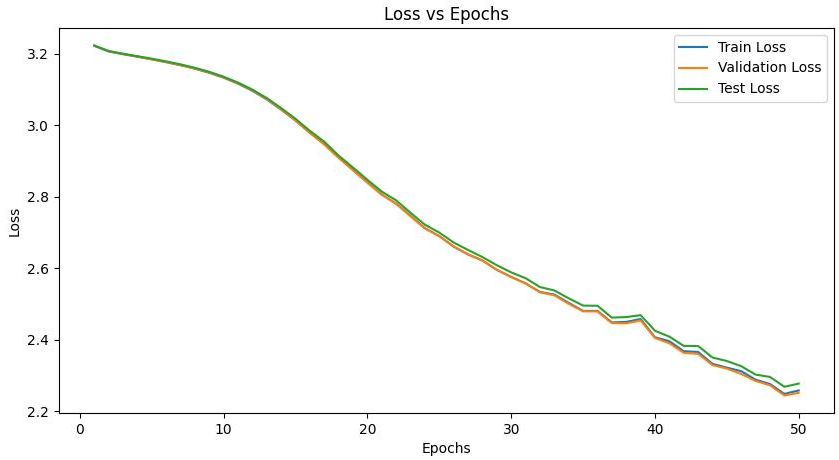
Observations

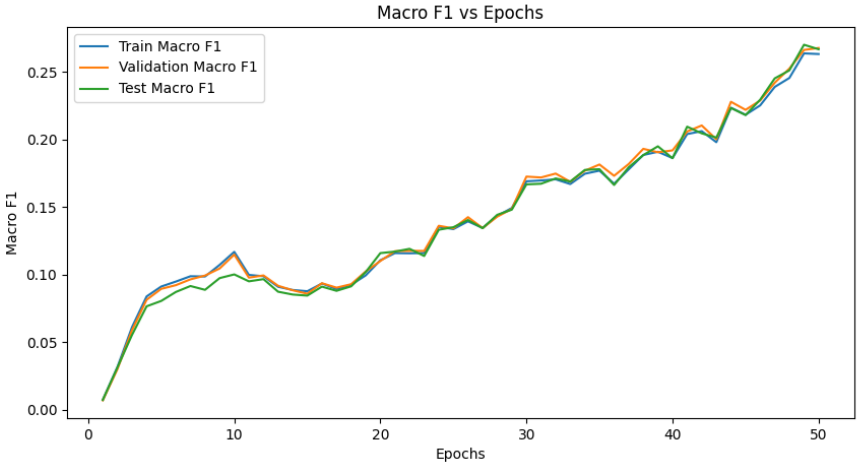
Accuracy and loss trends are similar in inbuilt batch normalization and custom batch normalization. Sanity checks successful

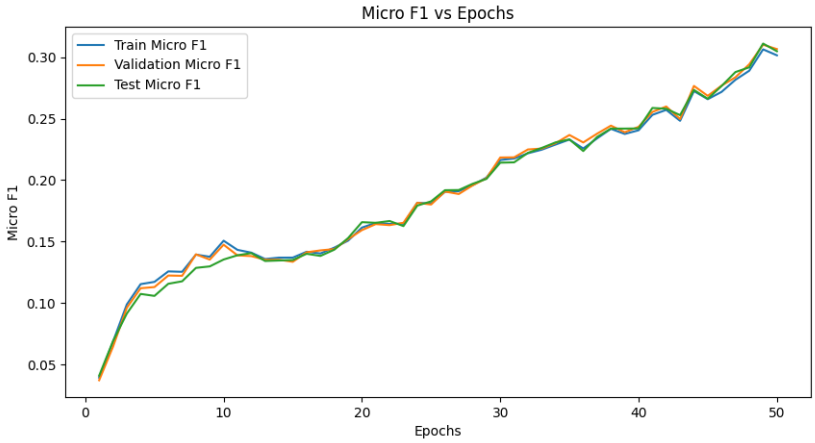
Resnet with custom Instance Normalization

```
{
    "train": {
        "train_loss": 2.258979521367861,
        "train_accuracy": 0.30149233436448314,
        "train_micro_f1": 0.30149233436448314,
        "train_macro_f1": 0.2634082776809527
    },
    "val": {
        "val_loss": 2.2521173999336734,
        "val_accuracy": 0.3065493878521336,
        "val_micro_f1": 0.3065493878521336,
        "val_macro_f1": 0.26789698679498636
    },
    "test": {
        "test_loss": 2.278038212086292,
        "test_accuracy": 0.3048,
        "test_micro_f1": 0.3048,
        "test_macro_f1": 0.2669304659778836
    }
}
```

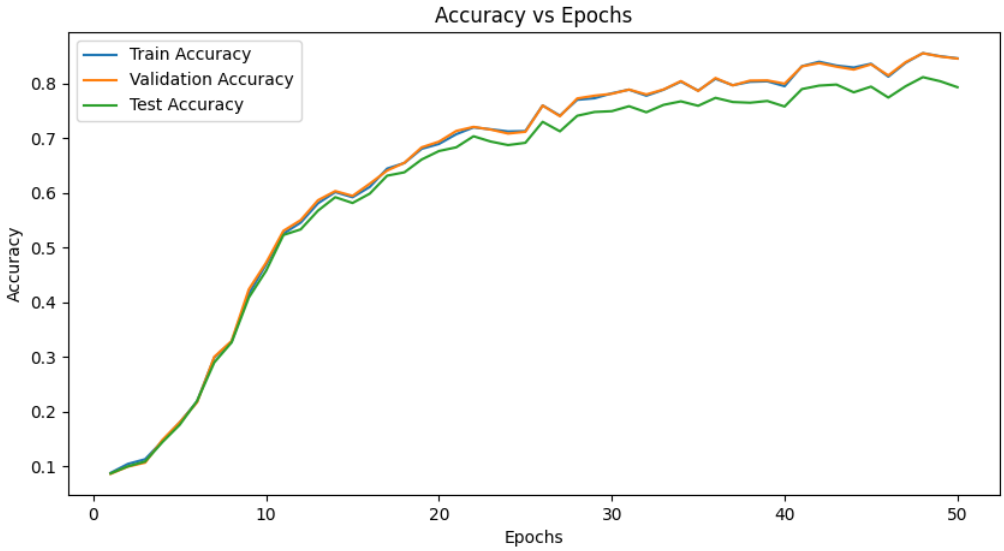


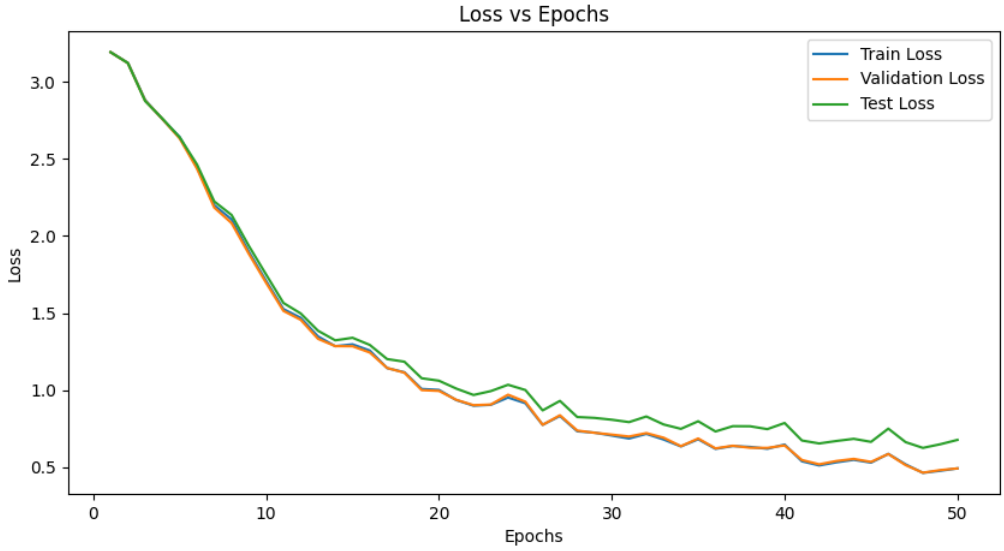


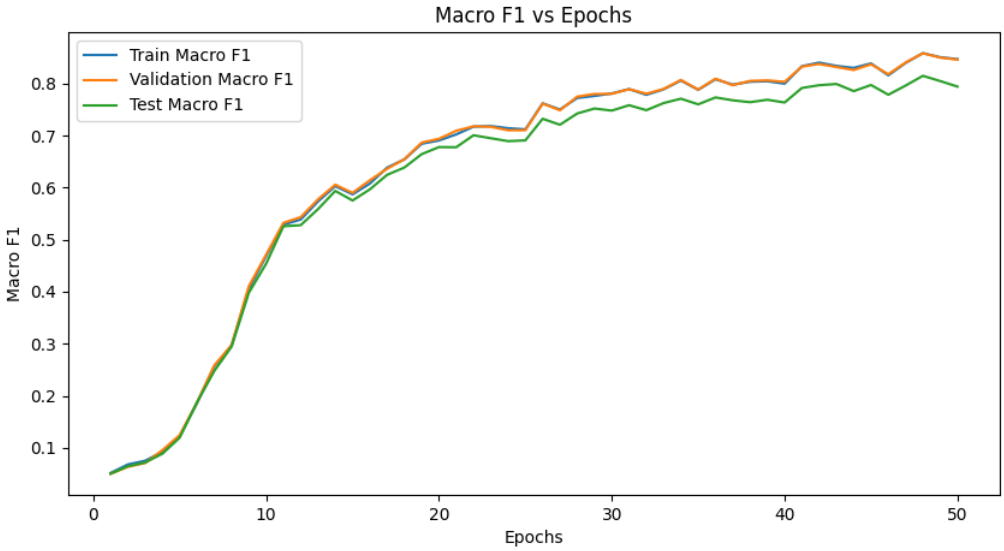


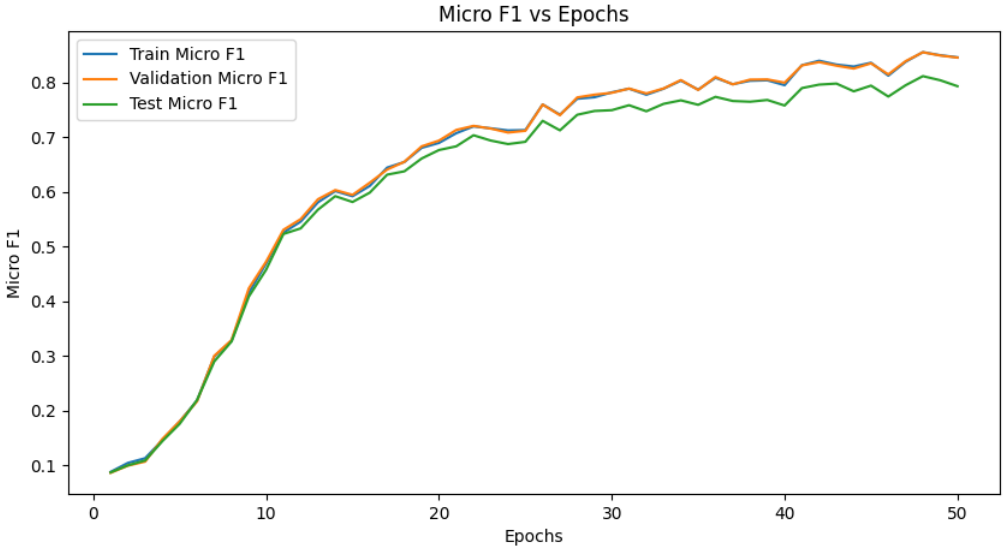
Resnet with custom Batch Instance Normalization


```
{
    "train": {
        "train_loss": 0.4920850399233725,
        "train_accuracy": 0.8464153380698236,
        "train_micro_f1": 0.8464153380698236,
        "train_macro_f1": 0.8471753352557143
    },
    "val": {
        "val_loss": 0.49186680516929226,
        "val_accuracy": 0.8459526922619756,
        "val_micro_f1": 0.8459526922619756,
        "val_macro_f1": 0.8462407178970895
    },
    "test": {
        "test_loss": 0.6768601452416562,
        "test_accuracy": 0.7936,
        "test_micro_f1": 0.7936,
        "test_macro_f1": 0.7943609966934463
    }
}
```


Accuracy vs Epochs
Train Accuracy
Validation Accuracy
Test Accuracy
Accuracy
Epochs
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
50

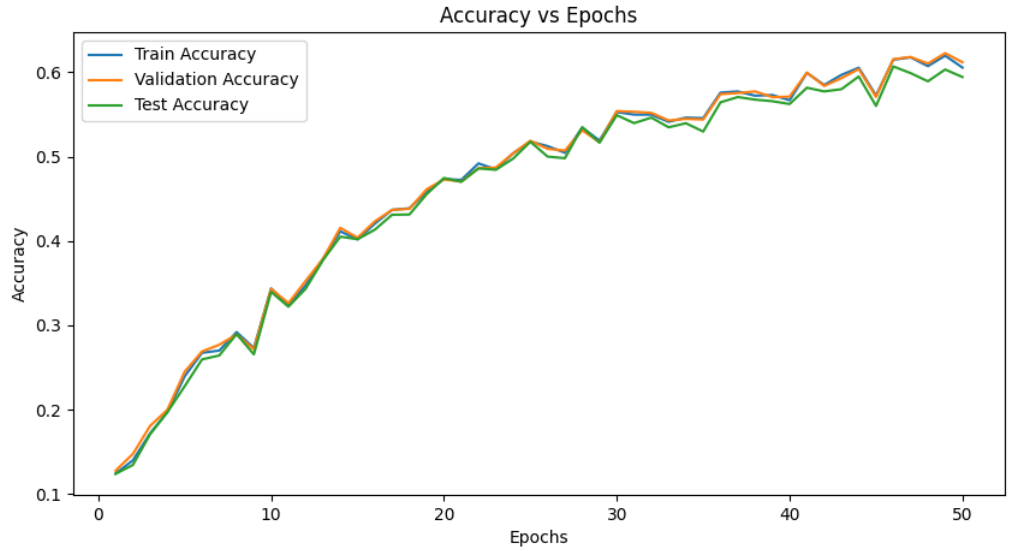

Loss vs Epochs
Train Loss
Validation Loss
Test Loss
Loss
Epochs
0.5
1.0
1.5
2.0
2.5
3.0
0
10
20
30
40
50

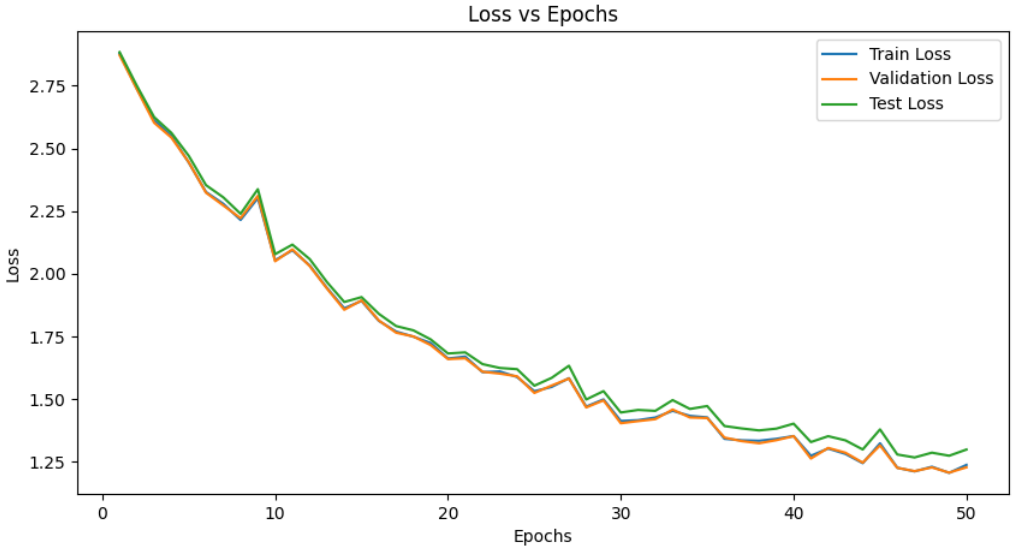


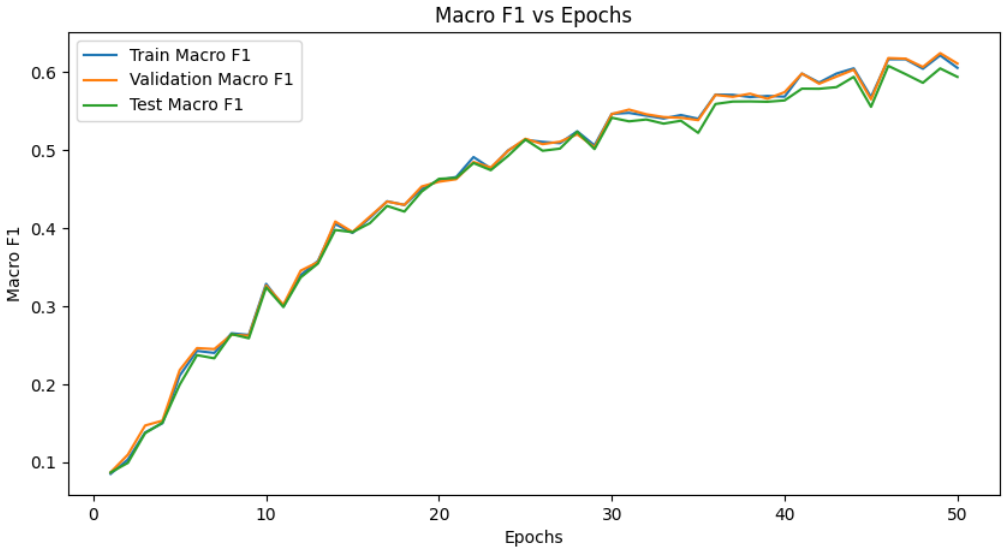


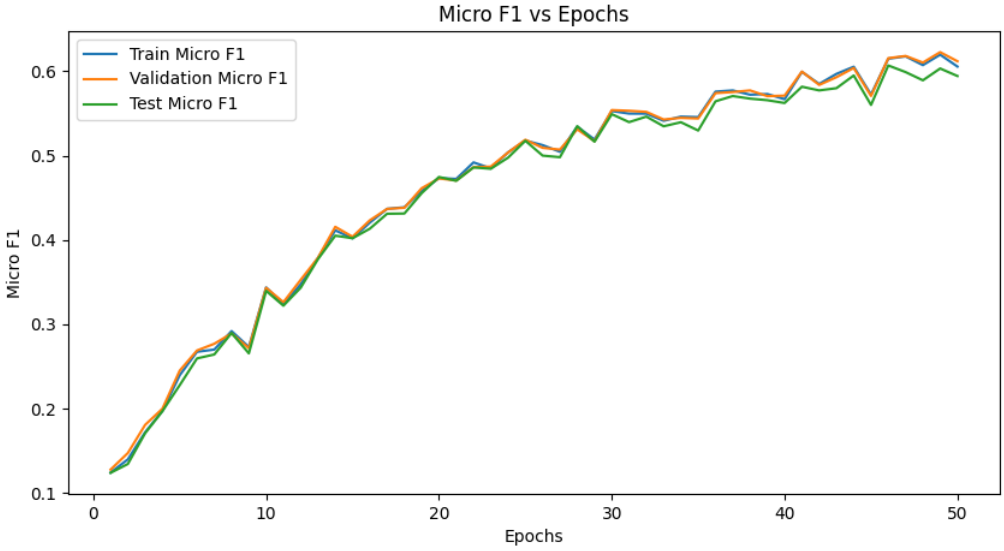
Resnet with custom Layer Normalization

```
{
    "train": {
        "train_loss": 1.2381730638768362,
        "train_accuracy": 0.6056361967569772,
        "train_micro_f1": 0.6056361967569772,
        "train_macro_f1": 0.6055347365749276
    },
    "val": {
        "val_loss": 1.2285274758991633,
        "val_accuracy": 0.6119101390704862,
        "val_micro_f1": 0.6119101390704862,
        "val_macro_f1": 0.6110633059512213
    },
    "test": {
        "test_loss": 1.2994497316948912,
        "test_accuracy": 0.5944,
        "test_micro_f1": 0.5944,
        "test_macro_f1": 0.5939828893447232
    }
}
```


Accuracy vs Epochs
Train Accuracy
Validation Accuracy
Test Accuracy
Accuracy
Epochs
0.1
0.2
0.3
0.4
0.5
0.6
0
10
20
30
40
50

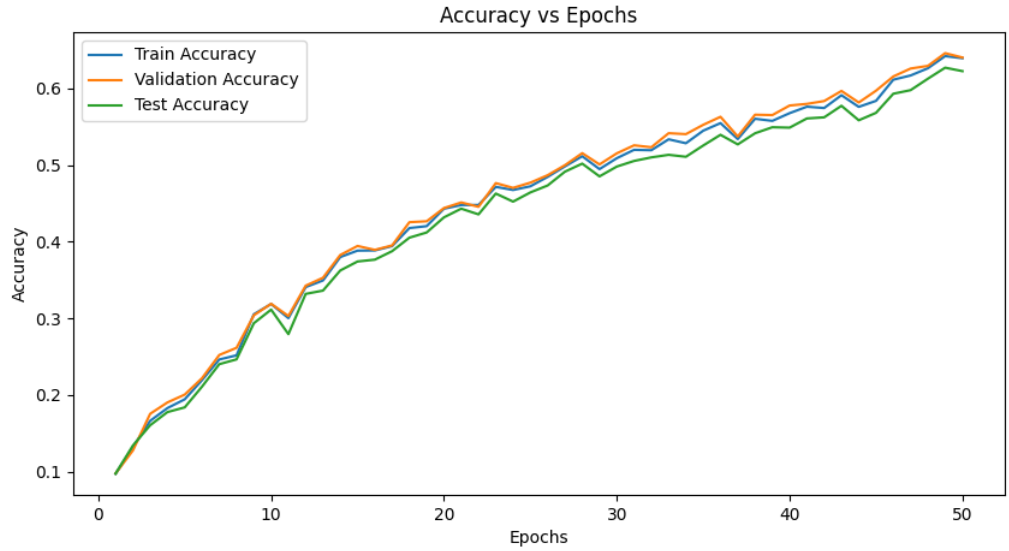

Loss vs Epochs
Train Loss
Validation Loss
Test Loss
Loss
Epochs
1.25
1.50
1.75
2.00
2.25
2.50
2.75
0
10
20
30
40
50

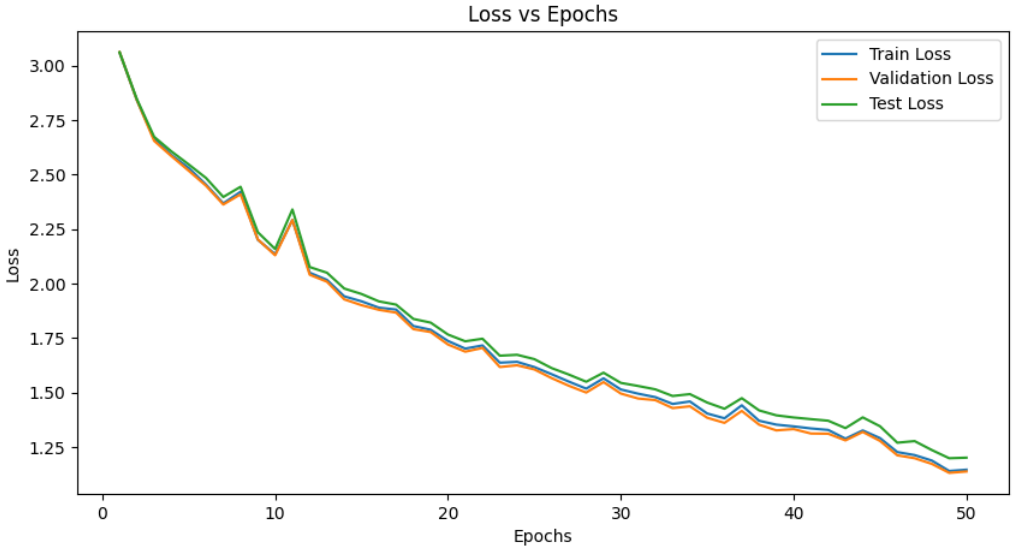


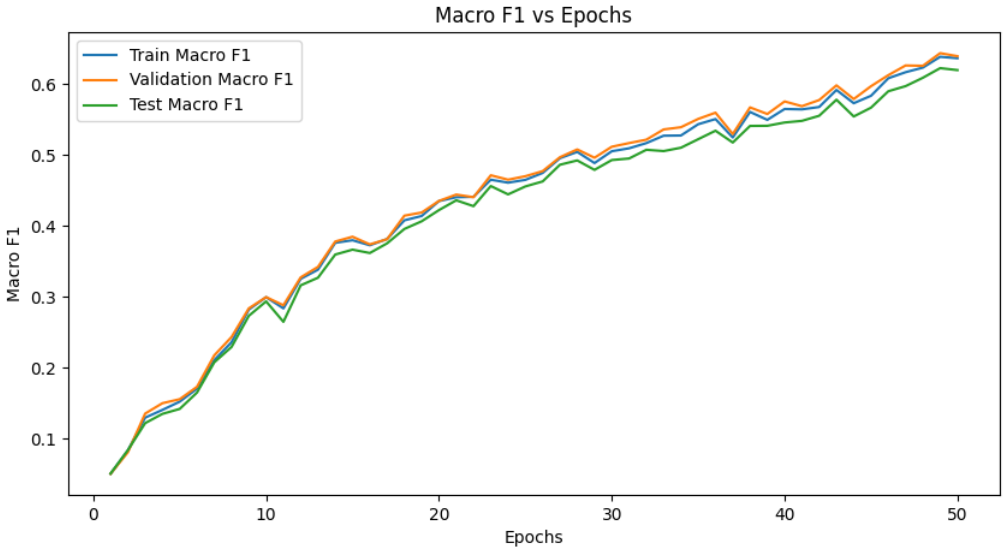


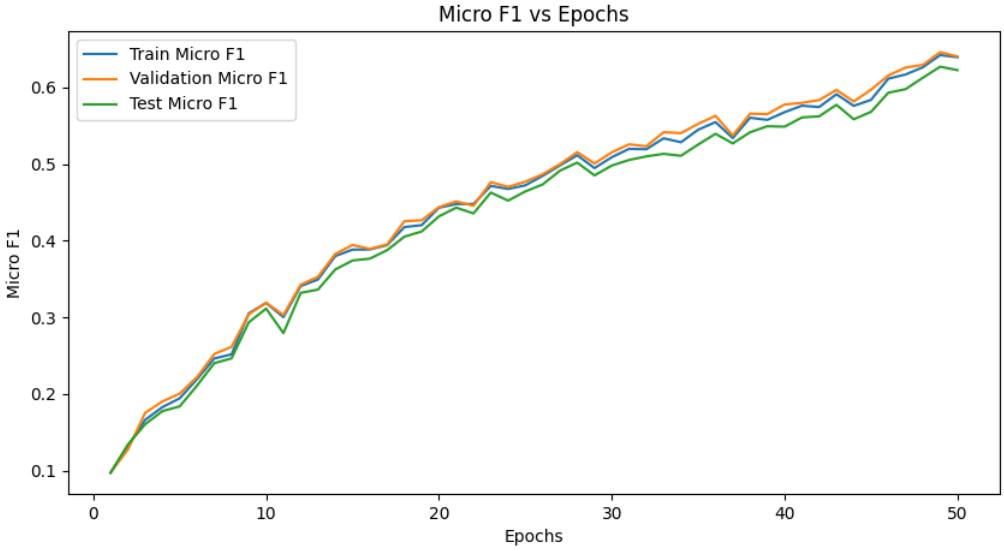
Resnet with custom Group Normalization

```
{
    "train": {
        "train_loss": 1.1465322592984075,
        "train_accuracy": 0.6394601760886562,
        "train_micro_f1": 0.6394601760886562,
        "train_macro_f1": 0.6369042811940904
    },
    "val": {
        "val_loss": 1.1383321373634918,
        "val_accuracy": 0.6403185546178534,
        "val_micro_f1": 0.6403185546178534,
        "val_macro_f1": 0.6397700240007888
    },
    "test": {
        "test_loss": 1.201749276100321,
        "test_accuracy": 0.6225333333333334,
        "test_micro_f1": 0.6225333333333334,
        "test_macro_f1": 0.6201463627378804
    }
}
```


Accuracy vs Epochs
Train Accuracy
Validation Accuracy
Test Accuracy
Accuracy
Epochs
0.1
0.2
0.3
0.4
0.5
0.6
0
10
20
30
40
50

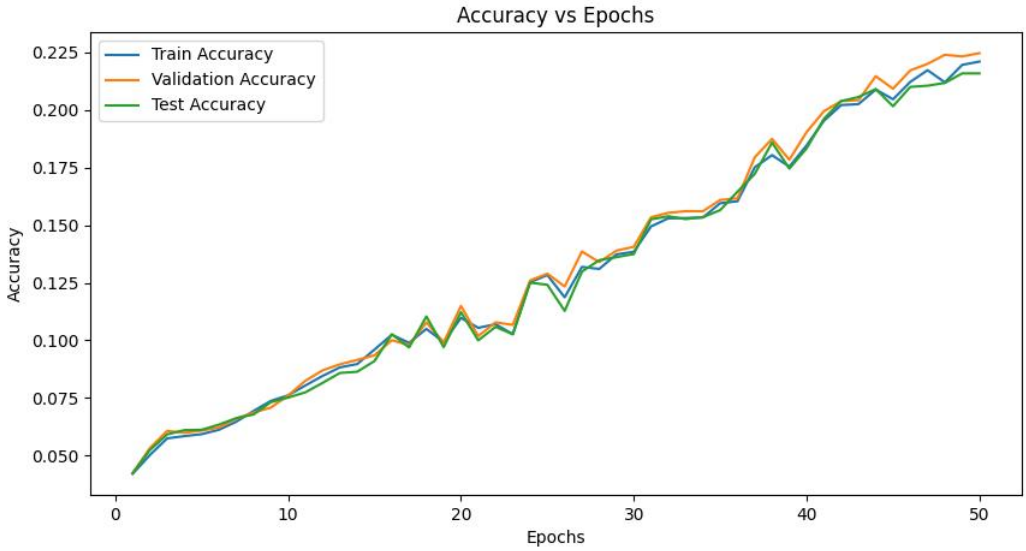

Loss vs Epochs
Train Loss
Validation Loss
Test Loss
Loss
Epochs
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
0
10
20
30
40
50

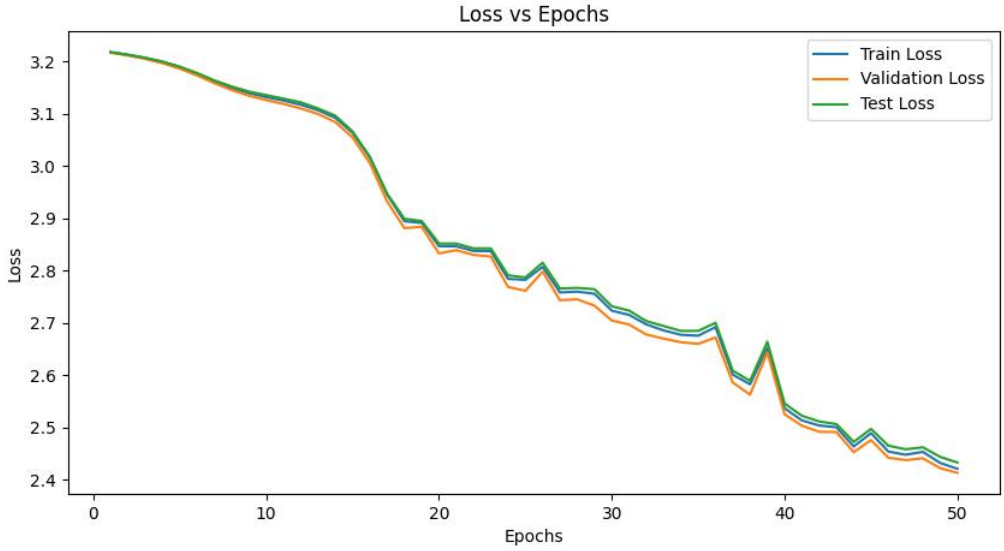


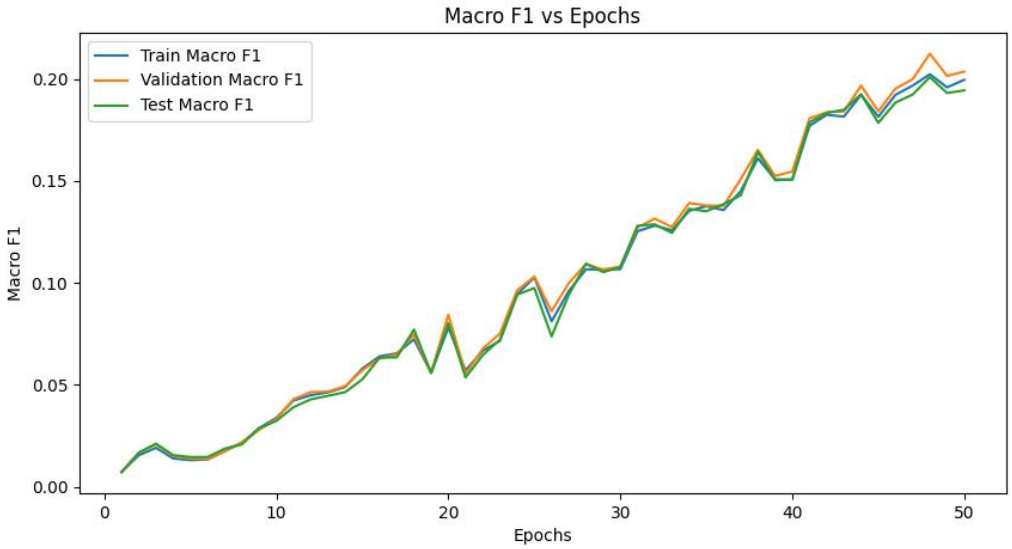


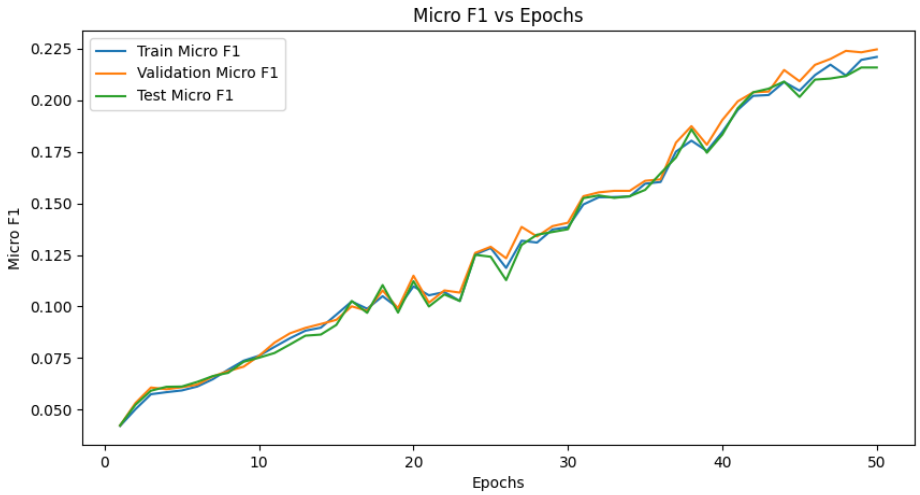
Resnet with custom No Normalization

```
{
    "train": {
        "train_loss": 2.4210832098255985,
        "train_accuracy": 0.2209946629499949,
        "train_micro_f1": 0.2209946629499949,
        "train_macro_f1": 0.19956954703592106
    },
    "val": {
        "val_loss": 2.4132905119725505,
        "val_accuracy": 0.22465232378461905,
        "val_micro_f1": 0.22465232378461905,
        "val_macro_f1": 0.2035995099600237
    },
    "test": {
        "test_loss": 2.4329213040940303,
        "test_accuracy": 0.21586666666666668,
        "test_micro_f1": 0.21586666666666668,
        "test_macro_f1": 0.19437245423564936
    }
}
```


Accuracy vs Epochs
Train Accuracy
Validation Accuracy
Test Accuracy
Accuracy
Epochs
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0
10
20
30
40
50


Loss vs Epochs
Train Loss
Validation Loss
Test Loss
Loss
Epochs
3.2
3.1
3.0
2.9
2.8
2.7
2.6
2.5
2.4
0
10
20
30
40
50





Observations

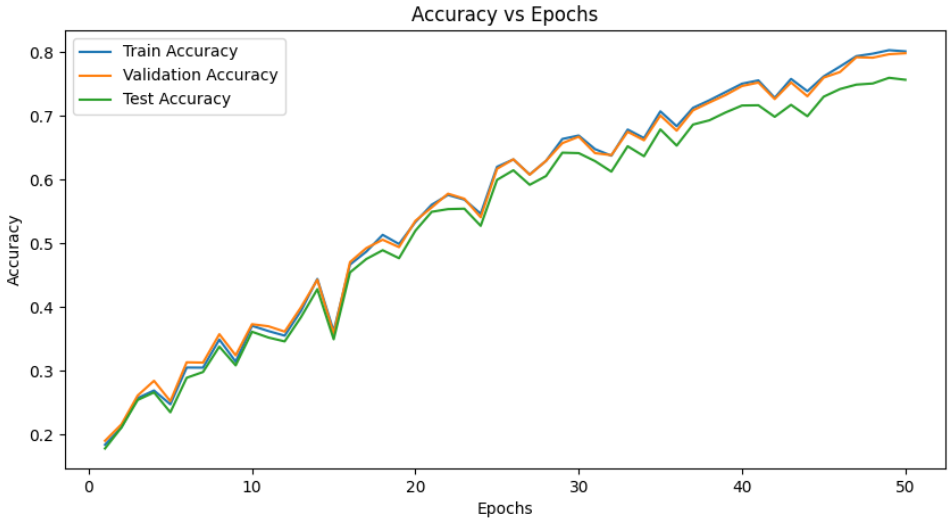
Normalization Type	Validation Accuracy	Test Accuracy
Inbuilt Batch Norm	0.6722928800665636	0.6493333333333333
Custom Batch Norm	0.6656365149173897	0.6286666666666667
Custom Instance Norm	0.3065493878521336	0.3048
Custom Batch Instance Norm	0.8459526922619756	0.7936
Custom Group Norm	0.6403185546178534	0.6225333333333334
Custom Layer Norm	0.6119101390704862	0.5944
No Norm	0.22465232378461905	0.21586666666666668

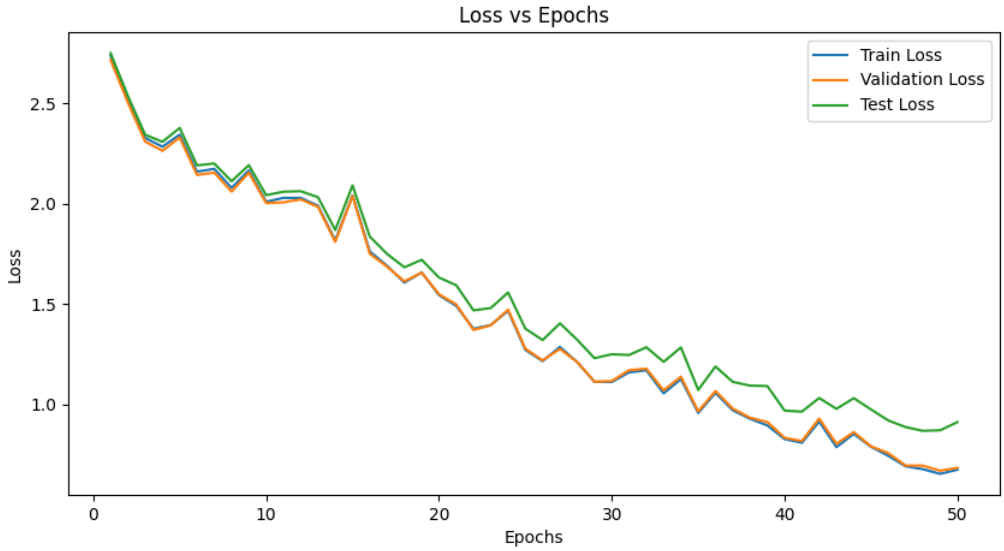
Batch Instance Norm performs better given the hyperparameters are fixed. No normalization setting slows down the training. Batch Norm and Group Norm produces decent results

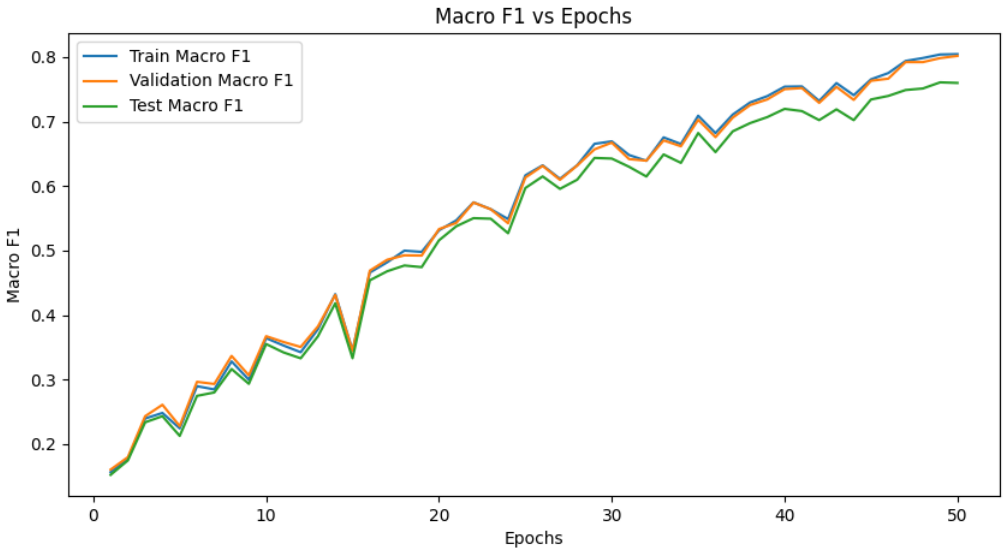
Impact of Batch Size

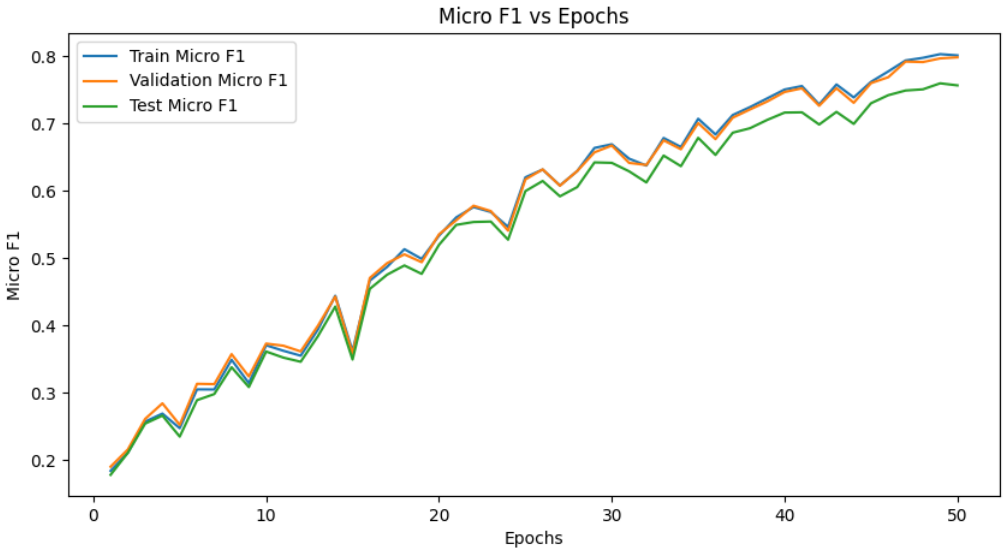
Resnet with custom Batch Normalization (BATCH SIZE 8)


```
{
    "train": {
        "train_loss": 0.6736252784112806,
        "train_accuracy": 0.8009654281537886,
        "train_micro_f1": 0.8009654281537886,
        "train_macro_f1": 0.804551115672139
    },
    "val": {
        "val_loss": 0.6829180782266889,
        "val_accuracy": 0.7980506359205991,
        "val_micro_f1": 0.7980506359205991,
        "val_macro_f1": 0.8018342125126184
    },
    "test": {
        "test_loss": 0.9109166109654456,
        "test_accuracy": 0.7564,
        "test_micro_f1": 0.7564,
        "test_macro_f1": 0.7599149575204979
    }
}
```

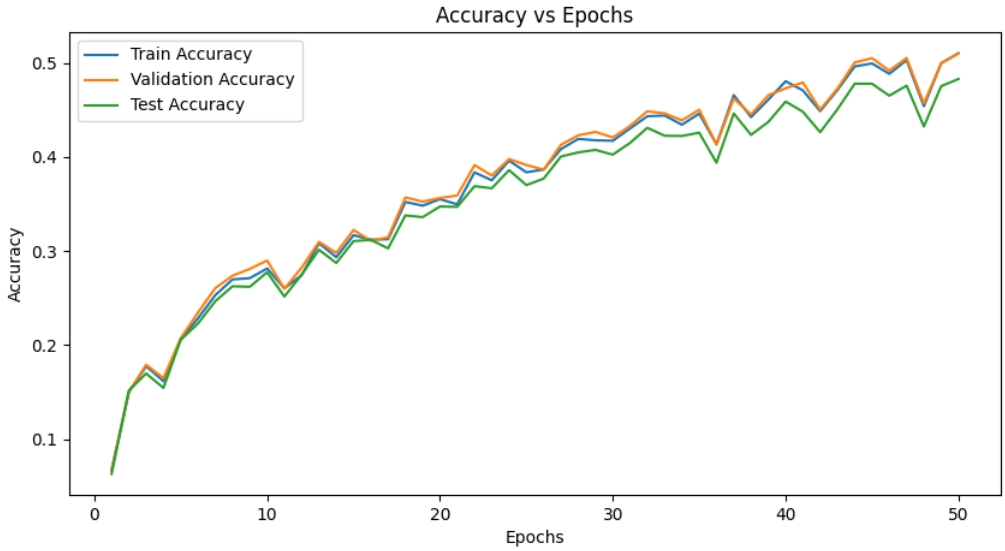

Loss vs Epochs
Train Loss
Validation Loss
Test Loss
Loss
Epochs
2.5
2.0
1.5
1.0
0
10
20
30
40
50

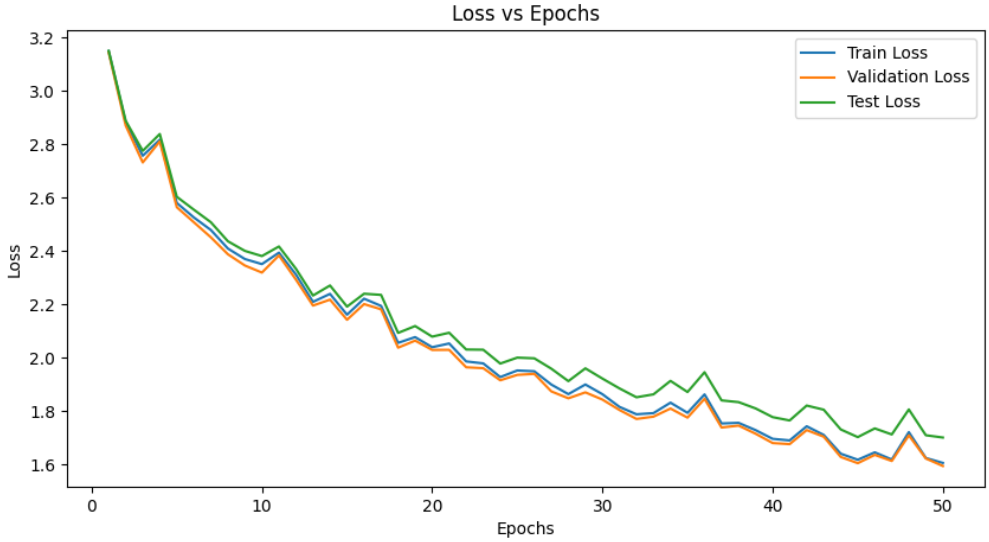

Macro F1 vs Epochs
Train Macro F1
Validation Macro F1
Test Macro F1
Macro F1
Epochs
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50

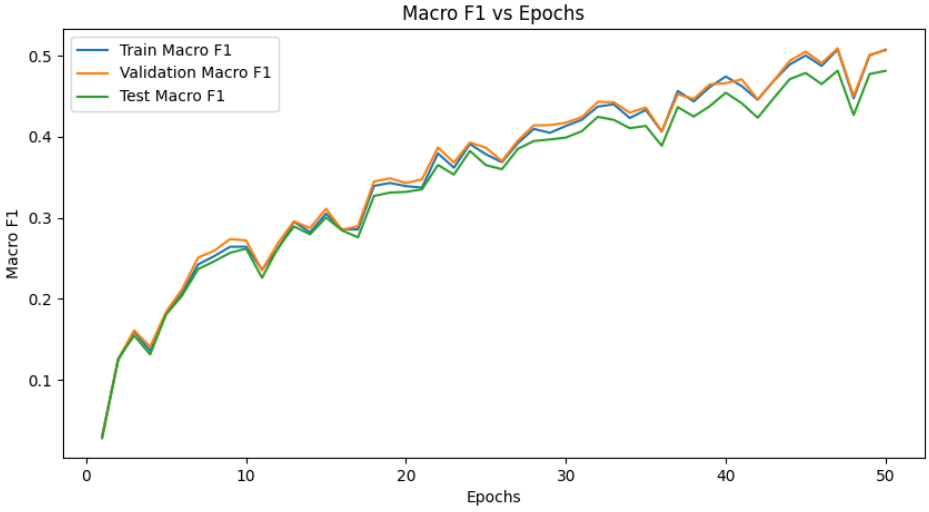

Micro F1 vs Epochs
Train Micro F1
Validation Micro F1
Test Micro F1
Micro F1
Epochs
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50

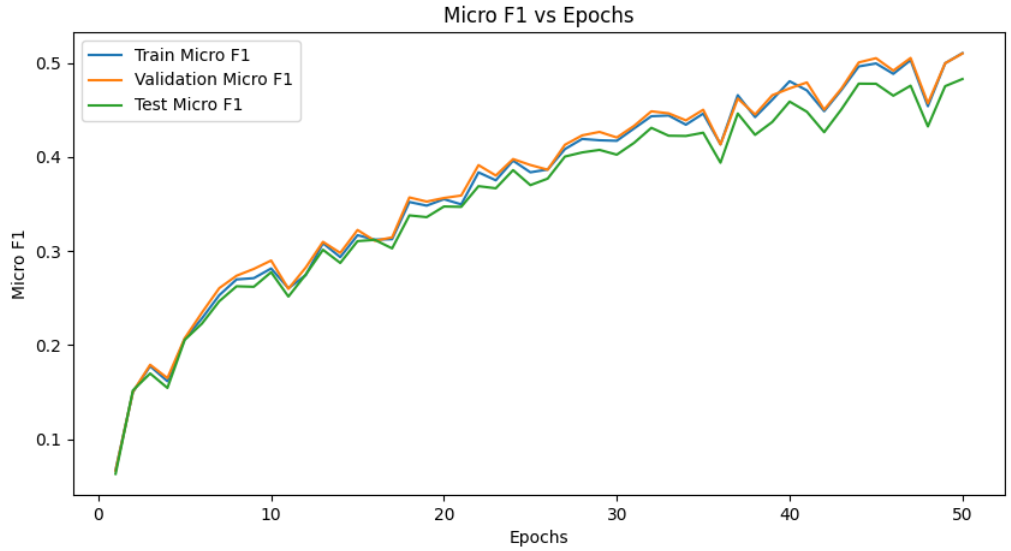
Resnet with custom Batch Normalization (BATCH SIZE 128)


```
{
    "train": {
        "train_loss": 1.605818880122641,
        "train_accuracy": 0.510283169595812,
        "train_micro_f1": 0.510283169595812,
        "train_macro_f1": 0.507422980849897
    },
    "val": {
        "val_loss": 1.594307384707711,
        "val_accuracy": 0.5099251158920718,
        "val_micro_f1": 0.5099251158920718,
        "val_macro_f1": 0.5065690504565279
    },
    "test": {
        "test_loss": 1.701052071684498,
        "test_accuracy": 0.4828,
        "test_micro_f1": 0.4828,
        "test_macro_f1": 0.4813478099457022
    }
}
```


Accuracy vs Epochs
Train Accuracy
Validation Accuracy
Test Accuracy
Accuracy
Epochs
0.1
0.2
0.3
0.4
0.5
0
10
20
30
40
50

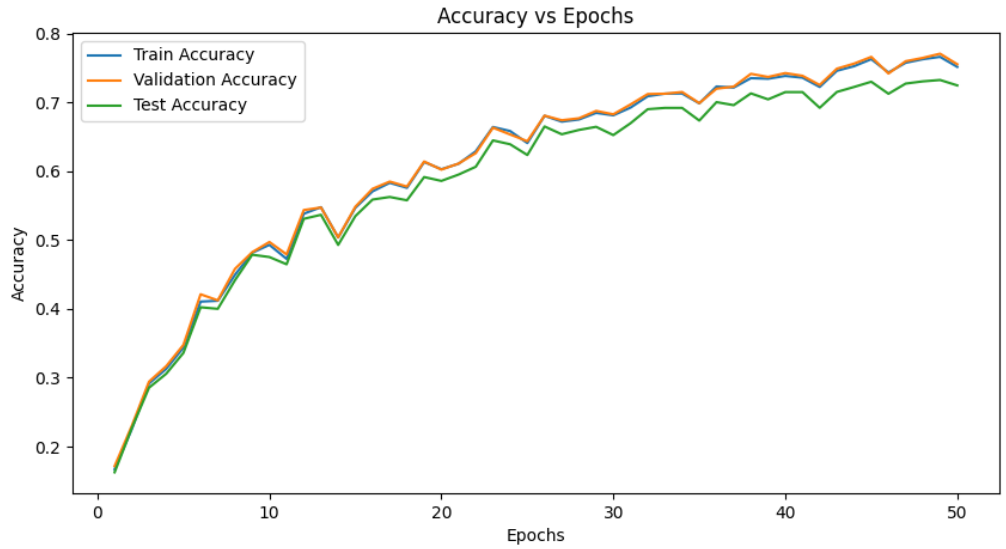

Loss vs Epochs
Train Loss
Validation Loss
Test Loss
Loss
Epochs
1.6
1.8
2.0
2.2
2.4
2.6
2.8
3.0
3.2
0
10
20
30
40
50

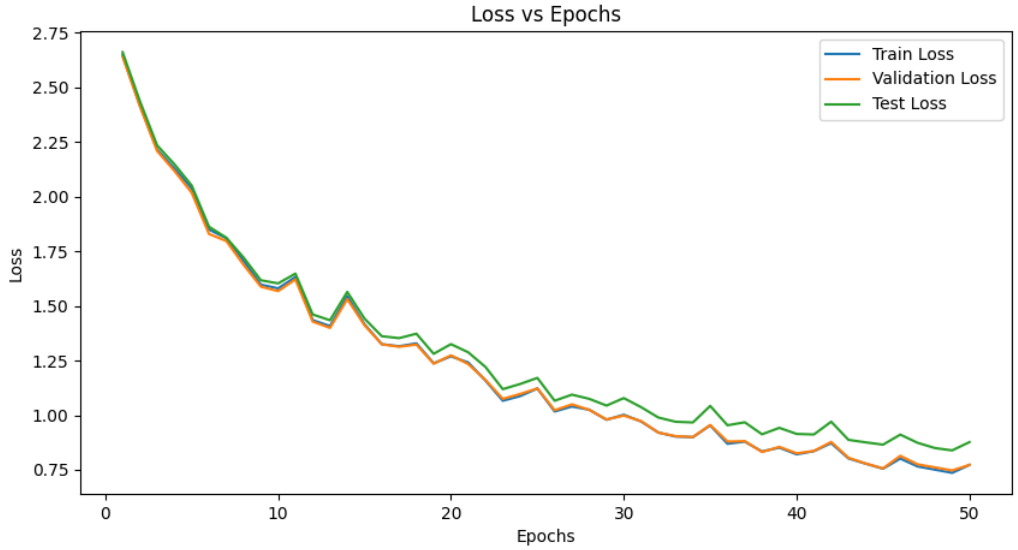

Macro F1 vs Epochs
Train Macro F1
Validation Macro F1
Test Macro F1
Macro F1
Epochs
0.1
0.2
0.3
0.4
0.5
0
10
20
30
40
50

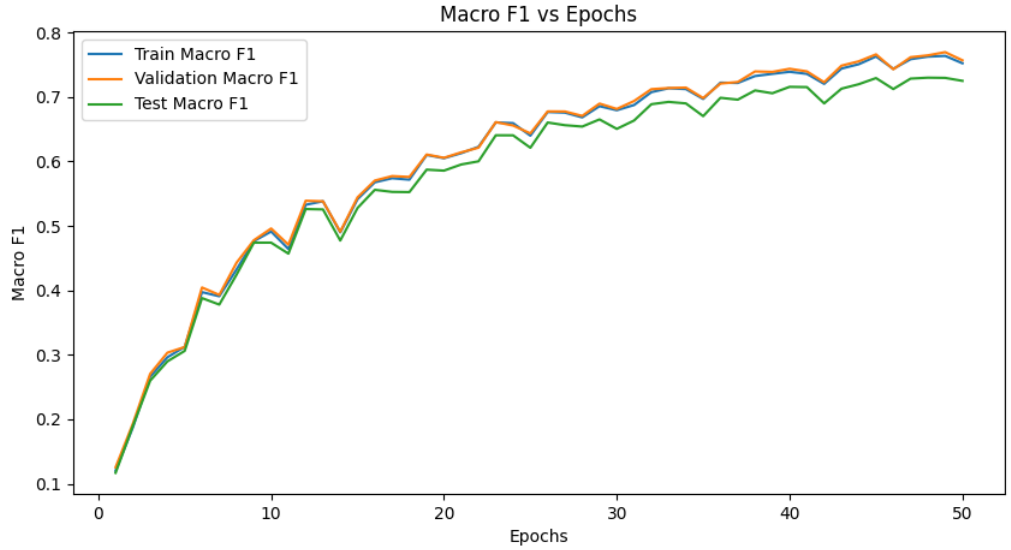


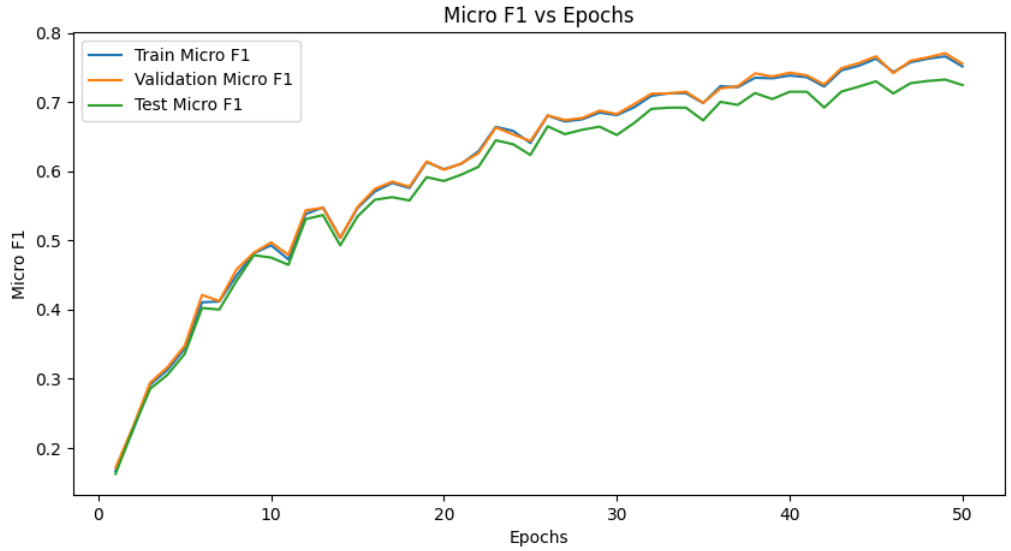
Resnet with custom Group Normalization (BATCH SIZE 8)


```
{
    "train": {
        "train_loss": 0.7723269656395754,
        "train_accuracy": 0.751436244348506,
        "train_micro_f1": 0.751436244348506,
        "train_macro_f1": 0.7524609442902027
    },
    "val": {
        "val_loss": 0.7725104815627629,
        "val_accuracy": 0.7552597171044811,
        "val_micro_f1": 0.7552597171044811,
        "val_macro_f1": 0.7570525645164285
    },
    "test": {
        "test_loss": 0.8767406503592473,
        "test_accuracy": 0.7245333333333334,
        "test_micro_f1": 0.7245333333333334,
        "test_macro_f1": 0.7251739366275344
    }
}
```


Accuracy vs Epochs
Train Accuracy
Validation Accuracy
Test Accuracy
Accuracy
Epochs
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
50

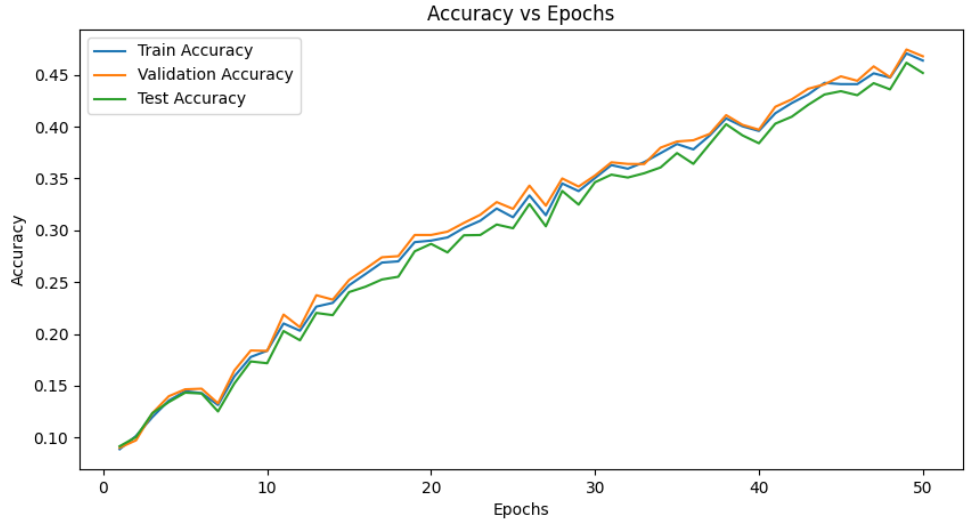

Loss vs Epochs
Train Loss
Validation Loss
Test Loss
Loss
Epochs
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
0
10
20
30
40
50

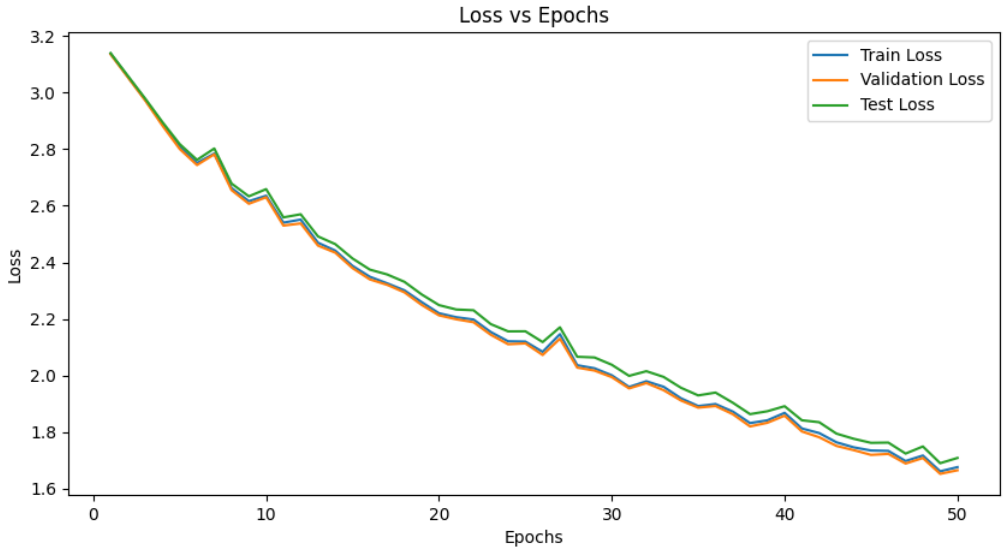

Macro F1 vs Epochs
Train Macro F1
Validation Macro F1
Test Macro F1
Macro F1
Epochs
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
10
20
30
40
50

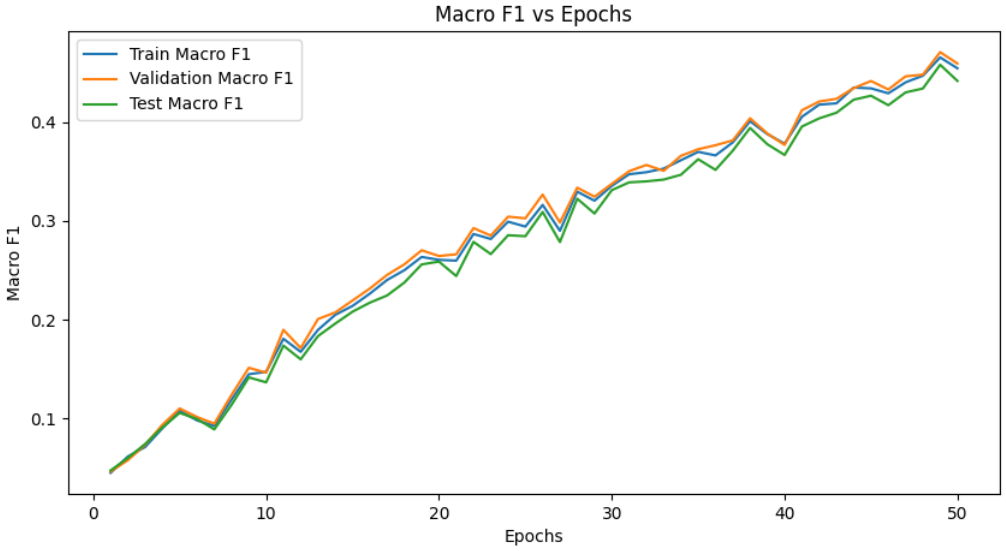


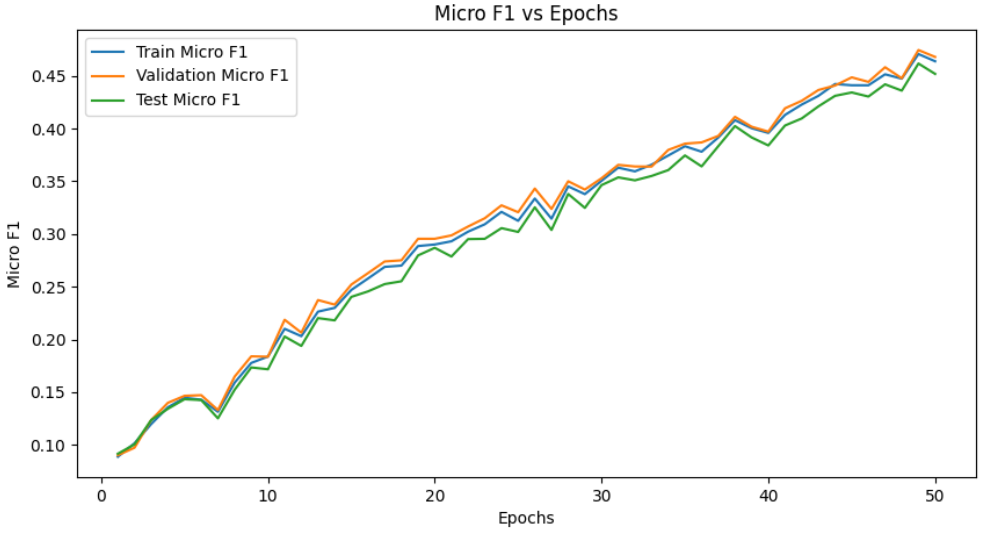
Resnet with custom Group Normalization (BATCH SIZE 128)


```
{
    "train": {
        "train_loss": 1.6758654127950254,
        "train_accuracy": 0.46384743515654214,
        "train_micro_f1": 0.46384743515654214,
        "train_macro_f1": 0.4543085500179798
    },
    "val": {
        "val_loss": 1.6647579579642324,
        "val_accuracy": 0.46796624271960063,
        "val_micro_f1": 0.46796624271960063,
        "val_macro_f1": 0.4591190485319773
    },
    "test": {
        "test_loss": 1.708329980656252,
        "test_accuracy": 0.4518666666666667,
        "test_micro_f1": 0.4518666666666667,
        "test_macro_f1": 0.4415523894003313
    }
}
```


Loss vs Epochs
Train Loss
Validation Loss
Test Loss
Loss
Epochs
3.2
3.0
2.8
2.6
2.4
2.2
2.0
1.8
1.6
0
10
20
30
40
50


Macro F1 vs Epochs
Train Macro F1
Validation Macro F1
Test Macro F1
Macro F1
Epochs
0.4
0.3
0.2
0.1
0
10
20
30
40
50


Micro F1 vs Epochs
Train Micro F1
Validation Micro F1
Test Micro F1
Micro F1
Epochs
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
10
20
30
40
50

Observations

Normalization Type	Batch Size	Validation Accuracy	Test Accuracy
Custom Batch Norm	8	0.7980506359205991	0.7564
Custom Batch Norm	128	0.5099251158920718	0.4828
Custom Group Norm	8	0.7552597171044811	0.7245333333333334
Custom Group Norm	128	0.4679662427196006	0.4518666666666667

It can be observed that Batch size 8 performs significantly better for this task than batch size 128, attributing to more steps taken.

Even within same batch size, Batch Norm consistently performs better than Group Norm

Best model

Hyperparameters used are:

N = 2

Batch size = 16

Epochs = 110 (after Early Stopping)

R = 25

Optimizer = Adam

Learning Rate = 0.0001

Normalization = Layer Norm

Resnet with custom Layer Normalization (BATCH SIZE 16)

Training Loss: 0.2727 Training Accuracy: 0.9160 Training Macro F1: 0.9163 Training Micro F1: 0.9160

Validation Loss: 0.2891 Validation Accuracy: 0.9109 Validation Macro F1: 0.9115 Validation Micro F1: 0.9109

Test Loss: 0.5051 Test Accuracy: 0.8524 Test Macro F1: 0.8532 Test Micro F1: 0.8524

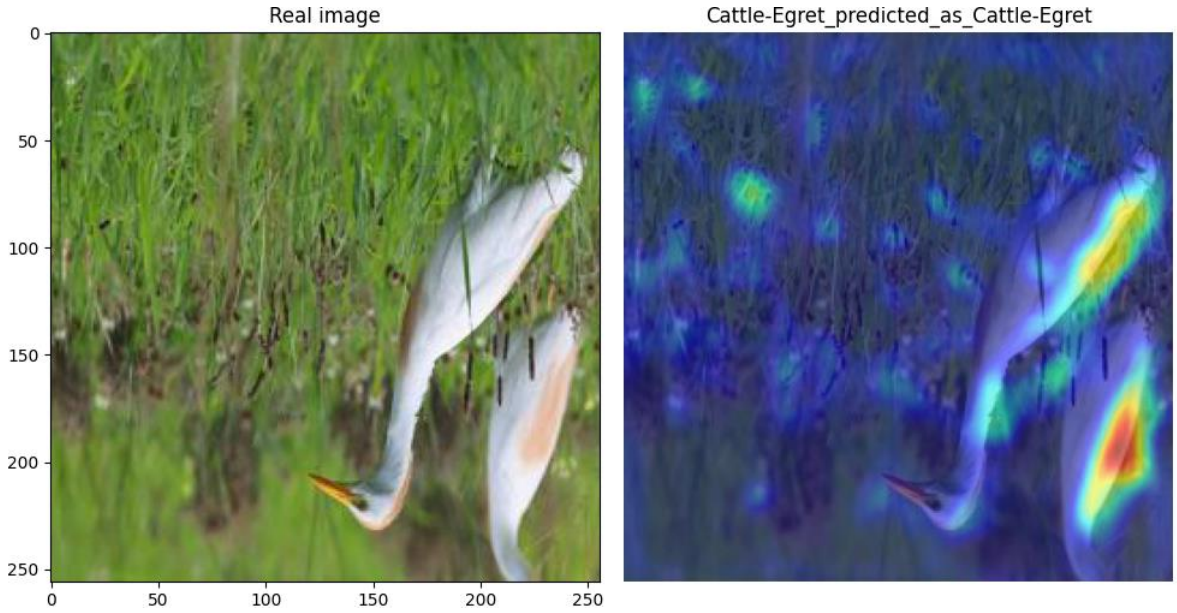
Please use the below command

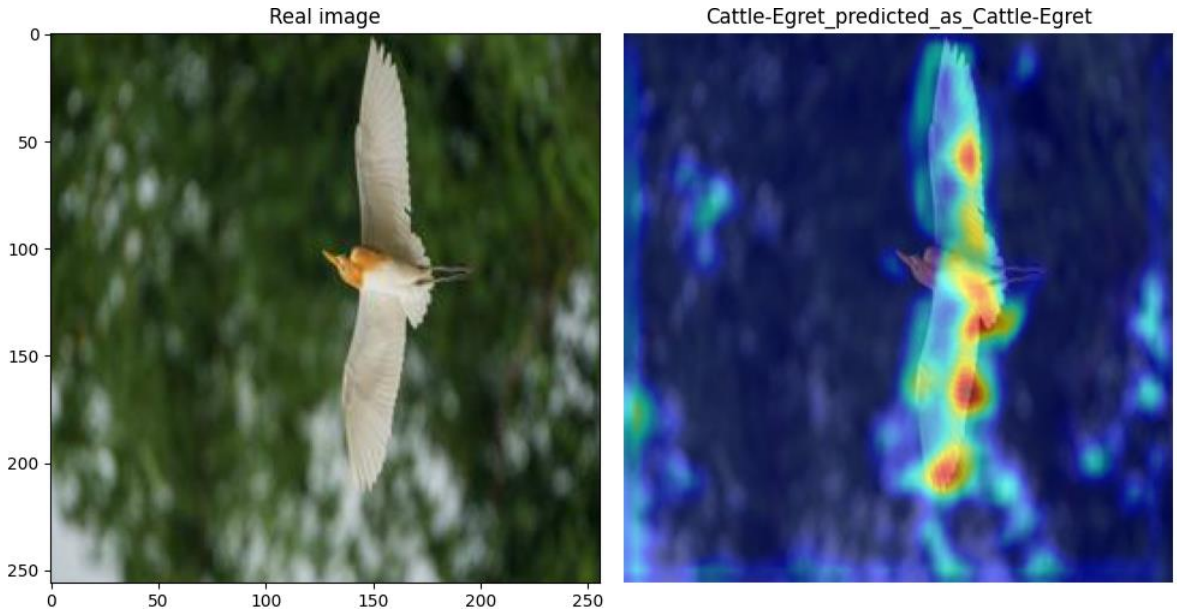
python3 infer.py --model_file models/best.pth --normalization ln --n 2 --test_data_file birds_test --output_file output_best.txt

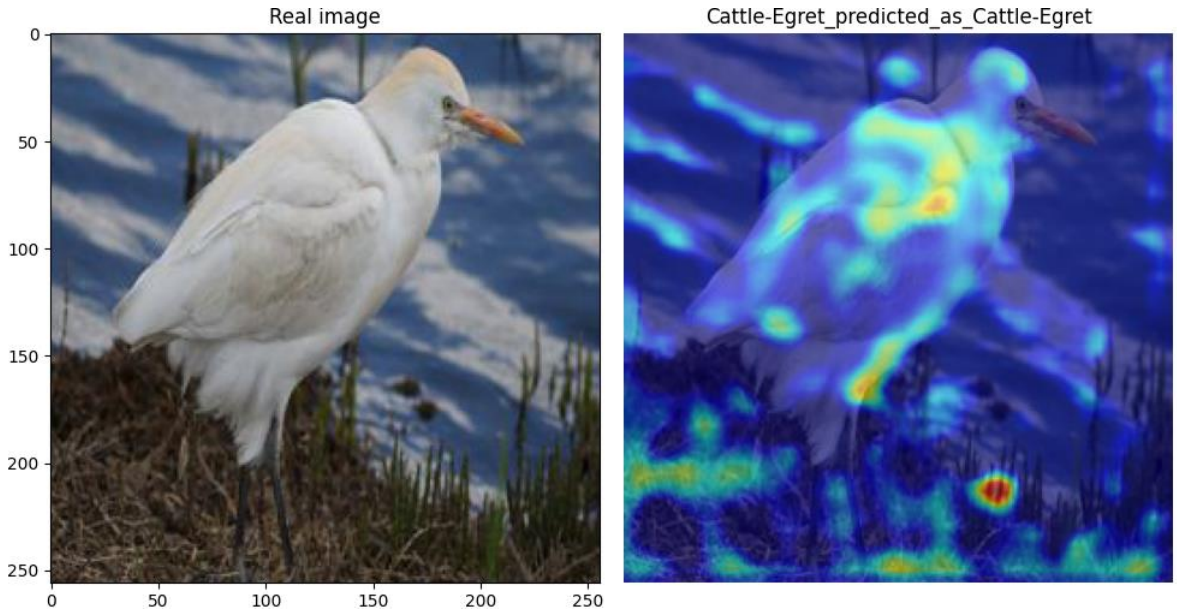
Visualize Models Thinking

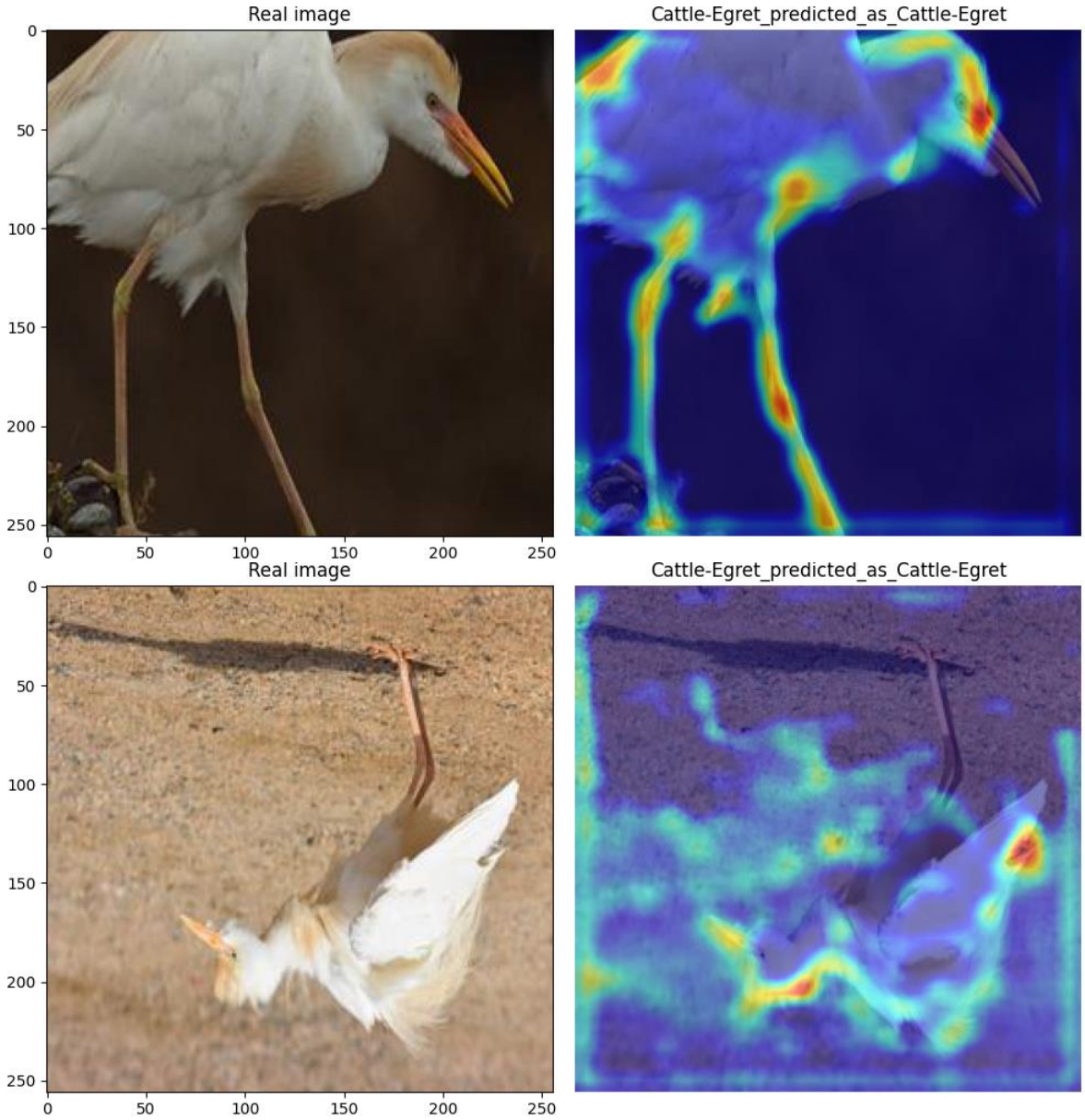
Cattle Egret

Correctly labelled samples

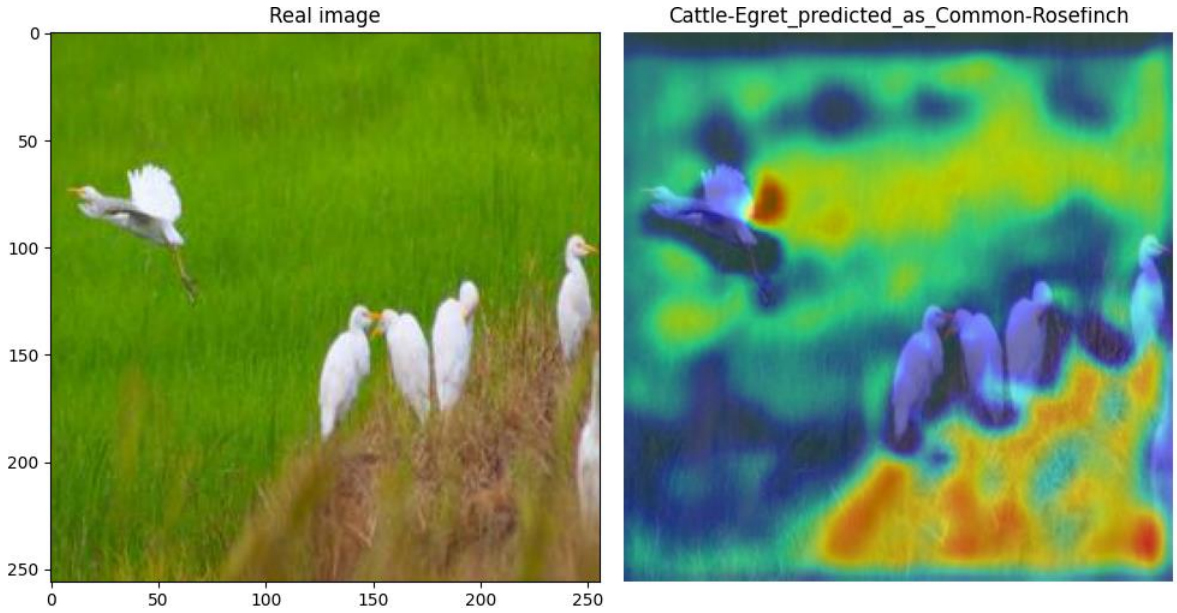

Real image
Cattle-Egret_predicted_as_Cattle-Egret
0
50
100
150
200
250

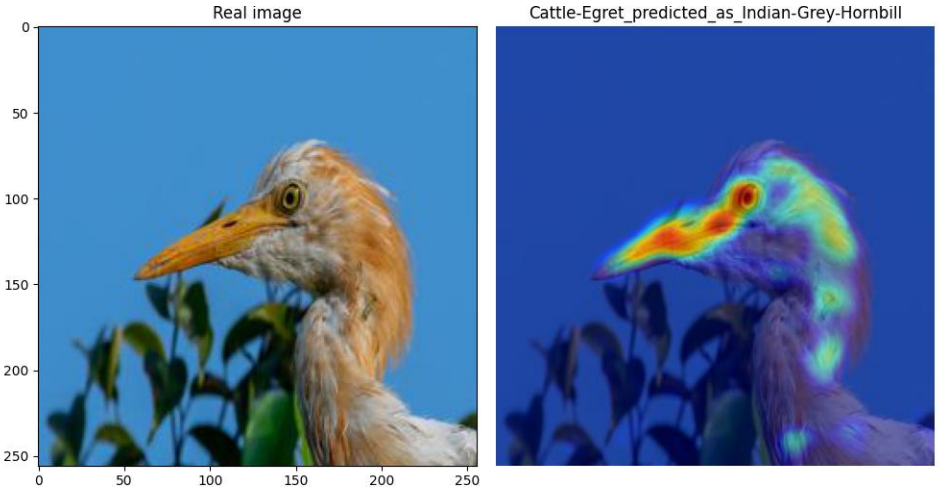

Real image
Cattle-Egret_predicted_as_Cattle-Egret
0
50
100
150
200
250

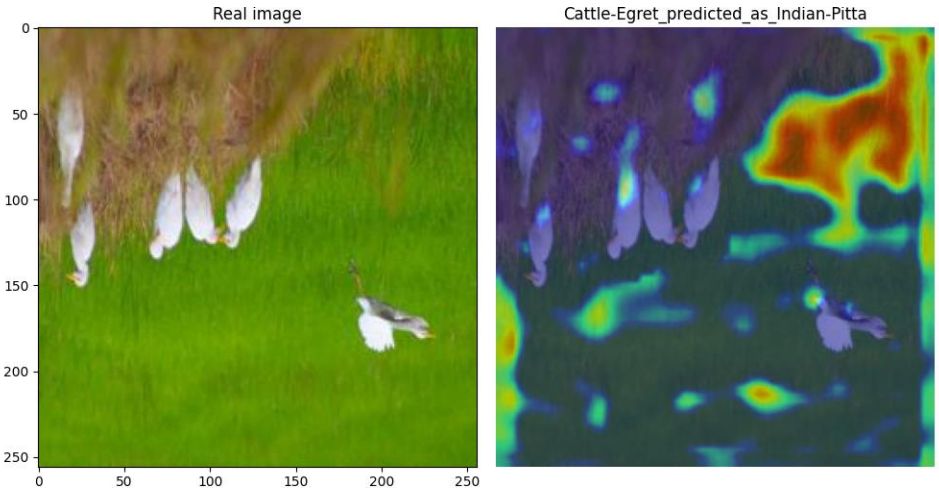

Real image
Cattle-Egret_predicted_as_Cattle-Egret
0
50
100
150
200
250

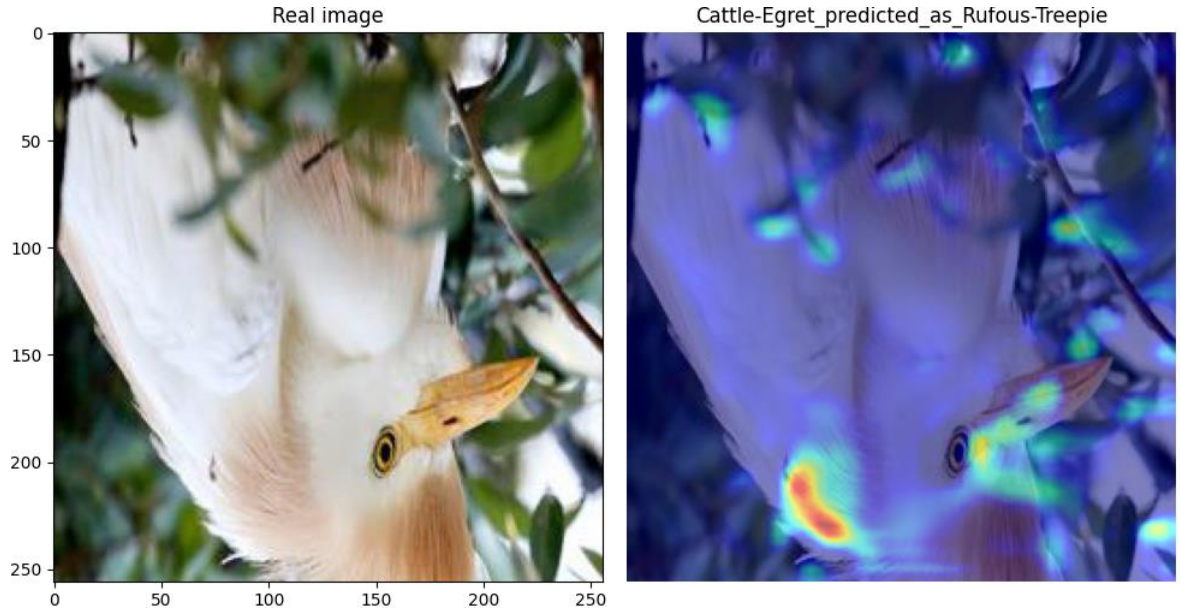


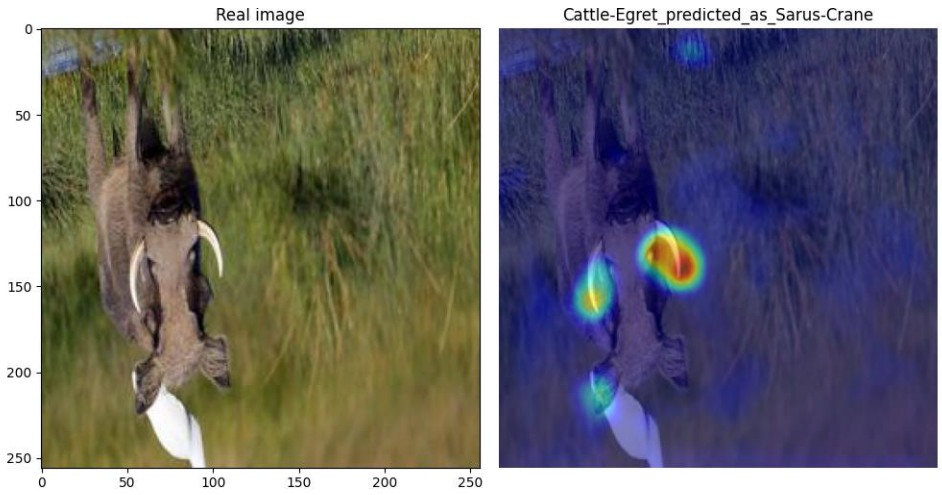
Incorrectly labelled samples


Real image
Cattle-Egret_predicted_as_Common-Rosefinch


Real image
Cattle-Egret_predicted_as_Indian-Grey-Hornbill

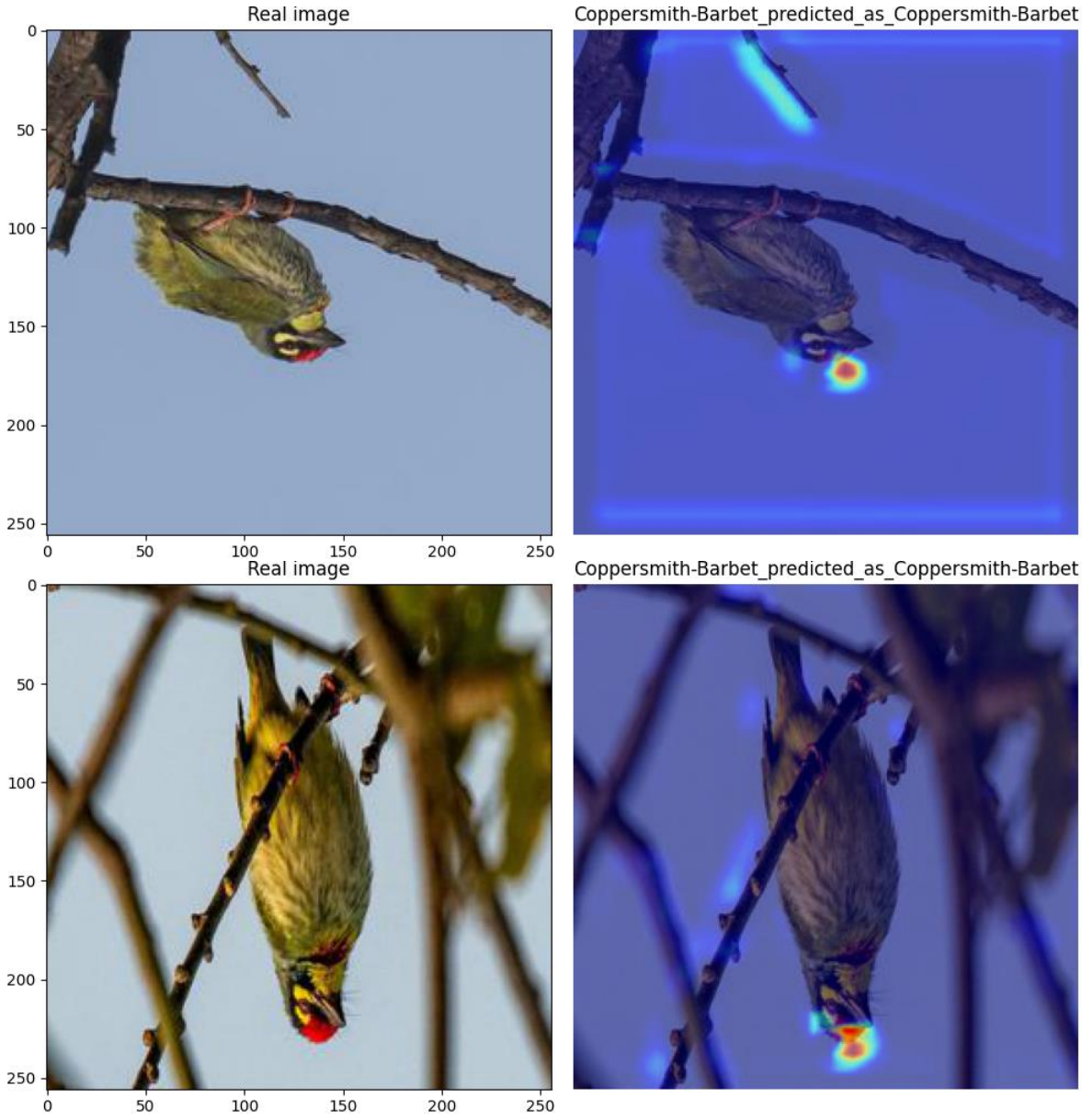

Real image
Cattle-Egret_predicted_as_Indian-Pitta

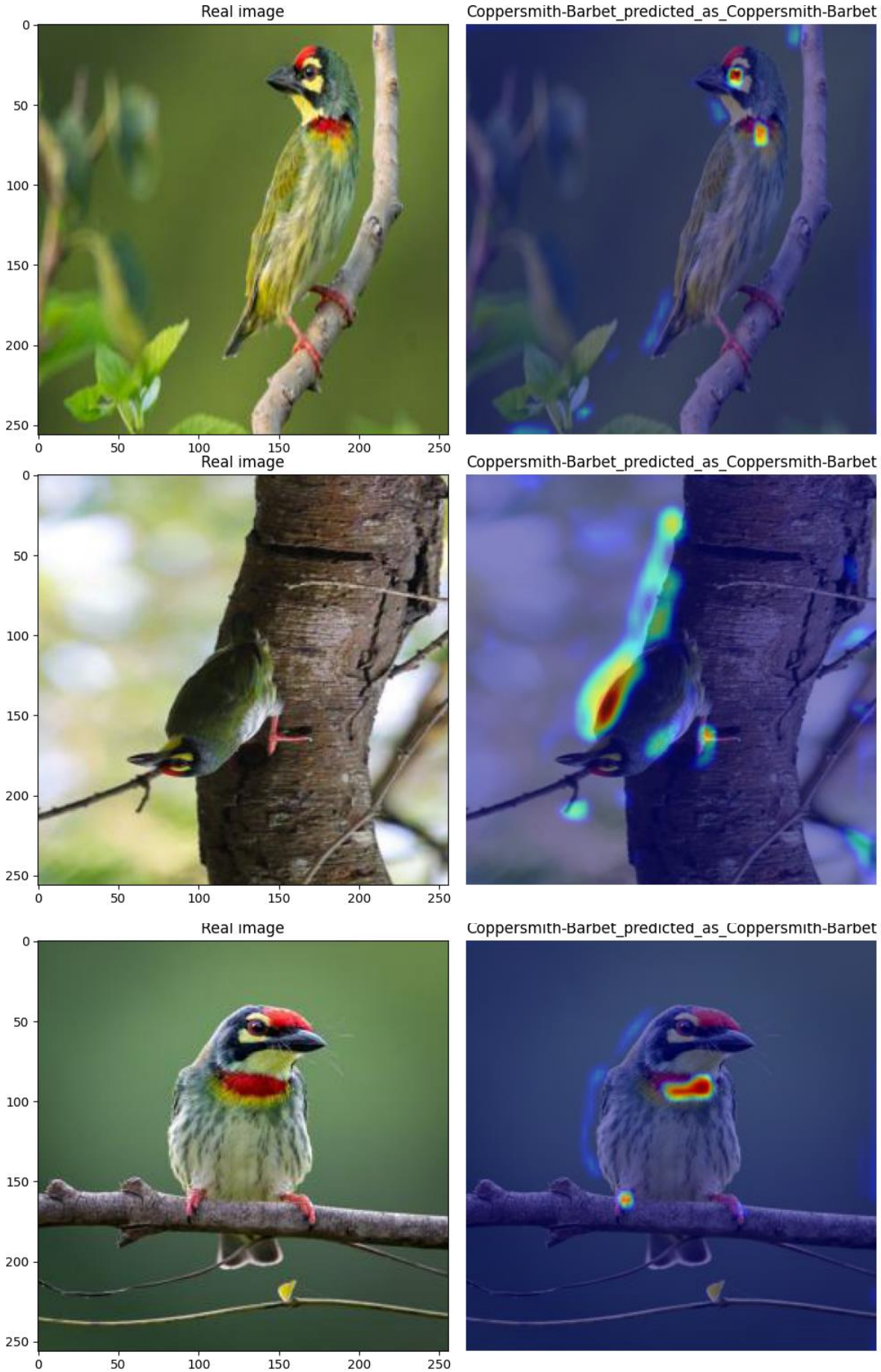




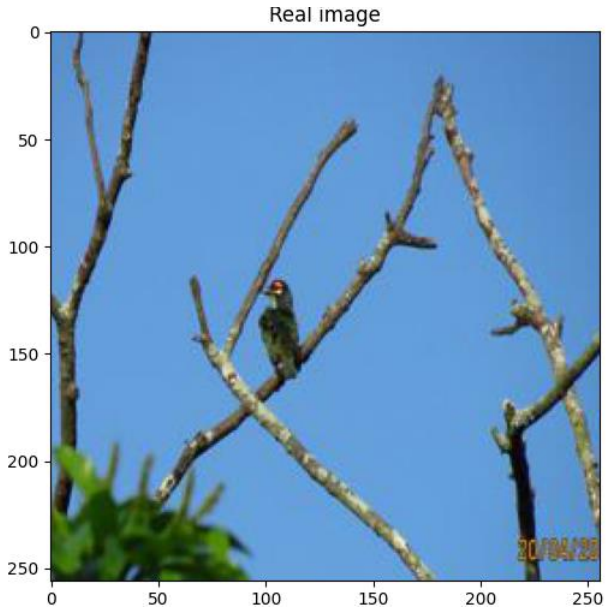
Coppersmith-Barbet

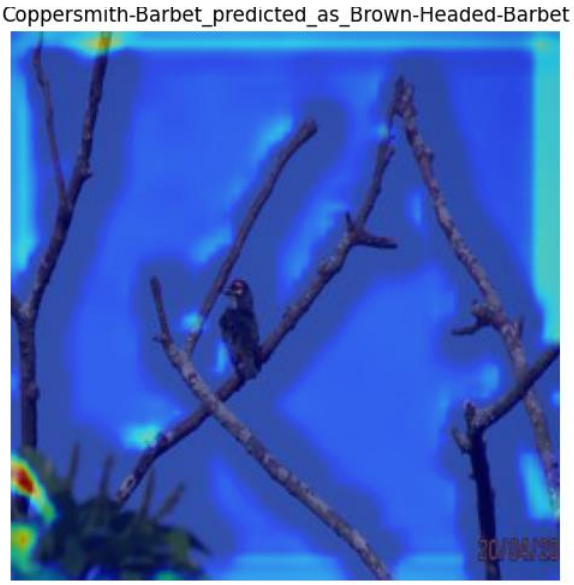
Correctly labelled samples

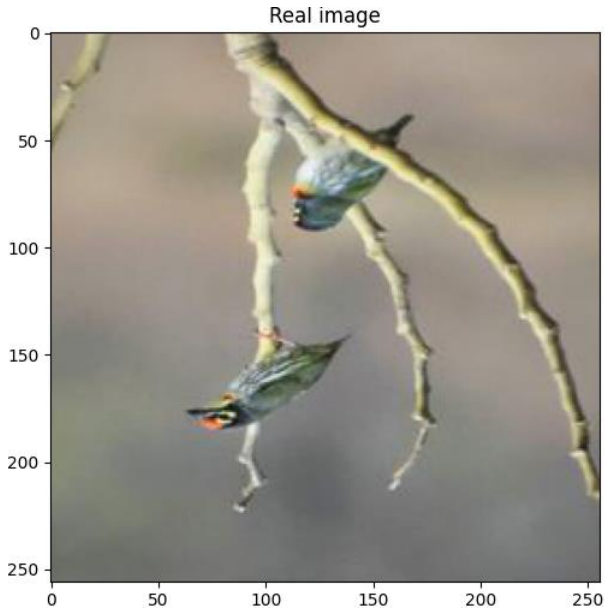

Real image
Coppersmith-Barbet_predicted_as_Coppersmith-Barbet
Real image
Coppersmith-Barbet_predicted_as_Coppersmith-Barbet
0
50
100
150
200
250

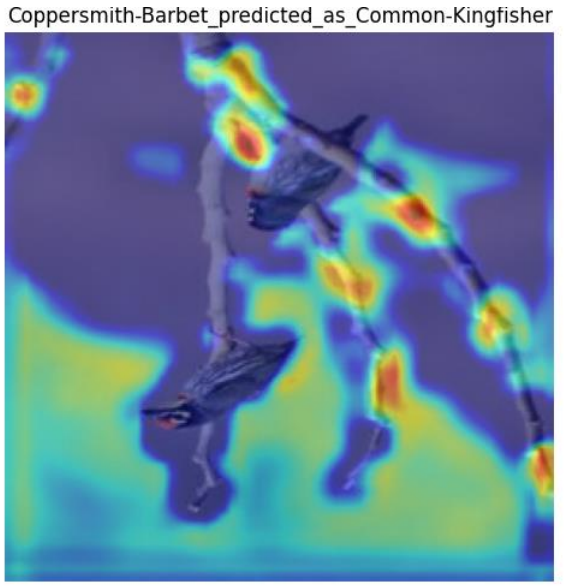

Real image
Coppersmith-Barbet_predicted_as_Coppersmith-Barbet
Real image
Coppersmith-Barbet_predicted_as_Coppersmith-Barbet
Real image
Coppersmith-Barbet_predicted_as_Coppersmith-Barbet
0
50
100
150
200
250

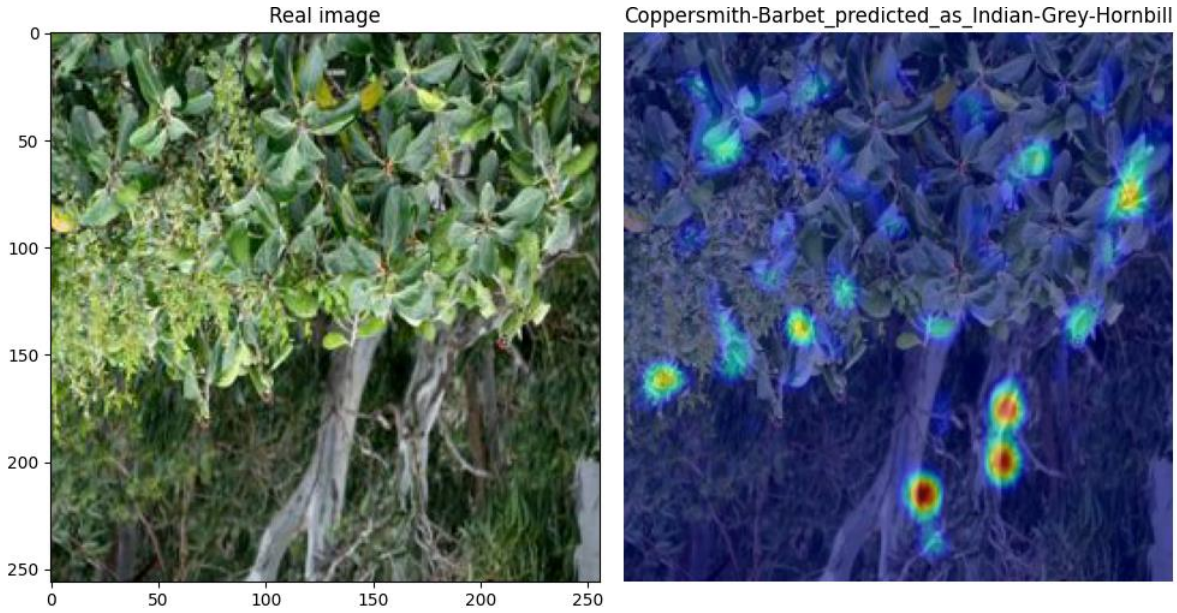
Incorrectly labelled samples

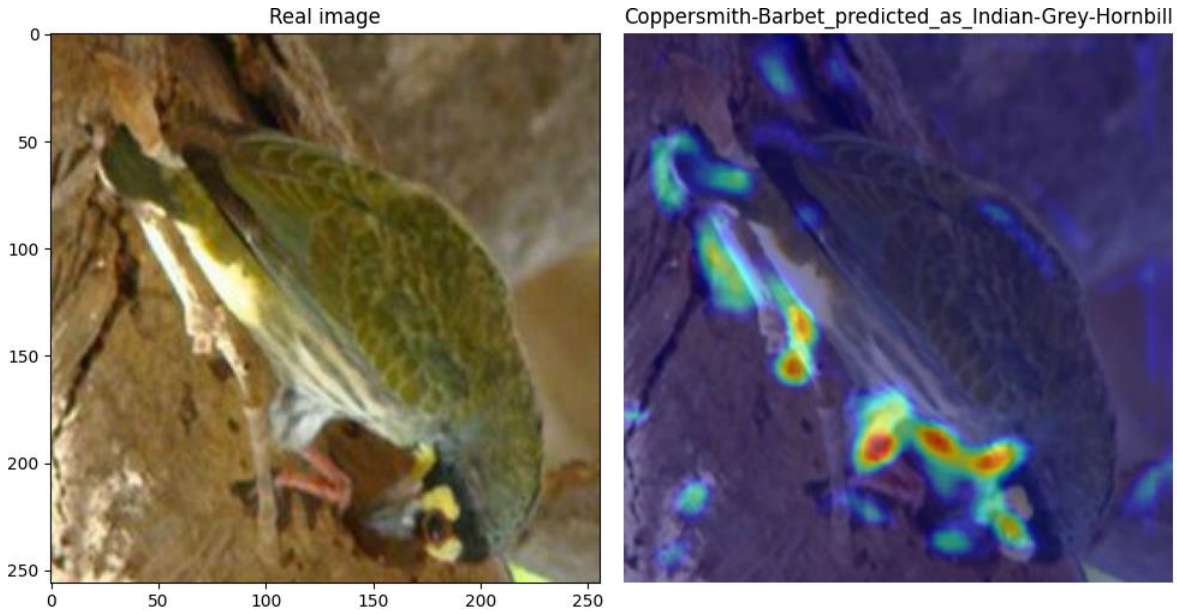


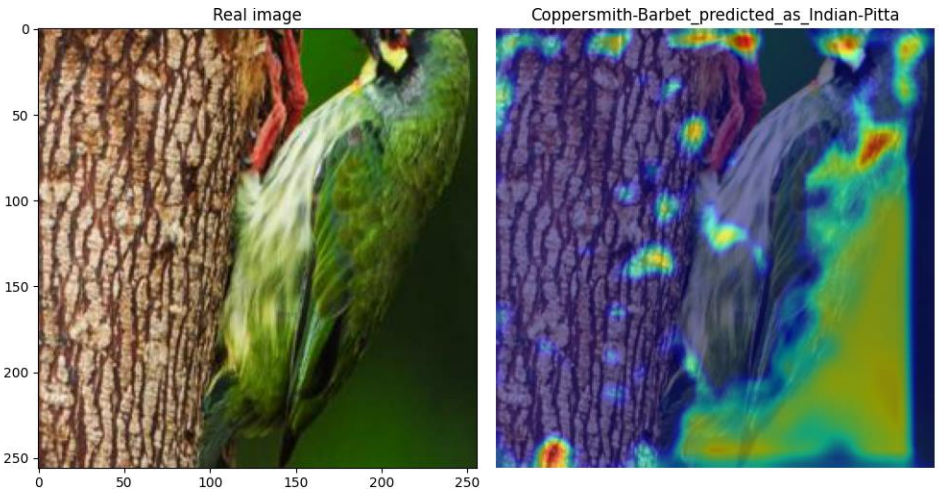






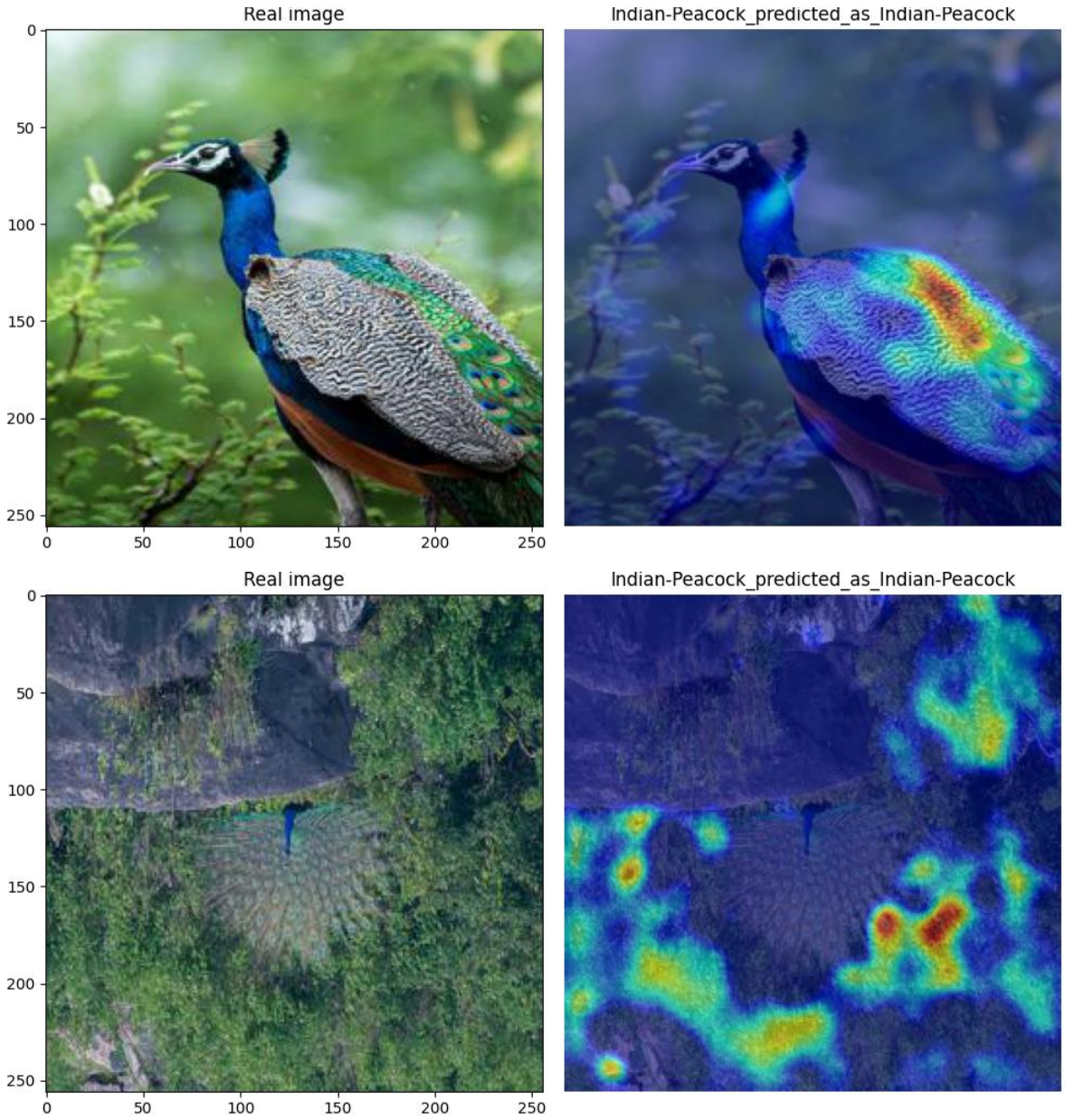


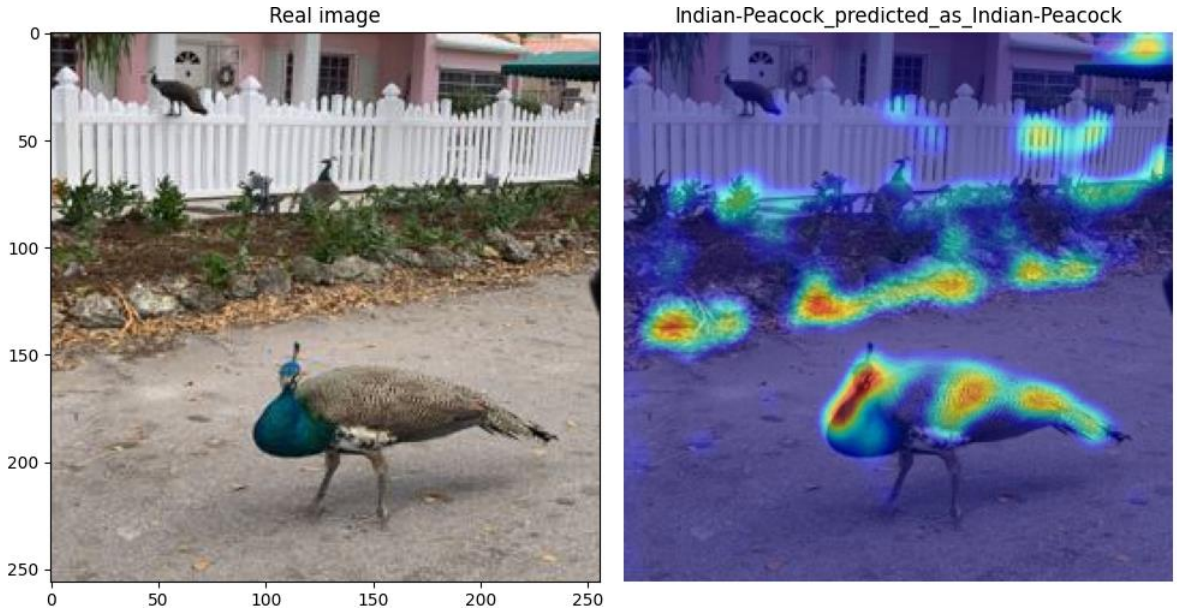


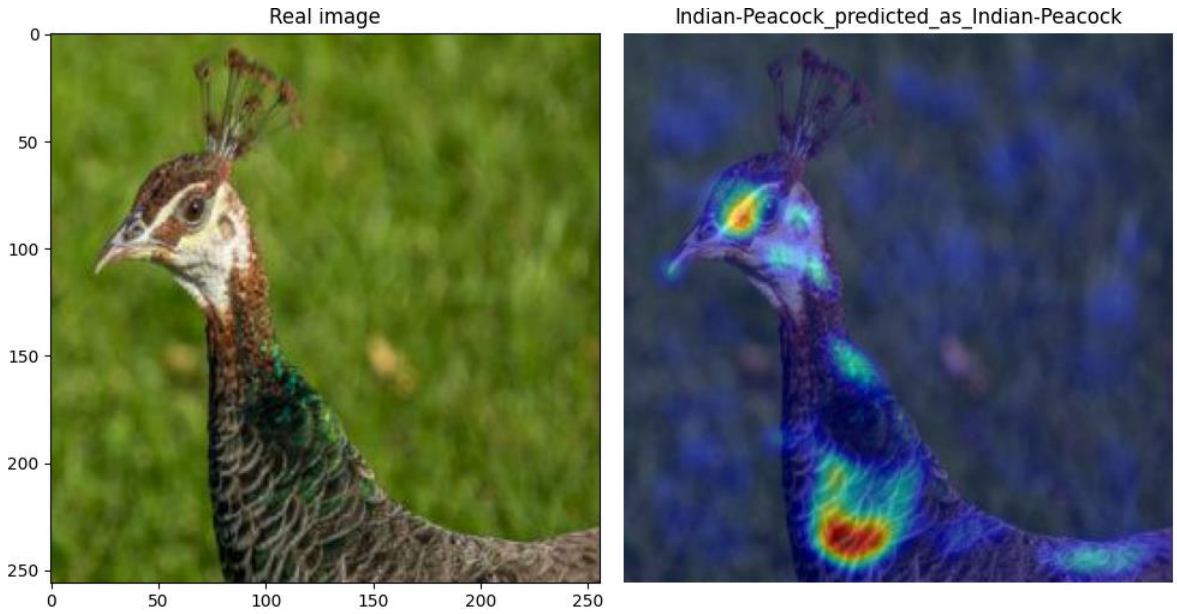


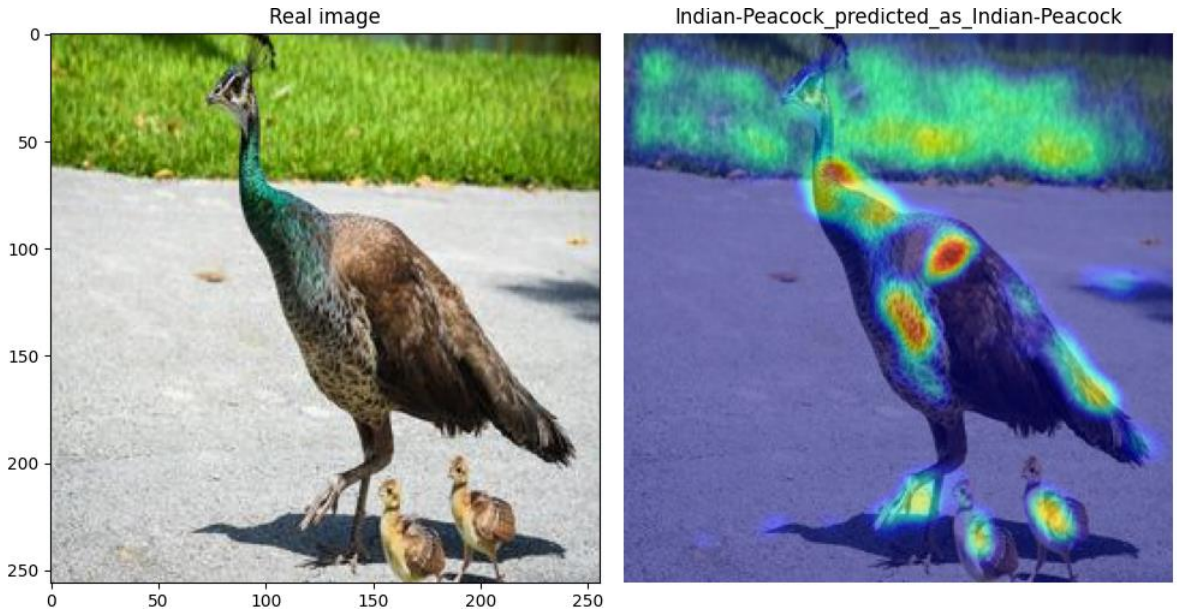
Indian Peacock

Correctly labelled samples

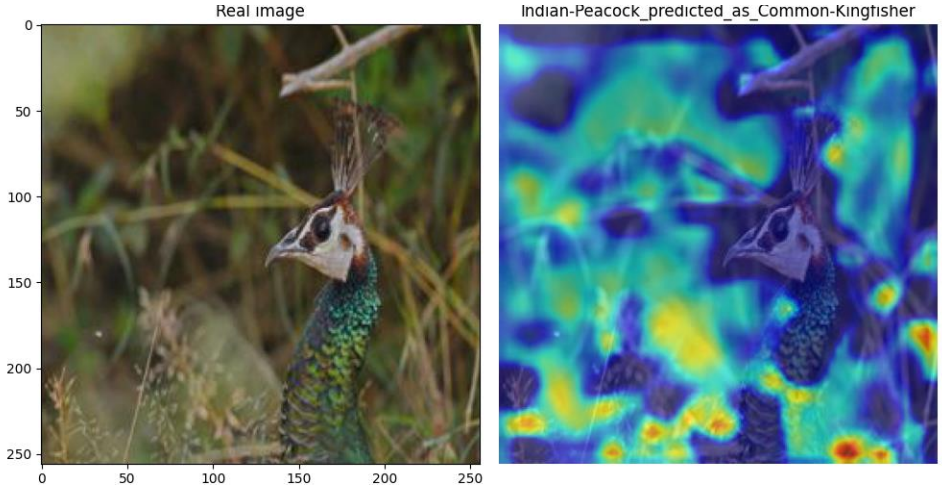


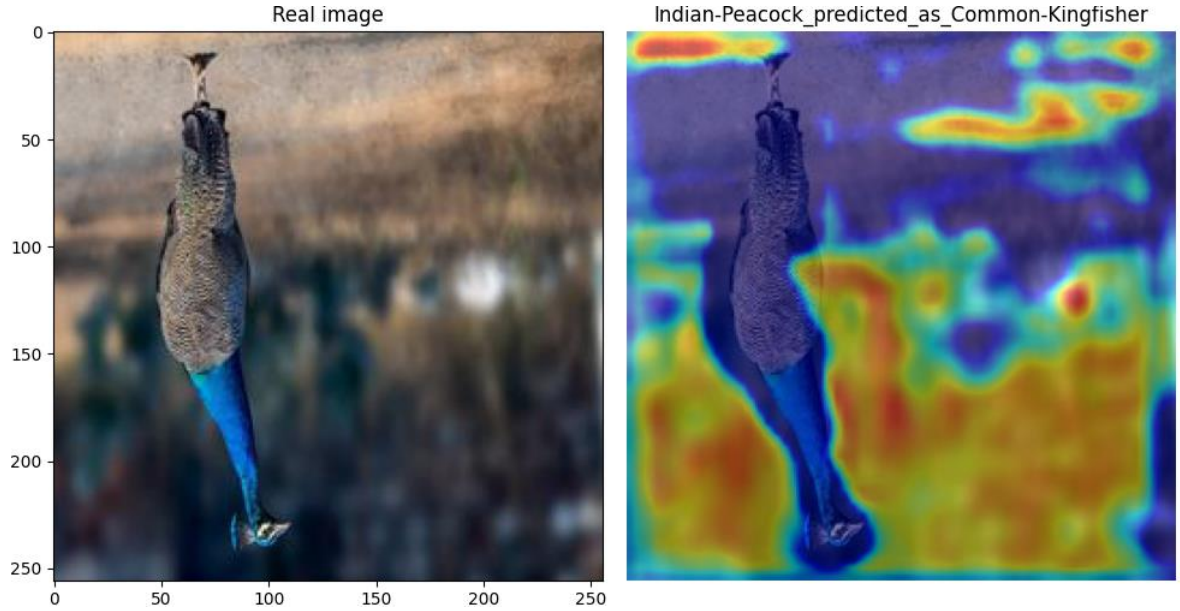

Real image
Indian-Peacock_predicted_as_Indian-Peacock
0
50
100
150
200
250

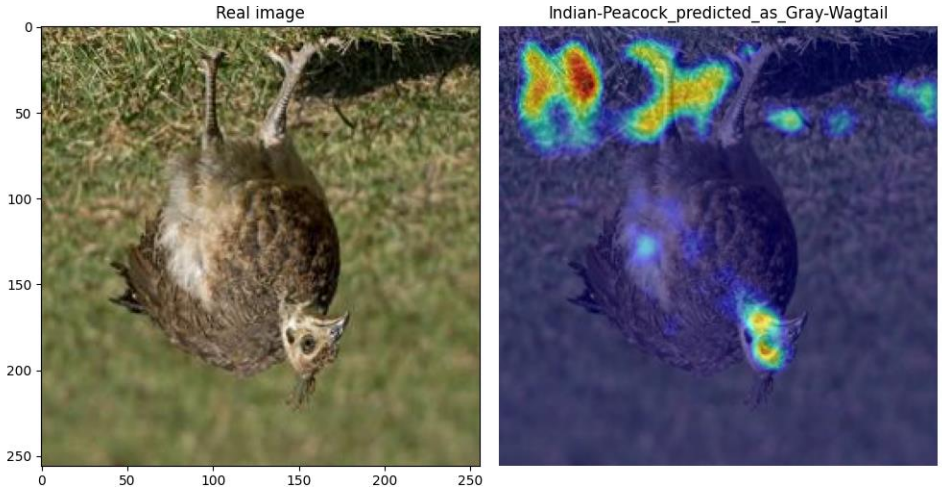

Real image
Indian-Peacock_predicted_as_Indian-Peacock
0
50
100
150
200
250

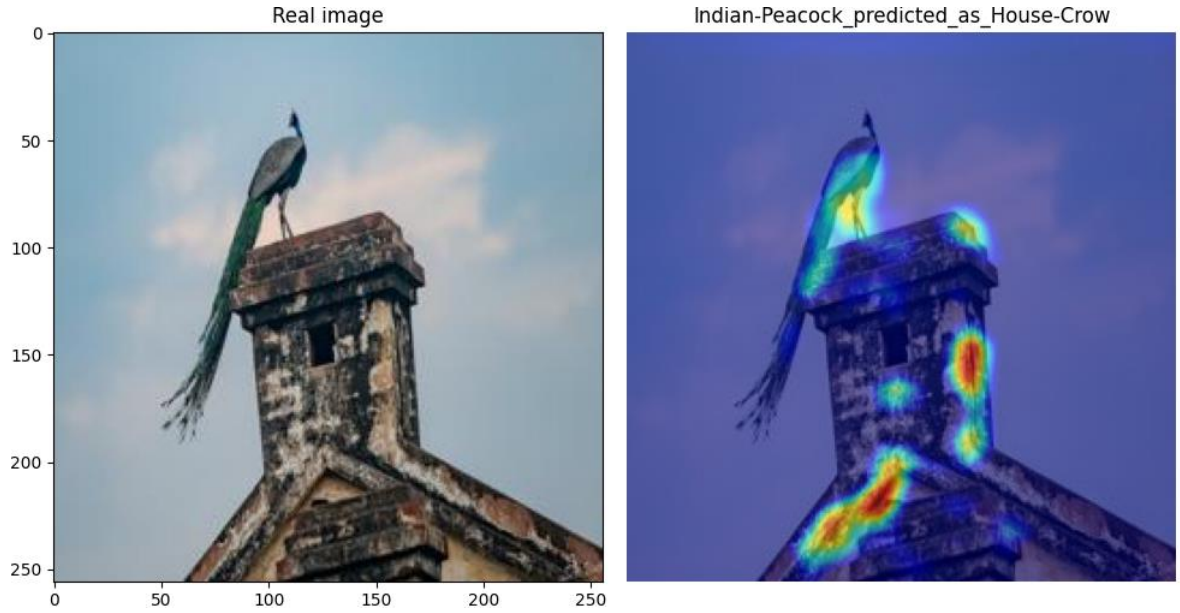

Real image
Indian-Peacock_predicted_as_Indian-Peacock
0
50
100
150
200
250

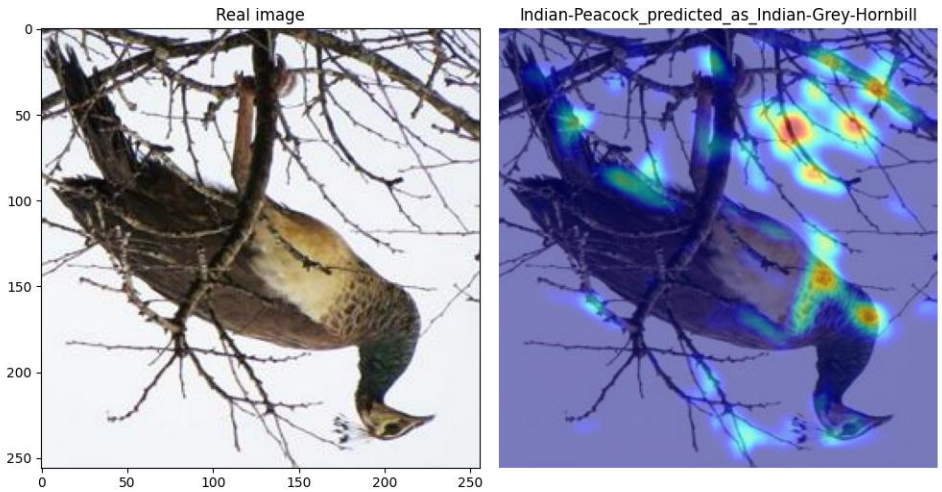
Incorrectly labelled samples





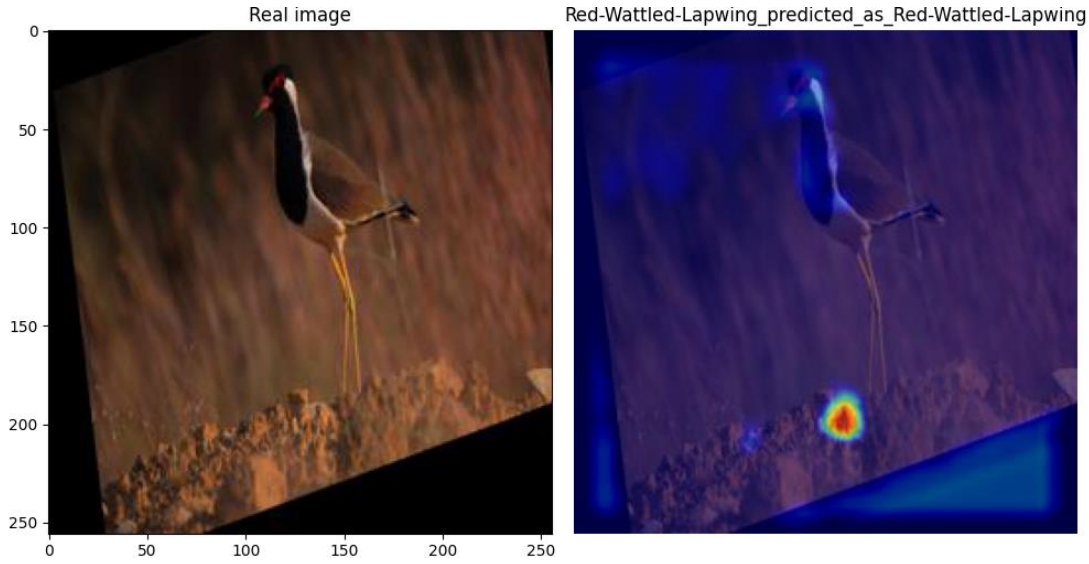


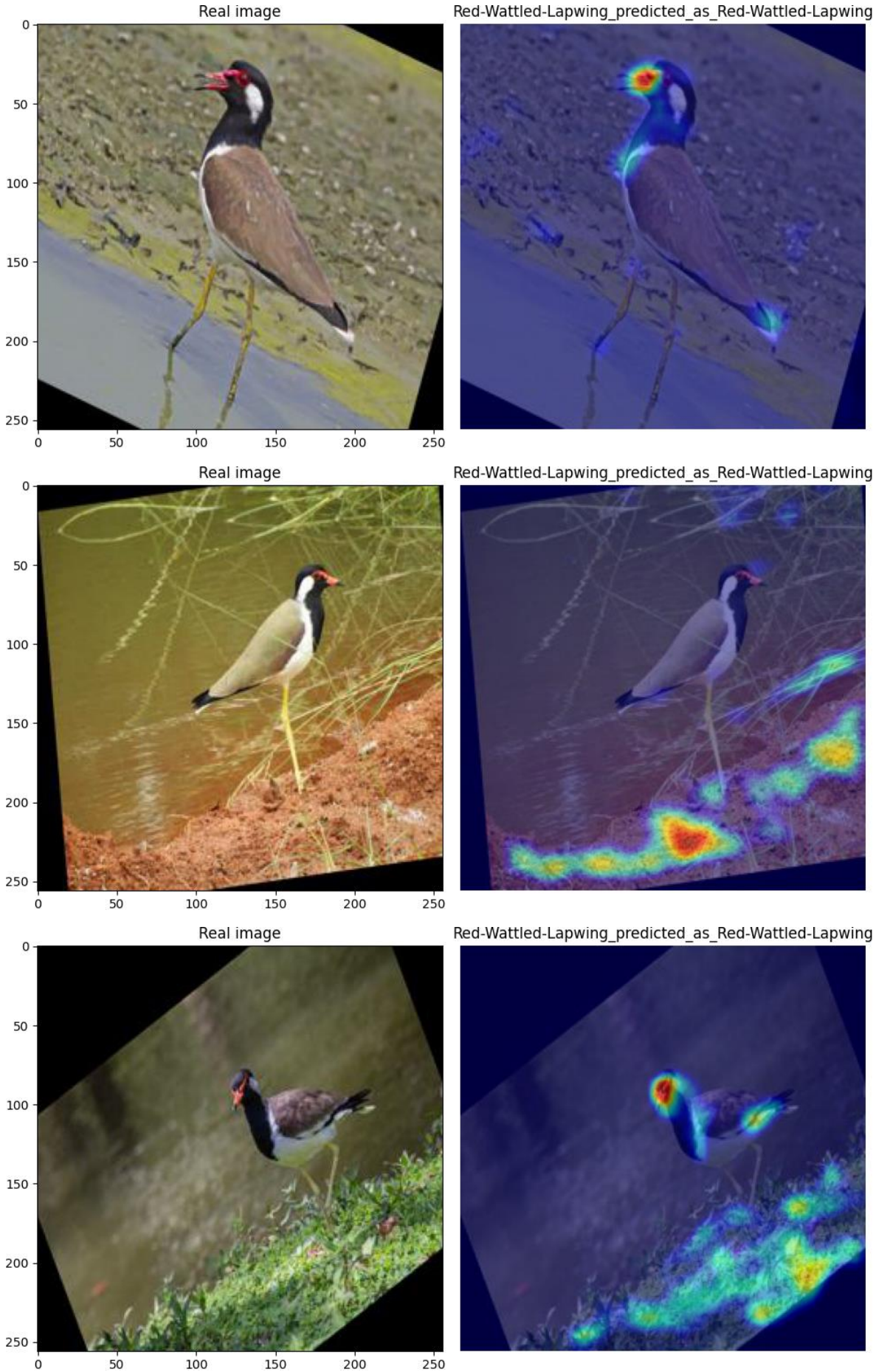


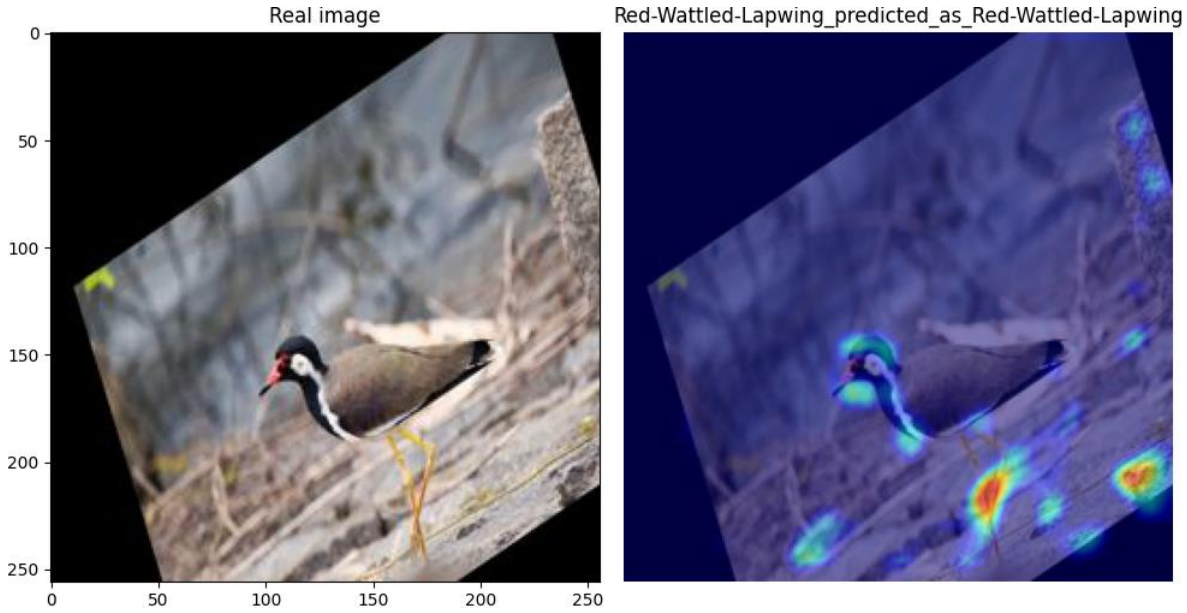


Red Wattled Lapwing

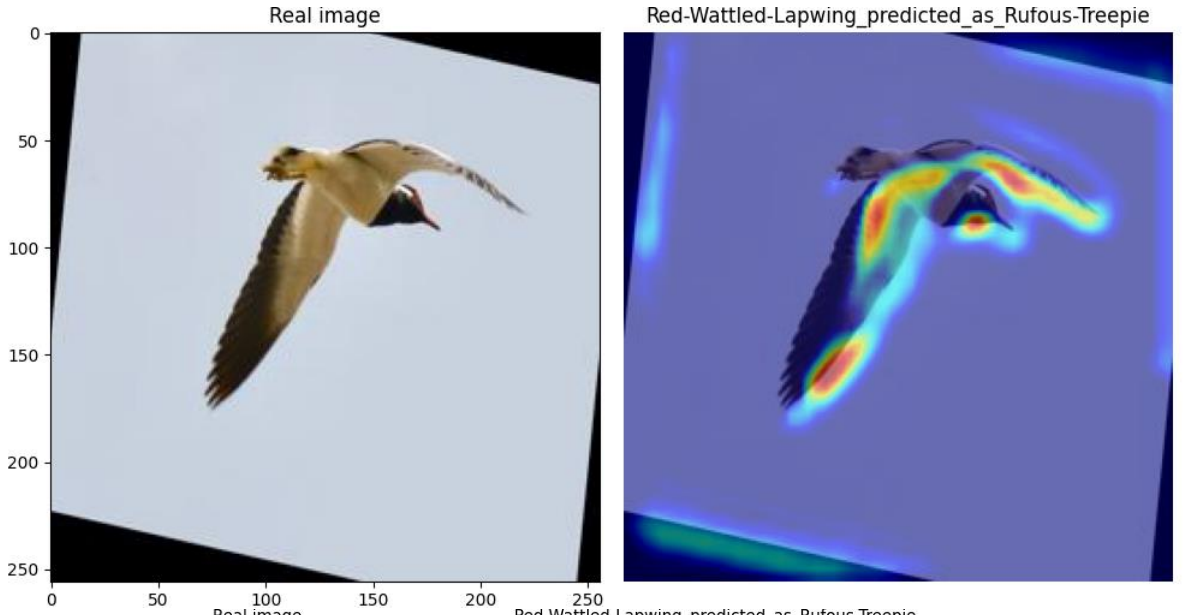
Correctly labelled samples

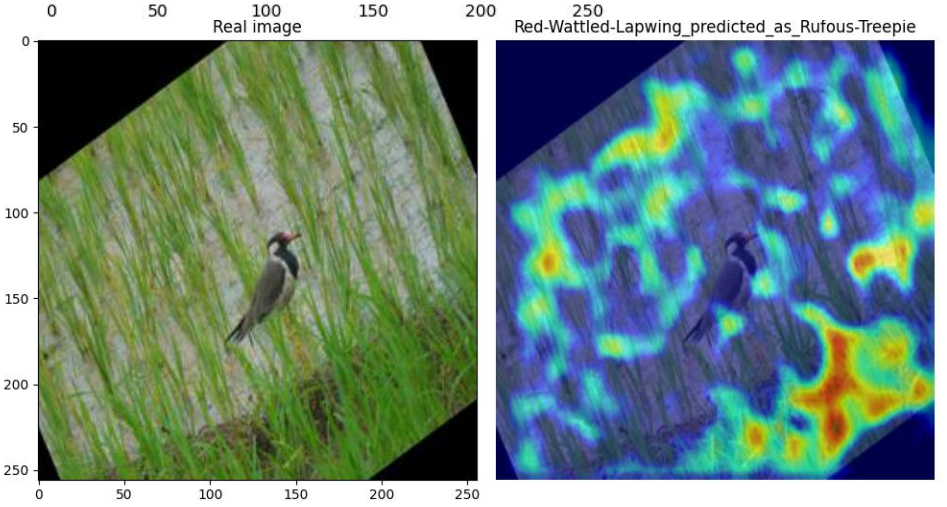


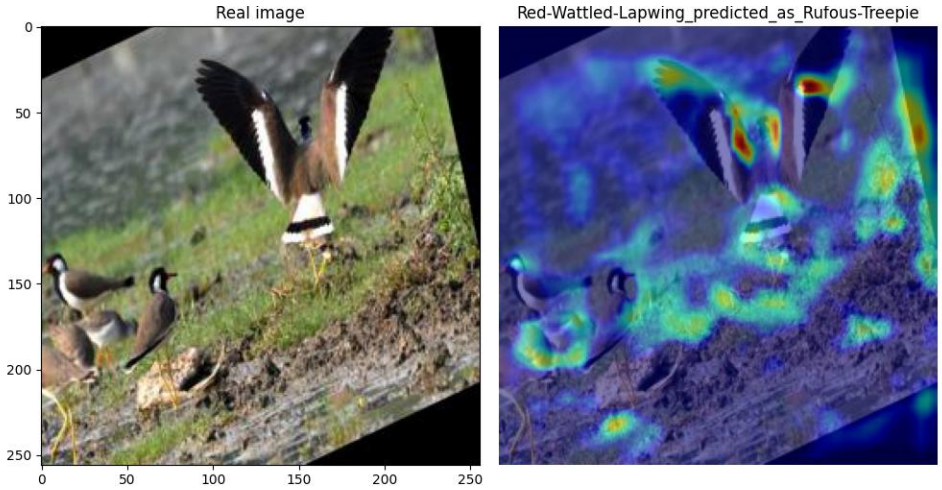

Real image
Red-Wattled-Lapwing_predicted_as_Red-Wattled-Lapwing
Real image
Red-Wattled-Lapwing_predicted_as_Red-Wattled-Lapwing
Real image
Red-Wattled-Lapwing_predicted_as_Red-Wattled-Lapwing
0
50
100
150
200
250

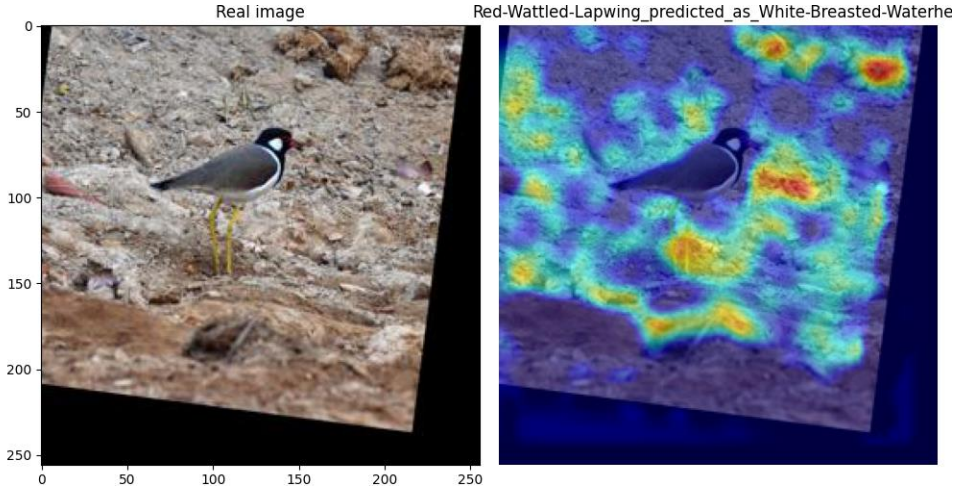


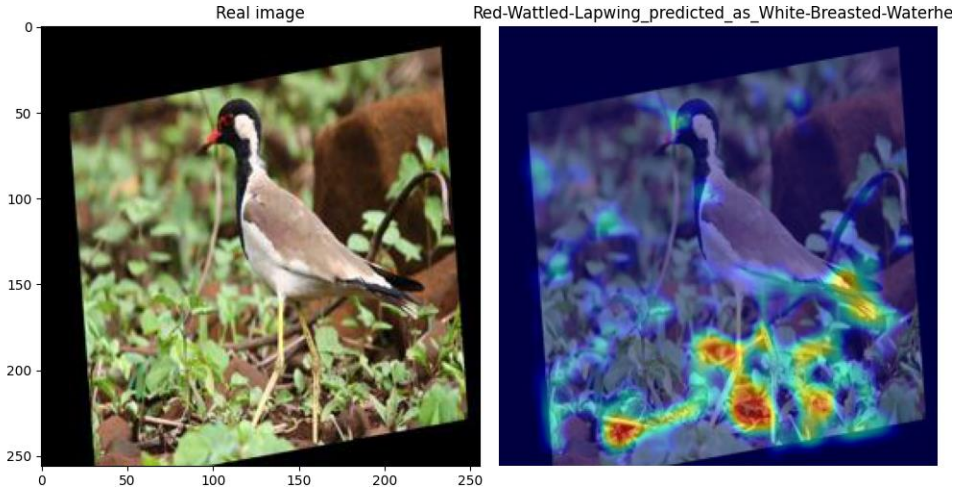
Incorrectly labelled samples





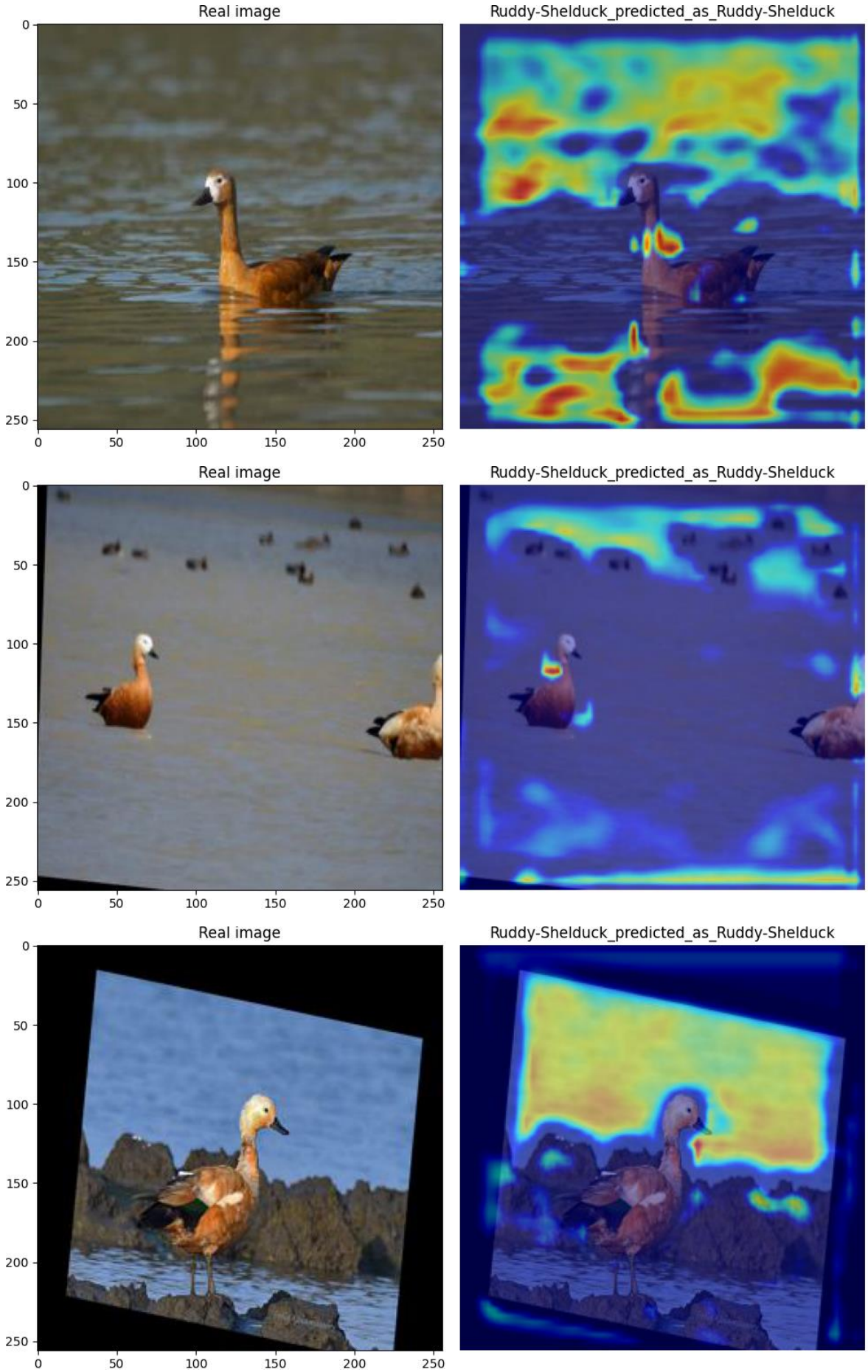


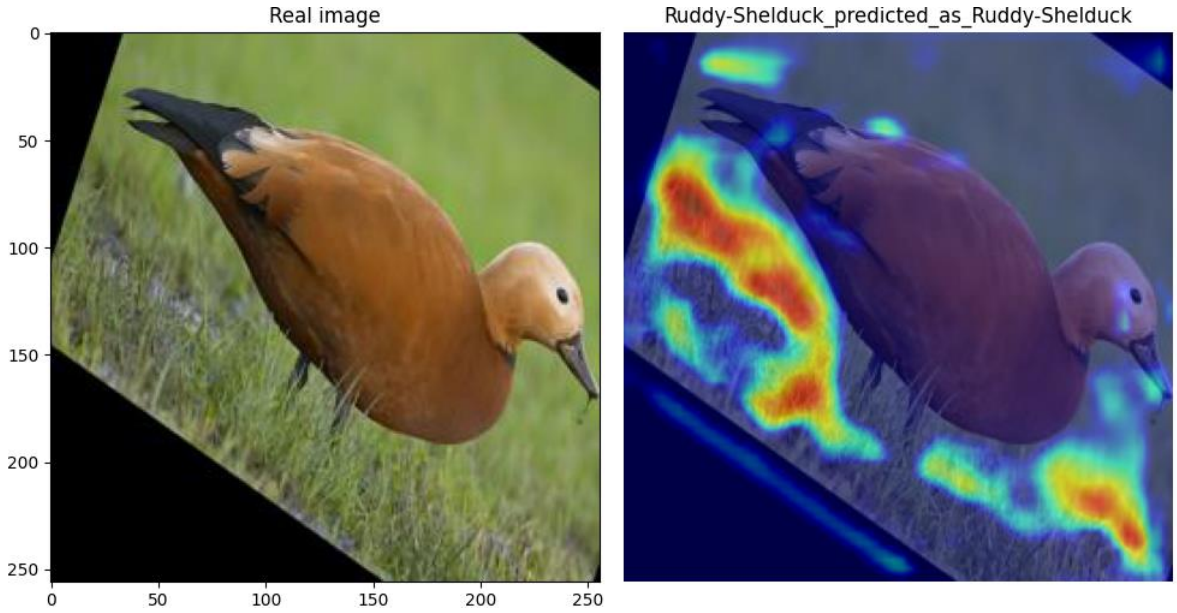


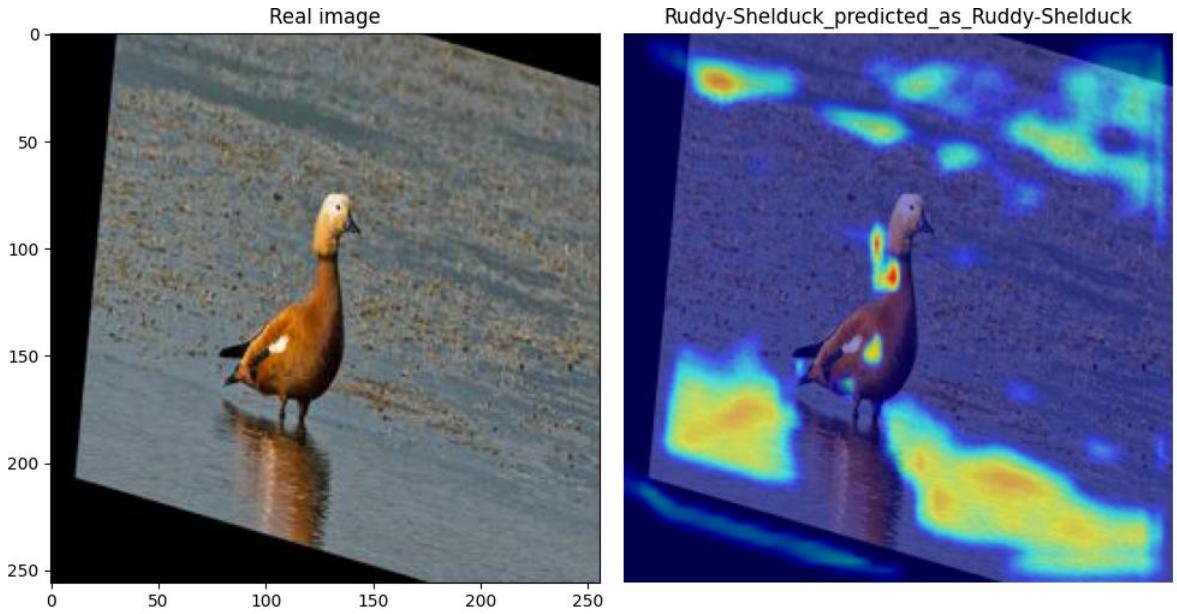


Ruddy Shelduck

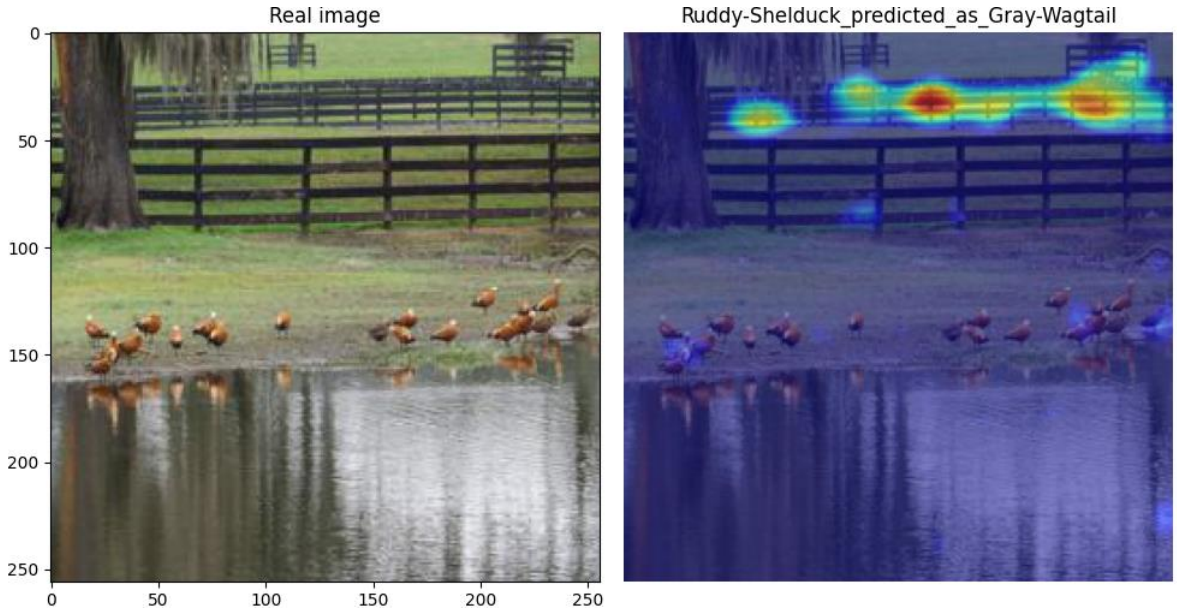
Correctly labelled samples

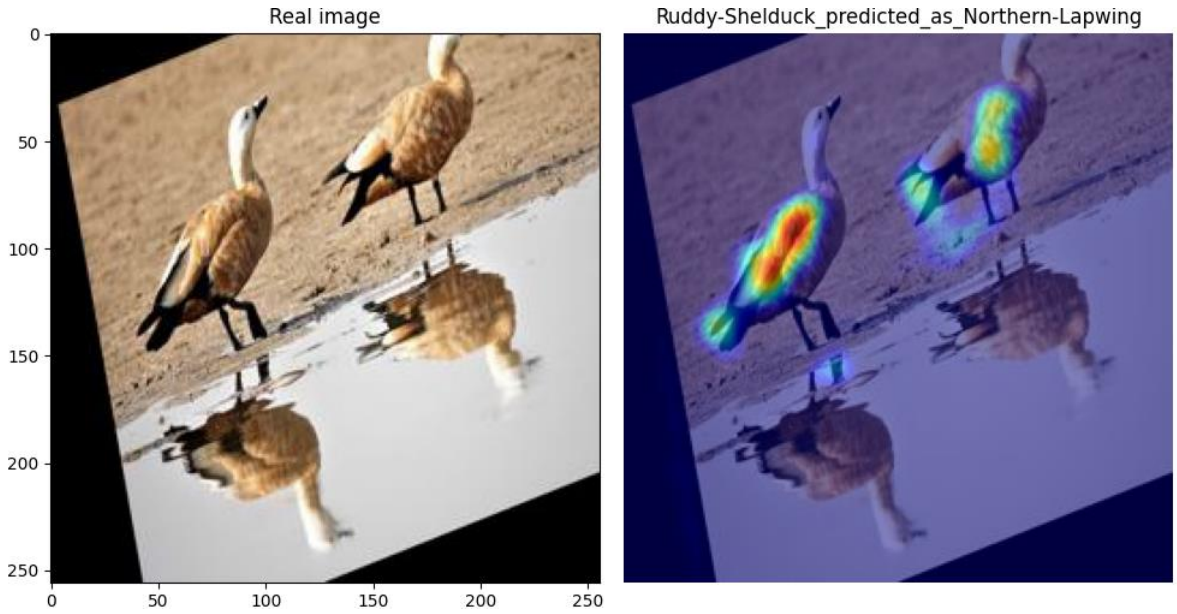

Real image
Ruddy-Shelduck_predicted_as_Ruddy-Shelduck
Real image
Ruddy-Shelduck_predicted_as_Ruddy-Shelduck
Real image
Ruddy-Shelduck_predicted_as_Ruddy-Shelduck
0
50
100
150
200
250



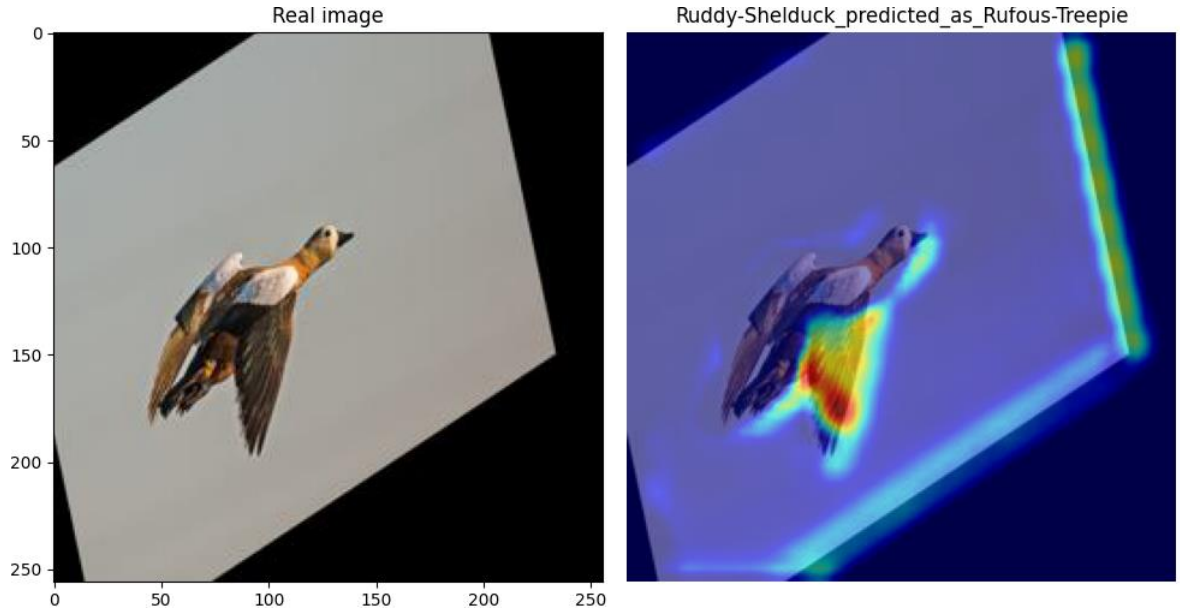


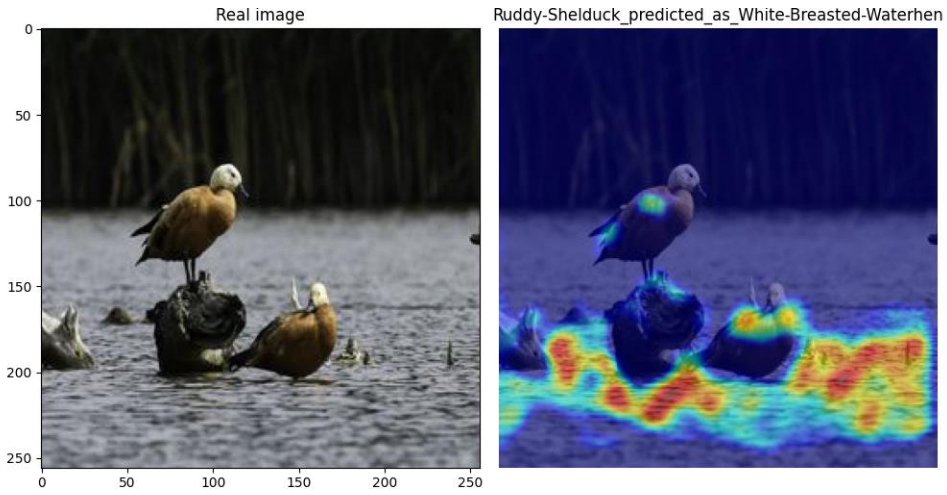
Incorrectly labelled samples


Real image
Ruddy-Shelduck_predicted_as_Gray-Wagtail
0
50
100
150
200
250


Real image
Ruddy-Shelduck_predicted_as_Northern-Lapwing
0
50
100
150
200
250

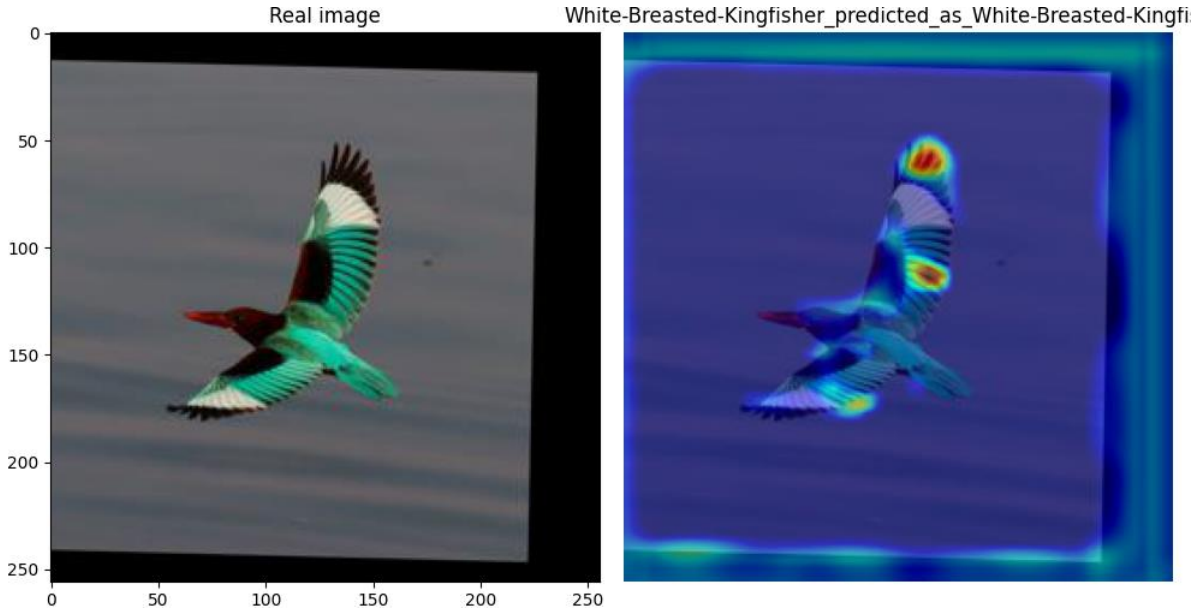

Real image
Ruddy-Shelduck_predicted_as_Northern-Lapwing
0
50
100
150
200
250

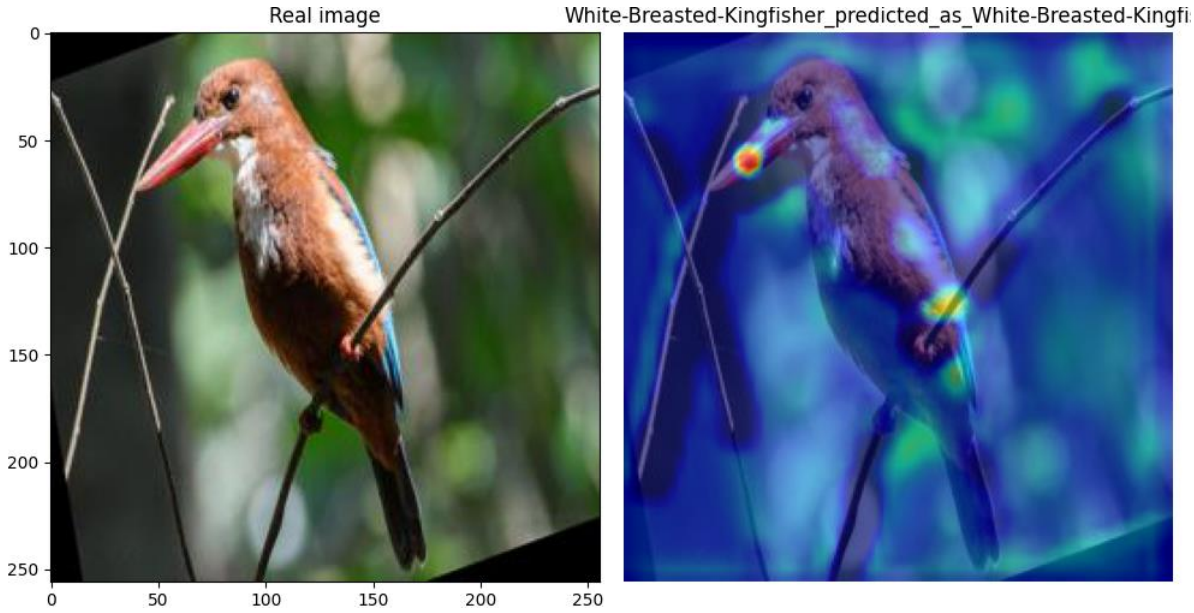


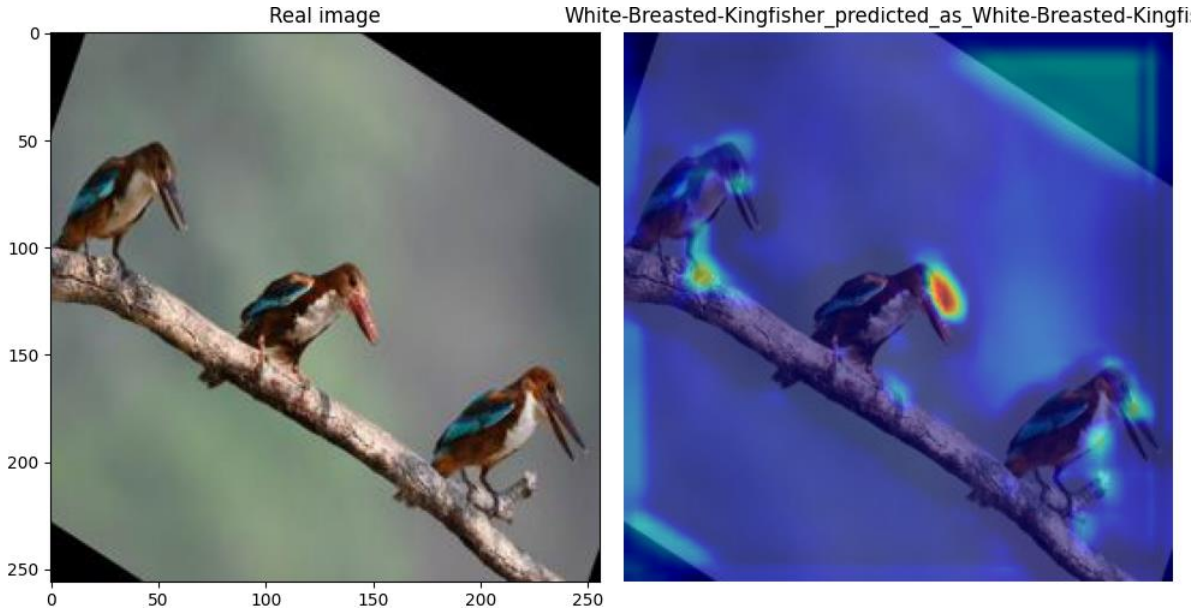


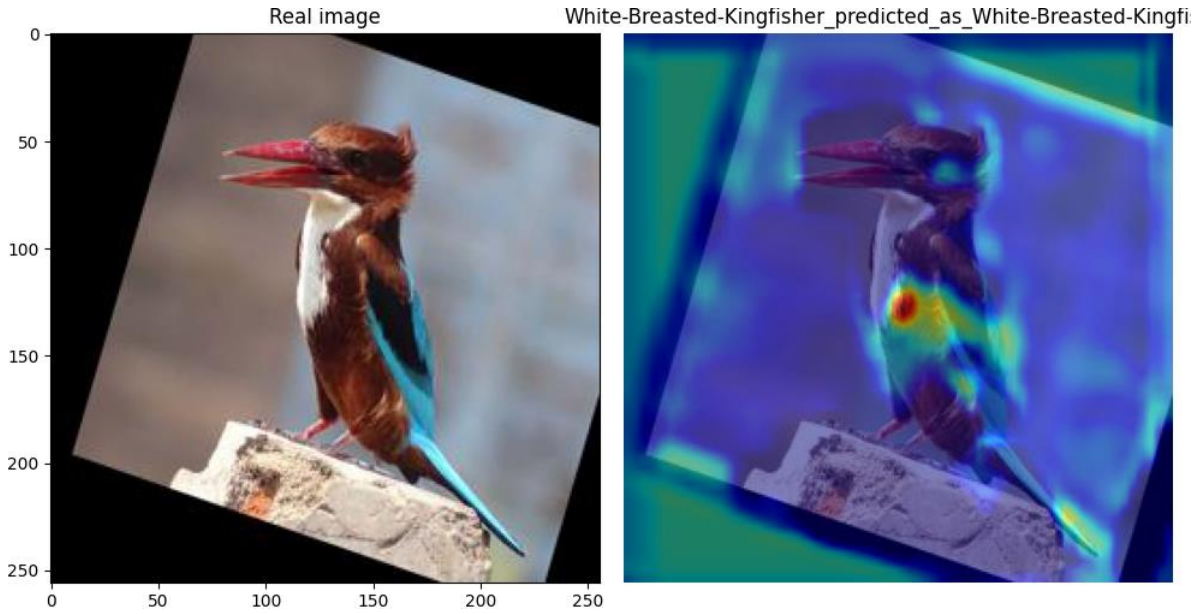
White Breasted kingfisher

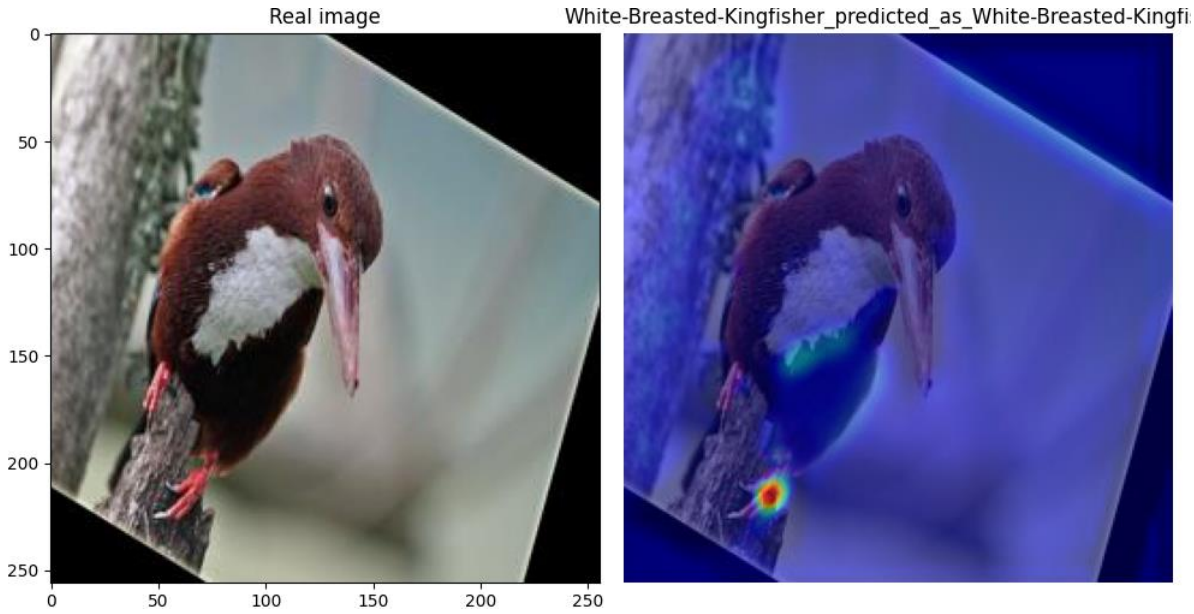
Correctly labelled samples


Real image
White-Breasted-Kingfisher_predicted_as_White-Breasted-Kingfi
0
50
100
150
200
250
0
50
100
150
200
250

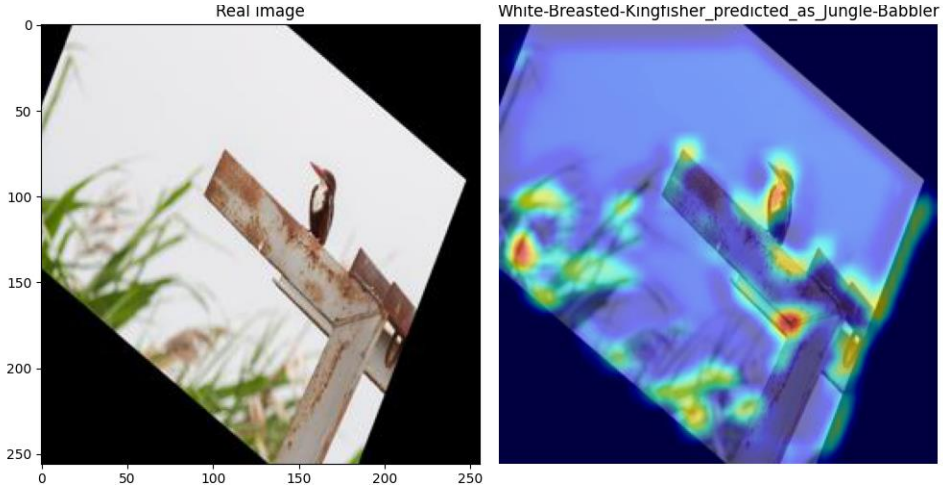

Real image
White-Breasted-Kingfisher_predicted_as_White-Breasted-Kingfi
0
50
100
150
200
250
0
50
100
150
200
250

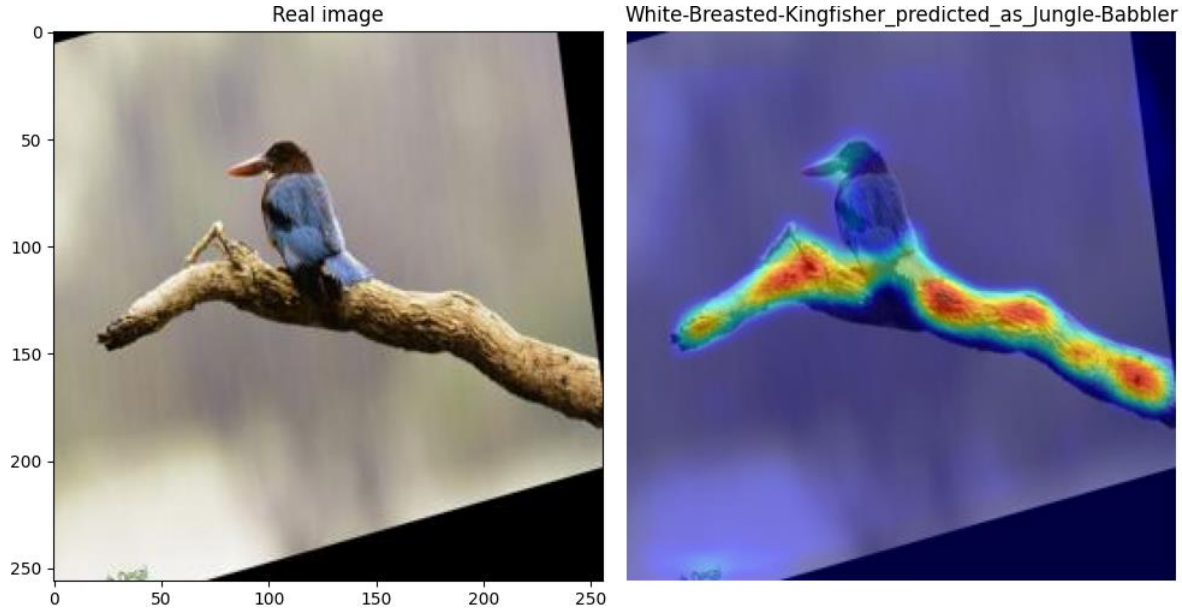

Real image
White-Breasted-Kingfisher_predicted_as_White-Breasted-Kingfi
0
50
100
150
200
250

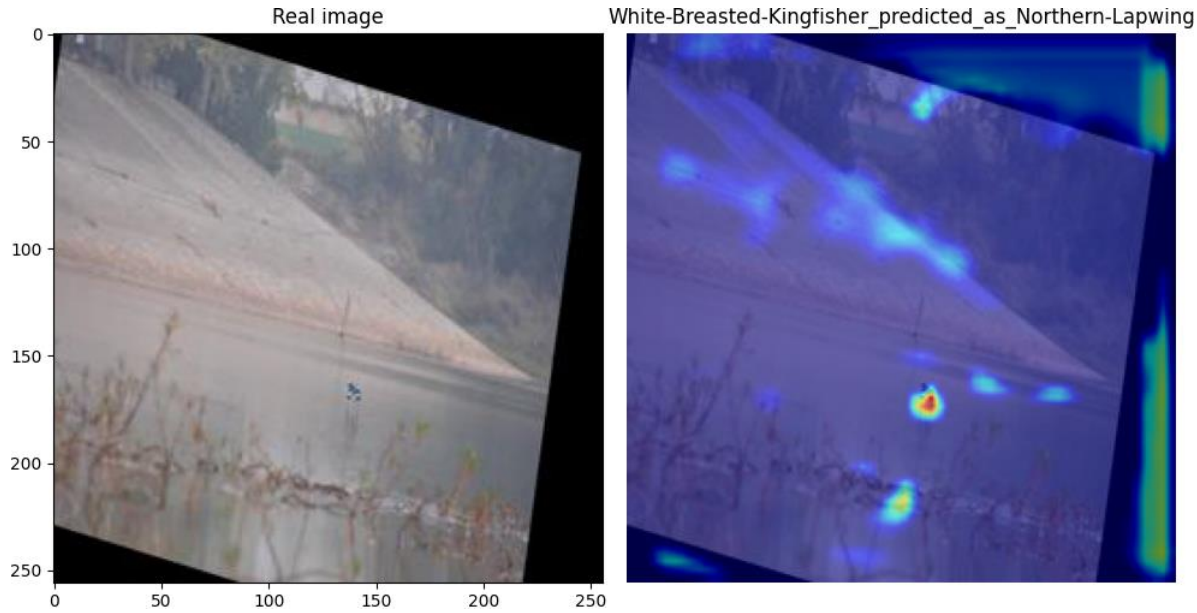

Real image
White-Breasted-Kingfisher_predicted_as_White-Breasted-Kingfi
0
50
100
150
200
250

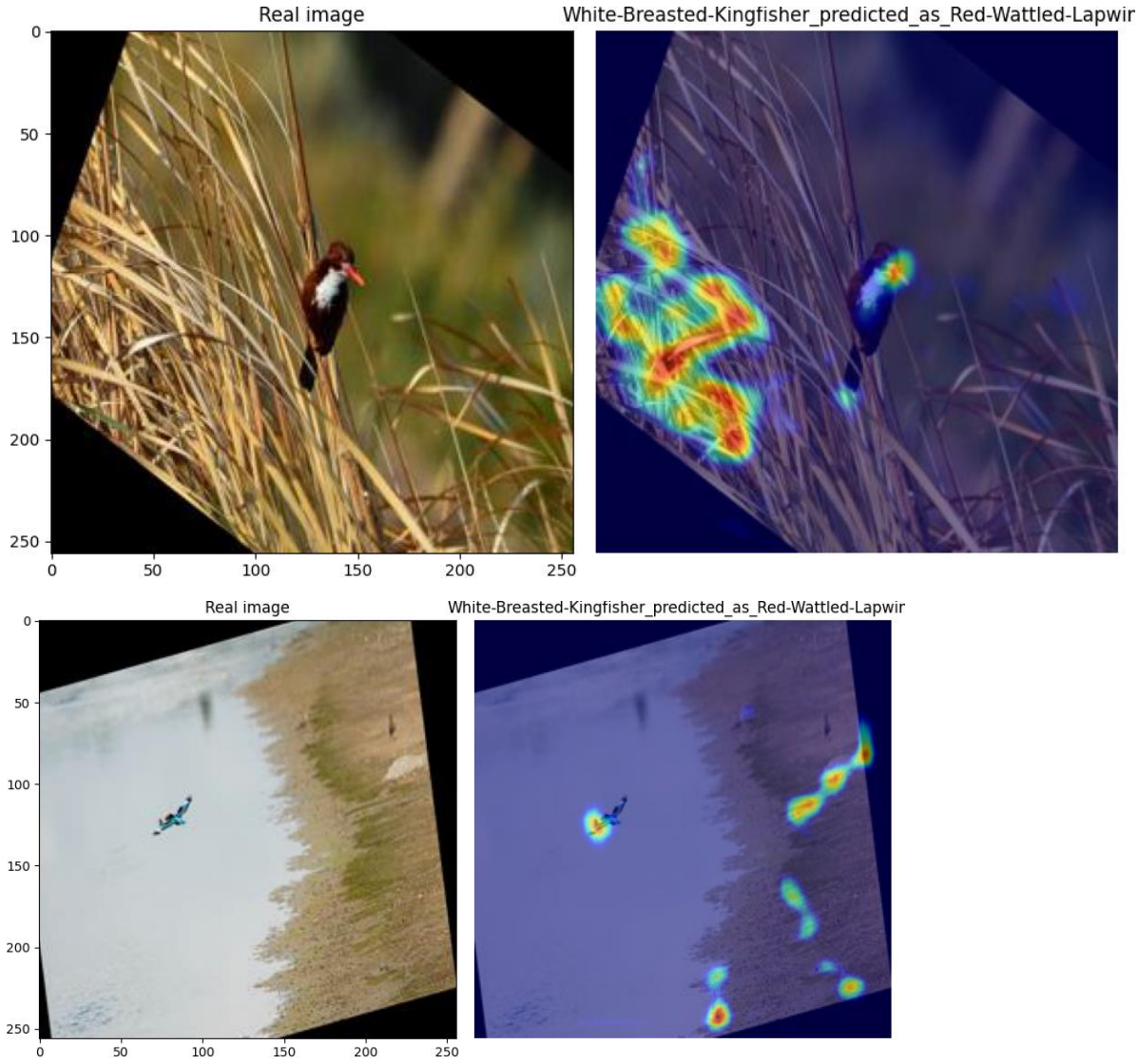

Real image
White-Breasted-Kingfisher_predicted_as_White-Breasted-Kingfi
0
50
100
150
200
250

Incorrectly labelled samples



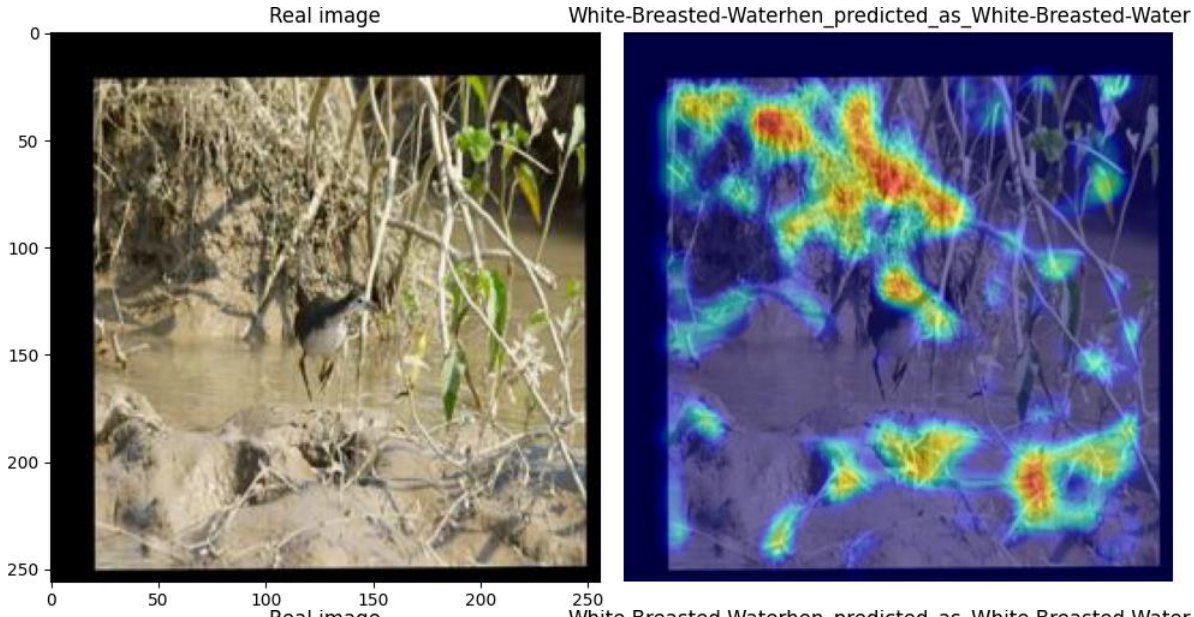


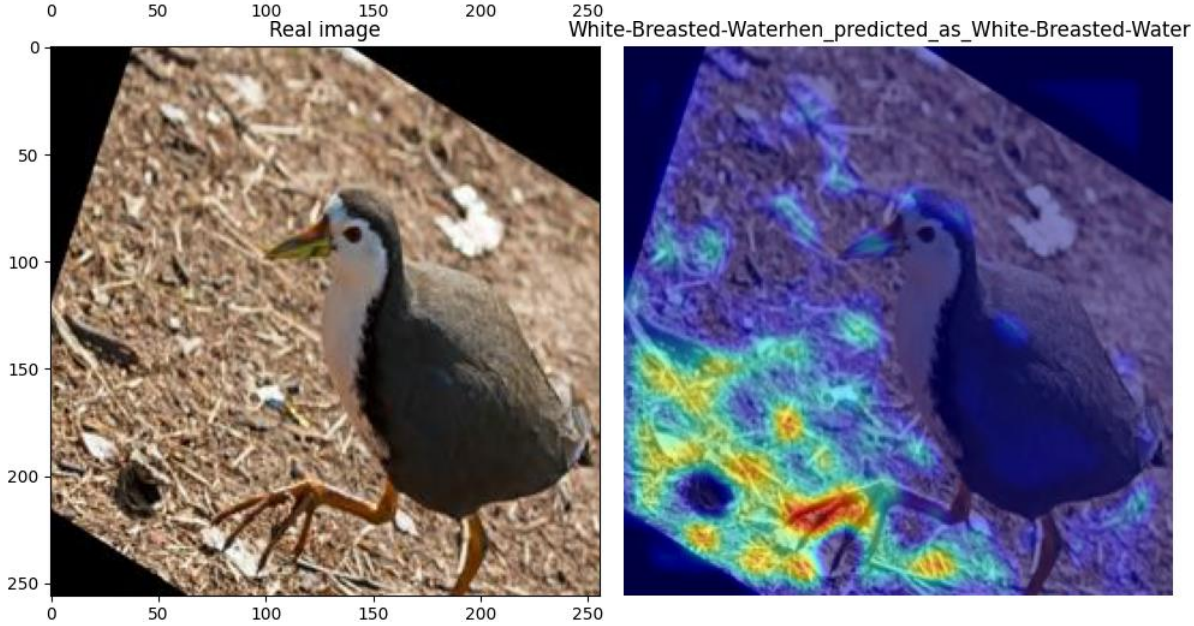


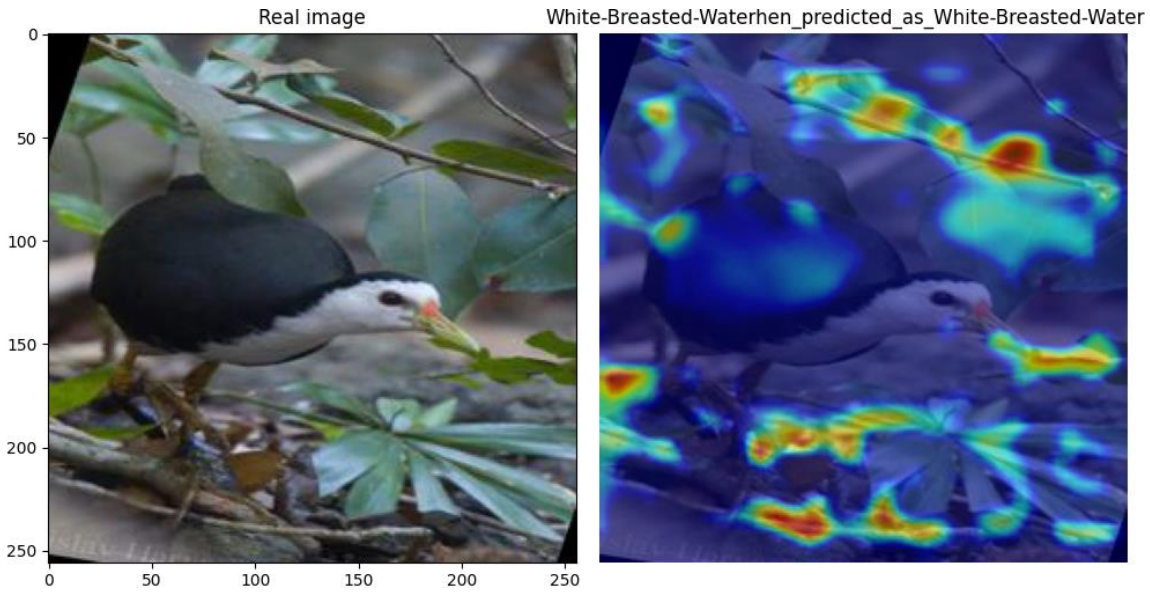


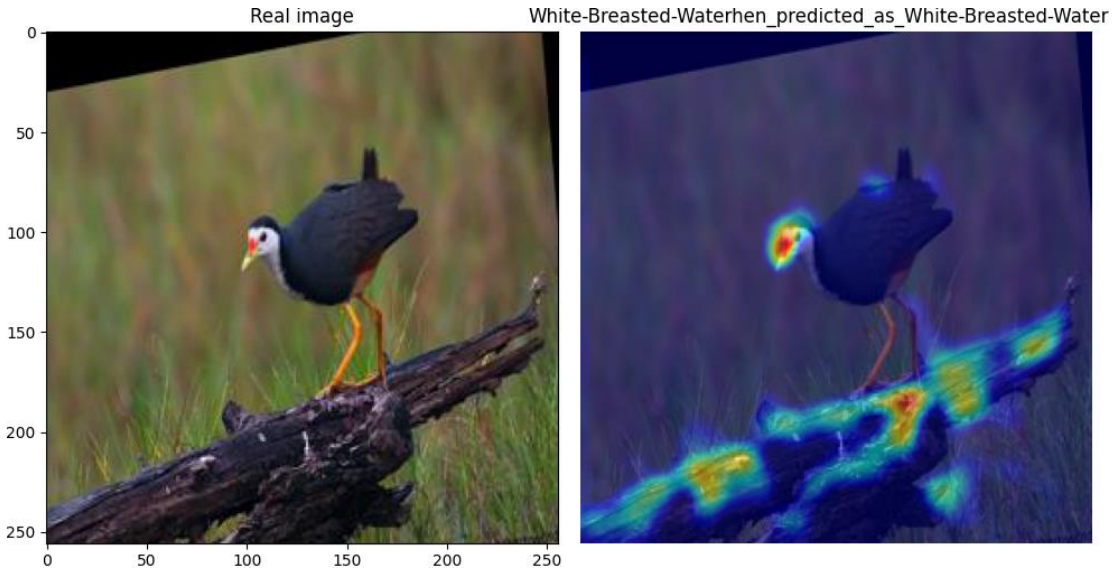
White breasted waterhen

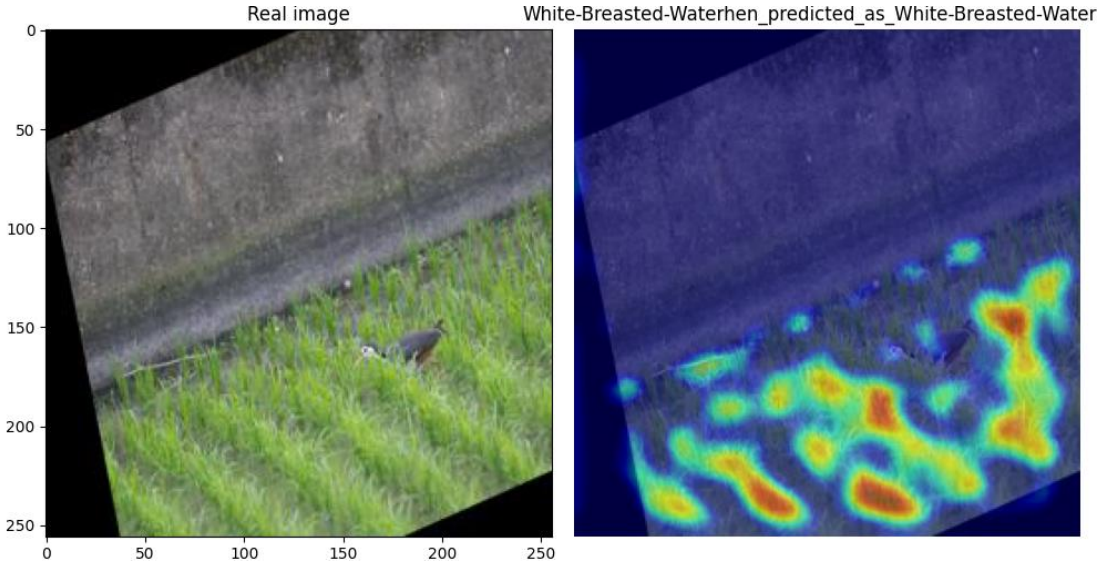
Correctly labelled samples


Real image
White-Breasted-Waterhen_predicted_as_White-Breasted-Water
0
50
100
150
200
250
0
50
100
150
200
250

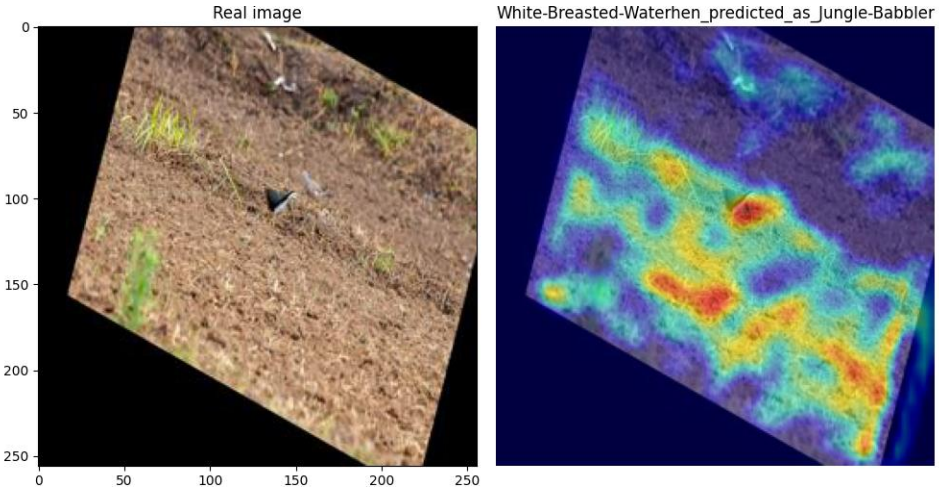

Real image
White-Breasted-Waterhen_predicted_as_White-Breasted-Water
0
50
100
150
200
250
0
50
100
150
200
250

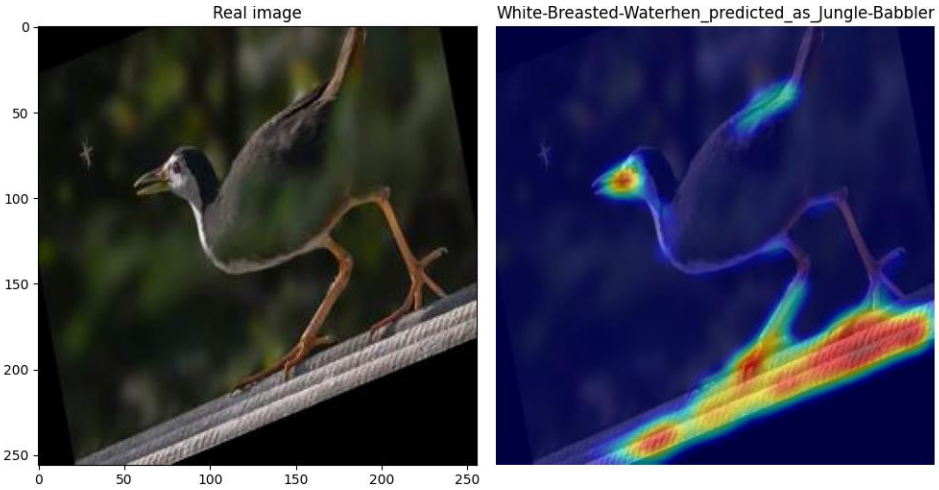

Real image
White-Breasted-Waterhen_predicted_as_White-Breasted-Water
0
50
100
150
200
250
0
50
100
150
200
250

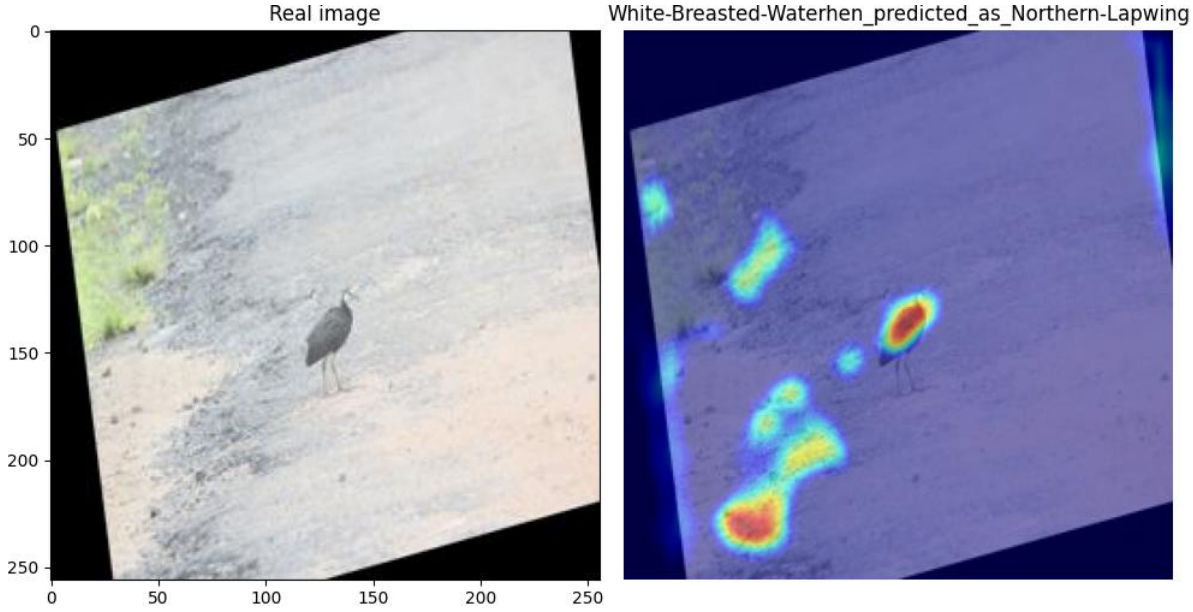


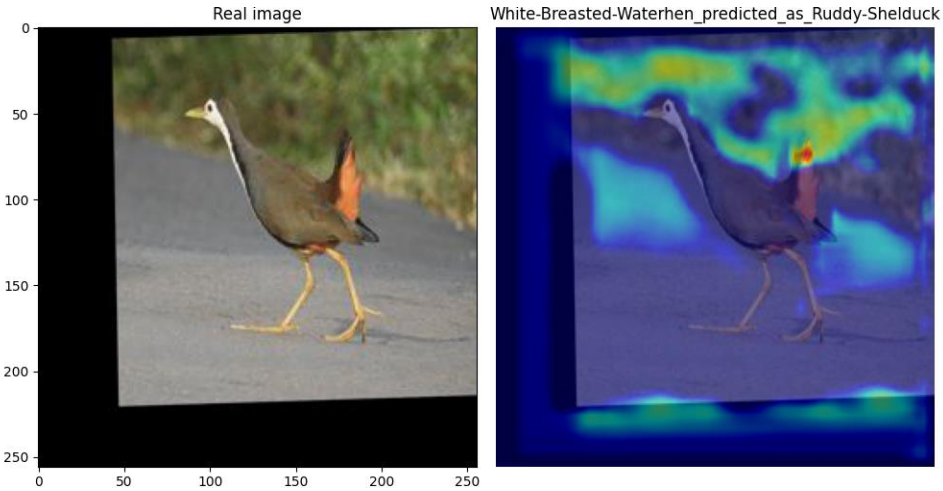


Incorrectly labelled samples




Real image
White-Breasted-Waterhen_predicted_as_Jungle-Babbler
0
50
100
150
200
250


Real image
White-Breasted-Waterhen_predicted_as_Northern-Lapwing
0
50
100
150
200
250


Real image
White-Breasted-Waterhen_predicted_as_Ruddy-Shelduck
0
50
100
150
200
250

Observations

It is interesting to note that Indian Peacock is classified based on its structure and blue colour, whereas in some cases like Ruddy-Shelduck, background plays a more important role in classification. Some of the misclassified examples are even difficult for human to distinguish.

COL775 Deep Learning

Assignment 1b

Name: T Karthikeyan

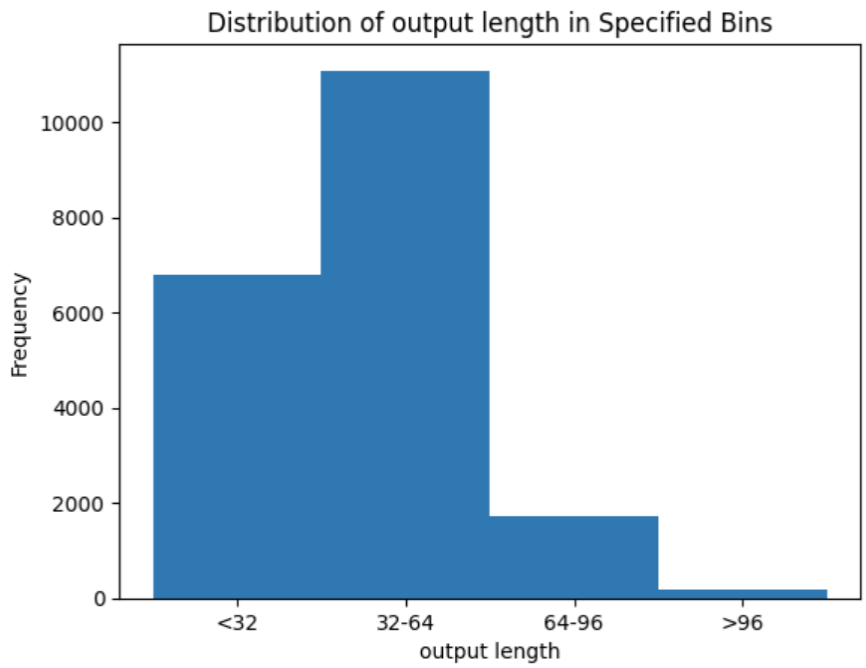
Entry No. 2023AIZ8140

Model weights: https://drive.google.com/drive/folders/1oSYXGoby6_SfVpIp5mDg9S7oTLKmnw89

Part 2

Text2Math

Dataset Exploration



Training data have #tokens in linear_formula of this distribution which made me decide to infer for a maximum length of 96

Seq 2 Seq architectures

For comparison across all the models, hyperparameters are kept constant

Hyperparameters used are:

Hidden dimension = 128 (In case of LSTM encoder)

Hidden dimension = 768 (In case of BERT encoder)

Batch size = 16

No. of layers of LSTM = 1

Embedding dimension = 100 (Glove embeddings (In case of LSTM encoder))

Embedding dimension = 768 (In case of BERT encoder)

Teacher forcing = 0.6

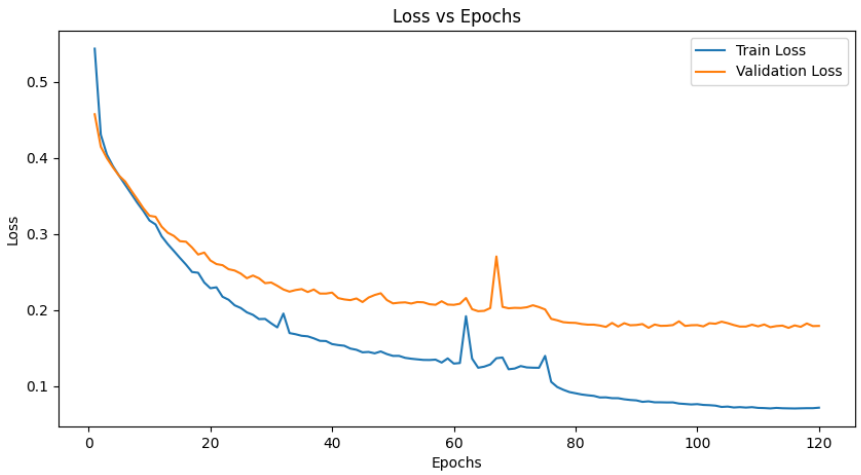
Beam width = 10

Scheduler = Reduce LR on Plateau

Optimizer = Adam

BiLSTM Encoder + LSTM decoder


```
{
    "train loss": 0.07142249218255234,
    "val loss": 0.17906585822923368,
    "Hyperparameters": {
        "HIDDEN SIZE": 128,
        "BATCH SIZE": 16,
        "EMBED DIM": 100,
        "LEARNING RATE": 0.001,
        "EPOCHS": 120
    }
}
```

```
    {
        "Problem": "? % of 360 = 108",
        "linear_formula": "multiply(n1,const_100)|divide(#0,n0)|",
        "answer": 30,
        "predicted": "multiply(n1,const_100)|multiply(n0,#0)|divide(#2,#1)|",
        "predicted_answer": null
    },
    {
        "Problem": "a train running at the speed of 120 km / hr crosses a pole in 18 seconds . what is the length of the train
?",
        "linear_formula": "multiply(n0,const_1000)|divide(#0,const_3600)|multiply(n1,#1)|",
        "answer": 600,
        "predicted": "multiply(n0,const_1000)|divide(#0,const_3600)|multiply(n1,#1)|",
        "predicted_answer": 600.0
    },
    {
        "Problem": "a train 100 meters long completely crosses a 300 meters long bridge in 45 seconds . what is the speed of
the train is ?",
        "linear_formula": "add(n0,n1)|divide(n2,const_3600)|divide(#0,const_1000)|divide(#2,#1)|",
        "answer": 32,
        "predicted": "add(n0,n1)|divide(n2,const_3600)|divide(#0,const_1000)|divide(#2,#1)|",
        "predicted_answer": 32.0
    },
```

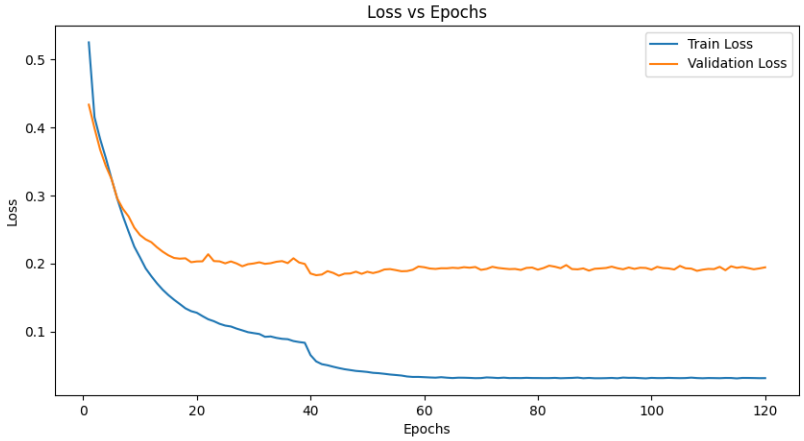

Observations

```
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch1/dev_predicted_final_bm10.json
  0%|                                                                                                                    | 0/2961 [00:00<?, ?it/s]
100%|████████████████████████████████████████████████████████████████████████████████████████████████████| 2961/2961 [00:01<00:00, 2231.47it/s]
Execution Accuracy: 47.04491725768322!!
Exact Match Accuracy: 45.187436676798384!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$
```

```
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch1/test_predicted_final_bm10.json
  0%|                                                                                                                    | 0/1925 [00:00<?, ?it/s]
 69%|████████████████████████████████████████████████████████████████                                                   | 1319/1925 [00:01<00:00, 946.72it/s]
100%|████████████████████████████████████████████████████████████████████████████████████████████████████| 1925/1925 [00:02<00:00, 931.55it/s]
Execution Accuracy: 48.72727272727273!!
Exact Match Accuracy: 46.18181818181818!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$
```

BiLSTM Encoder + Attention + LSTM decoder


```
{
    "train loss": 0.031341394861270175,
    "val loss": 0.19439149394090618,
    "Hyperparameters": {
        "HIDDEN SIZE": 128,
        "BATCH SIZE": 16,
        "EMBED DIM": 100,
        "LEARNING RATE": 0.001,
        "TEACHER_FORCING": 0.6,
        "EPOCHS": 120
    }
}
```



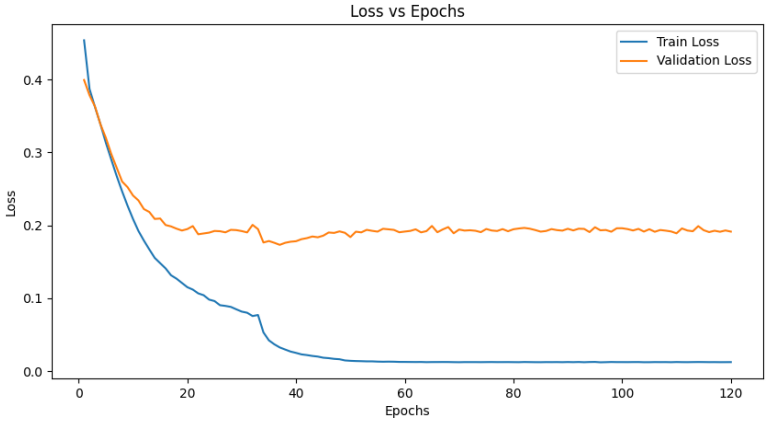

```
        "linear_formula": "divide(n3,const_100)|divide(n0,const_100)|divide(n2,const_100)|multiply(n1,#0)|multiply(n1,#1)|
subtract(#2,#0)|subtract(#3,#4)|divide(#6,#5)",
        "answer": 60,
        "predicted": "divide(n3,const_100)|divide(n0,const_100)|divide(n2,const_100)|multiply(n1,#0)|multiply(n1,#1)|subtract
(#2,#0)|subtract(#3,#4)|divide(#6,#5)|",
        "predicted_answer": 60.000000000000014
    },
    {
        "Problem": "a is 30 % more efficient than b . how much time they will working together take to complete a job which a
alone could have done in 23 days ?",
        "linear_formula": "divide(const_1,n1)|divide(n0,const_100)|add(#1,const_1)|multiply(n1,#2)|divide(const_1,#3)|add(#0,
#4)|inverse(#5)|",
        "answer": 13,
        "predicted": "divide(const_100,n1)|divide(n0,const_100)|add(#1,const_1)|multiply(n1,#2)|divide(const_1,#3)|add(#0,#4)|
inverse(#5)",
        "predicted_answer": null
    },
    {
        "Problem": "find the cost of fencing around a circular field of diameter 12 m at the rate of rs . 3.50 a meter ?",
        "linear_formula": "divide(n0,const_2)|circumface(#0)|multiply(n1,#1)|",
        "answer": 131.95,
        "predicted": "divide(n0,const_2)|circumface(#0)|multiply(n1,#1)|",
        "predicted_answer": 131.94689145077132
    },
```


Observations

```
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch2_tf6/dev_predicted_final_bm10.json
100%|| 2961/2961 [00:00<00:00, 39097.05it/s]
Execution Accuracy: 66.3627152988855!!
Exact Match Accuracy: 62.749071259709551!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch2_tf6/test_predicted_final_bm10.json
100%|| 1925/1925 [00:00<00:00, 37635.75it/s]
Execution Accuracy: 66.85714285714286!!
Exact Match Accuracy: 63.428571428571421!
```

BERT Encoder (Frozen) + Attention + LSTM decoder


```
{
    "train loss": 0.012309274305126471,
    "val loss": 0.19143003485195578,
    "Hyperparameters": {
        "HIDDEN SIZE": 768,
        "BATCH SIZE": 16,
        "EMBED DIM": 100,
        "LEARNING RATE": 0.001,
        "EPOCHS": 120
    }
}
```



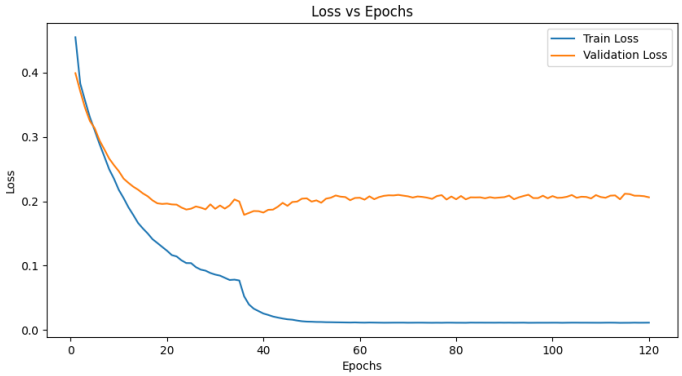

```
    {
        "Problem": "the average of 50 numbers id 20 . if two numbers , namely 45 and 55 are discarded , the average of the
remaining numbers is :",
        "linear_formula": "add(n2,n3)|multiply(n0,n1)|subtract(n0,const_2)|subtract(#1,#0)|divide(#3,#2)|",
        "answer": 18.75,
        "predicted": "add(n2,n3)|multiply(n0,n1)|subtract(n0,const_2)|subtract(#1,#0)|divide(#3,#2)|",
        "predicted_answer": 18.75
    },
    {
        "Problem": "the area of a triangle is with base 10 m and height 10 m ?",
        "linear_formula": "triangle_area(n0,n1)|",
        "answer": 50,
        "predicted": "triangle_area(n0,n1)|",
        "predicted_answer": 50.0
    },
    {
        "Problem": "in what time will a railway train 40 m long moving at the rate of 36 kmph pass a telegraph post on its way
?",
        "linear_formula": "multiply(n1,const_0_2778)|divide(n0,#0)|",
        "answer": 4,
        "predicted": "multiply(n1,const_0_2778)|divide(n0,#0)|",
        "predicted_answer": 3.9996800255979523
    },
```


Observations

```
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch3/dev_predicted_final_bm10.json
100%|                                                                                | 2961/2961 [00:00<00:00, 38535.49it/s]
Execution Accuracy: 79.5677136102668!!
Exact Match Accuracy: 75.07598784194529!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch3/test_predicted_final_bm10.json
100%|                                                                                | 1925/1925 [00:00<00:00, 10760.72it/s]
Execution Accuracy: 80.72727272727272!!
Exact Match Accuracy: 76.25974025974025!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$
```

BERT Encoder (Finetuned) + Attention + LSTM decoder


```
{
    "train loss": 0.011326356153075395,
    "val loss": 0.20625969171724332,
    "Hyperparameters": {
        "HIDDEN SIZE": 768,
        "BATCH SIZE": 16,
        "EMBED DIM": 100,
        "LEARNING RATE": 0.001,
        "EPOCHS": 120
    }
}
```




```
    {
        "Problem": "an art gallery has only paintings and sculptures . currently , 1 / 3 of the pieces of art are displayed ,
and 1 / 6 of the pieces on display are sculptures . if 1 / 3 of the pieces not on display are paintings , and 800 sculptures
are not on display , how many pieces of art does the gallery have ?",
        "linear_formula": "divide(n0,n1)|subtract(n0,#0)|divide(n6,#1)|divide(#2,#1)|",
        "answer": 1800,
        "predicted": "divide(n0,n1)|subtract(n0,#0)|divide(n6,#1)|divide(#2,#1)|",
        "predicted_answer": 1799.9999999999995
    },
    {
        "Problem": "if x and y are integers and | x - y | = 10 , what is the minimum possible value of xy ?",
        "linear_formula": "add(const_2,const_3)|negate(#0)|subtract(n0,#0)|multiply(#1,#2)|",
        "answer": -25,
        "predicted": "add(const_2,const_3)|negate(#0)|subtract(n0,#0)|multiply(#1,#2)",
        "predicted_answer": -25.0
    },
    {
        "Problem": "two trains , one from howrah to patna and the other from patna to howrah , start simultaneously . after
they meet , the trains reach their destinations after 25 hours and 16 hours respectively . the ratio of their speeds is ?",
        "linear_formula": "sqrt(n1)|sqrt(n0)|divide(#0,#1)|",
        "answer": 0.8,
        "predicted": "sqrt(n1)|sqrt(n0)|divide(#0,#1)|",
        "predicted_answer": 0.8
    },
```


Observations

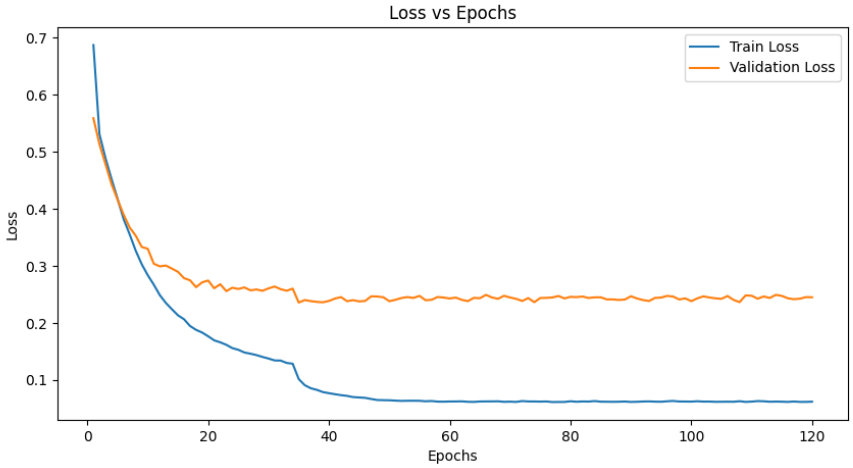
Architecture	Exact Accuracy (val set) (in %)	Exec Accuracy (val set) (in %)	Exact Accuracy (test set) (in %)	Exec Accuracy (test set) (in %)
LSTM-LSTM	45.18	47.04	46.18	48.72
LSTM-ATTN-LSTM	62.74	66.36	63.42	66.85
BERT-ATTN-LSTM (frozen)	75.07	79.56	76.25	80.72
BERT-ATTN-LSTM (finetuned)	75.14	79.56	76.93	81.29

It could be verified that adding the attention mechanism to LSTM LSTM architecture improves the result. BERT (frozen) performs significantly better than LSTM encoder. BERT (finetuned) performs slightly better than BERT (frozen)

Effect of Teacher Forcing probability

BiLSTM Encoder + Attention + LSTM decoder (TF = 0.3)

```
{
    "train loss": 0.06216915837122412,
    "val loss": 0.24521532733874615,
    "Hyperparameters": {
        "HIDDEN SIZE": 128,
        "BATCH SIZE": 16,
        "EMBED DIM": 100,
        "LEARNING RATE": 0.001,
        "TEACHER_FORCING": 0.3,
        "EPOCHS": 120
    }
}
```

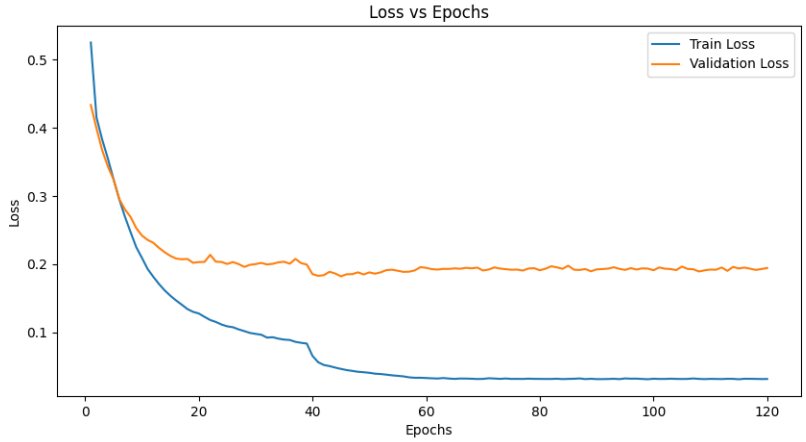




```
    {
        "Problem": "a collection of books went on sale , and 2 / 3 of them were sold for $ 3.50 each . if none of the 40
remaining books were sold , what was the total amount received for the books that were sold ?",
        "linear_formula": "divide(n2,n1)|divide(const_1,n1)|multiply(n3,#0)|divide(#2,#1)|multiply(n0,#3)|",
        "answer": 280,
        "predicted": "multiply(n0,n1)|divide(#0,const_2)|multiply(n2,#1)|add(n3,#2)",
        "predicted_answer": null
    },
    {
        "Problem": "excluding stoppages , the speed of a train is 60 kmph and including stoppages it is 40 kmph . of how many
minutes does the train stop per hour ?",
        "linear_formula": "divide(n1,n0)|multiply(#0,const_60)|subtract(const_60,#1)|",
        "answer": 20,
        "predicted": "divide(n1,n0)|multiply(#0,const_60)|subtract(const_60,#1)|",
        "predicted_answer": 20.0
    },
    {
        "Problem": "an error 10 % in excess is made while measuring the side of a square . now what is the percentage of error
in the calculated area of the square ?",
        "linear_formula": "add(n0,const_100)|square_area(const_100)|square_area(#0)|subtract(#2,#1)|multiply(#3,const_100)|
divide(#4,#1)|",
        "answer": 21,
        "predicted": "add(n0,const_100)|square_area(const_100)|square_area(#0)|subtract(#2,#1)|multiply(#3,const_100)|divide
(#4,#1)|",
        "predicted_answer": 21.0
    },
```


```
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch2_tf3/dev_predicted_final_bm10.json
100%|| 2961/2961 [00:00<00:00, 39388.94it/s]
Execution Accuracy: 66.46403242147923!!
Exact Match Accuracy: 63.627152988855116!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch2_tf3/test_predicted_final_bm10.json
  0%|                                                                                                  | 0/1925 [00:00<?, ?it/s]
100%|| 1925/1925 [00:01<00:00, 1637.71it/s]
Execution Accuracy: 66.85714285714286!!
Exact Match Accuracy: 64.67532467532467!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$
```

BiLSTM Encoder + Attention + LSTM decoder (TF = 0.6)

```
{
    "train loss": 0.031341394861270175,
    "val loss": 0.19439149394090618,
    "Hyperparameters": {
        "HIDDEN SIZE": 128,
        "BATCH SIZE": 16,
        "EMBED DIM": 100,
        "LEARNING RATE": 0.001,
        "TEACHER_FORCING": 0.6,
        "EPOCHS": 120
    }
}
```

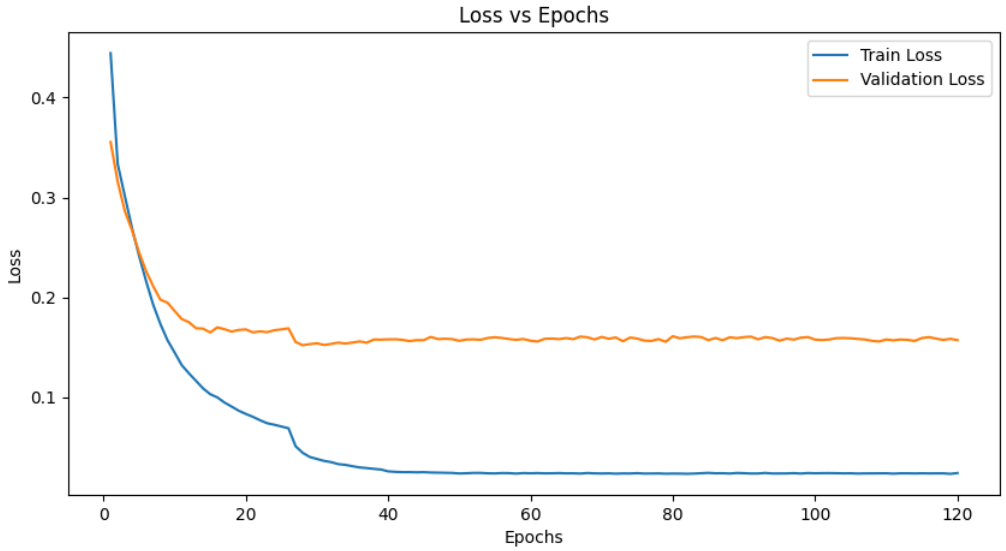




```
        "linear_formula": "divide(n3,const_100)|divide(n0,const_100)|divide(n2,const_100)|multiply(n1,#0)|multiply(n1,#1)|
subtract(#2,#0)|subtract(#3,#4)|divide(#6,#5)",
        "answer": 60,
        "predicted": "divide(n3,const_100)|divide(n0,const_100)|divide(n2,const_100)|multiply(n1,#0)|multiply(n1,#1)|subtract
(#2,#0)|subtract(#3,#4)|divide(#6,#5)|",
        "predicted_answer": 60.000000000000014
    },
    {
        "Problem": "a is 30 % more efficient than b . how much time they will working together take to complete a job which a
alone could have done in 23 days ?",
        "linear_formula": "divide(const_1,n1)|divide(n0,const_100)|add(#1,const_1)|multiply(n1,#2)|divide(const_1,#3)|add(#0,
#4)|inverse(#5)|",
        "answer": 13,
        "predicted": "divide(const_100,n1)|divide(n0,const_100)|add(#1,const_1)|multiply(n1,#2)|divide(const_1,#3)|add(#0,#4)|
inverse(#5)",
        "predicted_answer": null
    },
    {
        "Problem": "find the cost of fencing around a circular field of diameter 12 m at the rate of rs . 3.50 a meter ?",
        "linear_formula": "divide(n0,const_2)|circumface(#0)|multiply(n1,#1)|",
        "answer": 131.95,
        "predicted": "divide(n0,const_2)|circumface(#0)|multiply(n1,#1)|",
        "predicted_answer": 131.94689145077132
    },
```


```
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch2_tf6/dev_predicted_final_bm10.json
100%|████████████████████████████████████████████████████████████| 2961/2961 [00:00<00:00, 39097.05it/s]
Execution Accuracy: 66.3627152988855!!
Exact Match Accuracy: 62.74907125970955!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch2_tf6/test_predicted_final_bm10.json
100%|████████████████████████████████████████████████████████████| 1925/1925 [00:00<00:00, 37635.75it/s]
Execution Accuracy: 66.85714285714286!!
Exact Match Accuracy: 63.428571428571421!!
```

BiLSTM Encoder + Attention + LSTM decoder (TF = 0.9)

```
{
    "train loss": 0.024446109197988636,
    "val loss": 0.15736342709191065,
    "Hyperparameters": {
        "HIDDEN SIZE": 128,
        "BATCH SIZE": 16,
        "EMBED DIM": 100,
        "LEARNING RATE": 0.001,
        "TEACHER_FORCING": 0.9,
        "EPOCHS": 120
    }
}
```


```
    {
        "Problem": "a bag marked at $ 125 is sold for $ 120 . the rate of discount is ?",
        "linear_formula": "subtract(n0,n1)|divide(#0,n0)|multiply(#1,const_100)",
        "answer": 4,
        "predicted": "subtract(n1,n0)|divide(#0,n0)|multiply(#1,const_100)|",
        "predicted_answer": null
    },
    {
        "Problem": "the radius of a cylindrical vessel is 8 cm and height is 3 cm . find the whole surface of the cylinder ?",
        "linear_formula": "surface_cylinder(n0,n1)|",
        "answer": 553.1,
        "predicted": "surface_cylinder(n0,n1)|",
        "predicted_answer": 552.9203070318035
    },
    {
        "Problem": "the marks obtained by vijay and amith are in the ratio 6 : 5 and those obtained by amith and abhishek in
the ratio of 3 : 2 . the marks obtained by vijay and abhishek are in the ratio of ?",
        "linear_formula": "multiply(n0,n2)|multiply(n1,n3)|divide(#0,#1)|",
        "answer": 1.8,
        "predicted": "multiply(n0,n2)|multiply(n1,n3)|divide(#0,#1)|",
        "predicted_answer": 1.8
    },
```


```
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch2_tf9/dev_predicted_final_bm10.json
100%|| 2961/2961 [00:00<00:00, 40629.74it/s]
Execution Accuracy: 51.469098277608914!!
Exact Match Accuracy: 47.88922661263087!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/
Assignment_1b/data/outputs/arch2_tf9/test_predicted_final_bm10.json
100%|| 1925/1925 [00:00<00:00, 24001.15it/s]
Execution Accuracy: 53.1948051948052!!
Exact Match Accuracy: 49.76623376623376!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$
```

Observations

Architecture	Exact Accuracy	Exec Accuracy	Exact Accuracy	Exec Accuracy

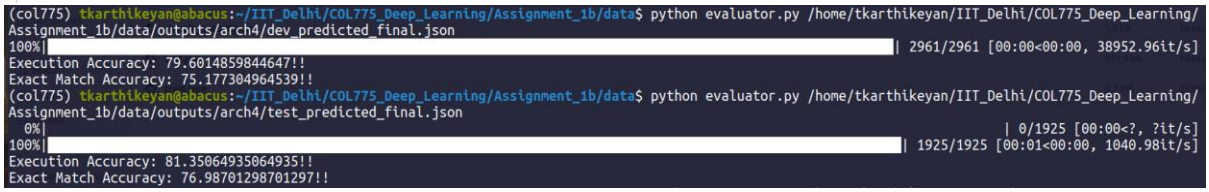
(LSTM-ATTN-LSTM)	(val set) (in %)	(val set) (in %)	(test set) (in %)	(test set) (in %)
Teacher forcing ratio 0.3	63.62	66.46	64.67	66.85
Teacher forcing ratio 0.6	62.74	66.36	63.42	66.85
Teacher forcing ratio 0.9	47.88	51.46	49.76	53.19

It could be seen from the loss curves that higher teacher forcing ratio makes the model converge faster but that is not translating increase in accuracy. Teacher forcing ratio 0.3 gives the best result.

Effect of Beam Size

BERT Encoder (Finetuned) + Attention + LSTM decoder (Beam width = 1)


```
    {
        "Problem": "when working alone , painter w can paint a room in 2 hours , and working alone , painter x can paint the
same room in z hours . when the two painters work together and independently , they can paint the room in 3 / 4 of an hour .
what is the value of z ?",
        "linear_formula": "add(n0,n1)|subtract(n2,n0)|divide(const_1,#0)|add(#2,#1)|",
        "answer": 2.2,
        "predicted": "add(n0,n1)|subtract(n2,n0)|divide(const_1,#0)|add(#2,#1)|",
        "predicted_answer": 2.2
    },
    {
        "Problem": "what is the dividend . divisor 17 , the quotient is 8 and the remainder is 5",
        "linear_formula": "multiply(n0,n1)|add(n2,#0)|",
        "answer": 141,
        "predicted": "multiply(n0,n1)|add(n2,#0)|",
        "predicted_answer": 141.0
    },
    {
        "Problem": "a searchlight on top of the watch - tower makes 3 revolutions per minute . what is the probability that a
man appearing near the tower will stay in the dark for at least 15 seconds ?",
        "linear_formula": "multiply(const_1,const_60)|divide(#0,n0)|divide(n1,#1)|subtract(const_1,#2)|",
        "answer": 0.25,
        "predicted": "multiply(const_1,const_60)|divide(#0,n0)|divide(n1,#1)|subtract(const_1,#2)|",
        "predicted_answer": 0.25
    },
```

BERT Encoder (Finetuned) + Attention + LSTM decoder (Beam width = 10)

```
    {
        "Problem": "an art gallery has only paintings and sculptures . currently , 1 / 3 of the pieces of art are displayed , and 1 / 6 of the pieces on display are sculptures . if 1 / 3 of the pieces not on display are paintings , and 800 sculptures are not on display , how many pieces of art does the gallery have ?",
        "linear_formula": "divide(n0,n1)|subtract(n0,#0)|divide(n6,#1)|divide(#2,#1)|",
        "answer": 1800,
        "predicted": "divide(n0,n1)|subtract(n0,#0)|divide(n6,#1)|divide(#2,#1)|",
        "predicted_answer": 1799.9999999999995
    },
    {
        "Problem": "if x and y are integers and | x - y | = 10 , what is the minimum possible value of xy ?",
        "linear_formula": "add(const_2,const_3)|negate(#0)|subtract(n0,#0)|multiply(#1,#2)|",
        "answer": -25,
        "predicted": "add(const_2,const_3)|negate(#0)|subtract(n0,#0)|multiply(#1,#2)",
        "predicted_answer": -25.0
    },
    {
        "Problem": "two trains , one from howrah to patna and the other from patna to howrah , start simultaneously . after they meet , the trains reach their destinations after 25 hours and 16 hours respectively . the ratio of their speeds is ?",
        "linear_formula": "sqrt(n1)|sqrt(n0)|divide(#0,#1)|",
        "answer": 0.8,
        "predicted": "sqrt(n1)|sqrt(n0)|divide(#0,#1)|",
        "predicted_answer": 0.8
    },
```

```
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data/outputs/arch4/dev_predicted_final_bm10.json
100%|████████████████████████████████████████| 2961/2961 [00:00<00:00, 38139.40it/s]
Execution Accuracy: 79.5677136102668!!
Exact Match Accuracy: 75.1435325903411!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data/outputs/arch4/test_predicted_final_bm10.json
  0%|                                        | 0/1925 [00:00<?, ?it/s]
100%|████████████████████████████████████████| 1925/1925 [00:01<00:00, 1721.67it/s]
Execution Accuracy: 81.2987012987013!!
Exact Match Accuracy: 76.93506493506493!!
```

BERT Encoder (Finetuned) + Attention + LSTM decoder (Beam width = 20)

```
    {
        "Problem": "in a kilometer race , a beats b by 48 meters or 12 seconds . what time does a take to complete the race ?",
        "linear_formula": "divide(n0,n1)|multiply(const_1,const_1000)|divide(#1,#0)|subtract(#2,n1)|",
        "answer": 238,
        "predicted": "divide(n0,n1)|multiply(const_1,const_1000)|divide(#1,#0)|subtract(#2,n1)|",
        "predicted_answer": 238.0
    },
    {
        "Problem": "in a school of 650 boys , 44 % of muslims , 28 % hindus , 10 % sikhs and the remaining of other communities . how many belonged to the other communities ?",
        "linear_formula": "add(n1,n2)|add(n3,#0)|subtract(const_100,#1)|multiply(n0,#2)|divide(#3,const_100)",
        "answer": 117,
        "predicted": "add(n1,n2)|add(n3,#0)|subtract(const_100,#1)|multiply(n0,#2)|divide(#3,const_100)|",
        "predicted_answer": 117.0
    },
    {
        "Problem": "a can do a piece of work in 4 hours ; b and c together can do it in 3 hours , while a and c together can do it 2 hours . how long will b alone take to do it ?",
        "linear_formula": "divide(const_1,n1)|divide(const_1,n2)|divide(const_1,n0)|subtract(#1,#2)|subtract(#0,#3)|divide(const_1,#4)|",
        "answer": 12,
        "predicted": "divide(const_1,n1)|divide(const_1,n2)|divide(const_1,n0)|subtract(#1,#2)|subtract(#0,#3)|divide(const_1,#4)",
        "predicted_answer": 12.000000000000004
    },
```

```
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data/outputs/arch4/dev_predicted_final_bm20.json
100%|████████████████████████████████████████| 2961/2961 [00:00<00:00, 37926.84it/s]
Execution Accuracy: 79.6014859844647!!
Exact Match Accuracy: 75.177304964539!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$ python evaluator.py /home/tkarthikeyan/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data/outputs/arch4/test_predicted_final_bm20.json
  0%|                                        | 0/1925 [00:00<?, ?it/s]
100%|████████████████████████████████████████| 1925/1925 [00:01<00:00, 1662.57it/s]
Execution Accuracy: 81.35064935064935!!
Exact Match Accuracy: 76.98701298701297!!
(col775) tkarthikeyan@abacus:~/IIT_Delhi/COL775_Deep_Learning/Assignment_1b/data$
```

Observations

Architecture	Exact Accuracy (val set) (in %)	Exec Accuracy (val set) (in %)	Exact Accuracy (test set) (in %)	Exec Accuracy (test set) (in %)

BERT-ATTN-LSTM (finetuned)				
Beam width 1	75.17	79.60	76.98	81.35
Beam width 10	75.14	79.56	76.93	81.29
Beam width 20	75.17	79.60	76.98	81.35

It's quite surprising that beam width 1 and beam width 20 have got exact same accuracy whereas beam width 10 is slightly lower. Maybe the task at our hand is not creative generation because of which we are not able to appreciate beam width or maybe the model have learnt properly to generate the next token that greedy search itself is sufficient

Assignment 2

PART 1: Object Centric Learning with Slot Attention

Model link:

https://drive.google.com/file/d/1PDChJox2DH9DpMOvdDcXkMqBaKqW4Udg/view?usp=sharing

Submission by:

T Karthikeyan 2023AIZ8140 and Anirban Ray 2023AIZ8073

Encoder(ResNet 18)

```python
class Encoder(nn.Module):
    '''
        Input dim: [B, 3, 128, 128]
        Output dim: [B, 11, 64]
    '''
    def __init__(self):
        super().__init__()
        self.Resnet = ResNet()
        self.SlotAttn = SlotAttention()

    def forward(self, x):
        x = self.Resnet(x) # (B, 1024, 64)
        slots = self.SlotAttn(x) # (B, K, 64)

        return slots
```

Slot Attention

```python
def forward(self, inputs, n_slots = 11):
    B, _, _ = inputs.shape
    n_slots = n_slots if n_slots is not None else self.n_slots

    mu = self.slots_mu.expand(B, n_slots, -1) # (B, K, DIM)
    sigma = self.slots_sigma.expand(B, n_slots, -1) # (B, K, DIM)

    slots = torch.normal(mu, sigma) # (B, K, DIM) .. sample from the normal distribution elementwise

    inputs = self.layer_norm_in(inputs) # (B, N, DIM)

    key_vec, val_vec = self.K(inputs), self.V(inputs) # both key and val are of dimension (B, N, DIM)

    for _ in range(self.iterations):
        slots_prev = slots

        slots = self.layer_norm_slots(slots) # (B, K, DIM)

        query_vec = self.Q(slots) # (B, K, DIM)

        dot_result = torch.einsum('bkd,bnd->bkn', query_vec, key_vec) * self.scale # (B, K, N) .... query
is (B, K, DIM) and key is (B, N, DIM) and it is converted to (B, K, N) #Einstein summation convention

        attn = dot_result.softmax(dim=1) + self.eps  # (B, K, N)

        attn = attn / attn.sum(dim=-1, keepdim=True) # (B, K, N) .. for the weighted mean

        updates = torch.einsum('bnd,bkn->bkd', val_vec, attn) # (B, K, DIM) .... val is (B, N, DIM) and
attn is (B, K, N) and it is converted to (B, K, DIM) #Einstein summation convention

        slots = self.gru(
            updates.reshape(-1, self.dim),
            slots_prev.reshape(-1, self.dim)
        ) # (B*K, DIM)

        slots = slots.reshape(B, -1, self.dim) # (B, K, DIM)

        slots_mlp = self.layer_norm_pre_mlp(slots) # (B, K, DIM)

        slots_mlp = self.mlp(slots_mlp) # (B, K, DIM)

        slots = slots + slots_mlp # (B, K, DIM)

    return slots
```

Decoder (Spatial Broadcast based)

```python
class Decoder(nn.Module):
    '''
        Input dim: [B, 11, 64]
        Output dim: [B, 3, 128, 128]
    '''
    def __init__(self):
        super().__init__()
        self.DeCNN = DeCNN()
        self.softmax = nn.Softmax(dim=1)

    def forward(self, slots):
        y = self.DeCNN(slots) # (B, K, 128, 128, 4)

        re_constructed_images, masks = y.split([3,1], dim=-1) # (B, K, 128, 128, 3) and (B, K, 128, 128, 1)
        masks = self.softmax(masks) # To normalize across slots

        re_constructed_image = torch.sum(re_constructed_images * masks, dim=1)  # (B, 128, 128, 3)

        re_constructed_image = re_constructed_image.permute(0,3,1,2) # (B, 3, 128, 128)
        return re_constructed_image, re_constructed_images, masks
```

Some samples from the CLEVERTex dataset

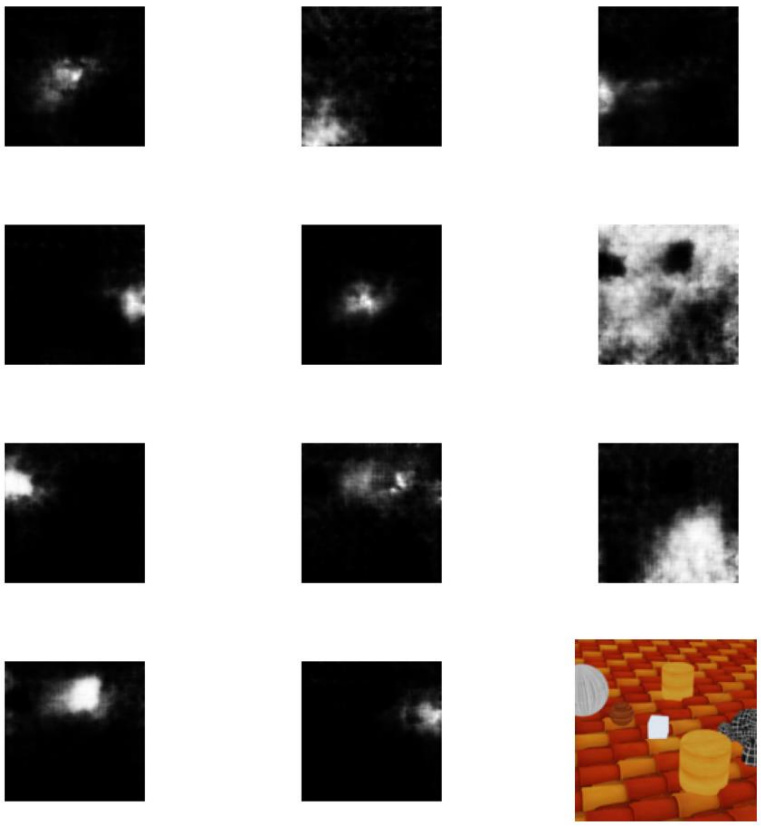
Hyperparameters

```
BATCH_SIZE = 64
LEARNING_RATE = 0.0004
```

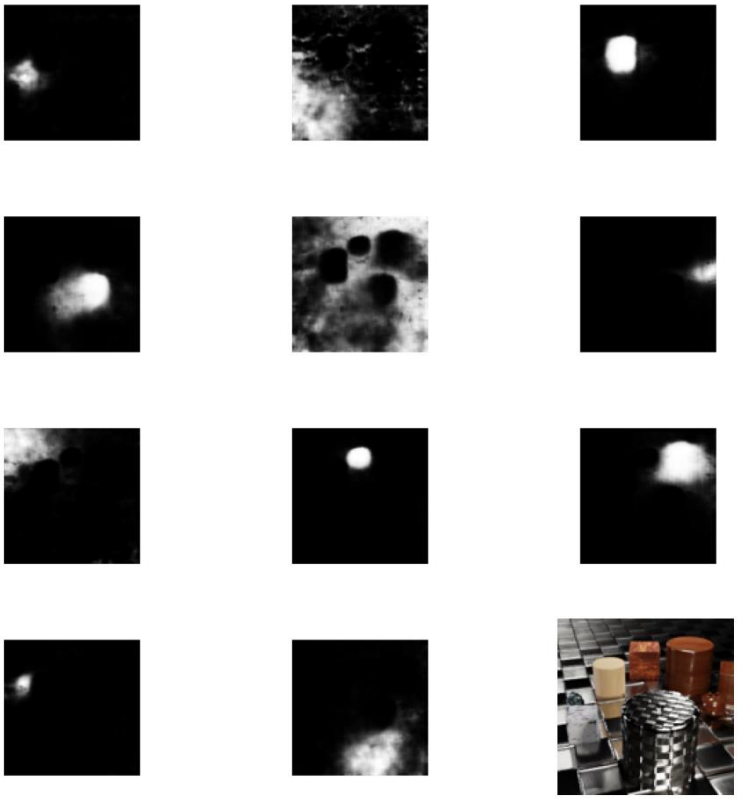
```
EPOCHS = 400
SAVE_STEPS = 5
NUM_WORKERS = 16
SLOT SIZE = 64
```

Generated Masks

Reconstruction masks



Reconstruction masks



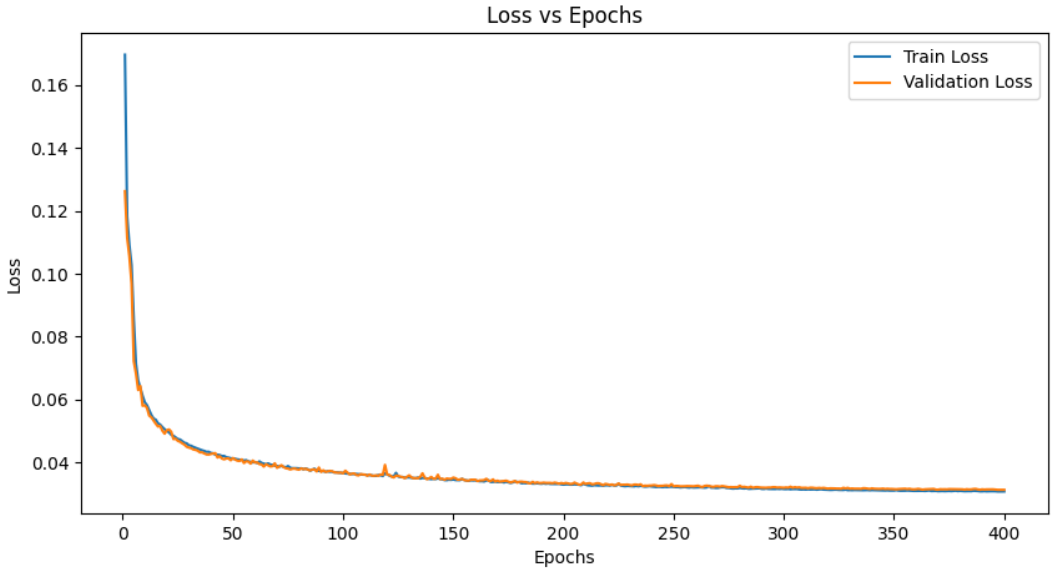
Reconstructed Images

ARI Score on Validation Set:

```
(col775_a1) tkarthikeyan@wonderland:~/IIT_DELHI/COL775_Deep_Learning/Assignment_2$ python eval.py
Device is  cuda
model loaded!
100%|| 10000/10000 [10:34<00:00, 15.76it/s]
Validation ARI score is  0.23533628491542186
(col775_a1) tkarthikeyan@wonderland:~/IIT_DELHI/COL775_Deep_Learning/Assignment_2$
```

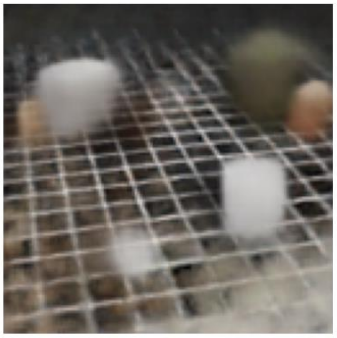
ARI Score: 0.23533628491542186

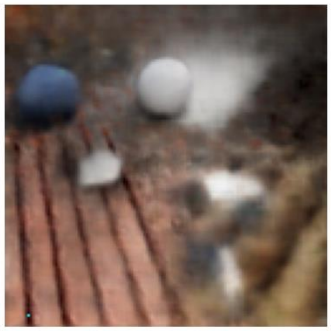
Loss Curve



final_train loss: **0.03072331871688366**
final_val loss: **0.031248664898667365**

Compositional Generation





Clean FID metric:

```
(col775_a1) tkarthikeyan@wonderland:~/IIT_DELHI/COL775_Deep_Learning/Assignment_2b$ python
Python 3.10.13 (main, Sep 11 2023, 13:44:35) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from cleanfid import fid
>>> fdir1 = "/home/tkarthikeyan/IIT_DELHI/COL775_Deep_Learning/Assignment_2/dataset/images/val"
>>> fdir2 = "/home/tkarthikeyan/IIT_DELHI/COL775_Deep_Learning/Assignment_2/outputs/batch64_400epochs/val_new"
>>> score = fid.compute_fid(fdir1, fdir2)
compute FID between two folders
FID val : 100%|| 313/313 [00:52<00:00,  5.94it/s]
FID val_new : 100%|| 313/313 [00:54<00:00,  5.75it/s]
>>> score
212.12462489411405
>>>
```

Clean FID score: 212.12462489411405

Assignment 2

PART 2: Slot Learning using Diffusion based Decoder

Model link: https://drive.google.com/file/d/1c-fEmjk4jnbifJOImpNaK4Qs1rlRtBZp/view?usp=sharing

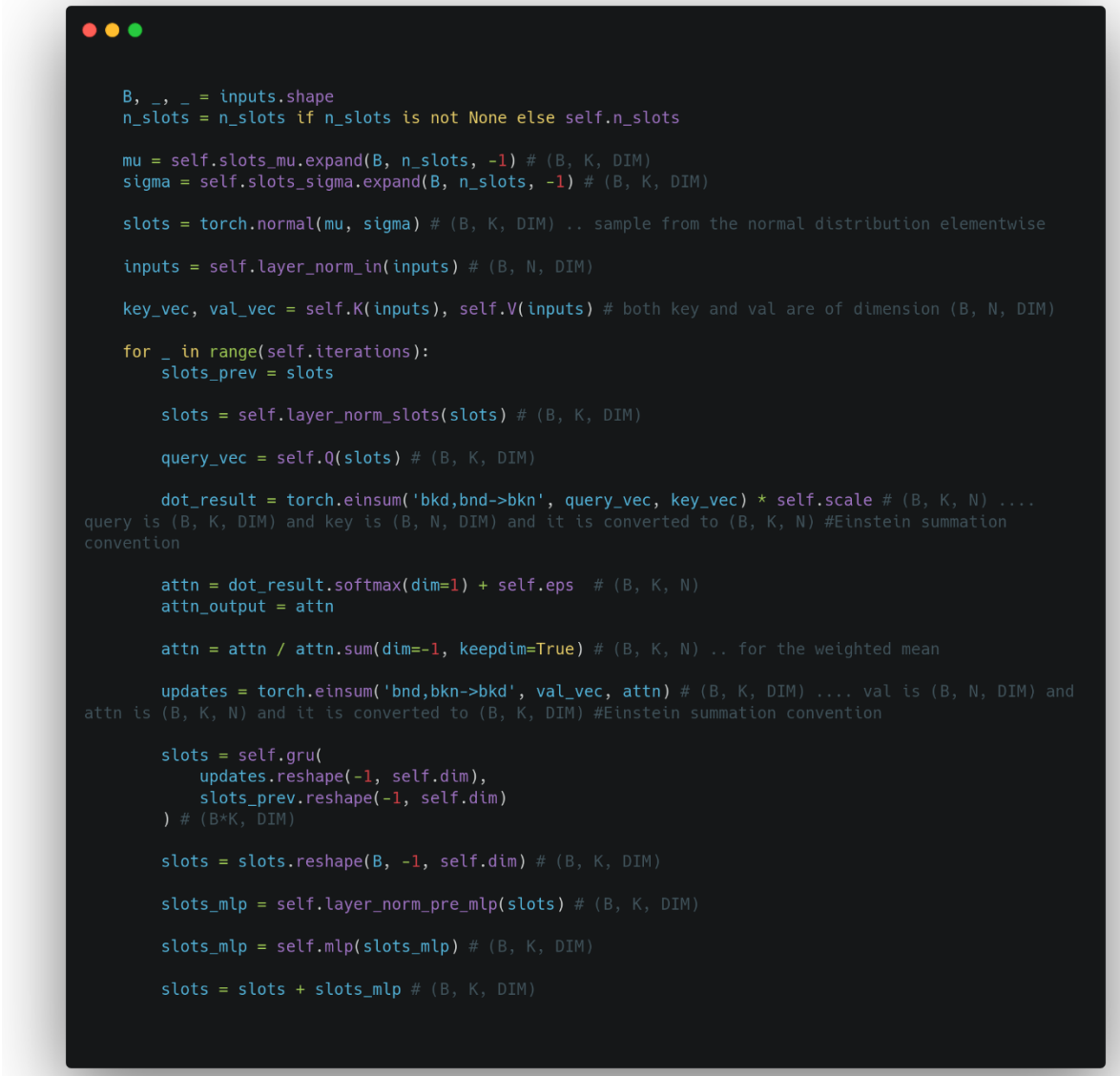
Submission by:

T Karthikeyan 2023AIZ8140 and Anirban Ray 2023AIZ8073

Encoder (ResNet 18)

```
class Encoder(nn.Module):
    '''
        Input dim: [B, 3, 128, 128]
        Output dim: [B, 11, 192]
    '''
    def __init__(self):
        super().__init__()
        self.Resnet = ResNet()
        self.SlotAttn = SlotAttention()

    def forward(self, x):
        x = self.Resnet(x) # (B, 1024, 64)
        slots, attn_output = self.SlotAttn(x) # (B, K, 64)

        return slots, attn_output
```

Slot attention



```
    B, _, _ = inputs.shape
    n_slots = n_slots if n_slots is not None else self.n_slots

    mu = self.slots_mu.expand(B, n_slots, -1) # (B, K, DIM)
    sigma = self.slots_sigma.expand(B, n_slots, -1) # (B, K, DIM)

    slots = torch.normal(mu, sigma) # (B, K, DIM) .. sample from the normal distribution elementwise

    inputs = self.layer_norm_in(inputs) # (B, N, DIM)

    key_vec, val_vec = self.K(inputs), self.V(inputs) # both key and val are of dimension (B, N, DIM)

    for _ in range(self.iterations):
        slots_prev = slots

        slots = self.layer_norm_slots(slots) # (B, K, DIM)

        query_vec = self.Q(slots) # (B, K, DIM)

        dot_result = torch.einsum('bkd,bnd->bkn', query_vec, key_vec) * self.scale # (B, K, N) ....
query is (B, K, DIM) and key is (B, N, DIM) and it is converted to (B, K, N) #Einstein summation
convention

        attn = dot_result.softmax(dim=1) + self.eps  # (B, K, N)
        attn_output = attn

        attn = attn / attn.sum(dim=-1, keepdim=True) # (B, K, N) .. for the weighted mean

        updates = torch.einsum('bnd,bkn->bkd', val_vec, attn) # (B, K, DIM) .... val is (B, N, DIM) and
attn is (B, K, N) and it is converted to (B, K, DIM) #Einstein summation convention

        slots = self.gru(
            updates.reshape(-1, self.dim),
            slots_prev.reshape(-1, self.dim)
        ) # (B*K, DIM)

        slots = slots.reshape(B, -1, self.dim) # (B, K, DIM)

        slots_mlp = self.layer_norm_pre_mlp(slots) # (B, K, DIM)

        slots_mlp = self.mlp(slots_mlp) # (B, K, DIM)

        slots = slots + slots_mlp # (B, K, DIM)
```

Decoder (UNET based)

```
    def forward(self, x, timestep_emb, context):
        conv1_out = self.conv1(x)
        r1_out = self.R1(conv1_out, timestep_emb)
        r2_out = self.R2(r1_out, timestep_emb)
        d1_out = self.D1(r2_out, timestep_emb)
        r3_out = self.R3(d1_out, timestep_emb)
        t1_out = self.T1(r3_out, context)
        r4_out = self.R4(t1_out, timestep_emb)
        t2_out = self.T2(r4_out, context)
        d2_out = self.D2(t2_out, timestep_emb)
        r5_out = self.R5(d2_out, timestep_emb)
        t3_out = self.T3(r5_out, context)
        r6_out = self.R6(t3_out, timestep_emb)
        t4_out = self.T4(r6_out, context)
        d3_out = self.D3(t4_out, timestep_emb)
        r7_out = self.R7(d3_out, timestep_emb)
        t5_out = self.T5(r7_out, context)
        r8_out = self.R8(t5_out, timestep_emb)
        t6_out = self.T6(r8_out, context)

        r9_out = self.R9(t6_out, timestep_emb)
        t7_out = self.T7(r9_out, context)
        r10_out = self.R10(t7_out, timestep_emb)

        r11_out = self.R11(torch.cat((t6_out, r10_out), dim = 1), timestep_emb)
        t8_out = self.T8(r11_out, context)
        r12_out = self.R12(torch.cat((t5_out, t8_out), dim = 1), timestep_emb)
        t9_out = self.T9(r12_out, context)
        r13_out = self.R13(torch.cat((d3_out, t9_out), dim = 1), timestep_emb)
        t10_out = self.T10(r13_out, context)
        u1_out = self.U1(t10_out, timestep_emb)
        r14_out = self.R14(torch.cat((t4_out, u1_out), dim = 1), timestep_emb)
        t11_out =self.T11(r14_out, context)
        r15_out = self.R15(torch.cat((t3_out, t11_out), dim = 1), timestep_emb)
        t12_out = self.T12(r15_out, context)
        r16_out = self.R16(torch.cat((d2_out, t12_out), dim = 1), timestep_emb)
        t13_out = self.T13(r16_out, context)
        u2_out = self.U2(t13_out, timestep_emb)
        r17_out = self.R17(torch.cat((t2_out, u2_out), dim = 1), timestep_emb)
        t14_out = self.T14(r17_out, context)
        r18_out = self.R18(torch.cat((t1_out, t14_out), dim = 1), timestep_emb)
        t15_out = self.T15(r18_out, context)
        r19_out = self.R19(torch.cat((d1_out, t15_out), dim = 1), timestep_emb)
        t16_out = self.T16(r19_out, context)
        u3_out = self.U3(t16_out, timestep_emb)
        r20_out = self.R20(torch.cat((r2_out, u3_out), dim = 1), timestep_emb)
        r21_out = self.R21(torch.cat((r1_out, r20_out), dim = 1), timestep_emb)
        r22_out = self.R22(torch.cat((conv1_out, r21_out), dim = 1),
timestep_emb)out = self.norm(r22_out)
        conv2_out = self.conv2(norm_out)

        return conv2_out
```

Residual block

```
    def forward(self, feature, time):
        #feature is [B, C, H, W]
        #time is [1, time_emb_size]

        out_feature = self.groupnorm1(feature) #[B, inC, H, W]
        out_feature = F.silu(out_feature) #[B, inC, H, W]

        if self.mode == "normal":
            out_feature = self.conv1(out_feature) #[B, outC, H, W]
        elif self.mode == "down":
            out_feature = self.avg(out_feature) #[B, outC, H/2, W/2]
        elif self.mode == "up":
            out_feature = F.interpolate(out_feature, scale_factor=2, mode='bilinear') #[B, outC, 2H, 2W]

        out_time = F.silu(time) #[1, time_emb_size]
        out_time = self.linear(out_time) #[1, outC]
        out_time = out_time.unsqueeze(dim = -1).unsqueeze(dim = -1) #[1, outC, 1, 1]

        out = out_feature + out_time #[B, outC, H*, W*]

        residual = out
        out = self.groupnorm2(out) #[B, outC, H*, W*]
        out = F.silu(out) #[B, outC, H*, W*]
        out = self.dropout(out) #[B, outC, H*, W*]
        out = self.conv2(out) #[B, outC, H*, W*]

        return out + residual
```

Transformer block

```
class TransformerBlock(nn.Module):
    '''
    Input: [B, C, H, W]
    Output: [B, C, H, W]
    '''
    def __init__(self, dim, num_heads, dim_head, context_dim=None):
        super().__init__()

        #Self Attention
        self.attn1 = Attention(query_dim = dim, heads = num_heads, dim_head = dim_head)

        #Cross Attention
        self.attn2 = Attention(query_dim=dim, context_dim=context_dim, heads=num_heads,
dim_head=dim_head)  # is self-attn if context is none

        #FFN
        self.ff = FeedForwardNetwork(dim)

        self.norm1 = nn.LayerNorm(dim)
        self.norm2 = nn.LayerNorm(dim)
        self.norm3 = nn.LayerNorm(dim)

    def forward(self, x, context=None):
        x = self.norm1(self.attn1(x)) + x
        x = self.norm2(self.attn2(x, context=context)) + x
        x = self.norm3(self.ff(x)) + x
        return x
```

Decoding (Ancestral Sampling)

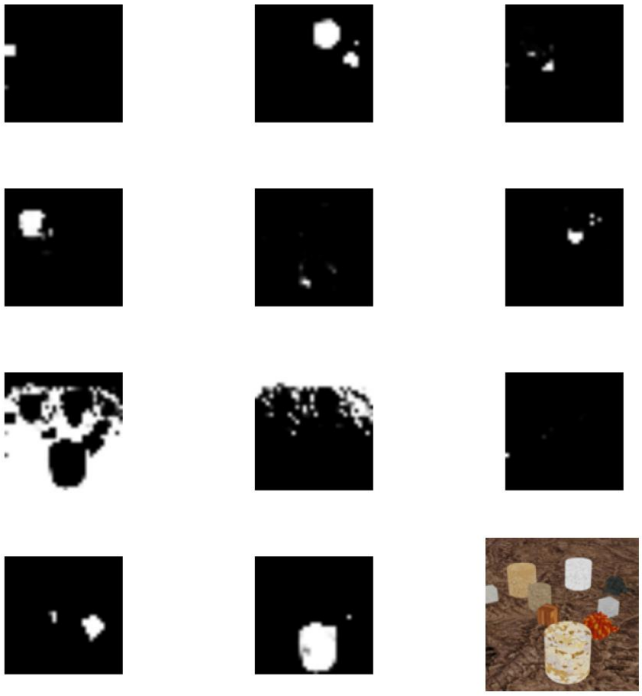
```
slots, attn_masks = encoder(image)
attn_masks = attn_masks.reshape((1,11,32,32))
rescaled_attn_masks = F.interpolate(attn_masks, size=(128,128), mode='bilinear', align_corners=False)

temp_out = torch.randn((1, 3, 32, 32)).to(device)

for ts in tqdm(reversed(range(0, 1000))):
    sinusoidal_timestep = SP(torch.tensor([ts])).to(device)

    unet_out = decoder(temp_out, sinusoidal_timestep, slots)

    temp_out = p_sample(unet_out, temp_out, torch.full((1,), ts, device=device, dtype=torch.long), ts,
betas, sqrt_one_minus_alphas_cumprod, sqrt_recip_alphas, posterior_variance)

reconstructed_image = vae.decode(temp_out).clamp(-1,1)
rimage = reconstructed_image.squeeze(dim = 0).permute(1,2,0).cpu().numpy()
rimage = (((rimage/2.0) + 0.5) * 255.0).astype(np.uint8)

store_all_plots_slotattn(rimage, rescaled_attn_masks, img_name, args.output_dir)
```

Hyperparameters

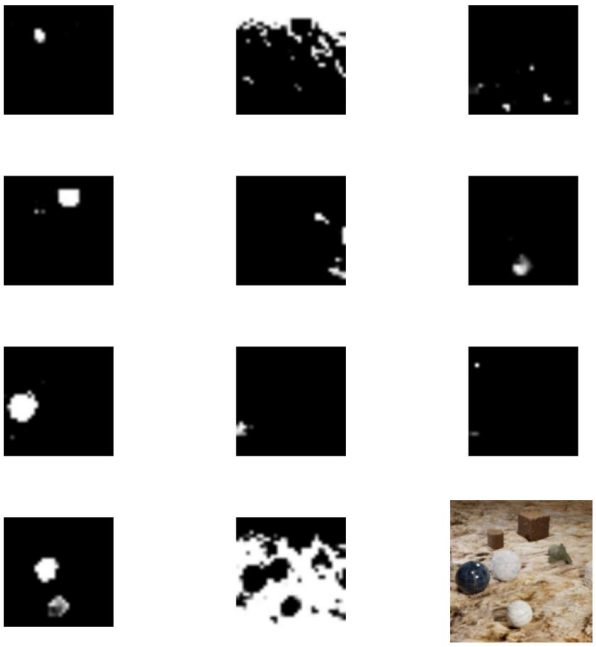
```
BATCH_SIZE = 64
LEARNING_RATE = 0.0002
EPOCHS = 300
SAVE_STEPS = 5
NUM_WORKERS = 16
SLOT SIZE = 192
```

Generated Masks

Reconstruction masks



Reconstruction masks



Reconstructed Images



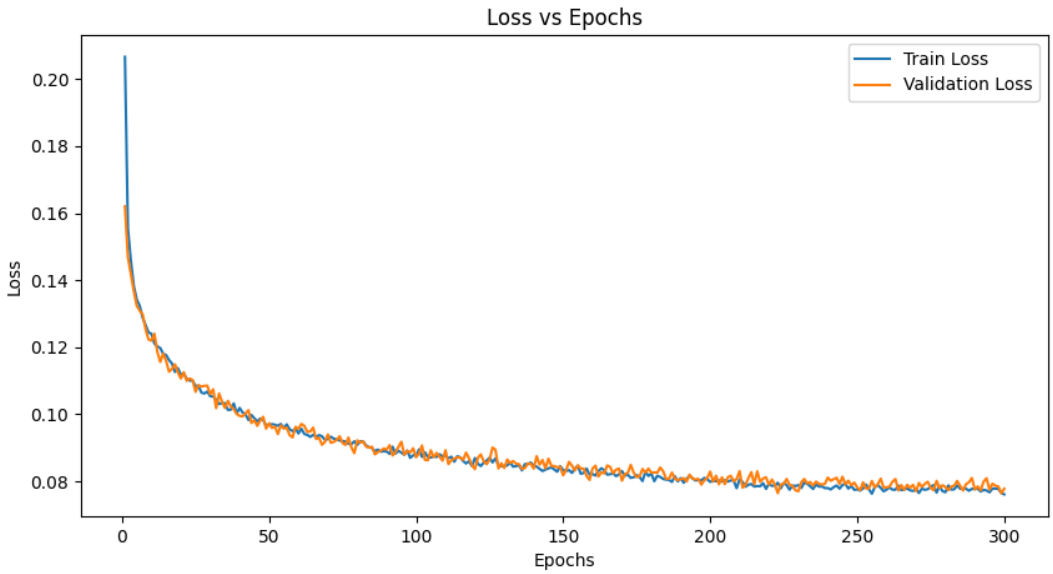



ARI Score on Validation Set:

```
(col775_a1) tkarthikeyan@wonderland:~/IIT_DELHI/COL775_Deep_Learning/Assignment_2b_new$ python eval.py
Device is  cuda
making 'vanilla' attention with 256 in_channels
Working with z of shape (1, 3, 32, 32) = 3072 dimensions.
making 'vanilla' attention with 256 in_channels
model loaded!
100%|██████████| 10000/10000 [10:20<00:00, 16.12it/s]
Validation ARI score is  0.3149918072068831
(col775_a1) tkarthikeyan@wonderland:~/IIT_DELHI/COL775_Deep_Learning/Assignment_2b_new$
```

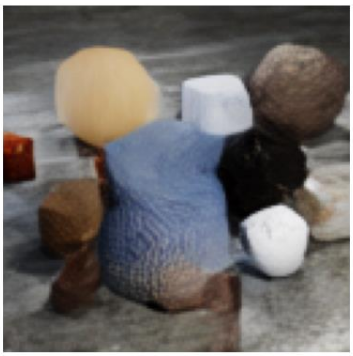
ARI Score: 0.3149918072068831

Loss Curve



final_train loss: 0.0760888967514038
final_val loss: 0.07777002657864504

Compositional Generation



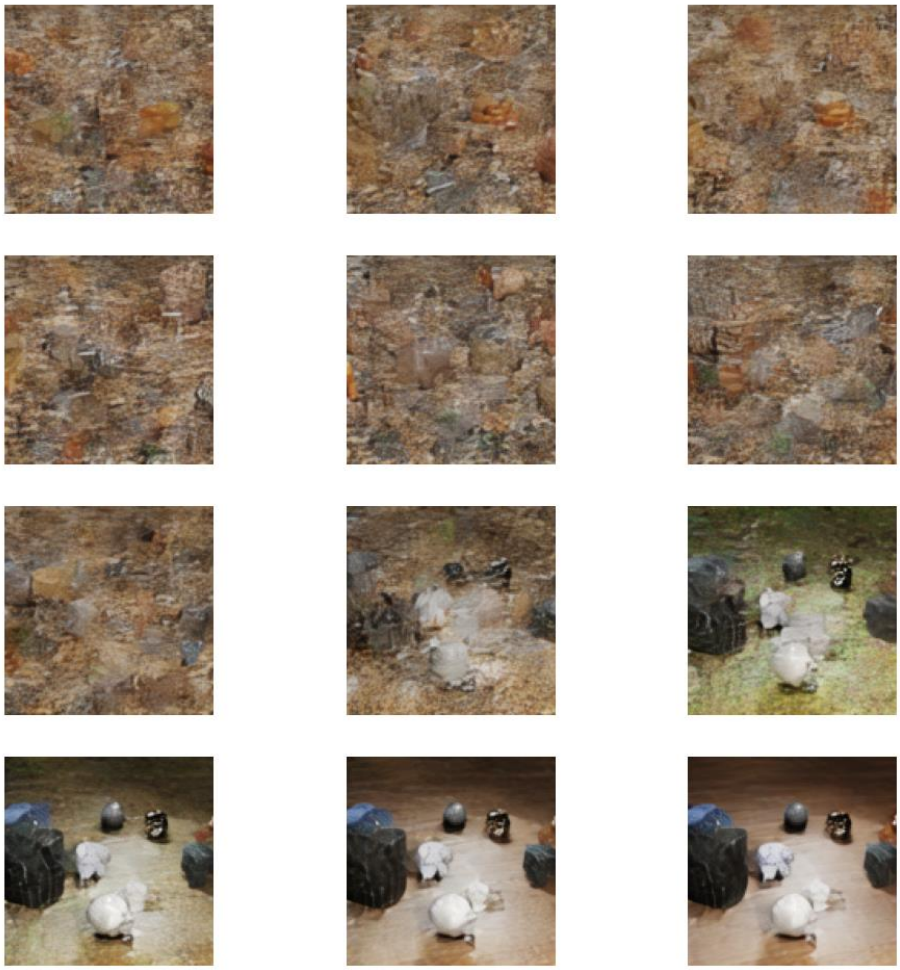
Clean FID metric:

```
(col775_a1) tkarthikeyan@wonderland:~/IIT_DELHI/COL775_Deep_Learning/Assignment_2b$ python
Python 3.10.13 (main, Sep 11 2023, 13:44:35) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from cleanfid import fid
>>> fdir1 = "/home/tkarthikeyan/IIT_DELHI/COL775_Deep_Learning/Assignment_2/dataset/images/val"
>>> fdir2 = "/home/tkarthikeyan/IIT_DELHI/COL775_Deep_Learning/Assignment_2b_new/outputs/batch64_300epochs/val_new"
>>> score = fid.compute_fid(fdir1, fdir2)
compute FID between two folders
FID val : 100%|████████████████████████████████████████████████████████████████████████| 313/313 [01:51<00:00,  2.82it/s]
FID val_new : 100%|████████████████████████████████████████████████████████████████████| 120/120 [00:24<00:00,  4.94it/s]
>>> score
129.53712739559978
>>>
```

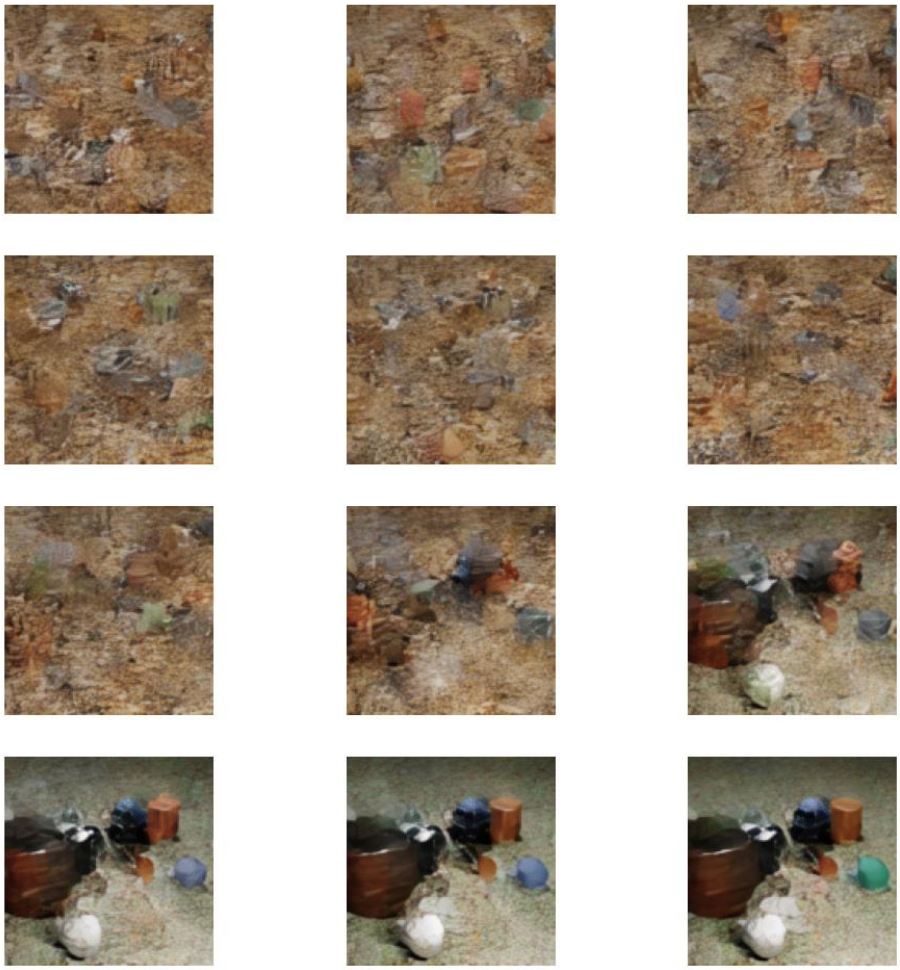
Clean FID score: 129.53712739559978

Step by Step Images:

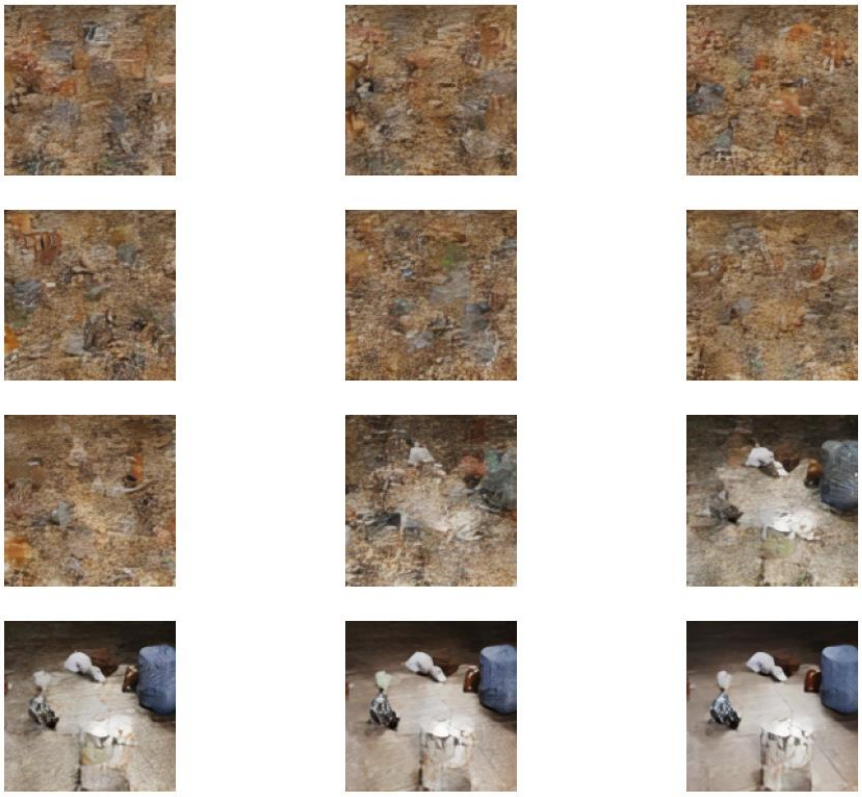
Diffusion process over the timesteps (Read top left to bottom right)



Diffusion process over the timesteps (Read top left to bottom right)



Diffusion process over the timesteps (Read top left to bottom right)



Diffusion process over the timesteps (Read top left to bottom right)