DL Assignment 1 Part1

Utkarsh Sharma

March 2024

https://drive.google.com/drive/folders/1ITzFKa_MG8Id8U2V50Th2xPKKhzv2peo?usp=drive_link

1 Image Classification using ResNet Architecture

Implementation of ResNet Architecture: We will construct a ResNet architecture from scratch in PyTorch. The total number of layers in the network will be determined by the formula 6n+2. This includes the initial convolutional layer processing input images of size 256×256, followed by n layers with feature map size 256×256, another set of n layers with feature map size 128×128, and finally n layers with feature map size 64×64. The fully connected output layer will consist of r units, where r represents the number of classes. The number of filters in the three sets of n hidden layers after the first convolutional layer will be 16, 32, and 64, respectively. Residual connections will be established between each block of 2 layers, except for the first convolutional layer and the output layer. Convolutions will employ a filter size of 3×3, inspired by the VGG net architecture [10]. Down-sampling will be achieved using convolutional layers with a stride of 2, and appropriate zero padding will be applied to maintain the feature map size. The final hidden layer will perform a mean pooling operation over all features before feeding into the output layer. Further details about the architecture can be found in Section 4.2 of the ResNet paper.

Training Procedure: We will train a ResNet architecture with n = 2 on the provided dataset. The training will be conducted for 50 epochs using a batch size of 32. The number of classes (r) is set to 25. We will employ the SGD optimizer with an initial learning rate of 10^{-4}. Learning rate scheduling or decay will be applied as deemed appropriate. Additionally, we may experiment with alternative optimizers apart from SGD.

Dataset Description: The image dataset used was `Indian-Birds-Dataset` with 25 classifications. Train data has 30,000 images while validation has 7500 images each image is of different resolution.

1.1 Implementation Details

- **ResNet Architecture:**
  - The ResNet architecture is implemented following the guidelines presented in the original ResNet paper [1].
  - The architecture consists of convolutional layers, residual blocks, batch normalization layers, ReLU activation functions, adaptive average pooling, and fully connected layers.
  - The ResNetBlock class defines a residual block with optional downsampling using convolutional layers. It incorporates batch normalization and ReLU activation functions.
  - The ResNet class defines the overall ResNet model, which consists of multiple residual blocks stacked together, followed by adaptive average pooling and fully connected layers.
- **Data Preprocessing:**
  - The torchvision.transforms.Compose function is used to define a sequence of transformations applied to the input images. These transformations include resizing the images to (256, 256) dimensions, converting them to PyTorch tensors, and normalizing the pixel values.
  - The datasets.ImageFolder class from torchvision is utilized to load the image dataset. It automatically assumes the directory structure where each subdirectory represents a class and loads images accordingly.
- **Data Loaders:**
  - The torch.utils.data.DataLoader class is employed to create data loaders for the training, testing, and validation datasets. These data loaders enable efficient batching, shuffling, and parallel data loading using multiple workers.
- **Model Training:**
  - The ResNet18 function initializes the ResNet model instance with the specified values of n (number of layers) and r (number of classes).
  - The model is trained using the training dataset with a batch size of 32 and for 50 epochs. It employs the Stochastic Gradient Descent (SGD) optimizer with an initial learning rate of 10^{-4}.
  - Learning rate scheduling or decay is applied as appropriate to optimize the training process.
- **Approaches:**

– It adopts a modular approach by encapsulating the residual block and the entire ResNet architecture within separate classes, enhancing code readability and maintainability.

– The implementation follows the guidelines and architectural principles outlined in the ResNet paper [1] to ensure compatibility and adherence to the original design.

1.1.1 Results

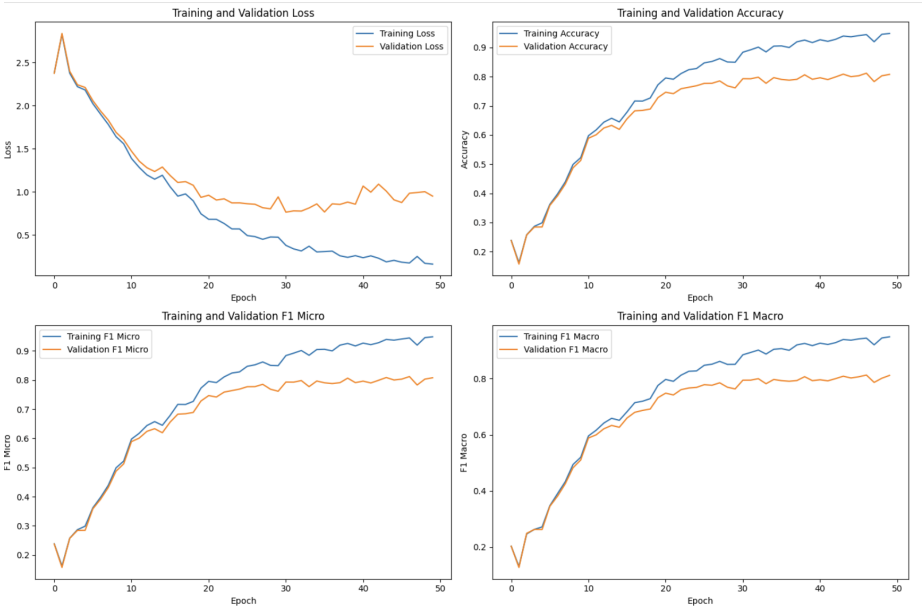

Figure 1: Enter Caption

Table 1: BatchNorm (default) Results

Metric	**Performance**		
	Train	**Validation**	**Test**
Accuracy	0.7962	0.7844	0.7832
Micro F1 Score	0.7643	0.7584	0.7322
Macro F1 Score	0.7671	0.7586	0.7362

1.2 Impact of Normalization

In this part, we implement from scratch the following Normalisation schemes as a subclass of torch.nn.Module -

1. (a) Batch Normalization (BN)
2. (b) Instance Normalization (IN)
3. (c) Batch-Instance Normalization (BIN)
4. (d) Layer Normalization (LN)
5. (e) Group Normalization (GN)
6. (f) No Normalisation (NN)

1.2.1 BN

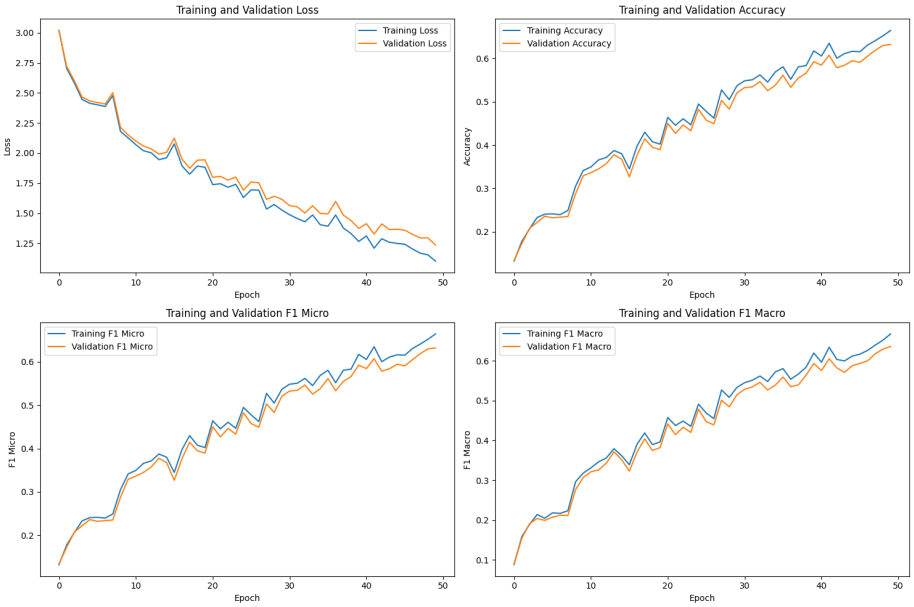

Table 2: BatchNorm Results

Metric	**Performance**		
	Train	**Validation**	**Test**
Accuracy	0.6646	0.6574	0.6323
Micro F1 Score	0.6646	0.6574	0.6323
Macro F1 Score	0.6676	0.6597	0.6361

Observation (Sanity Check): We can see that our implementation of BatchNorm performs similar to the default BatchNorm usage. (Within 4%)

1.2.2 IN

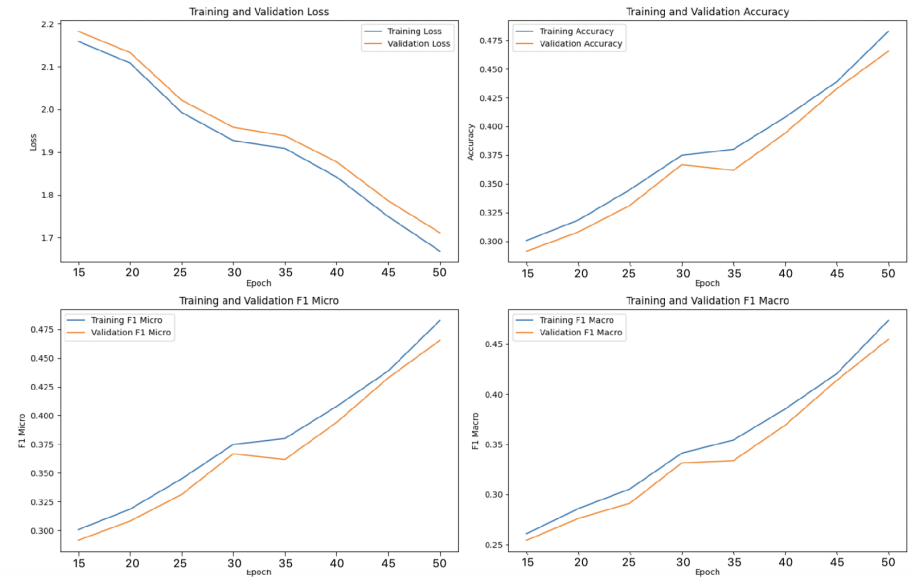

Figure 2: Instance Norm Metrics

Table 3: Statistics of Instance Normalization (IN) Model

Metric	**Performance**		
	Train	**Validation**	**Test**
Accuracy	0.6828	0.6884	0.6655
Micro F1 Score	0.6828	0.6884	0.6655
Macro F1 Score	0.6735	0.6770	0.6546

1.2.3 BIN

For Batch-Instance-Normalisation, we only ran 32 epochs since we got good accuracy within only 32 epochs.

Figure 3: Batch Instance Norm Results

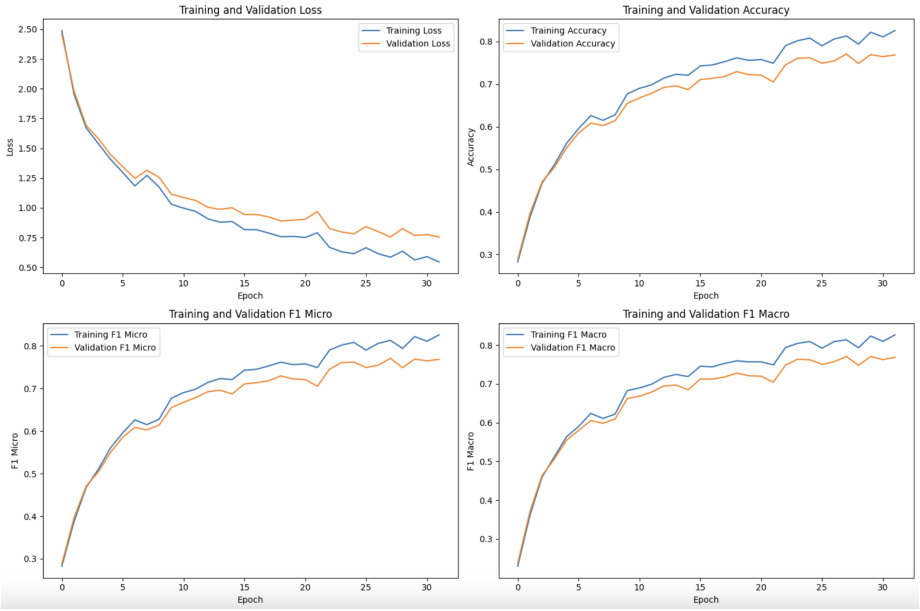

Figure 4: Results vs Epochs

Table 4: Results of Batch Instance Norm

Metric	Train	Test	Validation
Accuracy	0.826	0.769	0.826
Micro F1 Score	0.826	0.769	0.826
Macro F1 Score	0.827	0.769	0.827

1.2.4 LN

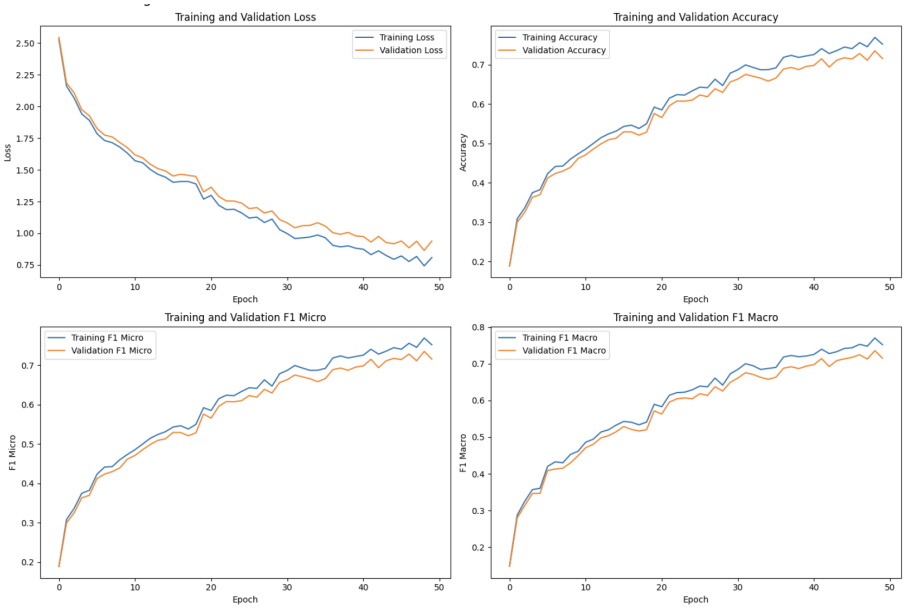

Figure 5: Layer Normalization Diagram

Table 5: Layer Normalization Results

Metric	**Performance**		
	Train	**Validation**	**Test**
Accuracy	0.7521	0.7528	0.7157
Micro F1 Score	0.7521	0.7528	0.7157
Macro F1 Score	0.7521	0.7535	0.7152

1.2.5 GN

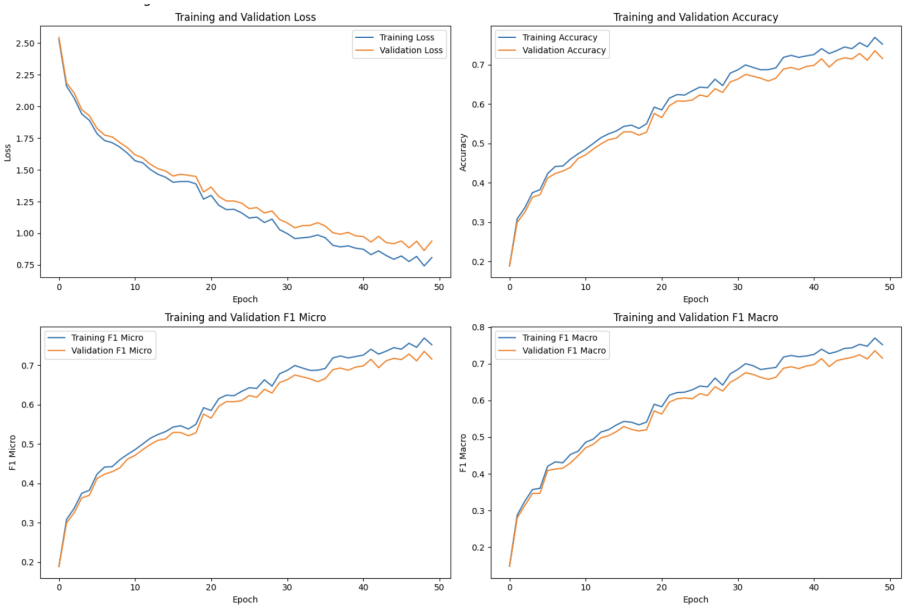

Figure 6: Group Normalization Results

Table 6: GroupNorm Results

Metric	**Train**	**Validation**	**Test**
Accuracy	0.6597	0.6654	0.6403
Micro F1 Score	0.6597	0.6654	0.6403
Macro F1 Score	0.6533	0.6598	0.6325

1.2.6 NN

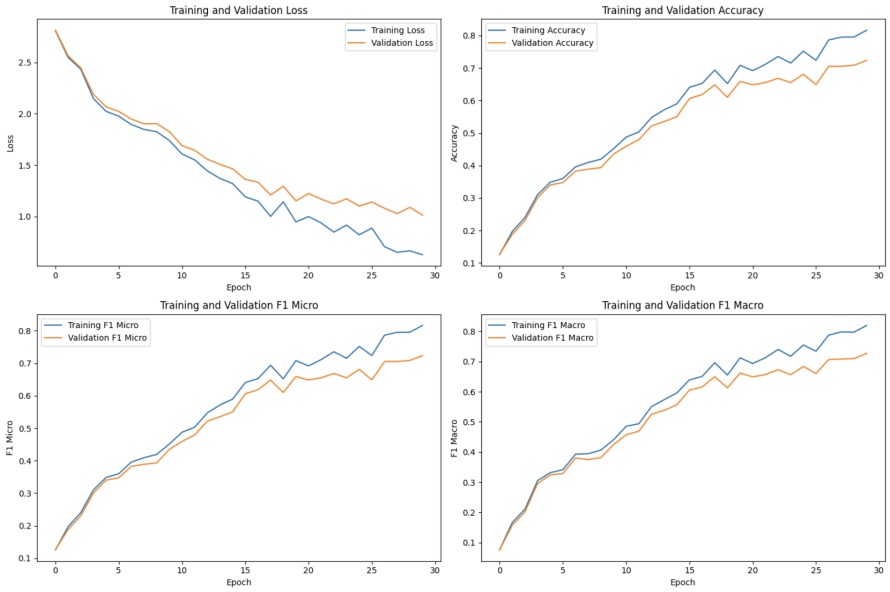

Figure 7: Enter Caption

Observation: We can see that No Normalisation gives us the poorest result.

Table 7: No Normalization Results

Metric			
	Train	**Validation**	**Test**
Accuracy	0.6675	0.5683	0.5676
Micro F1 Score	0.5975	0.5983	0.5976
Macro F1 Score	0.5868	0.5866	0.5873

1.3 Optimisers comparison

Experimentation with SGD tells us that **0.9 momentum** with learning rate $2e^{-3}$ performs the best.
Further, the best performing optimiser was Adam which was used for the final training of all the models.

1.4 Comparison

We observe that **Batch Instance Norm** performed the best under uniform optimiser conditions, achieving a low loss comparable to others in only under

20 epochs. Further, it also had the best accuracy.
Next in performance were Layer Normalisation and Group Normalisation.

Table 8: Comparison of Model Accuracies

Model	**Accuracy**		
	Train	**Validation**	**Test**
BN	0.7962	0.7844	0.7832
IN	0.6828	0.6884	0.6655
NN	0.6675	0.5683	0.5676
BIN	0.8260	0.7604	0.8262
GN	0.6597	0.6654	0.6403
LN	0.7521	0.7528	0.7157

1.5 Variation of GroupNorm batchsizes

Below, we discuss how the batch sizes affect BN vs GN

1.5.1 Group Norm Batch size 8

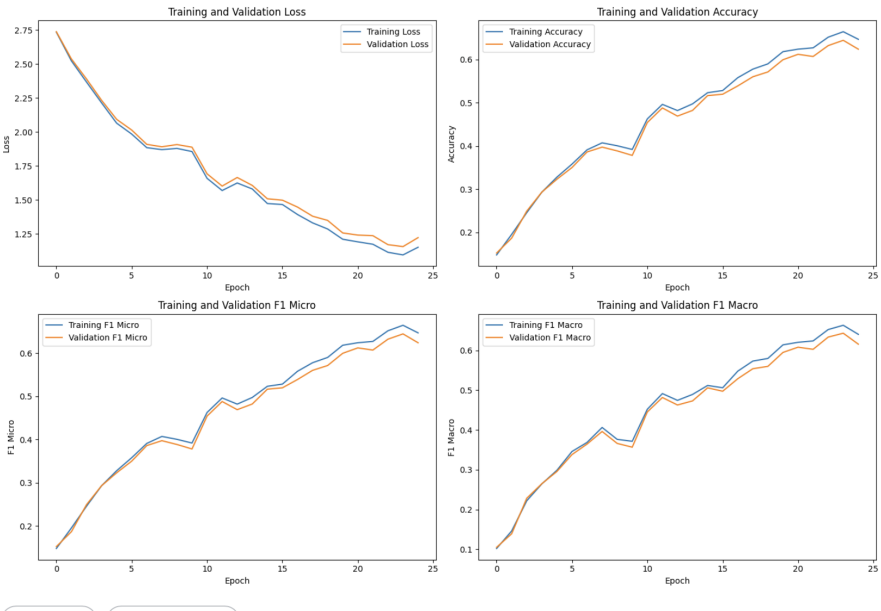

Figure 8: Losses and Accuracies for Group Norm with Batch Size 8

Table 9: Statistics of GroupNorm Model with Batch Size 8

Metric	**Performance**		
	Train	**Validation**	**Test**
Accuracy	0.6468	0.6528	0.6239
Micro F1 Score	0.6468	0.6528	0.6239
Macro F1 Score	0.6405	0.6470	0.6158

1.5.2 GroupNorm BatchSize 128

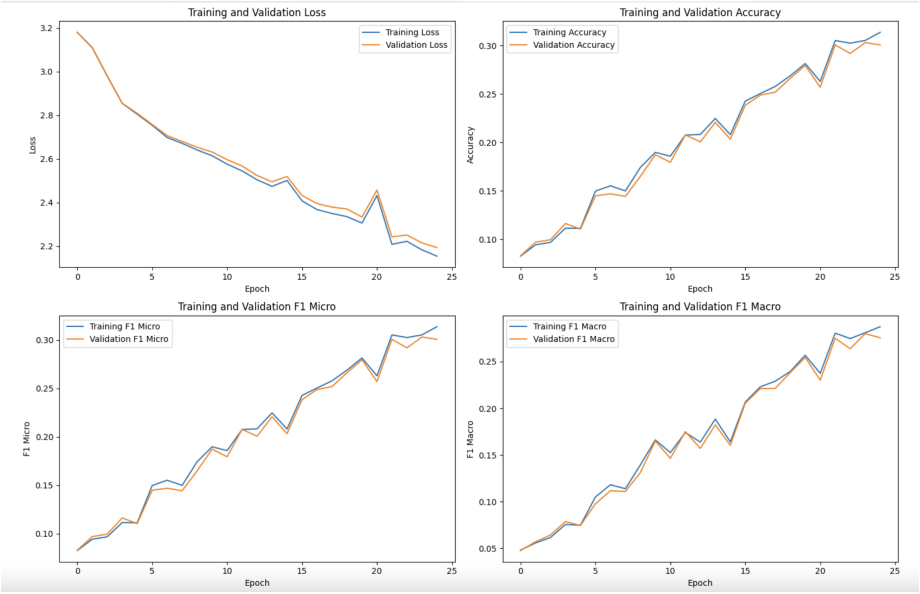

Figure 9: Results for Group Norm with Batch Size=128

Note that these results were taken for the same number of epochs as were used for Batch size of 8. Thus, it has a relatively lower accuracy.

Group Norm however **performs good mostly regardless of what batch size is used**. (Good accuracies for batch sizes 8, 32 and 128 all.)

Table 10: GroupNorm Results (Batch Size = 128)

Metric			
	Train	**Validation**	**Test**
Accuracy	0.5138	0.5132	0.5008
Micro F1 Score	0.5138	0.5132	0.5008
Macro F1 Score	0.5874	0.5879	0.5756

1.5.3 BatchNorm Batch Size 8

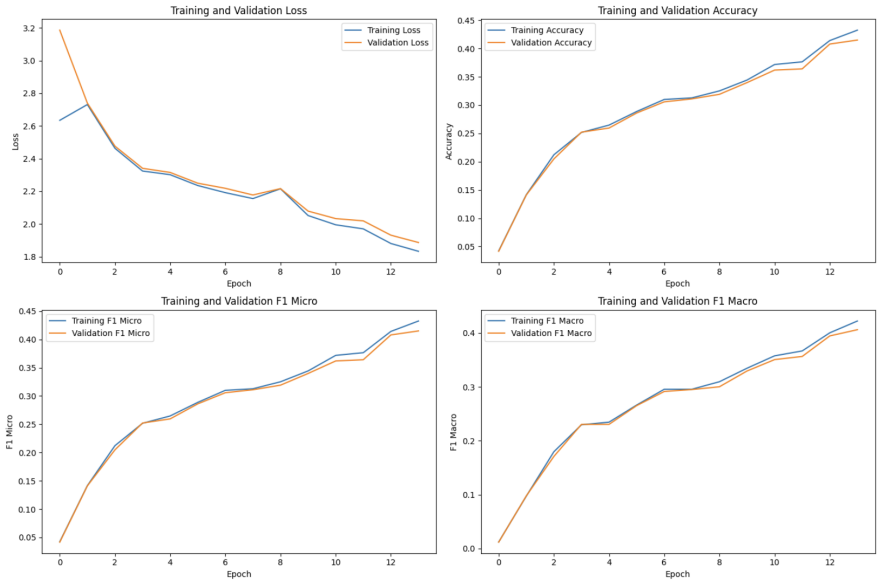

Figure 10: Enter Caption

Table 11: Batch Normalization with Batch Size 8

Metric			
Accuracy	0.4338	0.4337	0.4159
Micro F1 Score	0.4338	0.4337	0.4159
Macro F1 Score	0.4313	0.4292	0.4142

1.5.4 Batch Norm Batch Size 128

Trained on 35 epochs only to compare with Batch Norm Batch Size 8 that was also trained on 35 epochs (since training it takes a large amount of time).

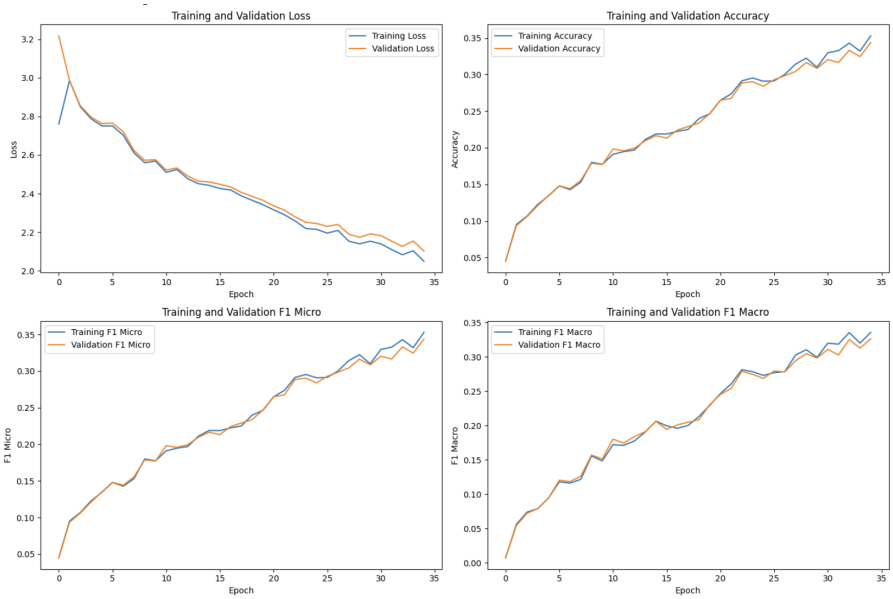

Figure 11: Enter Caption

Table 12: Batch Norm Results (Batch Size: 128)

Metric			
	Train	**Validation**	**Test**
Accuracy	0.3529	0.3593	0.3437
Micro F1 Score	0.3529	0.3593	0.3437
Macro F1 Score	0.3359	0.3424	0.3263

1.5.5 Conclusion

We find that Batch Normalization (BN) performs well at **medium** (32) and **large** batch sizes (128).

1.6 Visualising Features

Figure 12: correct cattle Eggrett

Figure 13: Incorrect Cattle Eggrett

Observations -

1. It identifies the Cattle Eggrett mainly by its white body
2. The incorrect classifications look at the white background at wrong places

Figure 14: Red-Wattled-Lapwing correct

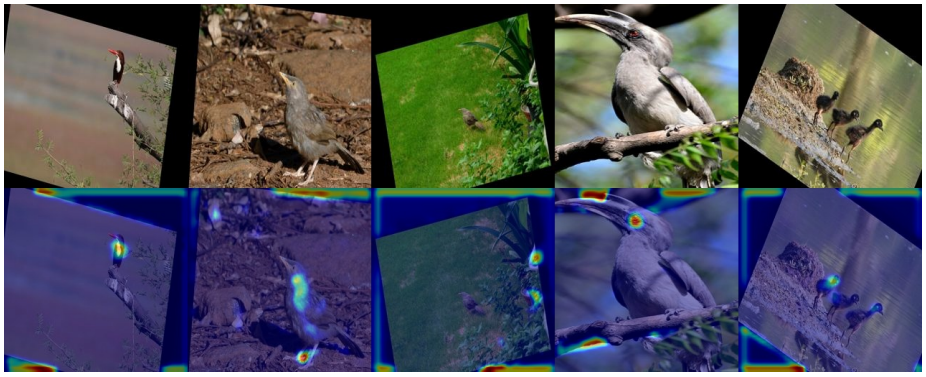

Figure 15: Red-Wattled-Lapwing incorrect

Observations -

1. It identifies the bird from its characteristic neck-head part
2. The incorrect classifications look at similar shapes translated or rotated in different places of the wrong birds

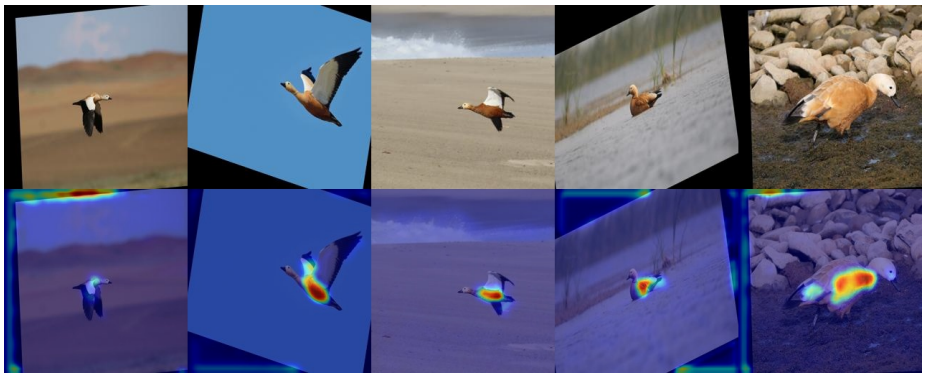

Figure 16: Ruddy-Shelduck-correct.jpg

Figure 17: Ruddy-Shelduck-incorrect.jpg

Observations -

1. It identifies the Rudy Shelduck mainly by its body

Figure 18: White-Breasted-Kingfisher correct.jpg

Figure 19: White-Breasted-Kingfisher incorrect.jpg

Observations -

1. It identifies the White Breasted Kingfisher mainly by its beak
2. The incorrect classifications look at beaks of different birds that look similar

Figure 20: White-Breasted-Waterhen$_{c}orrect.jpg$

Figure 21: White-Breasted-Waterhen-incorrect.jpg

Figure 22: Indian Peacock correct

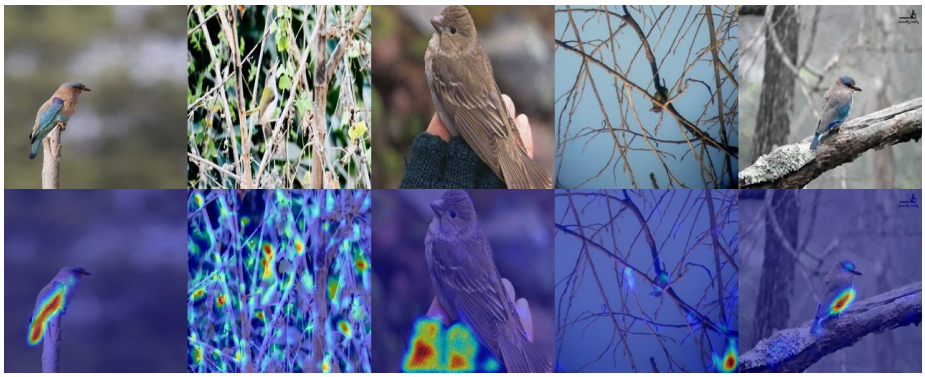

Figure 23: Indian Peacock incorrect

Observations -

1. It identifies the Indian Peacock from its long blue neck