ResNet Implementation and Normalization Impact Analysis

Anant Kumar

0.1 ResNet Architecture

The ResNet architecture consists of a series of convolutional layers followed by residual blocks. The input to the network is an RGB image with dimensions of 256x256 pixels. The first convolutional layer has 3 input channels and 16 output channels, with a kernel size of 3x3 and stride of 1. Batch normalization is applied after each convolutional layer to stabilize the training process.

The residual blocks are the building blocks of ResNet. Each residual block consists of two convolutional layers, each followed by batch normalization and ReLU activation. The shortcut connection bypasses these convolutional layers and directly adds the input to the output of the second convolutional layer. This helps in overcoming the vanishing gradient problem during training.

The network architecture is divided into three stages. The first stage contains n residual blocks with 16 output channels, maintaining the input size of 256x256 pixels. The second stage contains n residual blocks with 32 output channels, downsampling the feature maps to 128x128 pixels using convolutional layers with a stride of 2. The third stage contains n residual blocks with 64 output channels, further downsampling the feature maps to 64x64 pixels.

After the third stage, global average pooling is applied to reduce the spatial dimensions of the feature maps to 1x1. Finally, a fully connected layer with r units, where r is the number of classes (25 in this case), produces the output logits.

0.2 Training and Evaluation

The model is trained using Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.0001. The model is trained for 50 epochs with early stopping based on validation accuracy. The training process involves iterating over the dataset in batches of 32, calculating the loss using CrossEntropyLoss, and updating the model parameters using backpropagation.

0.2.1 Adam Optimizer Details

For the experiments conducted with the Adam optimizer, a learning rate of 0.0001 and a batch size of 32 were used.

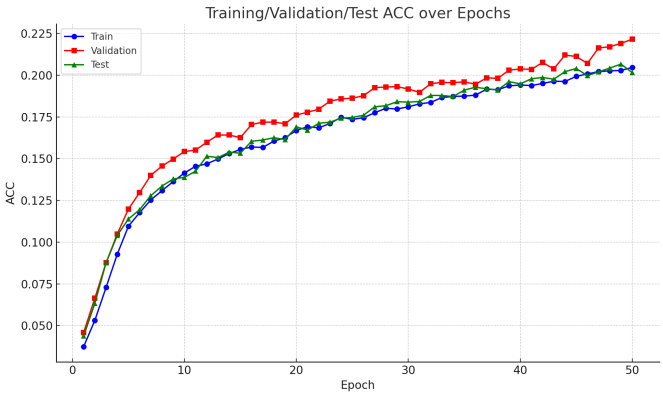


Figure 1: Accuracies over epochs

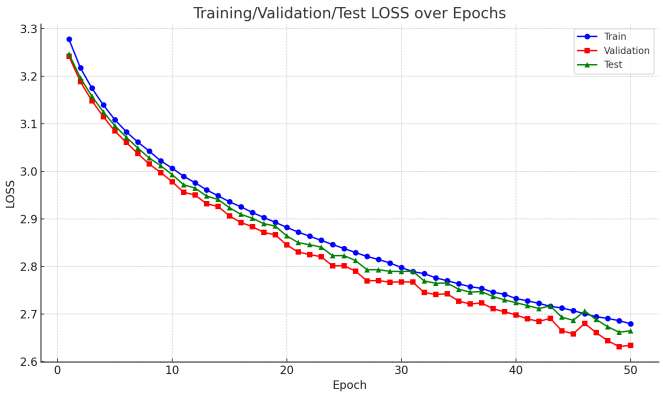


Figure 2: Losses over Epochs

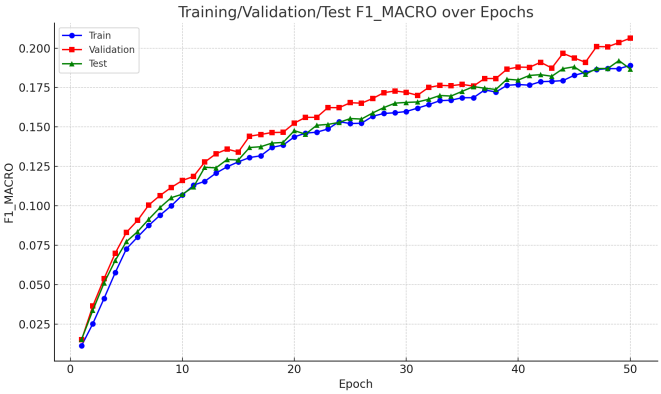


Figure 3: F1 macro over epochs

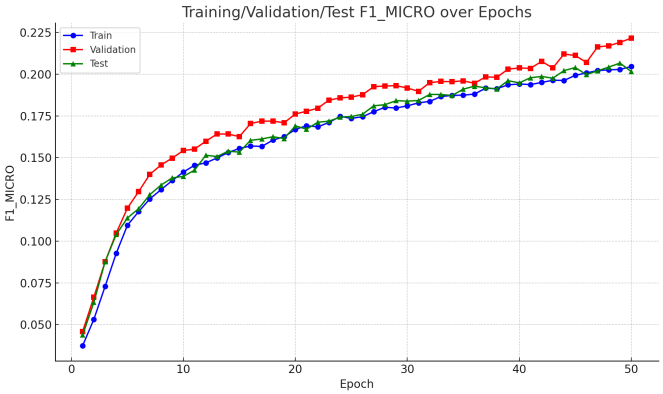


Figure 4: F1 micro over epochs

Table 1: Performance Metrics After Epoch 50

Metric	Train	Validation	Test
Accuracy	0.2046	0.2216	0.2016
Loss	2.6794	2.6342	2.6646
F1 Micro	0.2046	0.2216	0.2016
F1 Macro	0.1890	0.2063	0.1867

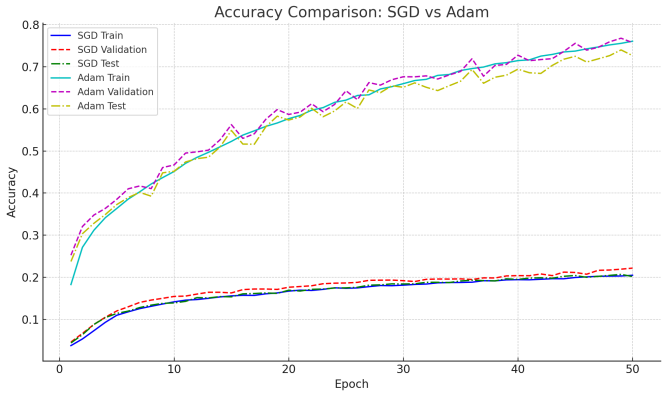


Figure 5: Accuracies over epochs (ADAM) vs (SGD)

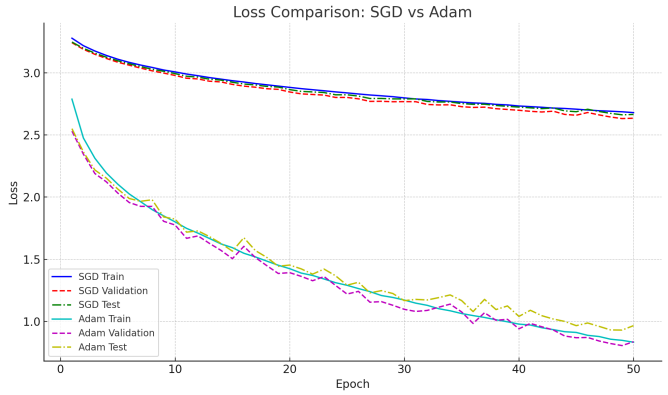


Figure 6: Losses over epochs (ADAM) vs (SGD)

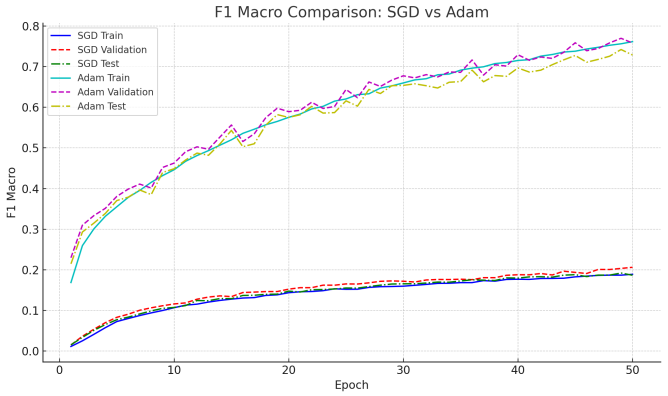


Figure 7: F1 macro over epochs (ADAM) vs (SGD)

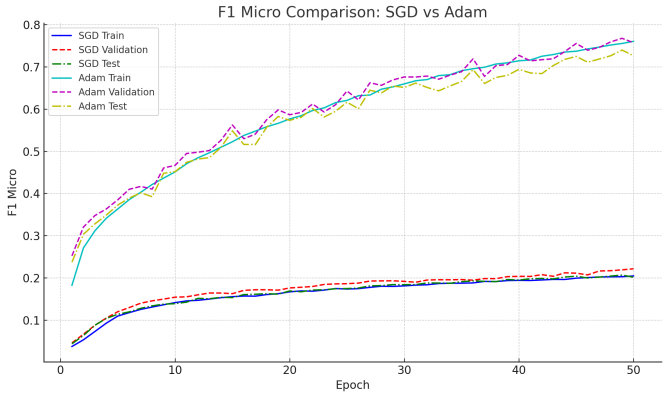


Figure 8: F1 micro over epochs (ADAM) vs (SGD)

Table 2: Performance Metrics Comparison after Epoch 50: SGD vs Adam

Metric	**Dataset**	**SGD**	**Adam**
Accuracy	Train	0.2046	0.7608
	Validation	0.2216	0.7575
	Test	0.2016	0.7267
Loss	Train	2.6794	0.8322
	Validation	2.6342	0.8364
	Test	2.6646	0.9657
F1 Micro	Train	0.2046	0.7608
	Validation	0.2216	0.7575
	Test	0.2016	0.7267
F1 Macro	Train	0.1890	0.7615
	Validation	0.2063	0.7582
	Test	0.1867	0.7286

1 Impact of Normalization

1.1 Batch Normalization (BN)

- Integrated after each convolutional layer as ‘nn.BatchNorm2d‘.
- Normalizes the output of convolutional layers across the batch dimension.
- Stabilizes learning by reducing internal covariate shift.

1.2 Instance Normalization (IN)

- Applied after convolutional layers as ‘MyInstanceNorm2d‘.
- Normalizes each channel in each data sample separately.
- Often used in style transfer models to normalize image contrast.

1.3 Batch-Instance Normalization (BIN)

- Combines the effects of BN and IN using ‘MyBatchInstanceNorm2d‘.
- Dynamically balances the batch and instance normalization benefits.
- Can improve generalization by leveraging both inter- and intra-batch information.

1.4 Layer Normalization (LN)

- Implemented as ‘MyLayerNorm‘, normalizing across the channel dimension.
- Useful for tasks where batch size is small or varies.
- Ensures consistency of normalization across varying batch sizes.

1.5 Group Normalization (GN)

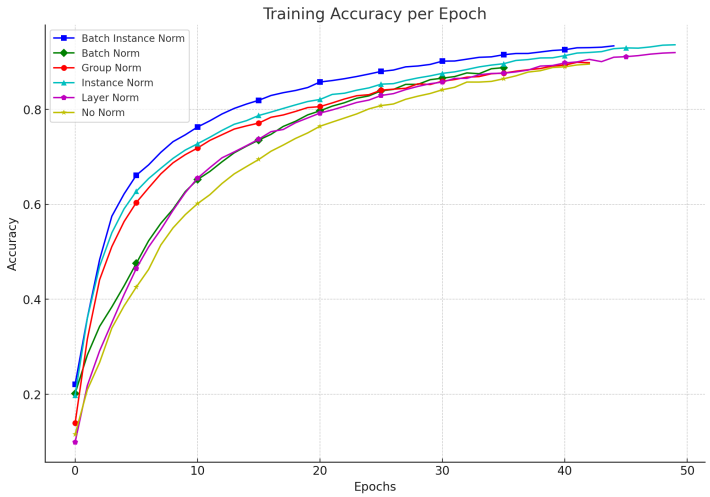
- Applied via ‘MyGroupNorm‘, splitting channels into groups for normalization.
- Independent of batch sizes, making it suitable for small batch scenarios.
- Balances computation between channel-based and batch-based normalization.

1.6 No Normalization (NN)

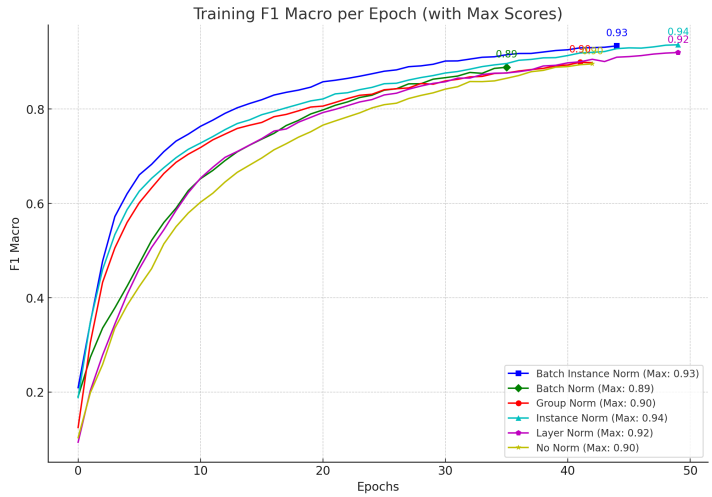
- Omission of normalization layers in the architecture.
- Provides a baseline to understand the impact of normalization on performance.
- Can lead to more challenging training dynamics and potential overfitting.

1.6.1 Training Metrics with Various Normalization Schemes

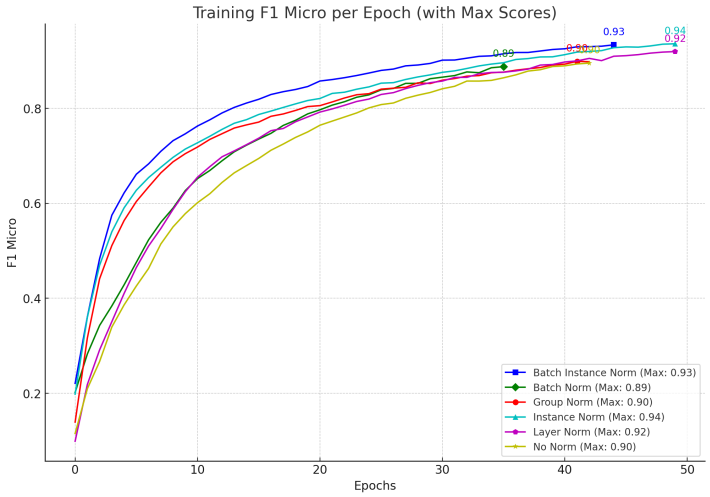
For the experiments conducted with various normalization schemes, Adam optimizer with a learning rate of 0.001 and a batch size of 32 was used.



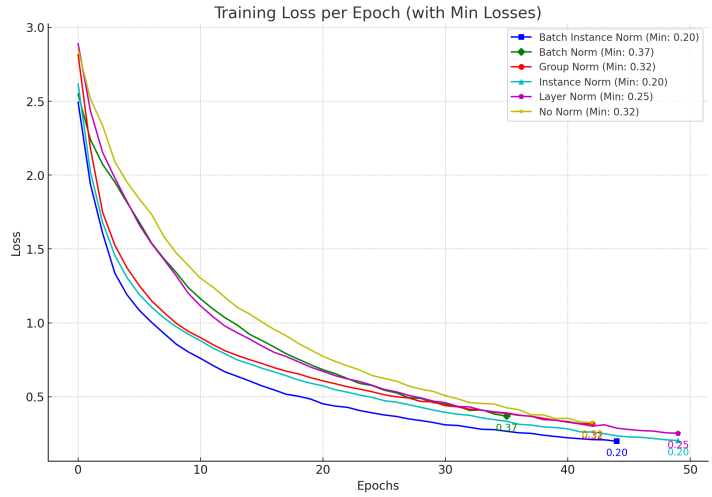
((a)) Training Accuracies



((b)) Training F1 Macro



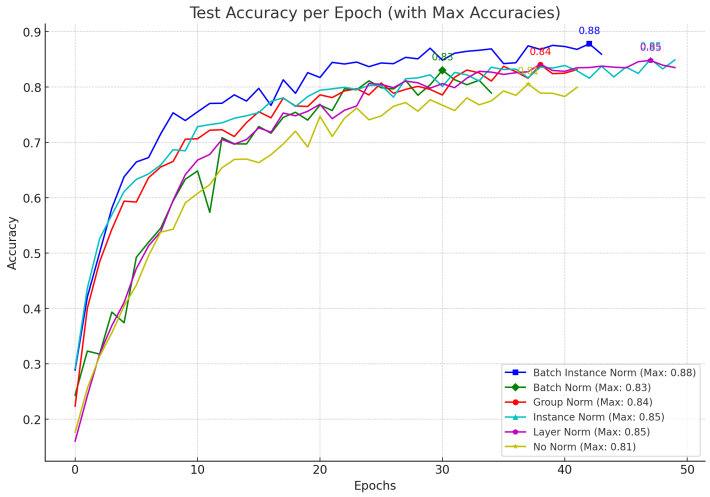
((c)) Training F1 Micro



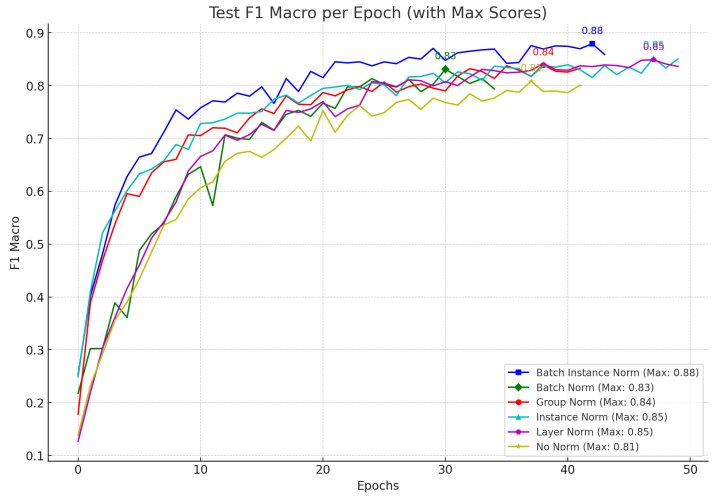
((d)) Training Loss

Figure 9: Training metrics for model performance.

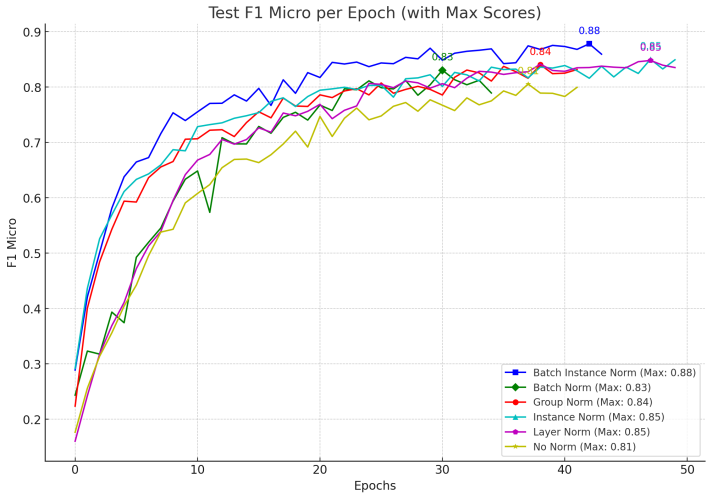
1.6.2 Testing Metrics



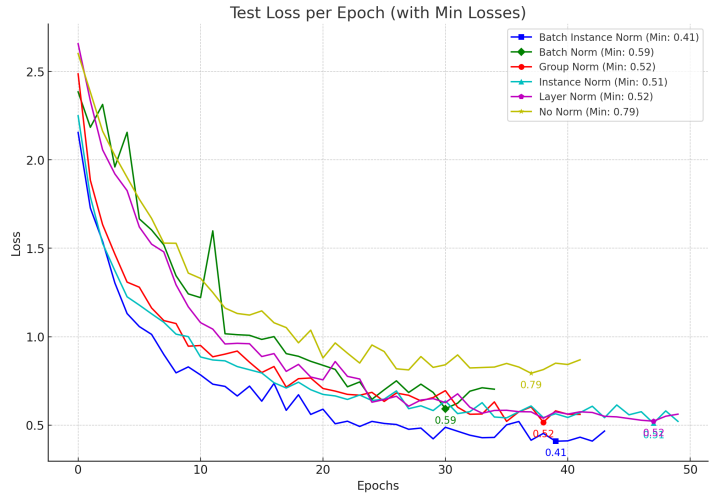
((a)) Test Accuracies



((b)) Test F1 Macro

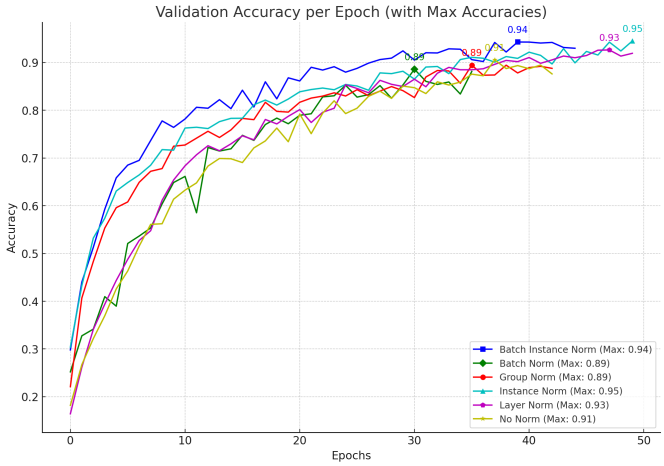


((c)) Test F1 Micro

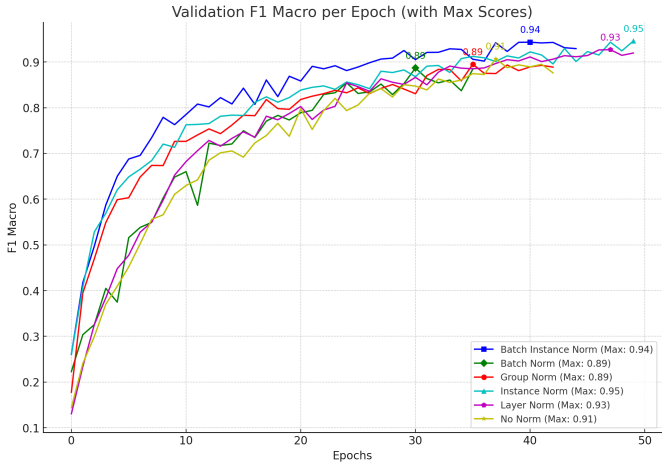


((d)) Test Loss

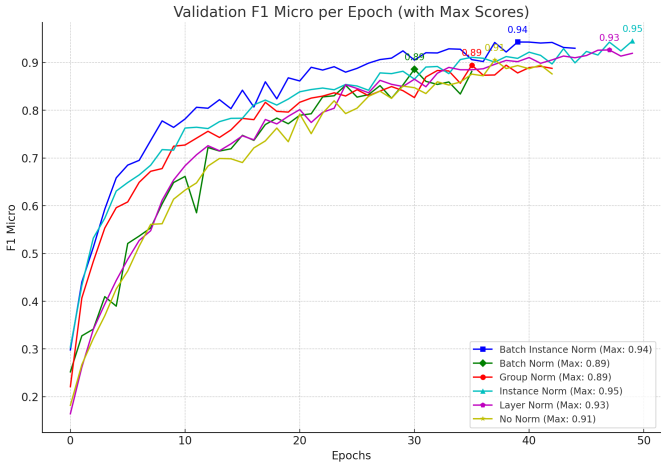
Figure 10: Testing metrics for model performance.



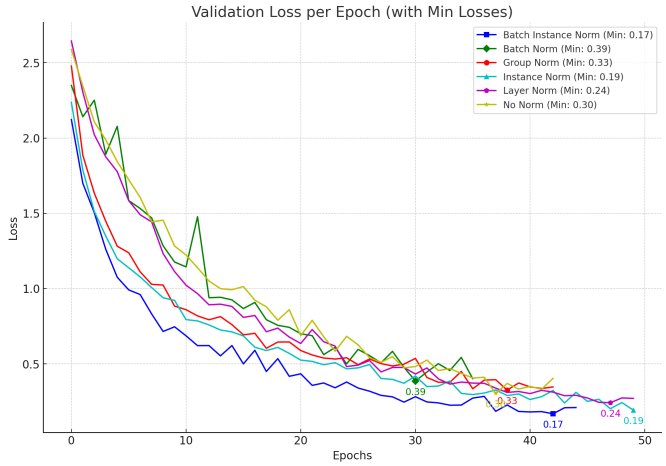
((a)) Validation Accuracies



((b)) Validation F1 Macro



((c)) Validation F1 Micro



((d)) Validation Loss

Figure 11: Validation performance metrics for all models across epochs.

Table 3: Summary of Performance Metrics for Different Normalization Methods

	Accuracy			F1 Macro			F1 Micro		
Normalization Method	Train	Test	Val	Train	Test	Val	Train	Test	Val
Batch Instance Norm	0.93	0.88	0.94	0.93	0.88	0.94	0.93	0.88	0.94
Batch Norm	0.89	0.83	0.89	0.89	0.83	0.89	0.89	0.83	0.89
Group Norm	0.90	0.84	0.89	0.90	0.84	0.89	0.90	0.84	0.89
Instance Norm	0.94	0.85	0.95	0.94	0.85	0.95	0.94	0.85	0.95
Layer Norm	0.92	0.85	0.93	0.92	0.85	0.93	0.92	0.85	0.93
No Norm	0.90	0.81	0.91	0.90	0.81	0.91	0.90	0.81	0.91

Table 4: Summary of Loss Metrics for Different Normalization Methods

Normalization Method	Training Loss	Testing Loss	Validation Loss
Batch Instance Norm	0.20	0.41	0.17
Batch Norm	0.37	0.59	0.39
Group Norm	0.32	0.52	0.33
Instance Norm	0.20	0.51	0.19
Layer Norm	0.25	0.52	0.24
No Norm	0.32	0.79	0.30

2 Impact of Batch Size

2.1 Training Setup

Batch size 128 vs Batch size 8

2.2 Training and Evaluation

Train the BN and GN variants of the model with Batch Size 8 and compare them with the same variants trained with Batch Size 128. Consider early stopping based on validation accuracy.

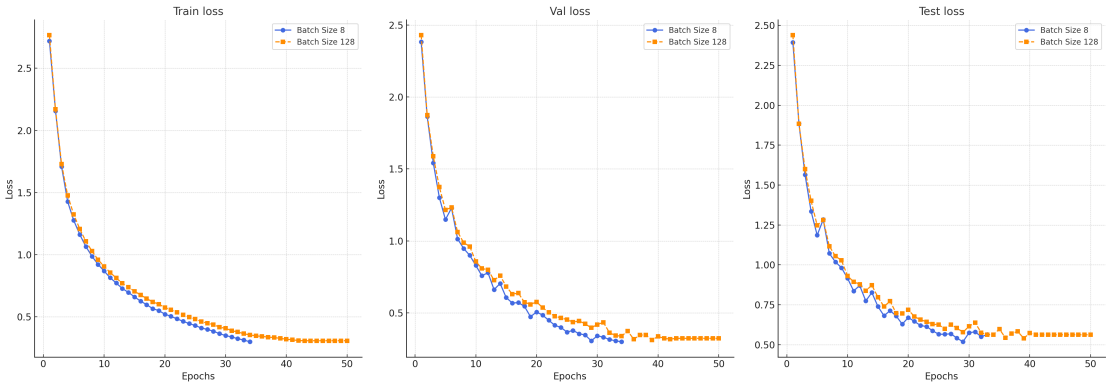


Figure 12: Comparison of Loss

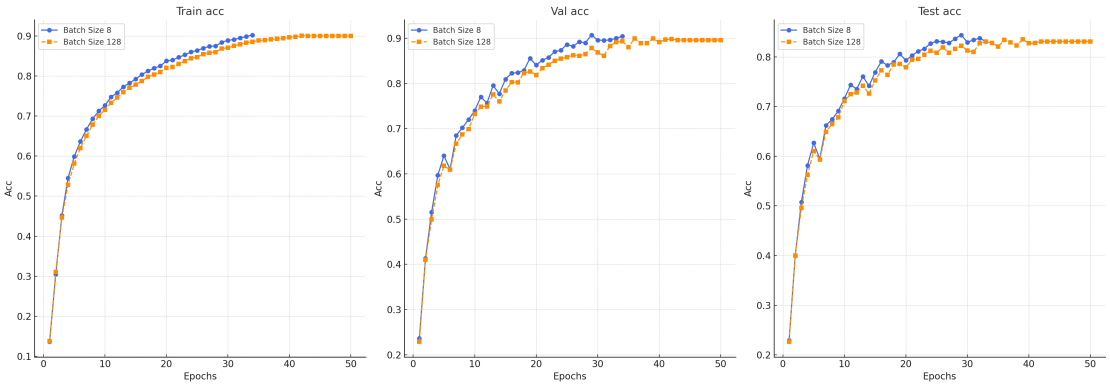


Figure 13: Comparison of Accuracy

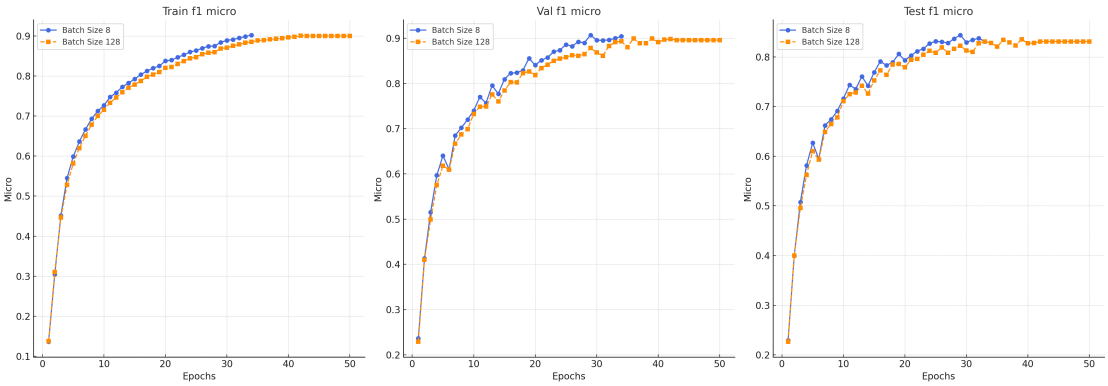


Figure 14: Comparison of F1 Micro

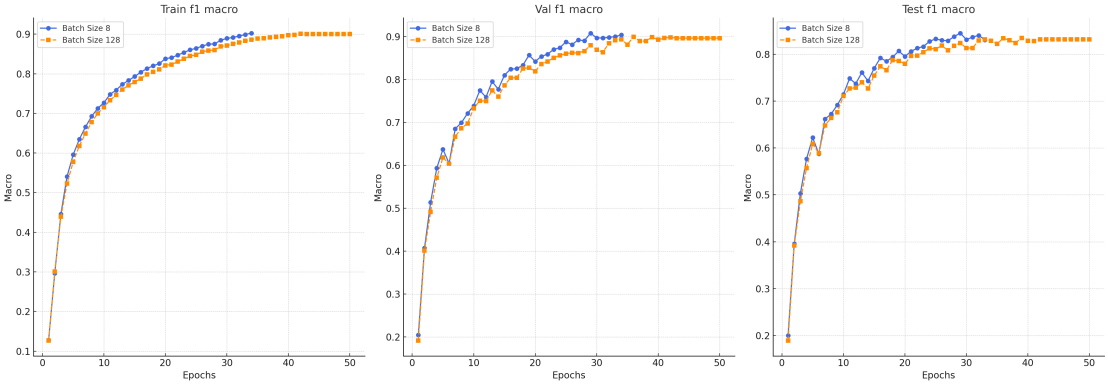


Figure 15: Comparison of F1 Macro

3 Adam vs SGD: A Comparative Analysis

The optimization algorithms, Adam and SGD, exhibit distinct characteristics and performance metrics when applied to deep learning models. My investigations reveal that:

- **Adam** significantly outperforms SGD in terms of convergence speed and handling sparse gradients. Through its adaptive learning rate mechanism, Adam facilitates faster and more stable training across various network architectures and datasets.
- **SGD**, while traditionally valued for its simplicity and effectiveness in large-scale settings, often requires more epochs to converge and is highly sensitive to the initial learning rate and schedule.

4 Impact of Normalization Techniques

Normalization techniques play a crucial role in stabilizing the training process and improving model performance. My analysis covers several normalization methods, including Batch Normalization (BN), Instance Normalization (IN), Layer Normalization (LN), and Group Normalization (GN), with the following insights:

- **Batch Normalization (BN)** proves effective in large-batch scenarios by normalizing the input features across the batch dimension, thus reducing internal covariate shift.
- **Instance Normalization (IN)** and **Layer Normalization (LN)** are particularly beneficial in tasks that require independent sample normalization, such as style transfer and generative models.
- **Group Normalization (GN)** stands out in its independence from batch size, offering consistent performance across varying batch sizes and proving especially advantageous in limited-resource settings or when employing highly variable batch sizes.

5 Influence of Batch Size on Group Normalization

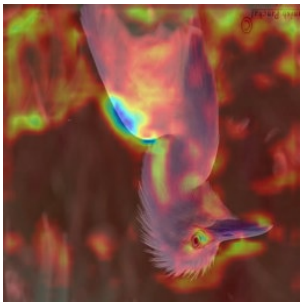
The impact of batch size on normalization techniques, especially GN, provides insightful revelations into efficient model training. Group Normalization, by design, offers robustness against the variability in batch size, making it an optimal choice for scenarios with constrained computational resources. My observations indicate that:

- GN maintains stable and high-performance metrics irrespective of the batch size, showcasing its adaptability and efficiency in diverse training conditions.
- The independence from batch size enables GN to provide consistent normalization benefits, thereby facilitating better generalization and performance across training and validation sets.
- Optimal configuration of group size within GN emerges as a critical factor for leveraging its full potential, necessitating empirical tuning based on the specific architecture and task at hand.

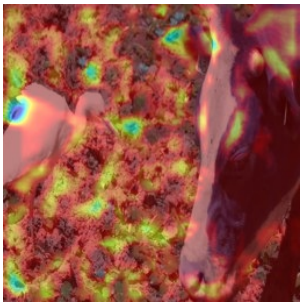
6 GRADCaM

- **Confirming Model Focus:** It confirms whether the model is focusing on the right areas of an image to make a decision. For example, in a bird species classification task, the highlighted areas should ideally be on the bird and particularly on the most distinctive features of that species.
- **Model Debugging:** If the model is not performing well, Grad-CAM can help diagnose whether the model is looking at the right features. If the heatmaps are highlighting irrelevant parts of an image, this could explain poor performance and suggest that further training or data cleaning is needed.
- **Understanding Model Behavior:** Grad-CAM allows us to gain insight into the model's behavior, even if we don't have a deep understanding of the inner workings of the model. By seeing what the model sees, we can build an intuition about why it makes certain decisions.

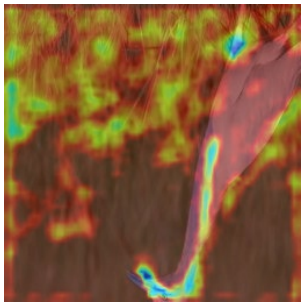
((a)) Image 1



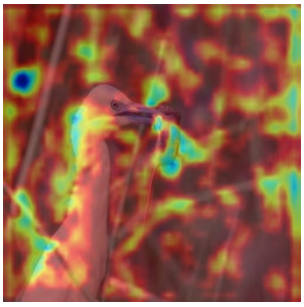
((b)) Image 2



((c)) Image 3



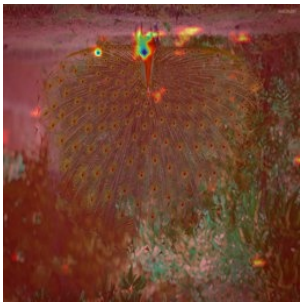
((d)) Image 4



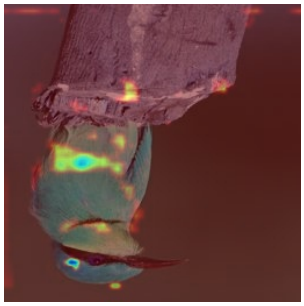
((e)) Image 5

((f)) Image 6

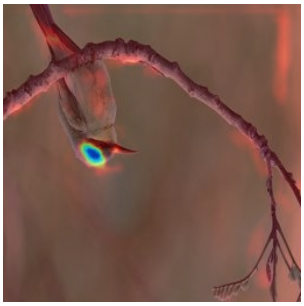
((g)) Image 7



((h)) Image 8



((i)) Image 9

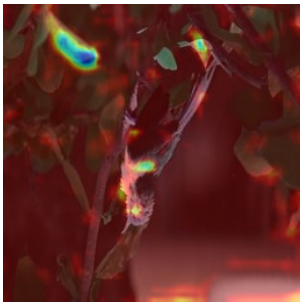


((j)) Image 10

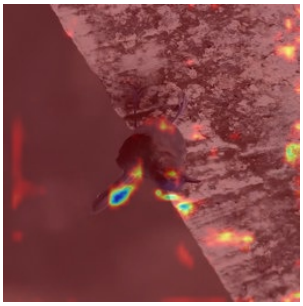
Correct Class Cattle-Egret

Incorrect Class

((a)) Image 1

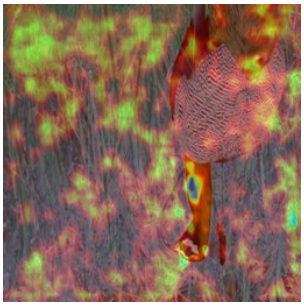


((b)) Image 2

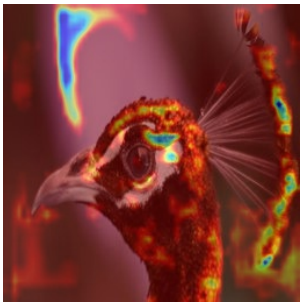


((c)) Image 3

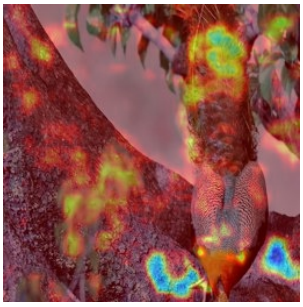
((d)) Image 4



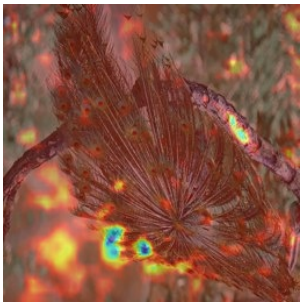
((e)) Image 5



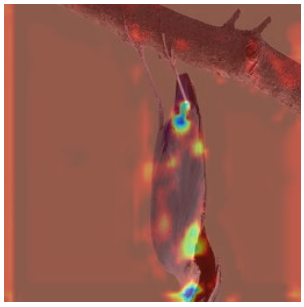
((f)) Image 6



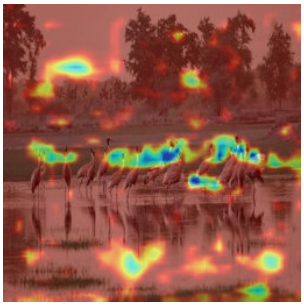
((g)) Image 7



((h)) Image 8



((i)) Image 9



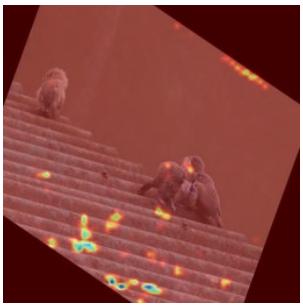
((j)) Image 10

Correct Class Peacock

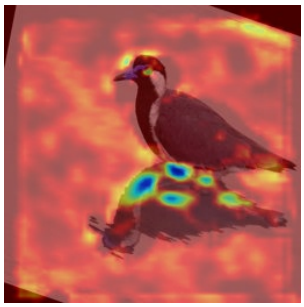
Incorrect Class

((a)) Image 1

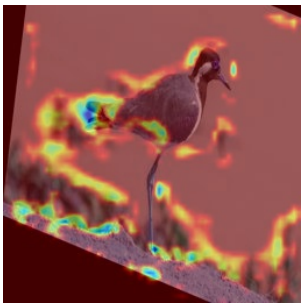
((b)) Image 2



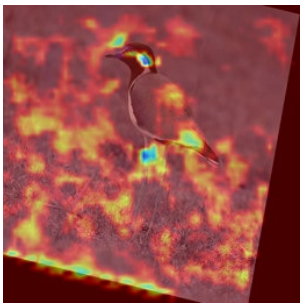
((c)) Image 3



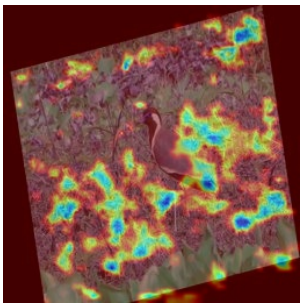
((d)) Image 4



((e)) Image 5

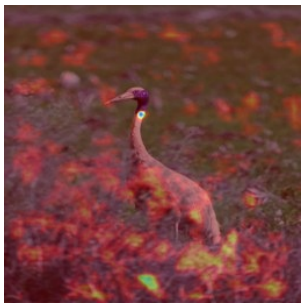


((f)) Image 6



((g)) Image 7

((h)) Image 8



((i)) Image 9

Correct Class Red Wattled Lapwing

Incorrect Class

((a)) Image 1

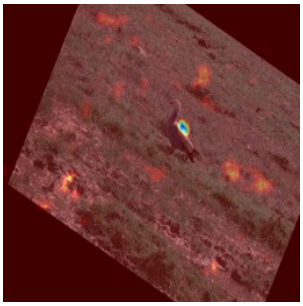
((b)) Image 2

((c)) Image 3

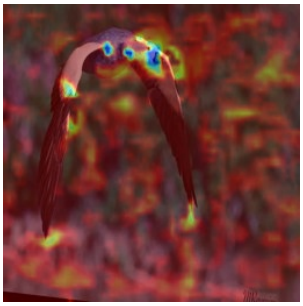
((d)) Image 4

((e)) Image 5

((f)) Image 6



((g)) Image 7



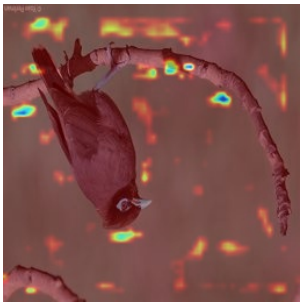
((h)) Image 8

((i)) Image 9

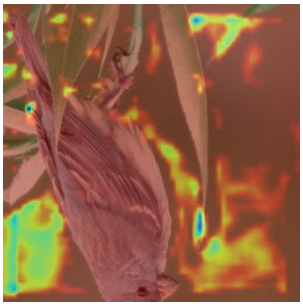
((j)) Image 10

Correct Class Rudy Shelduck

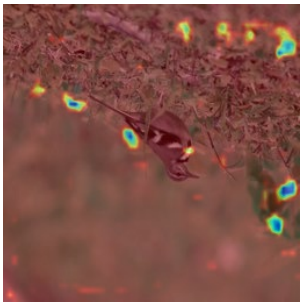
Incorrect Class



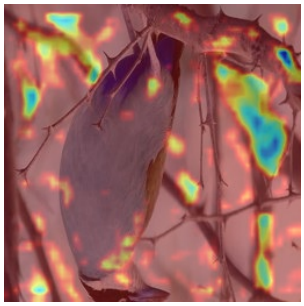
((a)) Image 1



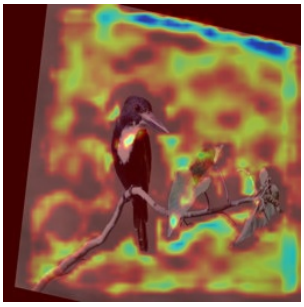
((b)) Image 2



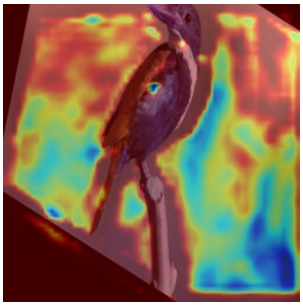
((c)) Image 3



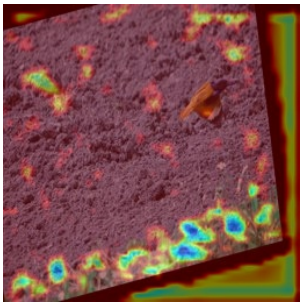
((d)) Image 4



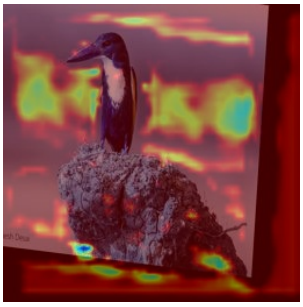
((e)) Image 5



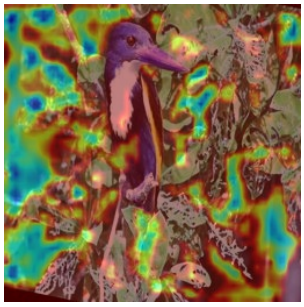
((f)) Image 6



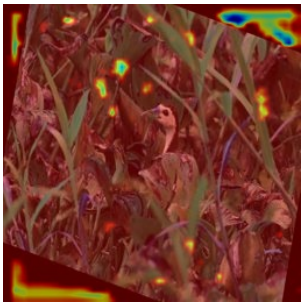
((g)) Image 7



((h)) Image 8



((i)) Image 9

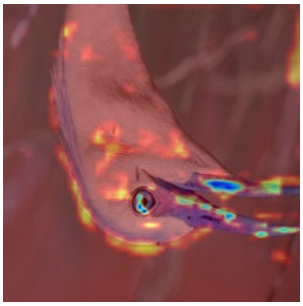


((j)) Image 10

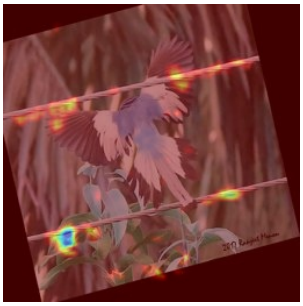
Correct Class White Breasted Kingfisher

Incorrect Class

((a)) Image 1

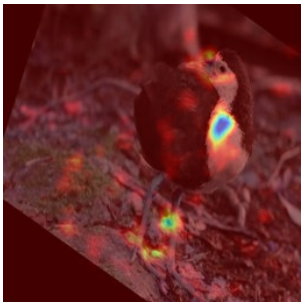


((b)) Image 2

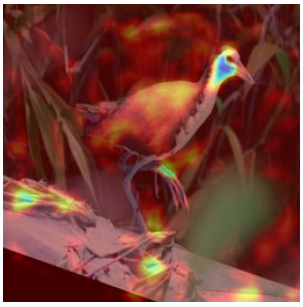


((c)) Image 3

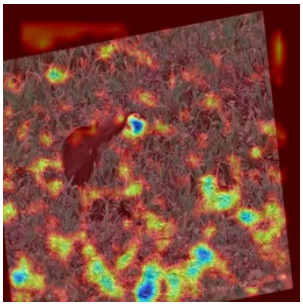
((d)) Image 4



((e)) Image 5



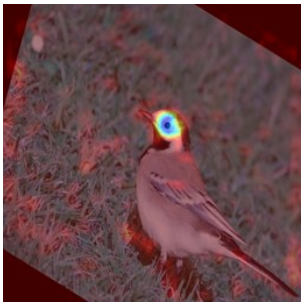
((f)) Image 6



((g)) Image 7

((h)) Image 8

((i)) Image 9



((j)) Image 10

Correct Class White Breasyed Waterhen

Incorrect Class

7 Conclusion

Through the comparative analysis of Adam vs SGD, alongside an in-depth examination of various normalization techniques and the nuanced impact of batch size, particularly on Group Normalization, I have garnered valuable insights into optimizing deep learning models. These findings underscore the importance of carefully selecting optimization algorithms and normalization methods in accordance with the specific requirements and constraints of the model and training environment.

part 1 models