Deep Learning

Assignment Report
Transfer Learning Optimization

Professor:
Dr. Eftekhari

Author:
Amirhossein Abolhasani

Spring 2025

Contents

1 Introduction

In recent years, deep learning has achieved significant advances in various fields, including computer vision and natural language processing. However, effective training of deep neural networks has always been accompanied by challenges, such as gradient fluctuations, the unstable gradient phenomenon, and prolonged network learning time. One effective approach to address these challenges is the use of transfer learning, where pre-trained models are adapted for new tasks.
In this context, techniques such as normalization have been introduced, which have a significant impact on learning time and gradient control. Methods such as Batch Normalization, Layer Normalization, and Filter Response Normalization were introduced with the primary goal of stabilizing activation value distributions and improving convergence rates. However, their impact on optimization quality and their effects on layer gradients require focused and systematic analysis.
In this assignment, with a focus on experimental and visual aspects, various normalization techniques and training conditions have been explored to achieve a deeper understanding of model behavior in parameter space and to highlight the strengths and weaknesses of each method.
To achieve these goals, in addition to local memory, the Weights & Biases cloud dashboard has been used for accurate recording of metrics, storing checkpoints, and comparisons. Additionally, 2D and 3D visualization techniques of the cost function have been employed to more precisely examine the relationship between cost function structure and model generalizability.

2 Methodology

To clarify the impact of different normalizations on the mentioned aspects, only one variable has been changed in each case, with the aim of ensuring a fair comparison in every experiment.

2.1 Experimental Settings

In examining normalization techniques used in transfer learning, a suitable pre-trained model and a quality benchmark dataset are required.
It should be noted that the only fixed hyperparameters throughout all experiments are the Seed (used for generating random numbers) and Epochs. Additionally, the Adam optimizer has been used in all experiments.

2.1.1 Pre-trained Model

MobileNet-V2 has been selected for this review. This model is a better choice due to its smaller size compared to other models such as Resnet or VGG.
This model was trained on the ImageNet-1k dataset. The architecture of this model is shown in Figure 1.

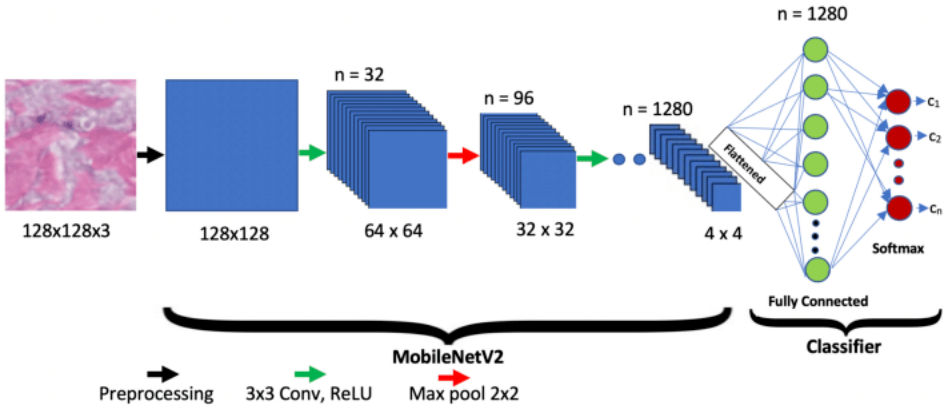


Figure 1: Architecture of the MobileNet-V2 model, which consists of two parts: Classifier and Backbone.

2.1.2 Dataset

The dataset used for the classification task is CIFAR-10, which includes images belonging to one of ten classes: airplane - automobile - bird - cat - deer - dog - frog - horse - ship - truck.

Feature	Description
Number of training samples	50000
Number of validation samples	8000
Number of test samples	2000
Image size	32×32
Number of channels per image	3
Number of classes	10

Table 1: Details of the CIFAR-10 dataset

Figure 2: Samples from the CIFAR-10 dataset

2.1.3 Classifier Heads

For the Backbone model, which is MobileNet-v2, one classifier network per normalization method has been fine-tuned, with the only difference being the type of normalization layer. Additionally, for better comparison of methods, a classifier network without a normalization layer has been used as a baseline in some experiments.

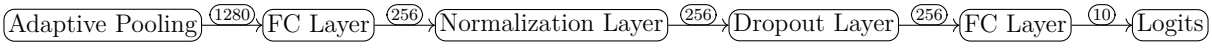


Figure 3: Classifier head architecture in all experiments; on each arrow, the input or output dimension is shown.

2.2 Examined Techniques

Normalization techniques Layer Normalization, Batch Normalization, and Filter Response Normalization have been examined under different conditions, with or without the Gradient Clipping technique.

2.3 Analysis and Implementation Tools

To examine the impact of various normalization methods on error, visualization of the cost function has been attempted both as a contour plot and as a 3D plot.
The impact of these methods on gradients has also been examined by visualizing the gradient magnitude (histogram) of each layer in each Epoch.
Logs of all experiments and runs are available in the Weights & Biases dashboard. Additionally, for saving models and checkpoints, this is done completely automatically both locally and in Weights & Biases. [1]

3 Experimental Results and Analyses

Three main experiments were conducted on each model, each comparing normalization techniques in specific conditions (as shown in a table in each section):

3.1 First Experiment: Convergence Rate Comparison

Under identical conditions and with a relatively appropriate learning rate for all classifiers, these networks were trained. A notable point is the faster convergence speed of models with normalization compared to the baseline model.

[1] Checkpoints are stored in an object of type Artifact.

Feature	Value
Number of Epochs	15
Learning Rate	0.001
Use of Gradient Clipping	No
Batch Size	64

Table 2: Hyperparameter values in the first experiment

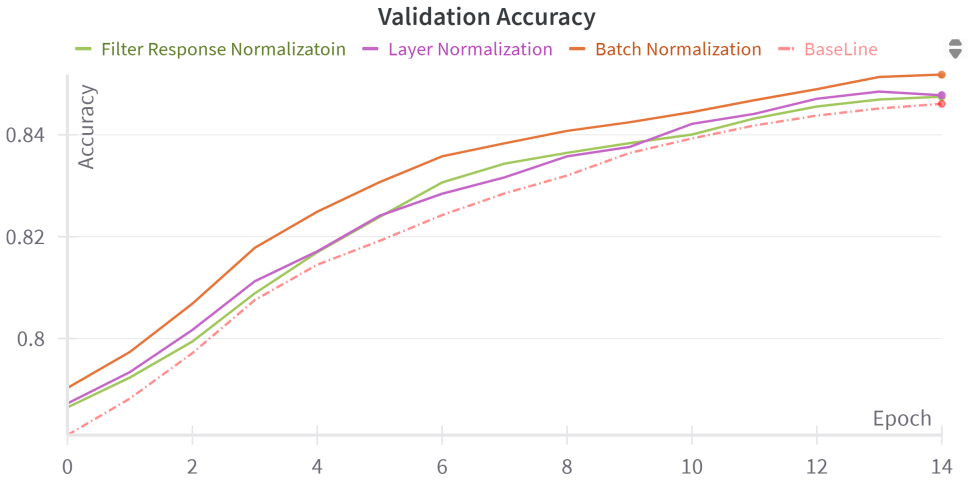
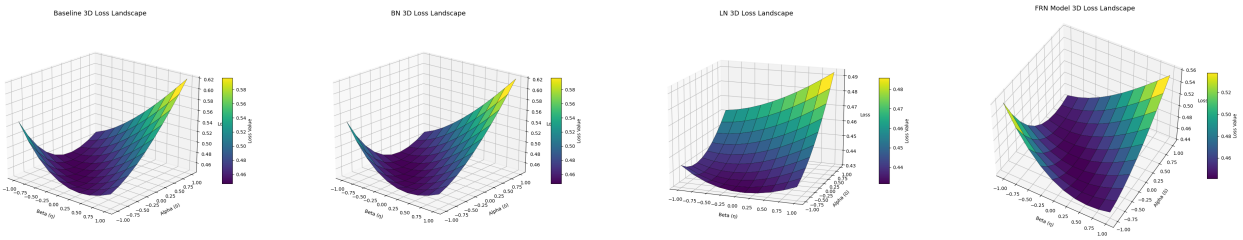


Figure 4: Accuracy plot in the first experiment (the plot has been smoothed and made interpretable using Running Average).

As seen in Figure 4, the convergence of models with normalization occurs faster than the baseline. This is particularly evident in more epochs (although this conclusion can also be drawn from Figure 4). The reason is that normalization directly affects smoothing the cost function surface, making optimization easier. In a way, normalization internally makes the cost function L-Lipschitz, meaning the cost function changes slowly and at most by L between any two points. In Figure 5, the 3D landscape of cost functions for each normalization technique and the baseline, focusing on perturbation of classifier head parameters, is shown.

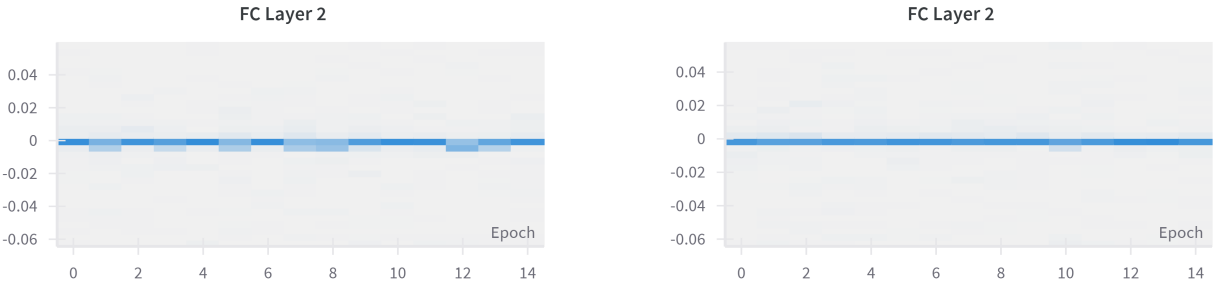


(a) Baseline model (b) Batch Normalization (c) Layer Normalization (d) Filter Response Normalization

Figure 5: Loss landscape in the first experiment

In analyzing the cost function landscape, all examined normalization techniques have relatively smooth cost function surfaces. The main reason for this observation is the use of transfer learning. When using a previously trained model, we essentially move to a location in parameter space close to the error minimum. Now, by changing a small number of parameters (which is essentially Fine-Tuning), we aim to improve model details on the new dataset or task. It appears this is the primary cause of the smoothness and lack of sharpness in the cost function across all methods and experiments (visualization of the cost function is still provided). It is expected that to better observe differences in the error landscape for each normalization method and the baseline, the model should be trained from scratch, which is unfortunately beyond the scope of this assignment due to processing limitations.

To examine gradient magnitudes in different methods, the gradient of the second FC layer is analyzed (Figure 6). One reason why changes are not clearly evident on a large scale could be the low learning rate and resulting stable learning. [1]



(a) Baseline model (b) Filter Response Normalization

Figure 6: Histogram of gradient magnitude per Epoch for the second FC layer

As seen in Figure 6, this plot shows the frequency of various gradient magnitudes across different epochs, meaning we observe gradient behavior during training. The applied normalization

[1] Empirical observation of this issue is available in Appendix 1 6.1.

technique has helped make the gradient L-Lipschitz, while in the baseline model, the frequency distribution is slightly wider.

3.2 Second Experiment: Impact of Gradient Clipping

One of the problems in neural networks is gradient vanishing or gradient explosion. A simple but well-known technique for controlling gradient magnitude is Gradient Clipping. This operation works by, during gradient calculation in the back propagation stage, setting the gradient magnitude to the threshold value if it exceeds a hyperparameter threshold. In this way, it attempts to control the gradient bounds.

Feature	Value
Number of Epochs	15
Learning Rate	0.001
Use of Gradient Clipping	Yes
Clipping Threshold	1.0
Batch Size	64

Table 3: Hyperparameter values in the second experiment

In this part, it was expected that this technique would lead to smoother model convergence, but this occurred specifically in Layer Normalization (Figure 7).

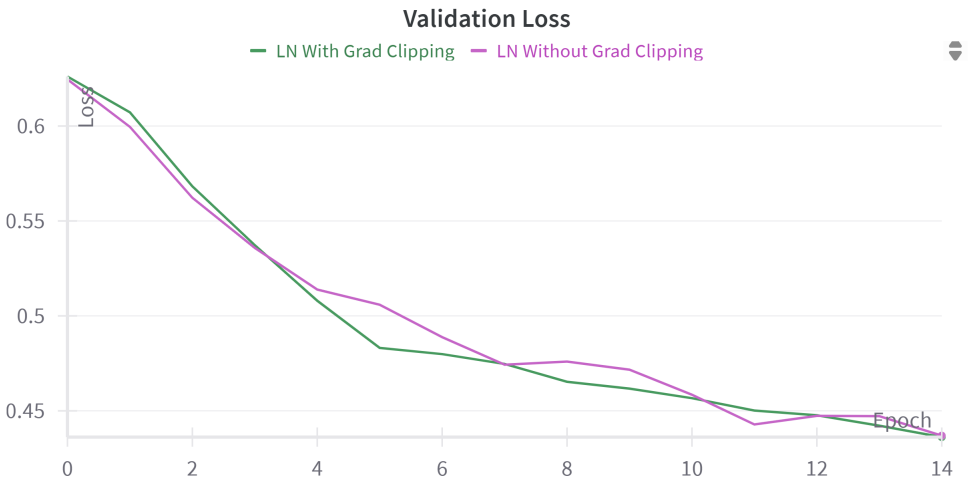
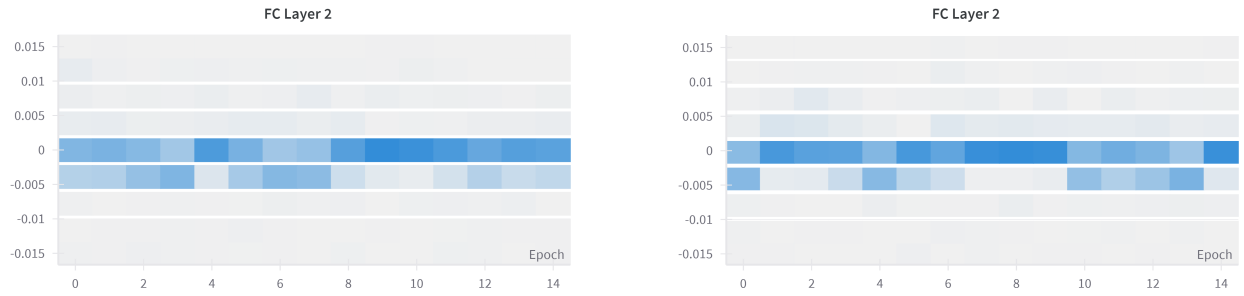


Figure 7: Validation error plot for Layer Normalization

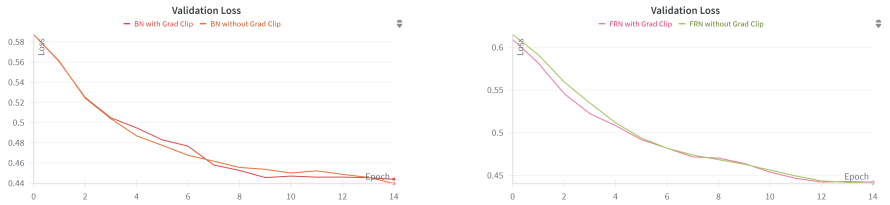
This is evident from the gradient histograms and their trends per Epoch (Figure 8).



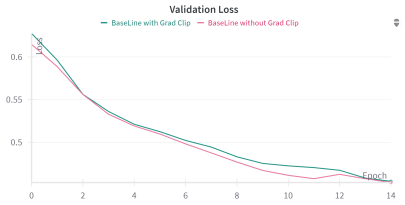
(a) Layer Normalization without Gradient Clipping (b) Layer Normalization with Gradient Clipping

Figure 8: Histogram of gradient magnitude per Epoch for the second FC layer

In other models, the comparison of obtained error behavior shows no noticeable change compared to before, which is due to the small learning rate used for training (Figure 9).



(a) Batch Normalization (b) Filter Response Normalization



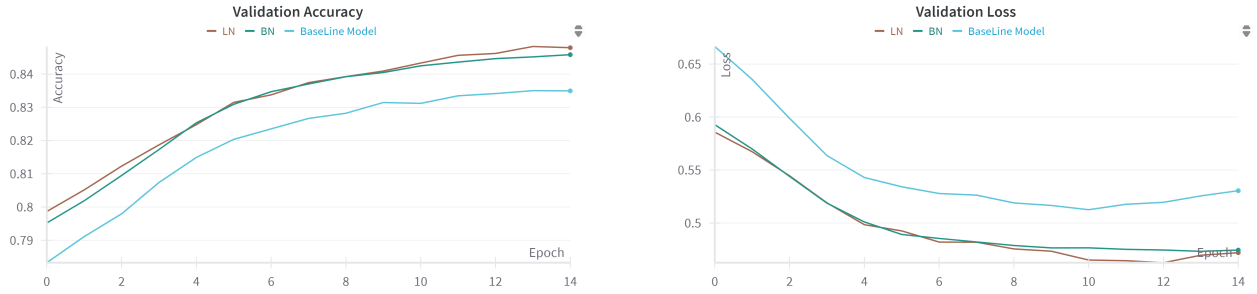
(c) Baseline model

Figure 9: Comparison of cost error trend for each model with and without Gradient Clipping

3.3 Third Experiment: Impact of Learning Rate

One of the most important applications of normalization techniques is accelerating the convergence process in network training. This was tested in this experiment. When the cost function becomes smooth, it is natural to move with larger steps (i.e., higher learning rate), leading to faster convergence.

In this experiment, only the learning rate was varied to demonstrate that under conditions where the baseline model cannot even converge, other models converge well (Figure 10).



(a) Accuracy plot

(b) Loss plot

Figure 10: Impact of learning rate on convergence and its speed

Feature	Value
Number of Epochs	15
Learning Rate	0.01
Use of Gradient Clipping	No
Batch Size	64

Table 4: Hyperparameter values in the third experiment

4 Discussion and Conclusion

In experiments aimed at examining the effects of various normalization techniques and the use of Gradient Clipping, notable differences in the behavior of some models were observed in terms of convergence, convergence speed, gradient stability, and final accuracy over a specific number of epochs.

In Table 5, a comparison of normalization methods is attempted. [1]

[1]Some items were not necessarily examined in the experiments, but mentioning them is useful.

Normalization Technique	**Strengths**	**Weaknesses**
Batch Normalization	+ Accelerates convergence + Stability in gradients + Improves generalization in many problems + Suitable for images	– Dependent on batch size – Poor performance in small batches – Complexity in use in RNNs and sequence-based models
Layer Normalization	+ Independent of batch size + Suitable for RNNs and sequence-to-sequence models + Simpler implementation in some architectures	– In convolutional models, usually not as effective as BN – May cause performance drop in classification tasks
Filter Response Normalization	+ Suitable for small and large batches + Increases numerical stability + Resolves some limitations of BN, such as batch size dependency + Successful use in Fixup ResNet and networks without BN	– Relatively new and less examined – Needs more precise parameter tuning (e.g., learnable epsilon) – More complex implementation in some frameworks (or even lack of implementation)

Table 5: Comparison of different normalization techniques

Gradient Clipping is one of the common techniques to prevent gradient explosion in training deep neural networks. In the performed experiments, it was found that using Gradient Clipping did not create much difference in the results. The main reason for this can be attributed to the low learning rate (0.001).

When the learning rate is relatively low, model optimization occurs with smaller steps, which reduces severe fluctuations in gradient values. In such a case, the risk of gradient explosion significantly decreases, and therefore the role of this operation becomes less prominent.
In other words, using a low learning rate has a similar effect to Gradient Clipping, keeping gradients within a safe range without the need for forced limitation. However, in other settings (such as high learning rates or very deep networks), using this method can still be crucial.

Another notable point is the difference between making the cost function L-Lipschitz, which occurs through normalization methods, and the Gradient Clipping method. In this regard, limiting the gradient in the cost function can occur fundamentally as a result of the function becoming

Lipschitz through normalization, or via Gradient Clipping, which only limits the gradient and has no impact on the cost function landscape. It can be concluded that the Gradient Clipping method is more effective when used alongside normalization methods and does not solve problems fundamentally on its own.

In analyzing the loss landscape, subtle differences among normalization techniques were observed. The most important cause of this smoothness in the visualized functions is the use of a pre-trained model, which has largely placed us in a suitable location in parameter space, and we essentially perform Fine-Tuning by moving slightly around the main minimum. Although all models had relatively smooth landscapes with only one local minimum, models with normalization showed gentler slopes around the minimum, indicating flatter minima and thus better generalization. These results align with existing theoretical hypotheses regarding the connection between landscape sharpness and model generalization capability.

5 Practical Recommendations

1. A learning rate around 0.01 to 0.1 can provide high convergence speed when training models with normalization layers.

2. If working with convolutional networks and large data batches, Batch Normalization appears to be the first choice. [1]

3. If batch sizes are small or the model is sequence-based (for example, RNN or Transformer architectures), it is better to use Layer Normalization.

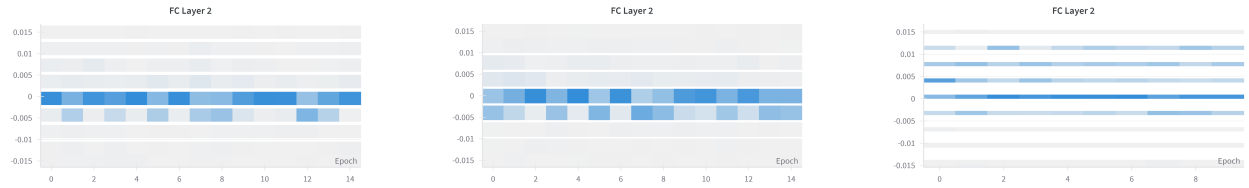
4. In any conditions where gradient behavior instability is likely, or the model's behavior in the training phase oscillates, using Gradient Clipping is recommended.

[1] More information on this is in 6.2.

6 Appendices

6.1 Effect of Learning Rate on Gradient Magnitude

Here, a small comparison is provided to demonstrate the effect of learning rate on the magnitude of the gradient in the second FC layer of the baseline classifier head.



(a) $lr = 0.001$ (b) $lr = 0.01$ (c) $lr = 0.2$

Figure 11: Effect of learning rate on convergence and its speed

As seen in Figure 11, as the learning rate decreases, gradient sizes gradually become smaller and more focused during training.

6.2 Dependency of Batch Normalization to Batch Sizes

In the pseudocode 1, it can be seen that the mean and variance are calculated for each batch of data. If the data are few in number, a good estimate of the real mean and variance in the data cannot be obtained. This indicates that the Batch Normalization method is highly reliant on the number of samples in each batch.

Algorithm 1 Batch Normalization (Training Mode)

Require: Mini-batch $\mathcal{B} = \{x_1, x_2, \ldots, x_m\}$, scale parameter γ, shift parameter β, small constant ε

Ensure: Normalized and transformed outputs $\{y_1, y_2, \ldots, y_m\}$

1: Compute batch mean: $\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$

2: Compute batch variance: $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$

3: **for** $i = 1$ to m **do**

4: Normalize: $\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}}$

5: Scale and shift: $y_i \leftarrow \gamma \hat{x}_i + \beta$

6: **end for**