IFT6135A - Homework #1

Report for the Practical Part of the Homework

Université de Montréal

Felix Wilhelmy

14 March 2025

Table des matières

1 Question 4.2 **4**
1.1 Results . . . 4
1.2 Discussion . . . 5

2 Question 4.3 **6**
2.1 Results . . . 6
2.2 Discussion . . . 7

3 Question 4.4 **8**
3.1 Results . . . 8
3.2 Discussion . . . 9

4 Question 4.5 **10**
4.1 Results . . . 10
4.2 Visualization . . . 10
4.3 Configurations . . . 10
4.4 Discussion . . . 11

5 Question 4.6 **13**
5.1 Results . . . 13
5.2 Visualisation . . . 13
5.3 Configurations . . . 13
5.4 Discussion . . . 14

6 Question 4.7 **18**
6.1 Results . . . 18
6.2 Configurations . . . 19
6.3 Discussion . . . 19

7 Question 4.8 **21**
7.1 Visualization . . . 21
7.2 Discussion . . . 22

Résumé

This report describes experiments on CIFAR10 image classification task using various neural network architectures (MLP, ResNet18, MLPMixer). The CIFAR10 dataset comprises 60000 color images in 10 classes. Experiments were conducted on Google Colab Pro utilizing GPU acceleration with CUDA. Initial experiments were performed over 15 epochs, but these were extended to 30 epochs to obtain more informative learning curves.

1 Question 4.2

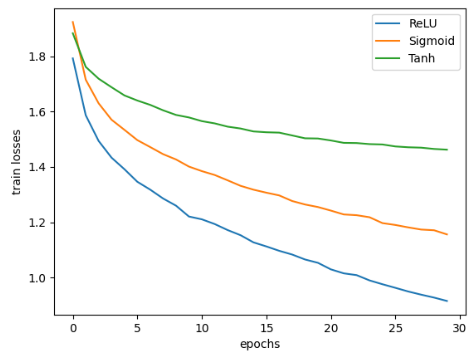
In this question, we evaluate the impact of three activation functions (ReLU, Sigmoid, and Tanh) on an MLP architecture that is kept identical across all experiments to isolate the effect of the non-linearity. The model has an input size of 3072, four hidden layers with sizes [1024, 512, 64, 64], and 10 output classes, totaling approximately 3.71 million trainable parameters.

1.1 Results

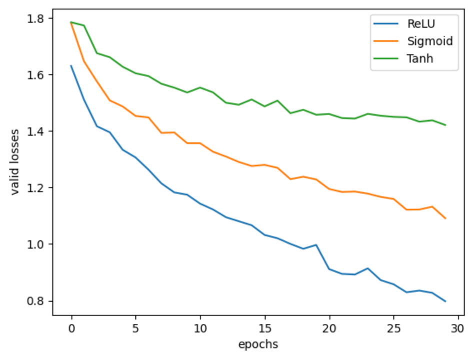
Below is the table and figures summarizing training results of the experiment. The results, presented in Table 1 and Figure 1, highlight the comparative performance of the three activation functions over 30 epochs. Additional analysis shows that ReLU consistently outperforms the other functions across training, validation, and test metrics.

Activation	Train Loss	Val Loss	Train Acc	Val Acc	Test Loss	Test Acc
ReLU	0.9150	0.7974	67.7%	72.4%	1.2438	58.9%
Sigmoid	1.1559	1.0913	58.8%	62.2%	1.3265	52.8%
Tanh	1.4625	1.4211	48.4%	51.1%	1.5202	46.7%

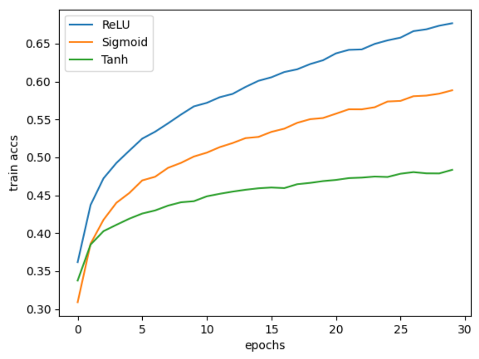
Table 1 – Final performance metrics for the MLP model with different activation functions. The results indicate that the ReLU configuration achieves the lowest losses and highest accuracies, while Sigmoid and Tanh lag behind.



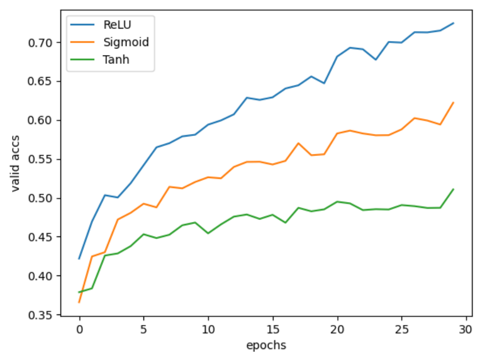
(a) Training Loss



(b) Validation Loss



(c) Training Accuracy



(d) Validation Accuracy

Figure 1 – Evolution curves of training loss (a), validation loss (b), training accuracy (c), and validation accuracy (d) for the MLP with different activation functions (ReLU, Sigmoid, Tanh). The legends in each subfigure denote the activation function used.

The results show that the ReLU activation function leads to better performance in terms of both convergence speed and final accuracy, whereas the Sigmoid and Tanh configurations converge more slowly and achieve lower accuracies.

1.2 Discussion

The **ReLU** function exhibits the best performance with a final accuracy of 58.9% on the test set. This can be explained by the fact that for any positive x, ReLU has a derivative equal to 1. This constant derivative ensures that gradients remain strong and do not diminish through the layers, allowing the network to learn quickly and effectively.
In contrast, **Sigmoid** and **Tanh** both lag behind in terms of performance, with final test accuracies of 52.8% and 46.7% respectively. This is due to their saturating behavior : the regions where these functions produce meaningful gradients are limited, and for large positive or negative x, the derivatives become very small. Small derivatives cause the gradients during backpropagation to vanish, leading to slower convergence. Furthermore, this effect becomes compounded as the network depth increases.
In conclusion, among the evaluated activation functions, **ReLU** is the better choice as it outperforms Sigmoid and Tanh in terms of convergence speed, accuracy, and generalization.

2 Question 4.3

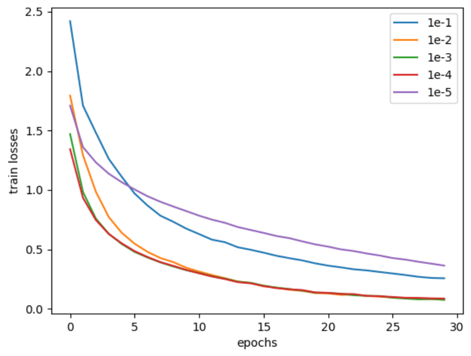
This question evaluates the impact of different learning rates on the performance of the ResNet18 architecture. The model configuration remains almost identical across all experiments, and only the learning rate is varied. We conducted experiments with the Adam optimizer using learning rates of [1e-1, 1e-2, 1e-3, 1e-4, 1e-5].

2.1 Results

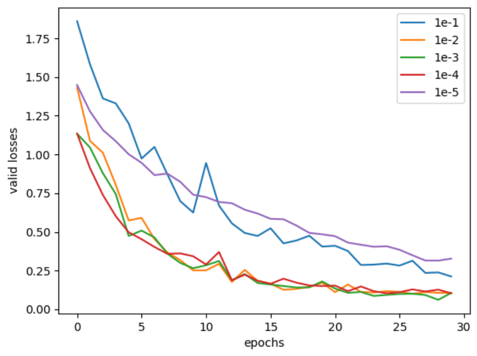
Below is the table and figures summarizing the training results of the experiment.

Learning Rate	Train Loss	Val Loss	Train Acc	Val Acc	Test Loss	Test Acc
0.1	0.2565	0.2125	91.21%	92.52%	0.5112	84.72%
0.01	0.0769	0.1055	97.36%	97.13%	0.5421	87.01%
0.001	0.0755	0.1062	97.28%	96.48%	0.4853	88.19%
0.0001	0.0859	0.1041	96.98%	96.91%	0.5055	86.45%
0.00001	0.3631	0.3276	87.78%	89.30%	0.7638	74.90%

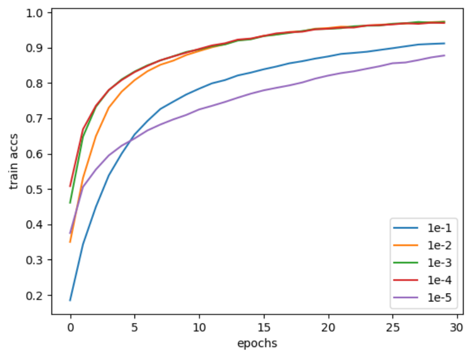
TABLE 2 – Final performance metrics for ResNet18 using different learning rates with the Adam optimizer. Intermediate learning rates (1e-2, 1e-3, 1e-4) yield the best performance, while extremely high or low learning rates degrade performance.



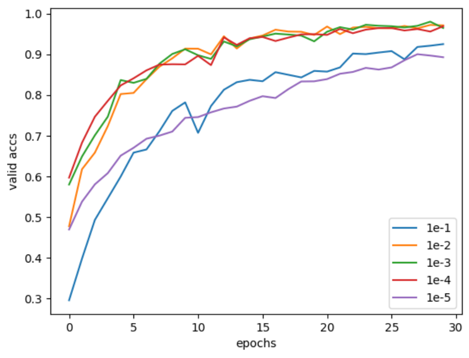
(a) Training Loss



(b) Validation Loss



(c) Training Accuracy



(d) Validation Accuracy

FIGURE 2 – Evolution curves of training loss (a), validation loss (b), training accuracy (c), and validation accuracy (d) for ResNet18 with different learning rates.

2.2 Discussion

The experimental results show that using a very high learning rate (1e-1) causes rapid initial improvements but leads to instability or even divergence later in training. Conversely, a very low learning rate (1e-5) results in very slow convergence and inferior final performance (74.90%). The best performance is achieved with the intermediate learning rates (1e-2, 1e-3, and 1e-4), with 1e-3 yielding the highest test accuracy of 88.19%.
Learning rate is a critical hyperparameter ; its proper tuning balances convergence speed and stability. In this setup, a moderate learning rate of around 1e-3 is recommended.

3 Question 4.4

In this question, we evaluate the impact of different patch sizes on the performance of the MLPMixer architecture. Three configurations are compared, where the only variable is the patch size [2, 4, 8] ; all other hyperparameters remain identical. The model is configured with an embedding dimension of 256, 4 Mixer blocks, no dropout, and GELU as the activation function. This setup isolates the effect of patch size on convergence and generalization.

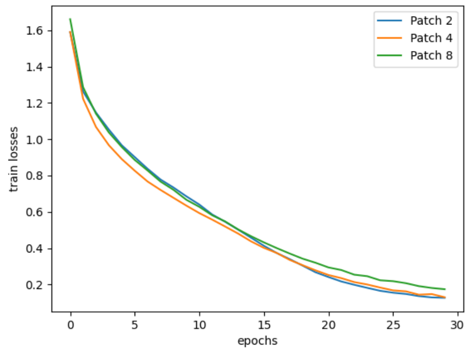
3.1 Results

Below is the table and figures summarizing the training results of the experiment.

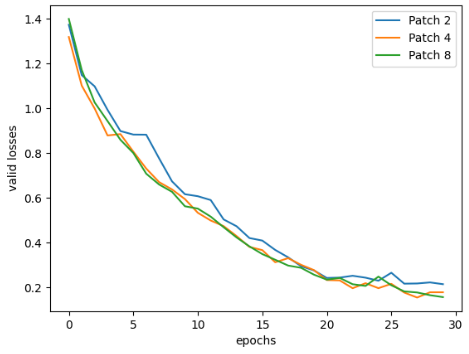
Patch Size	**Train Loss**	**Val Loss**	**Train Acc**	**Val Acc**	**Test Loss**	**Test Acc**
2	0.1266	0.2142	95.61%	93.89%	1.1291	73.64%
4	0.1296	0.1781	95.48%	95.20%	0.9936	75.95%
8	0.1738	0.1565	93.94%	96.05%	1.0589	73.72%

TABLE 3 – Final performance metrics (after 30 epochs of training) for MLPMixer with different patch sizes. The results suggest that patch size has a notable impact on convergence and generalization.

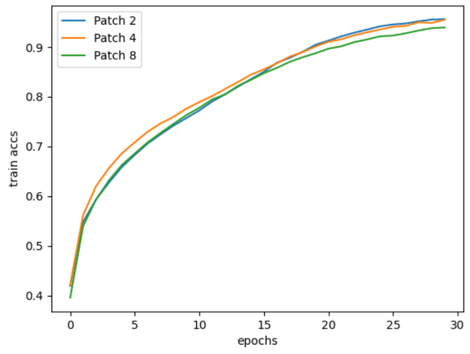
Figure 3 shows a 2×2 grid of evolution curves for training loss, validation loss, training accuracy, and validation accuracy for the three patch size configurations.



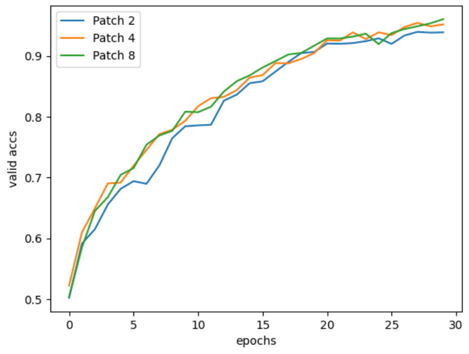
(a) Training Loss



(b) Validation Loss



(c) Training Accuracy



(d) Validation Accuracy

FIGURE 3 – Evolution curves of training loss (a), validation loss (b), training accuracy (c), and validation accuracy (d) for MLPMixer with different patch sizes.

3.2 Discussion

In an MLPMixer, a patch represents a small, non-overlapping sub-image (or token) that the image is divided into before processing by the model. A smaller patch size leads to a larger number of patches, which increases the number of parameters and training time. For example, for a configuration with a patch size of 2, the number of patches is calculated as

$$256 = \left(\frac{32}{2}\right)^2 .$$

This configuration results in a large MLP for token-mixing (dimensions 256→128→256 for each block) with the highest parameter count and the ability to capture more complex relations. However, as shown in Figure 3(b), the configuration with patch size 2 shows the worst validation performance and even plateaus during training (around epochs 4 and 11), suggesting early signs of overfitting. In contrast, with patch size 4, the model achieves improved validation accuracy (95.20%) and attains the lowest test loss (0.9936) along with the highest test accuracy (75.95%). The configuration with patch size 8 shows the lowest training accuracy (93.94%) but a high validation peak (96.05%), indicating that even with coarser patches, the model can capture sufficient spatial information for the CIFAR10 dataset.
In conclusion, among the three patch sizes evaluated, a patch size of 4 strikes the optimal balance between complexity and generalization.

4 Question 4.5

In this question, we explore the impact of various hyperparameter settings on the performance of ResNet18 for CIFAR-10. Four configurations are compared : a baseline using SGD with momentum, a variant with increased weight decay, a configuration with a high learning rate, and a run using AdamW. The models share the same architecture and dataset, with differences only in the optimizer settings.

4.1 Results

Below is the table of final performance metrics for the four configurations.

Configuration	**Train Loss**	**Val Loss**	**Train Acc**	**Val Acc**	**Test Loss**	**Test Acc**
Momentum (SGD) Baseline	0.0925	0.0968	96.68%	97.13%	0.5112	84.72%
Momentum (SGD) WD (1e-3)	0.1241	0.1232	95.70%	94.01%	0.4763	84.40%
Momentum (SGD) LR (1e-1)	0.3429	0.3262	88.16%	88.63%	0.4763	84.40%
AdamW LR (0.001)	0.0754	0.0925	97.43%	96.87%	0.4436	88.93%

TABLE 4 – Final performance metrics for ResNet18 under different configurations. The **Configuration** column presents the variations in the configuration in relation to the baseline configuration (SGD with momentum, LR=0.01, WD=5e-4). Here, **LR** stands for learning rate and **WD** stands for weight decay.

Figure 4 shows a 2×2 grid of evolution curves (training loss, validation loss, training accuracy, and validation accuracy) for the different configurations.

4.2 Visualization

For visualization, the first convolution layer's kernels are displayed in RGB. The weights were extracted, and a min-max normalization was applied to each 3x3 filter to scale values into the [0, 1] interval. The 64 filters are then shown in an 8 by 8 grid to inspect whether the filters focus on specific color channels or patterns.

4.3 Configurations

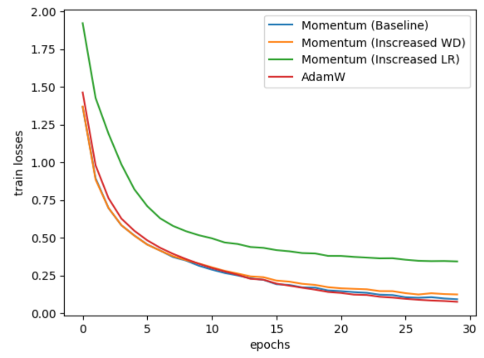
Baseline Configuration This configuration is chosen as the baseline because SGD with momentum is a widely recognized, stable optimizer for image classification.

- **Optimizer :** Momentum (SGD)
- **Momentum :** 0.9
- **Learning Rate :** 0.01
- **Weight Decay :** 5e-4

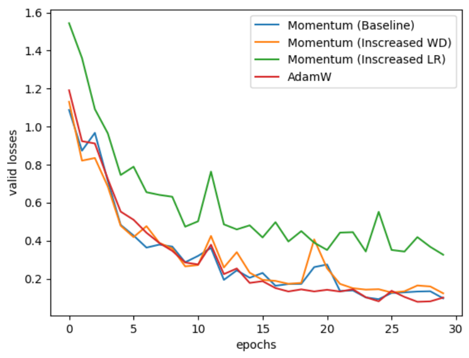
Momentum (SGD) with Increased Weight Decay This experiment increases the weight decay to 1e-3 to test if stronger regularization improves generalization.

Momentum (SGD) with High Learning Rate Increasing the learning rate to 0.1 evaluates whether a larger step size can accelerate convergence, albeit with the risk of instability.

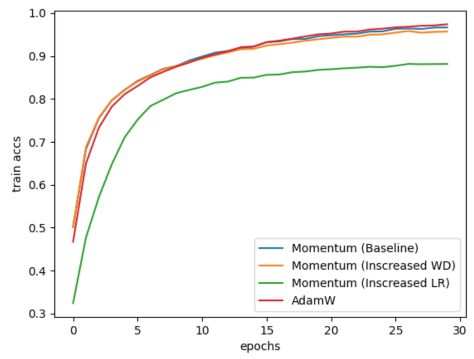
AdamW Using AdamW, with a learning rate of 0.001, leverages decoupled weight decay regularization which helps maintain a balance between learning rate and parameter updates, resulting in improved convergence and generalization.



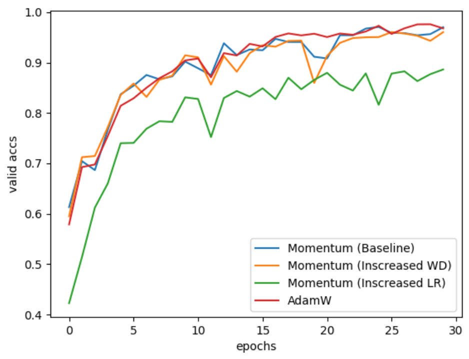
(a) Training Loss



(b) Validation Loss



(c) Training Accuracy



(d) Validation Accuracy

FIGURE 4 – Evolution curves for ResNet18 with different hyperparameter configurations.

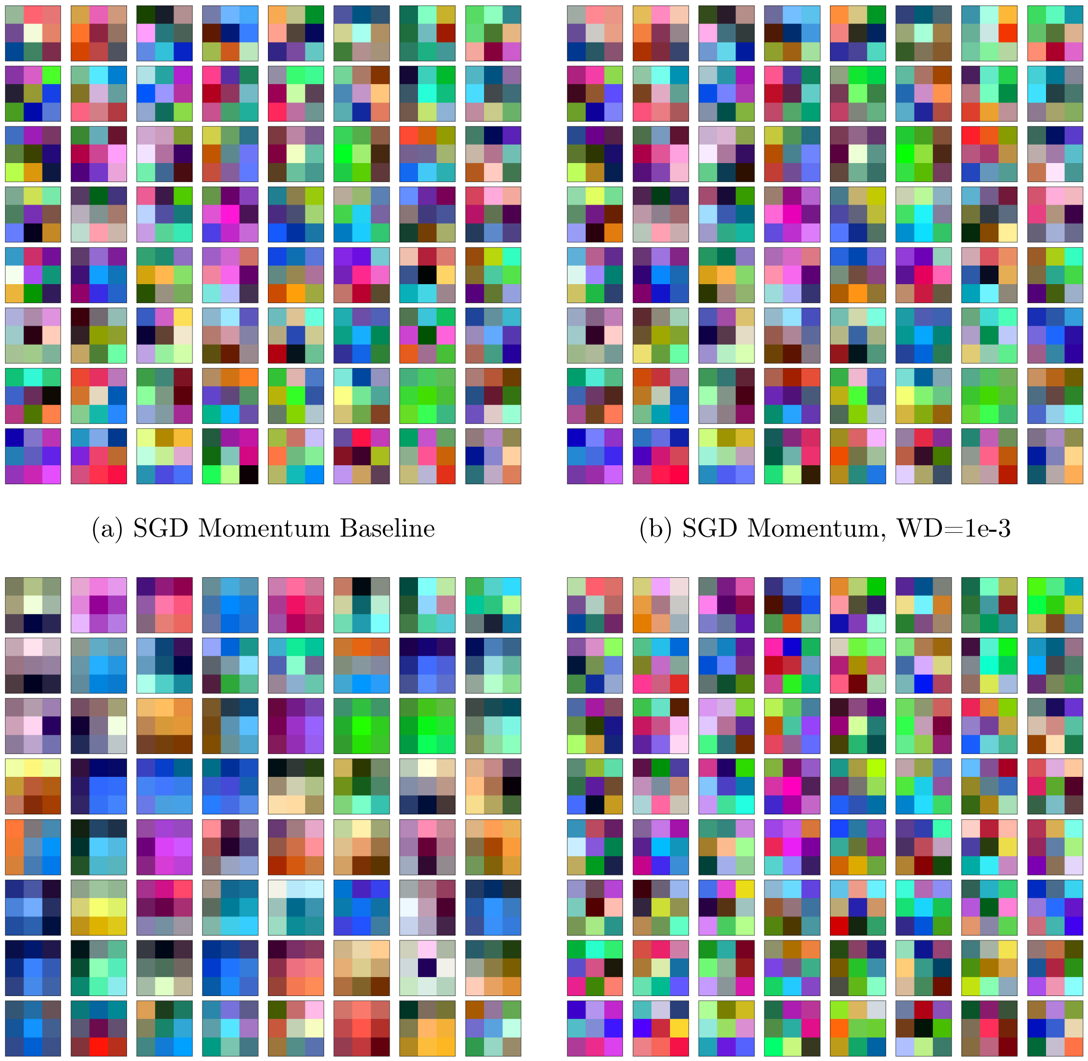
4.4 Discussion

Optimiser SGD with Momentum SGD with momentum works by accumulating a velocity vector in the direction of consistent gradients. This momentum term helps smooth the update steps and accelerates convergence in relevant directions while mitigating oscillations in the parameter updates. In our experiments, while SGD with momentum provides stable training, its convergence is less adaptive to the varying gradient magnitudes across layers compared to more modern optimizers.

Optimiser AdamW Switching the optimizer to AdamW with a learning rate of 0.001 provides the best performance among the tested configurations, yielding the lowest loss (0.4436) and the highest accuracy (88.93%) on the test set. AdamW decouples the weight decay from the gradient update, which results in more effective regularization and helps achieve faster convergence and better generalization, especially for deep networks.

Increasing Weight Decay By increasing weight decay to 1e-3, we aimed to enhance regularization. The training curves are similar to the baseline, indicating that while the extra weight decay did not drastically improve generalization, it did not hurt performance much either. The training loss is slightly higher, likely due to the stronger regularization.

Increasing Learning Rate Increasing the learning rate to 0.1 significantly alters the training dynamics. Although the model accelerates quickly at the start, it ends up oscillating and becomes unstable later during training. Even though the final test accuracy is 84.40%, which is close to the other Momentum configurations, its performance could be described as the worst of the experiment. This configuration might require further tuning or more epochs to fully benefit from the high learning rate.



(a) SGD Momentum Baseline

(b) SGD Momentum, WD=1e-3

(c) SGD Momentum, LR=1e-1

(d) AdamW

FIGURE 5 – Visualization of the first layer kernels for ResNet18 under different hyperparameter configurations : (a) Baseline (Momentum with LR=0.01 and WD=5e-4), (b) Momentum with increased weight decay (WD=1e-3), (c) Momentum with a higher learning rate (LR=0.1), and (d) AdamW.

In summary, the experiments indicate that while the baseline SGD with momentum performs well, employing AdamW with a learning rate of 0.001 further improves performance. Increasing weight decay offers modest gains, whereas an excessively high learning rate degrades performance. Therefore, for ResNet18 on CIFAR-10, we recommend using AdamW with a moderate learning rate (around 0.001).

5 Question 4.6

In this question, we investigate the effect of various configurations on the MLPMixer model. All models use a patch size of 4, as instructed. The baseline configuration uses an embedding dimension of 256, 4 Mixer blocks, no dropout, GELU activation, and an MLP ratio of [0.5, 4.0]. We compare this baseline against variants that incorporate dropout, increase the embedding dimension, or increase the number of mixer blocks (making the network deeper). We also explore visualization techniques for this type of model.

5.1 Results

Below is the table summarizing the final performance metrics for each MLPMixer configuration.

Configuration	Train Loss	Val Loss	Train Acc	Val Acc	Test Loss	Test Acc
Baseline	0.1296	0.1780	95.46%	95.20%	0.4229	88.12%
Dropout drop_rate=0.1	0.2289	0.2203	91.85%	92.99%	0.8907	75.78%
Increased Embedding embed_dim=512	0.1033	0.1512	96.33%	95.68%	1.0033	76.64%
Increased Depth num_blocks=8	0.1008	0.5279	96.41%	94.90%	1.0807	74.74%
Combined embed_dim=512 num_blocks=8	0.1038	0.1854	96.42%	94.77%	1.0475	75.33%
Combined Dropout embed_dim=512 num_blocks=8, drop_rate=0.1	0.1282	0.1518	95.51%	95.64%	0.9215	77.39%

Table 5 – Final performance metrics for MLPMixer variants. The baseline configuration (patch size=4, embed dim=256, 4 blocks, no dropout) achieves the best overall performance.

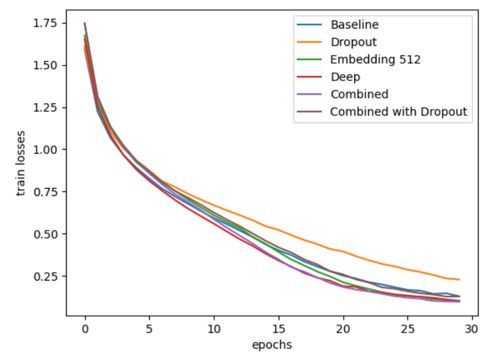
5.2 Visualisation

To make the visualisation more interesting, I decided on 2 points after some trial and errors. The first one is that I decided to use the cmap bwr for a better representation of the filters phases and deviations. The second being that I tried to do some sort of sorting based on their frequency. Sorting the filters so that smoother (low-frequency / paler) ones are shown first.

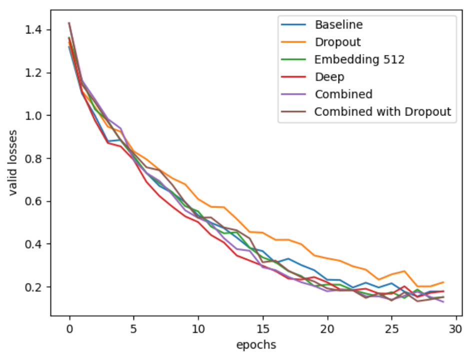
5.3 Configurations

Baseline Configuration This experiment was already trained as part of the question 4.4, so it was an obvious choice to include as a baseline for the other configurations in this experiment. Here is the full configuration :

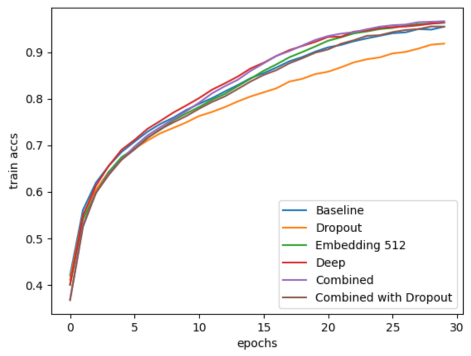
- **Optimizer :** AdamW
- **Epochs :** 30
- **Patch Size :** 128
- **Learning Rate :** 1e-3
- **Weight Decay :** 5e-4
- **Patch Size :** 4
- **Embedding Dimension :** 256
- **Number of Mixer Blocks :** 4
- **Dropout Rate :** 0.0
- **Activation :** GELU



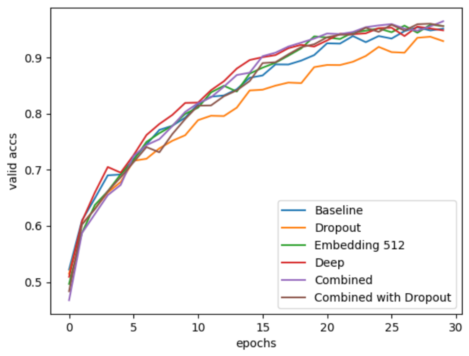
(a) Training Loss



(b) Validation Loss



(c) Training Accuracy



(d) Validation Accuracy

FIGURE 6 – Evolution curves of training loss (a), validation loss (b), training accuracy (c), and validation accuracy (d) for MLPMixer with different configurations. The legends in each subfigure denote the configuration used.

Dropout Configuration A dropout rate of 0.1 is introduced to regularize the model and reduce overfitting.

Increased Embedding Configuration The embedding dimension is increased from 256 to 512 to capture richer patch representations.

Deep Configuration (Increased Mixer Blocks) Doubling the number of Mixer blocks from 4 to 8 assesses whether increased depth improves token mixing.

Combined Configuration (Increased Embedding + Deep) This configuration combines the increased embedding dimension (512) and increased depth (8 blocks) to test the upper capacity of the model. The subsequent addition of dropout in the combined dropout variant is also evaluated.

5.4 Discussion

MLPMixer is an interesting architecture that uses modern training techniques such as skip connection (from ResNet) and layer normalisation (from Transformers) to maintain stable gradients and effective training even in deep networks, unlike a plain MLP on raw pixels which might suffer from vanishing or exploding gradients. Another advantage is the token/channel mixing technique, which makes the model both parameter-efficient and capable of learning complex interactions that a standard MLP (which would mix all pixels at once) cannot do effectively.

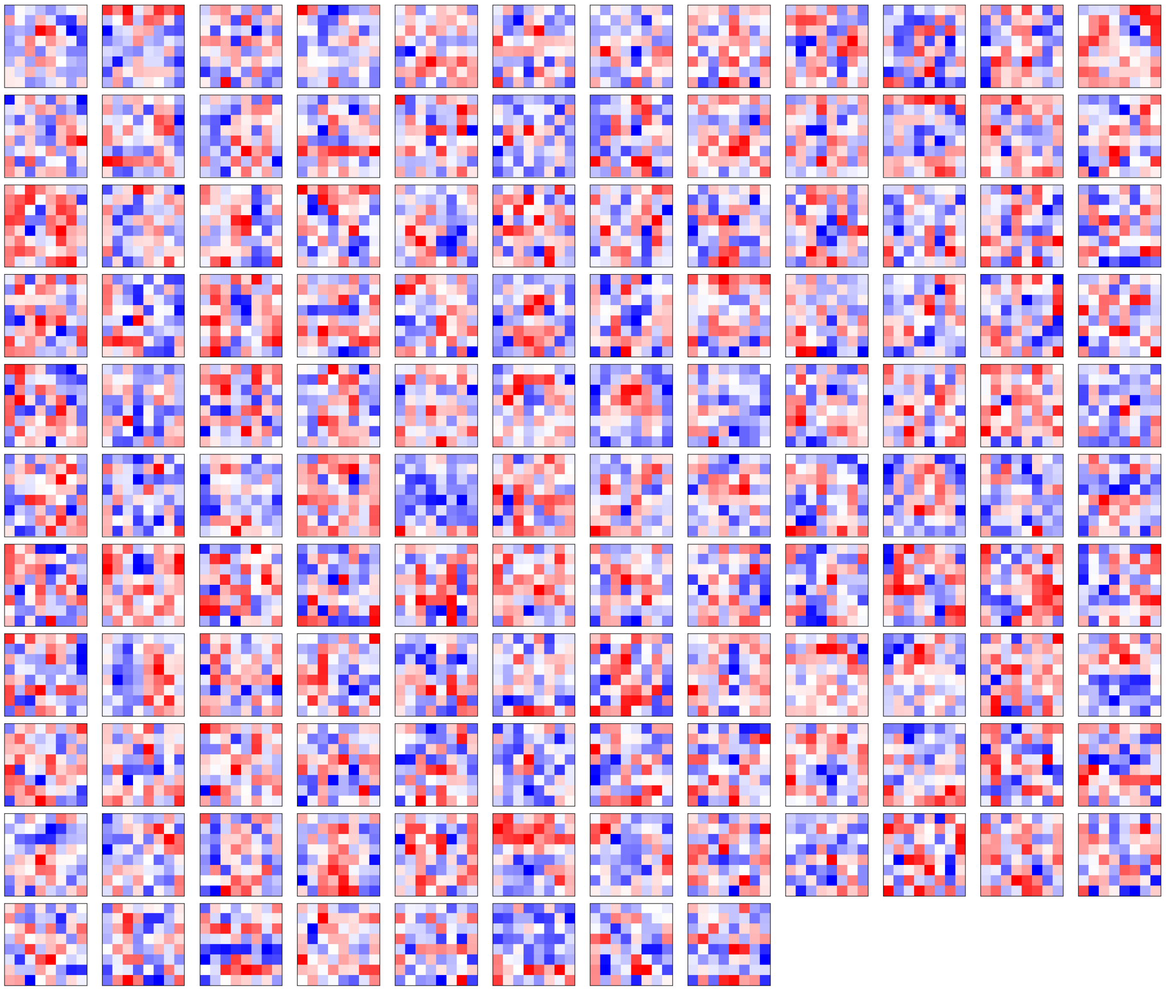
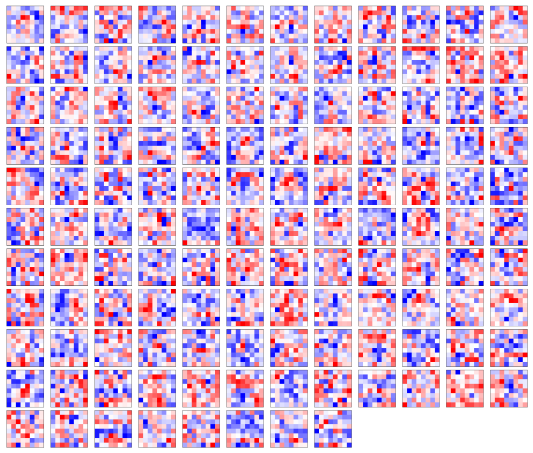
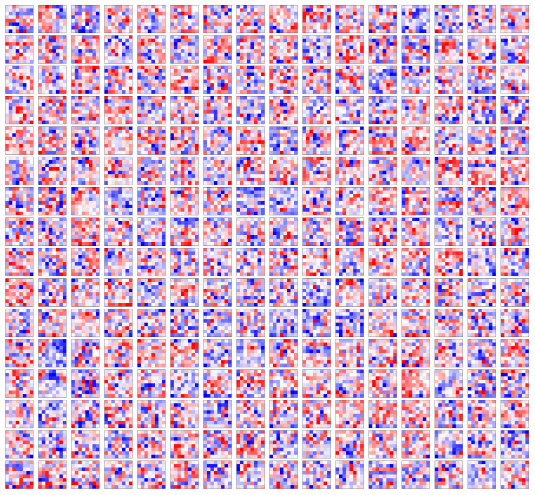


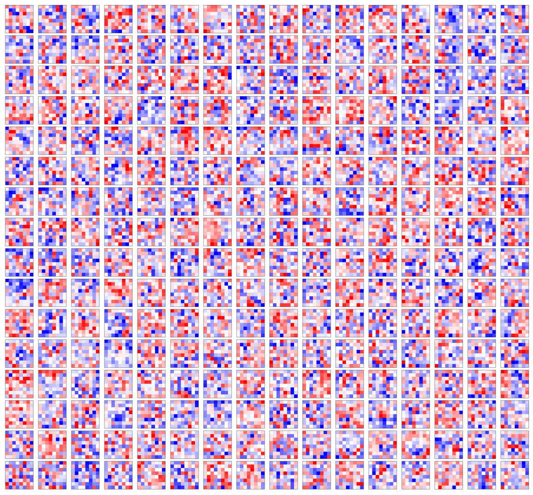
Figure 7 – Visualization of the token-mixing weights for the baseline MLPMixer configuration.



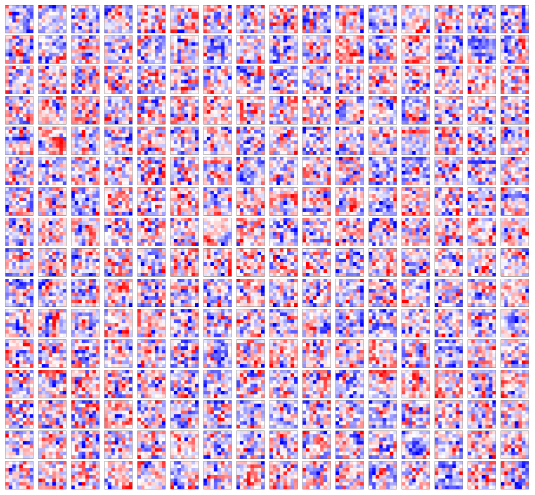
(a) MLPMixer with Dropout



(b) MLPMixer with Increased Embedding (512)



(c) Combined Variant



(d) Combined Variant with Dropout

Figure 8 – Visualization of the token mixing channel of the first Mixer block for different MLPMixer configurations.

Dropout With a smaller embedding, dropout worsened the performance, possibly because the network is not large enough to require heavy regularization. This was further confirmed when I ran the combined configuration. We can see that the performance does not improve much compared to other experiments even with the added complexity. But once you add the dropout in the next configuration, things are starting to look interesting.

Increasing Embedding Increasing the embedding dimensions seems to give a higher learning capacity to the model and slows and stabilises the convergence speed.

Increasing MixerBlocks Increasing the number of Mixer blocks increases the model size, but without increased regularisation, the additional capacity might hurt generalization.
It would have also been interesting to explore both simpler configurations and increased models with better regularisations. On one side, it would allow us to see if smaller MLPMixer models might have been better for the CIFAR10 task. On the other, from the experiments the biggest gains came from combining more channels and more depth with regularisations such as dropout. Without further exploration, it seems the baseline model remains a good trade-off between performance and computational

cost.

6 Question 4.7

In this question, we investigate how varying the width of the token-mixing and channel-mixing MLP layers (controlled by the MLP ratio) affects the generalization ability of the MLPMixer model. Four configurations are compared using an input size of 32, patch size of 4, embedding dimension of 256, 4 Mixer blocks, no dropout, and GELU activation.

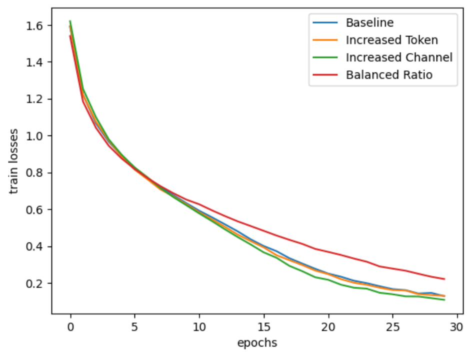
6.1 Results

Below is the table of final performance metrics for the four configurations, along with their respective MLP ratios.

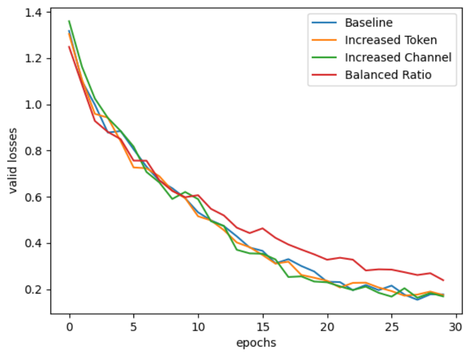
Configuration	Train Loss	Val Loss	Train Acc	Val Acc	Test Loss	Test Acc
Baseline Ratio : [0.5 ; 4.0]	0.1296	0.1780	95.48%	95.20%	0.9936	75.95%
Increased Token Ratio : [1.0 ; 4.0]	0.1311	0.1902	95.32%	95.45%	0.9685	77.22%
Increased Channel Ratio : [0.5 ; 8.0]	0.1096	0.1692	96.25%	95.14%	1.0106	76.91%
Balanced Ratio Ratio : [1 ; 1]	0.2223	0.2394	92.08%	92.01%	0.8979	75.69%

TABLE 6 – Final performance metrics for MLPMixer variants with different MLP ratios.

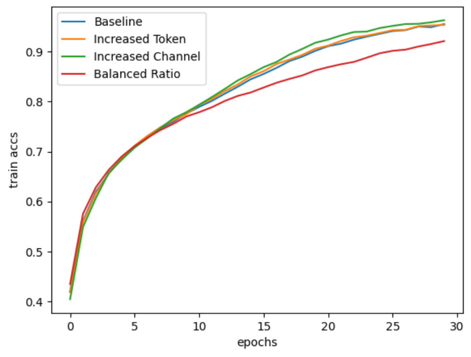
Figure 9 shows evolution curves for training loss, validation loss, training accuracy, and validation accuracy for the baseline configuration (similar trends were observed for the other variants).



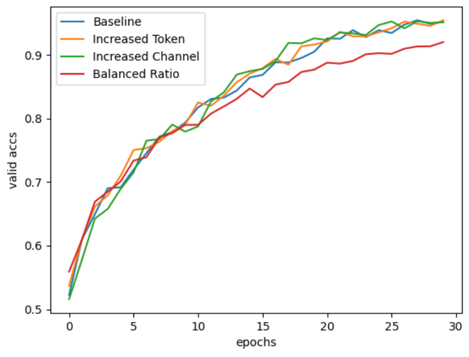
(a) Training Loss



(b) Validation Loss



(c) Training Accuracy



(d) Validation Accuracy

FIGURE 9 – Evolution curves for MLPMixer with different MLP ratios.

6.2 Configurations

Baseline In this configuration, the baseline model is used as a pretrained reference that is ready to use. It employs a patch size of 4 (yielding an 8×8 grid for a 32×32 image), an embedding dimension of 256, 4 MixerBlocks, a dropout rate of 0.1, and GELU as the activation function.

- **MLP Ratio :** [0.5, 4.0]
  - Token-Mixing : $0.5 \times 512 = 256$ units ;
  - Channel-Mixing : $4.0 \times 512 = 2048$ units.
- **Total Parameters :** Approximately 2.19 million.

Increased Token In this configuration, the capacity of the token-mixing MLP is doubled to better capture spatial (inter-patch) relationships. The increase in token-mixing capacity provides improved mixing of information across patches with only a modest increase in overall parameters.

- **MLP Ratio :** [1.0, 4.0]
  - Token-Mixing : $1.0 \times 512 = 512$ units ;
  - Channel-Mixing : $4.0 \times 512 = 2048$ units.
- **Total Parameters :** Approximately 2.25 million.

Increased Channel In this configuration, the width of the channel-mixing MLP is significantly increased to boost the model's ability to process intra-token features. While the token-mixing branch remains unchanged, the channel-mixing MLP's capacity is increased, enabling richer feature transformations at the cost of a higher parameter count.

- **MLP Ratio :** [0.5, 8.0]
  - Token-Mixing : $0.5 \times 512 = 256$ units ;
  - Channel-Mixing : $8.0 \times 512 = 4096$ units.
- **Total Parameters :** Approximately 4.29 million.

Balanced Ratio In this configuration, both the token-mixing and channel-mixing MLPs are allocated equal capacity by using an MLP ratio of [1.0, 1.0]. Although this balanced approach simplifies the model by treating spatial and feature processing symmetrically, it results in a substantial reduction in overall capacity, which may lead to underfitting.

- **MLP Ratio :** [1.0, 1.0]
  - Token-Mixing : $1.0 \times 512 = 512$ units ;
  - Channel-Mixing : $1.0 \times 512 = 512$ units.
- **Total Parameters :** Approximately 678K.

6.3 Discussion

We can view the MLP Ratio as a way to control the capacity of the model to capture interactions and features. As you increase the token-mixing layer, you will be able to capture more nuanced interactions between your patches or tokens. In other words, it's how patches communicate and subtly adjust each other's representations. On the other hand, the channel-mixing layer will be able to capture complex features within each individual patch. Basically, it's how detailed and informative the features will be extracted in the model (such as textures, edges, or patterns).

Increasing the Token-Mixing The increased token variant shows a slight improvement (test accuracy 77.2%) compared to the baseline (75.9%). The slight performance boost seen with the increased token configuration suggests that the baseline might have been slightly underpowered in its ability to mix tokens in the baseline configuration. Because the number of tokens is relatively small and its role is to coordinate global information, even a modest increase can help capture more nuanced interactions without a dramatic rise in overall parameters.

Increasing the Channel-Mixing Higher capacity (especially in the channel mixing branch) increases computational cost and training time per epoch because more multiplications and data movements are required. The lower training loss indicates enhanced capacity, but the similar validation and test performance suggest a trade-off where additional capacity may not translate into better generalization without more regularization.

Balancing the Ratio The balanced ratio model is computationally lighter and tends to achieve lower accuracies. This is mostly due to the fact that the channel capacity was reduced too much in the experiment. It might have been interesting to test a configuration with a higher balanced ratio, such as [4.0 ; 4.0]. The current balanced ratio limits the model's expressiveness, leading to underfitting.

7 Question 4.8

In this question, we compare the gradient flow (i.e., the norms of gradients at different layers) during backpropagation for three architectures : MLP, ResNet18, and MLPMixer. The configurations for each model are as follows :

— **MLP :** Input Size : 3072 ; Hidden Sizes : [1024, 512, 64, 64] ; Activation : ReLU.
— **ResNet18 :** Standard configuration of the ResNet18 model.
— **MLPMixer :** Patch Size : 4 ; Embedding Dimension : 256 ; Mixer Blocks : 4 ; Dropout : 0.0 ; Activation : GELU ; MLP Ratio : [0.5, 4.0].

7.1 Visualization

In order to analyse and understand the gradient flow of those models, I decided to generate figures to visualise the gradients. After my exploration, I ended up with 2 figures that I found interesting. First, a heatmap showing the evolution of the gradient norms for each layer across epochs. Then, aligned to its right, a horizontal bar chart that summarizes the mean and standard deviation of the gradient norms for each layer. This helped me understand at different levels what was going on in the gradients of my models.
Then I decided to remove some of the superfluous layers that didn't help with the visualisation process (such as layers with no parameters and normalization layers). Finally, I decided to split my data into two figures, one for the biases and one for the weights. I ended up only showing the weights in my report.

MLP Figure 10 shows the gradient flow of weights for the MLP architecture. In the heatmap, we observe a clear trend : the early layers (closer to input) have much darker coloration compared to later layers. This indicates that the magnitude of weight gradients in those input-proximal layers is significantly lower than in layers near the output. The output layer (final layer) shows the brightest intensities, as it directly receives the error signal and thus has the largest gradient updates. As we move backward toward the first layer, the colors fade, signaling that gradients are weakening as they propagate back. This pattern is consistent with vanishing gradients in a deep MLP. The earliest layers will learn slowly because their weight updates are tiny.

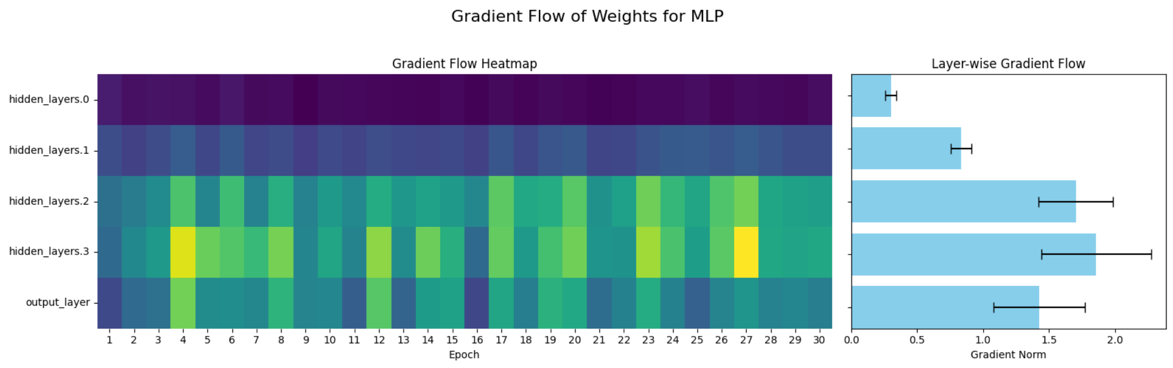


FIGURE 10 – Gradient flow of weights for the MLP architecture.

ResNet18 Figure 11 displays the gradient flow of weights for ResNet18. The gradient flow visualizations for ResNet18 illustrate how effectively this architecture maintains strong weight updates throughout its depth—even the deepest layers (close to the output) and the shallowest layers (close to input) all get their share of the error signal.
An interesting observation is the boundaries of residual blocks where a skip connection happens ; this momentarily intensifies, showing as a bright band in the figures. This demonstrates how the residual connections in ResNet models help boost the gradient error signal across the layers.

Note : The layer "layerX.0.shortcut_1" is actually the BatchNorm2d of the shortcut for each BasicBlock. This was the only normalization layer that wasn't removed from the figure because its name couldn't be filtered out.

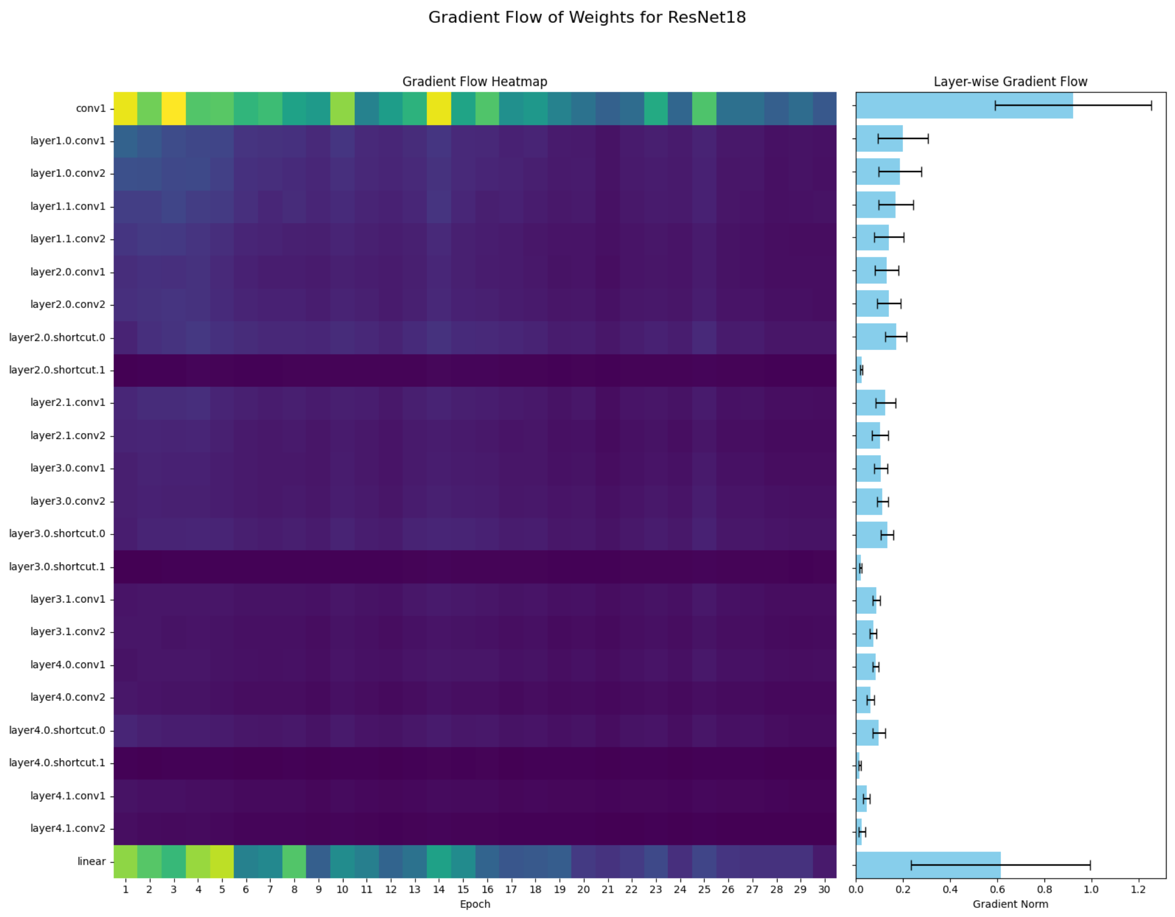


FIGURE 11 – Gradient flow of weights for the ResNet18 architecture.

MLPMixer Figure 12 presents the gradient flow of weights for the MLPMixer architecture (patch4 configuration). Overall, the gradient flow indicates a balanced distribution between the token-mixing and channel-mixing sub-layers. However, we can observe slight differences in gradient magnitudes between these sub-layers. If the token-mixing layers appear slightly darker than the subsequent channel-mixing layers (or vice versa), it may indicate that one type of sub-layer receives relatively smaller gradient updates. This interesting pattern could be due to the fact that channel-mixing layers, which mix features across channels, carry gradients more directly related to the loss, while token-mixing layers, which mix information across tokens, have gradients that are more diffused.

7.2 Discussion

In summary, examining gradient flow via heatmaps and weight-update distributions reveals that ResNet18's and MLPMixer's skip connections and residual blocks both foster strong, evenly-distributed gradients, enabling robust learning across all layers, even in deep networks. In contrast, the plain MLP, lacking these architectural aids, shows fading gradients in its early layers, which can hinder training. These visual insights underscore why ResNet-like architectures train deeper networks effectively and why even modern all-MLP architectures incorporate skip connections : balanced gradient flow is essential for fast, stable convergence.

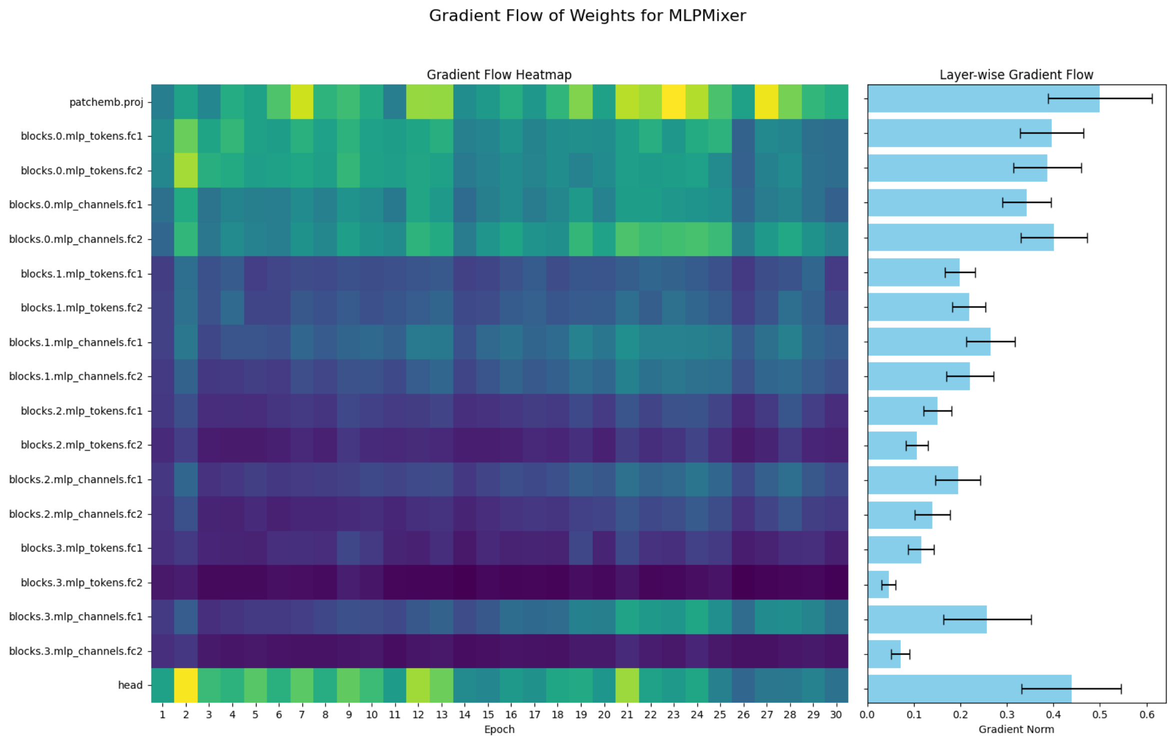


FIGURE 12 – Gradient flow of weights for the MLPMixer architecture (patch4 configuration).