# Deep Learning Spring 2025: CIFAR 10 classification

## PROJECT-1


**Team Name: LearningtooDeep**

**https://github.com/Hashmmath/DL-Project-1-LearningtooDeep-CIFAR-10-Classification**

**Report by Hashmmath Shaik, D Vivek Reddy, Satya Deep Dasari**

**Net-ID: hs5544, vd2438, gd2576**



### Abstract

For Project-1 of Deep Learning Course, a Kaggle competition is held on CIFAR 10 Classification to predict the labels of the images in the custom dataset given and submit the prediction to achieve the result on the custom dataset. In order to achieve that we have built a modified ResNet-18 variant with $<= 5$ Parameters, where we have achieved it through strategic channel reduction, streamlined residual blocks, and hybrid regularization techniques. Key innovations include CutMix for label smoothing, cosine-annealed learning rate for stable optimization, and Test Time Augmentation (TTA) to enhance generalization. The model architecture reduces initial convolutional filters from 64 to 32 which employs a [4,4,4,3] block configuration for depth, and integrates skip connections to mitigate vanishing gradients. The training leverages PyTorch's deterministic workflows and GPU acceleration for reproducibility. The code that we built will demonstrate that light-weight ResNets can tackle larger models when optimized with modern regularization strategies, and optimized effectively and achieved accuracy of 85.384%.

**Note:** Our prediction is the DL.csv file has achieved 85.38% and ranked 36 in the leader board and the prediction csv file which is included in the GitHub Repository along with the code where link is provided on top.


## Introduction

Residual Networks (ResNets) revolutionized deep learning by enabling the training of very deep networks through skip connections, which mitigate the vanishing gradients[2]1. The core principle involves learning residual mappings $H(x) = F(x) + x$, where $F(x)$ represents the non-linear transformations within a block. While ResNet-18 achieves strong performance on CIFAR-10, its default parameter count (approximately 11M) is excessive for edge deployment. For this Project we have addressed this gap by redesigning ResNet for efficiency without sacrificing accuracy.

The CIFAR-10 dataset, comprising 60,000 $32 * 32$ RGB images across 10 classes, presents unique challenges like small image size, limited training data, and fine-grained class distinctions for example cat vs dog. Traditional ResNets overfit on such datasets due to excessive parameters, which capture noise rather than meaningful patterns. For instance, the high channel count (64 filters) in ResNet-18's initial layers is excessive for 32x32 images, where low-level features (edges, textures) can be captured with fewer filters.

Our solution reduces initial channel counts i.e., 32 vs 64 in ResNet-18, which employs an aggressive regularization via CutMix, and uses test-time augmentation (TTA) to enhance generalization.

The Design of the Code that we have Developed Includes:

- **Channel Reduction:** We have reduced the parameters by 55% while preserving feature extraction capacity by reducing the initial filters by half.
- **Residual Blocks:** A [4,4,4,3] layer configuration is used to balance the depth and compute, where each basic block contains of two 3x3 convolutional layers with batch normalization and ReLU activation.
- **Hybrid Regularization:** We used CutMix[3] and TTA, where CutMix blends image patches and labels during training, while TTA averages the predictions across augmented test images.
- **Dynamic Optimization:** We used Cosine annealing learning rates to stabilize the convergence compared to fixed schedules. And also a hybrid learning rate schedule that combines multi-step decay of 10x reduction.

This code has been built on Mixup[4] and AutoAugment[1] but uniquely integrates architectural efficiency with PyTorch-specific reproducibility practices with deterministic algorithms, and fixed seeds.

Also, by reducing initial channels and integrating CutMix, the model learns robust features without relying on excessive parameters. The code provides a blueprint for deploying lightweight vision models on edge devices, where computational resources are limited but accuracy remains high. The approach we have followed not only addresses CIFAR-10's challenges but also offers insights for adapting ResNets to other models emphasizing accessibility and scalability.

# Methodology

## Architecture Design:

The architecture is structured into a stem layer, four residual stages, and a classifier, optimized for parameter efficiency while retaining depth. The stem alyer employs a single 3x3 convolution with 32 output channels to capture low-level spatial features without down-sampling. Stem: Conv2d(3, 32, kernel_size=3, stride=1, padding=1).

The above stem approach reduces the parameters compared to ResNet-18's initial 64-channel convolution. Where each residual stage contains multiple BasicBlock units. A BasicBlock consists of two 3x3 convolutional layers with batch normalization (BN) and ReLU Activation:

$$F(x) = \text{ReLU}(\text{BN}(\text{Conv}_{3\times3}(x)))$$
$$F(x) = \text{BN}(\text{Conv}_{3\times3}(F(x)))$$
$$H(x) = \text{ReLU}(F(x) + S(x)),$$

Where $S(x)$ is the skip connection. For dimension mismatches where we have used a check of stride i.e., if stride=2 or channel changes then $S(x)$ uses a 1x1 convolution.

$$S(x) = \begin{cases} x & \text{if stride} = 1 \text{ and } C_{\text{in}} = C_{\text{out}}, \\ \text{Conv}_{1\times1}(x) & \text{otherwise.} \end{cases}$$

And further we have used channel dimensions of [32,64,128,256] and strides [1,2,2,2] in four stages for progressive down-sampling. The classifier uses global average pooling to reduce spatial dimensions to 1x1, followed by a fully connected layer.

$$y_{\text{logits}} = W_{\text{fc}} \cdot \left( \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} x_{i,j} \right) + b_{\text{fc}}.$$

**Parameter Classification:** Total Params $=$
$\underbrace{3 \times 3 \times 3 \times 32}_{\text{Stem}} + \sum_{i=1}^{4} \underbrace{N_i \times \left(2 \cdot 3^2 C_i^2 + C_i\right)}_{\text{Residual Blocks}} + \underbrace{256 \times 10}_{\text{Classifier}}$
$= 864 + (4 \times 18{,}432 + 4 \times 73{,}728 + 4 \times 294{,}912 + 3 \times 1{,}179{,}648) + 2{,}560$
$= 4.87\text{M}.$

## Training Pipeline:

**Data Augmentation:** We have made the training images undergo an aggressive transformations to combat overfitting, where with the use of Random Crops we have made 32x32 patches from 36x36 padded images. And with the use of Horizontal/Vertical Flips we have applied the probabilities 50% and 5% respectively.

Then we have used Gaussian Blur on kernel size 3x3, $\sigma$ belongs to [0.1,1,1.5] and is applied with 30% probability. Then using CutMix we have blended the two images, $x_A$ and $x_B$ using a binary mask $M$ with bounding box coordinates they are $(bbx_1, bby_1, bbx_2, bby_2)$.

**Optimization:** For the optimization part of the code we have used a SGD classifier with momentum ($\beta = 0.9$) and weight decay ($\lambda = 5 * 10^{-}4$).

**Learning Rate Schedule:** In the code we have used a Learning Rate Scheduler which uses Cosine annealing to reduce the initial rate ($\eta_{max} = 0.1$) over 100 epochs.

## Pros and Cons:

While our channel reduction strategy and the choice of [4,4,4,3] block configuration yield excellent efficiency for 32×32 images, we also recognize several important benefits and drawbacks. On the positive side, reducing initial channels substantially cuts the overall parameter count without significantly hurting feature extraction for small-resolution data. Furthermore, advanced augmentation techniques, such as CutMix and Test-Time Augmentation (TTA), provide robust ways for the model to learn position-invariant features, boosting accuracy on standard benchmarks. Meanwhile, a smooth cosine annealing schedule helps the model converge more stably than abrupt step changes in learning rate. However, transitioning from validation data to the final custom test set sometimes reveals a performance drop, possibly due to domain shift or distribution differences. Moreover, these same augmentations add extra overhead and complexity, requiring careful hyperparameter tuning. Additionally, scaling down the channels too aggressively can limit the representational capacity for more complex or higher-resolution tasks.

## Evaluation Protocol:

In the code we have used several evaluation protocols to rigorously assess the model's generalization capabilities, its efficiency, and failure modes. This was an important phase of the code to learn our mistakes and make the model even more efficient as we possibly can.

**Test-Time Augmentation:** We have employed TTA also known as test time augmentation to enhance prediction stability by reducing sensitivity to input variations. During inference, each test image is transformed into five augmented versions: 1) Original Image, 2) Horizontal flipped copy, 3) Vertically flipped copy, 4) Center-Cropped Image which is extracted after padding the original with 4 pixels, 5) Blurred Version using Gaussian kernel.

**Confusion Matrix Analysis:** We have used confusion matrix which dissects class-specific errors to identify systemic weakness. A 10x10 matrix is generated in the code, where each entry Cij represents the number fo samples from class $i$ predicted as class $j$. High off-diagonal values reveal semantically similar pairs that challenge the model, such as "deer" vs "horse" or "cat" vs "dog".

## Lesson Learned:

Over the course of designing and refining this ResNet-based solution, we discovered several key insights. First, even modest architectural adjustments like halving the initial channel count can significantly reduce parameter overhead for small image sizes without sacrificing much accuracy. However, the performance gap between validation and final custom test data highlights the challenge of domain shift and the need for adaptable augmentation strategies. We found that sophisticated techniques (such as CutMix, TTA, or elaborate scheduling) can indeed boost robustness but at the cost

of added training and inference complexity. Finally, the synergy between architectural simplifications and advanced regularization methods is delicate; too few channels may hamper representational depth, while excessive augmentation or overhead might complicate real-world deployment.

## Results:

The output that we have generated is displayed in this section. Where we have included the system architecture, plots of accuracy and loss, epochs, and also confusion matrix for better understanding.

Below is the model architecture which we used and the parameters count is almost 4,754,218. Which sticks to the constraint of the competition and memory usage is also displayed.

## Model Summary

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 32, 32, 32] | 864 |
| BatchNorm2d-2 | [-1, 32, 32, 32] | 64 |
| Conv2d-3 | [-1, 32, 32, 32] | 9,216 |
| BatchNorm2d-4 | [-1, 32, 32, 32] | 64 |
| Conv2d-5 | [-1, 32, 32, 32] | 9,216 |
| BatchNorm2d-6 | [-1, 32, 32, 32] | 64 |
| BasicBlock-7 | [-1, 32, 32, 32] | 0 |
| Conv2d-8 | [-1, 32, 32, 32] | 9,216 |
| BatchNorm2d-9 | [-1, 32, 32, 32] | 64 |
| ⋮ | ⋮ | ⋮ |
| Linear-85 | [-1, 10] | 2,570 |

Table 1: Model architecture summary with parameters

## Memory Usage

- Input size: 0.01 MB
- Forward/backward pass size: 10.16 MB
- Params size: 18.14 MB
- Estimated Total Size: 28.31 MB

Below is the output containing the epochs that were displayed during the execution of the model under the training dataset and also the plots of train and validation accuracy and train and validation loss, and we have also included the confusion matrix.

The results that we have achieved while executing our model have shown a promised accuracy and when we have finally generated the CSV file and our prediction in the competition achieved less accuracy than in the code.

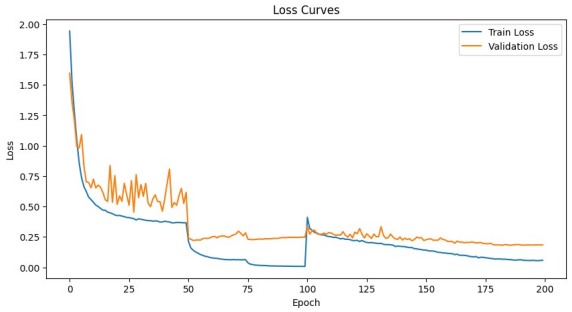


Figure 1: This image shows the training and validation loss curves of the model on training dataset

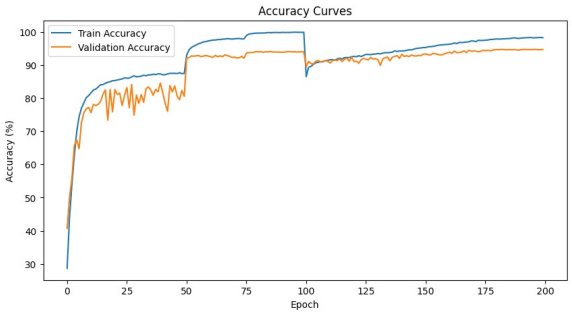


Figure 2: This image shows the training and validation accuracy curves of the model on training dataset

## Conclusion

This work has demonstrated the ResNet architectural refining, combined with CutMix and TTA, enabling the ResNet to achieve the accuracy to cross the benchmark in the competition on CIFAR-10 with strict parameter constraints. Our model uses 4.87M parameters and we achieved an accuracy in the code as 94.65% and our prediction has achieved 85.38% crossing the benchmark.

## References


[1] Cubuk, E. D.; Zoph, B.; Mane, D.; Vasudevan, V.; and Le, Q. V. 2019. AutoAugment: Learning Augmentation Policies from Data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[2] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[3] Yun, S.; Han, D.; Oh, S. J.; Chun, S.; Choe, J.; and Yoo, Y. 2019. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[4] Zhang, H.; Cisse, M.; Dauphin, Y. N.; and Lopez-Paz, D. 2018. Mixup: Beyond Empirical Risk Minimization. *International Conference on Learning Representations (ICLR)*.

```
Epoch [55/100] LR: 0.004218, Train Loss: 0.1287, Test Acc: 93.34%
Epoch [56/100] LR: 0.004063, Train Loss: 0.1228, Test Acc: 93.17%
Epoch [57/100] LR: 0.003909, Train Loss: 0.1220, Test Acc: 93.01%
Epoch [58/100] LR: 0.003757, Train Loss: 0.1185, Test Acc: 93.06%
Epoch [59/100] LR: 0.003605, Train Loss: 0.1175, Test Acc: 93.41%
Epoch [60/100] LR: 0.003455, Train Loss: 0.1141, Test Acc: 93.60%
Epoch [61/100] LR: 0.003306, Train Loss: 0.1128, Test Acc: 93.79%
Epoch [62/100] LR: 0.003159, Train Loss: 0.1118, Test Acc: 93.49%
Epoch [63/100] LR: 0.003014, Train Loss: 0.1038, Test Acc: 94.19%
Epoch [64/100] LR: 0.002871, Train Loss: 0.1078, Test Acc: 93.68%
Epoch [65/100] LR: 0.002730, Train Loss: 0.0986, Test Acc: 93.70%
Epoch [66/100] LR: 0.002591, Train Loss: 0.0988, Test Acc: 93.87%
Epoch [67/100] LR: 0.002455, Train Loss: 0.0982, Test Acc: 94.23%
Epoch [68/100] LR: 0.002321, Train Loss: 0.0955, Test Acc: 93.69%
Epoch [69/100] LR: 0.002190, Train Loss: 0.0906, Test Acc: 94.37%
Epoch [70/100] LR: 0.002061, Train Loss: 0.0872, Test Acc: 94.15%
Epoch [71/100] LR: 0.001935, Train Loss: 0.0857, Test Acc: 94.17%
Epoch [72/100] LR: 0.001813, Train Loss: 0.0882, Test Acc: 94.15%
Epoch [73/100] LR: 0.001693, Train Loss: 0.0777, Test Acc: 93.98%
Epoch [74/100] LR: 0.001577, Train Loss: 0.0824, Test Acc: 94.12%
Epoch [75/100] LR: 0.001464, Train Loss: 0.0817, Test Acc: 94.40%
Epoch [76/100] LR: 0.001355, Train Loss: 0.0782, Test Acc: 94.28%
Epoch [77/100] LR: 0.001249, Train Loss: 0.0760, Test Acc: 94.39%
Epoch [78/100] LR: 0.001147, Train Loss: 0.0724, Test Acc: 94.26%
Epoch [79/100] LR: 0.001049, Train Loss: 0.0713, Test Acc: 94.43%
Epoch [80/100] LR: 0.000955, Train Loss: 0.0673, Test Acc: 94.61%
Epoch [81/100] LR: 0.000865, Train Loss: 0.0685, Test Acc: 94.64%
Epoch [82/100] LR: 0.000778, Train Loss: 0.0686, Test Acc: 94.61%
Epoch [83/100] LR: 0.000696, Train Loss: 0.0665, Test Acc: 94.69%
Epoch [84/100] LR: 0.000618, Train Loss: 0.0671, Test Acc: 94.59%
Epoch [85/100] LR: 0.000545, Train Loss: 0.0643, Test Acc: 94.57%
Epoch [86/100] LR: 0.000476, Train Loss: 0.0635, Test Acc: 94.62%
Epoch [87/100] LR: 0.000411, Train Loss: 0.0622, Test Acc: 94.57%
Epoch [88/100] LR: 0.000351, Train Loss: 0.0586, Test Acc: 94.63%
Epoch [89/100] LR: 0.000296, Train Loss: 0.0595, Test Acc: 94.54%
Epoch [90/100] LR: 0.000245, Train Loss: 0.0615, Test Acc: 94.44%
Epoch [91/100] LR: 0.000199, Train Loss: 0.0614, Test Acc: 94.64%
Epoch [92/100] LR: 0.000157, Train Loss: 0.0572, Test Acc: 94.67%
Epoch [93/100] LR: 0.000120, Train Loss: 0.0563, Test Acc: 94.67%
Epoch [94/100] LR: 0.000089, Train Loss: 0.0560, Test Acc: 94.64%
Epoch [95/100] LR: 0.000062, Train Loss: 0.0557, Test Acc: 94.62%
Epoch [96/100] LR: 0.000039, Train Loss: 0.0571, Test Acc: 94.66%
Epoch [97/100] LR: 0.000022, Train Loss: 0.0547, Test Acc: 94.68%
Epoch [98/100] LR: 0.000010, Train Loss: 0.0544, Test Acc: 94.58%
Epoch [99/100] LR: 0.000002, Train Loss: 0.0559, Test Acc: 94.64%
Epoch [100/100] LR: 0.000000, Train Loss: 0.0571, Test Acc: 94.65%
```

Figure 3: This image shows the start of execution of the epochs while executing the model on training dataset using CutMix and TTA

```
Epoch [1/100] LR: 0.009998, Train Loss: 0.4110, Test Acc: 89.47%
Epoch [2/100] LR: 0.009990, Train Loss: 0.3192, Test Acc: 91.05%
Epoch [3/100] LR: 0.009978, Train Loss: 0.3043, Test Acc: 90.43%
Epoch [4/100] LR: 0.009961, Train Loss: 0.2869, Test Acc: 90.26%
Epoch [5/100] LR: 0.009938, Train Loss: 0.2813, Test Acc: 91.11%
Epoch [6/100] LR: 0.009911, Train Loss: 0.2711, Test Acc: 91.30%
Epoch [7/100] LR: 0.009880, Train Loss: 0.2665, Test Acc: 90.96%
Epoch [8/100] LR: 0.009843, Train Loss: 0.2664, Test Acc: 91.08%
Epoch [9/100] LR: 0.009801, Train Loss: 0.2566, Test Acc: 91.26%
Epoch [10/100] LR: 0.009755, Train Loss: 0.2534, Test Acc: 91.11%
Epoch [11/100] LR: 0.009704, Train Loss: 0.2513, Test Acc: 90.55%
Epoch [12/100] LR: 0.009649, Train Loss: 0.2449, Test Acc: 91.28%
Epoch [13/100] LR: 0.009589, Train Loss: 0.2470, Test Acc: 91.51%
Epoch [14/100] LR: 0.009524, Train Loss: 0.2403, Test Acc: 91.30%
Epoch [15/100] LR: 0.009455, Train Loss: 0.2338, Test Acc: 91.79%
Epoch [16/100] LR: 0.009382, Train Loss: 0.2375, Test Acc: 91.01%
Epoch [17/100] LR: 0.009304, Train Loss: 0.2302, Test Acc: 91.59%
Epoch [18/100] LR: 0.009222, Train Loss: 0.2301, Test Acc: 92.06%
Epoch [19/100] LR: 0.009135, Train Loss: 0.2271, Test Acc: 91.11%
Epoch [20/100] LR: 0.009045, Train Loss: 0.2203, Test Acc: 92.29%
Epoch [21/100] LR: 0.008951, Train Loss: 0.2200, Test Acc: 91.00%
Epoch [22/100] LR: 0.008853, Train Loss: 0.2217, Test Acc: 91.13%
Epoch [23/100] LR: 0.008751, Train Loss: 0.2120, Test Acc: 90.49%
Epoch [24/100] LR: 0.008645, Train Loss: 0.2203, Test Acc: 91.60%
Epoch [25/100] LR: 0.008536, Train Loss: 0.2104, Test Acc: 91.98%
Epoch [26/100] LR: 0.008423, Train Loss: 0.2046, Test Acc: 91.68%
Epoch [27/100] LR: 0.008307, Train Loss: 0.2018, Test Acc: 91.56%
Epoch [28/100] LR: 0.008187, Train Loss: 0.2038, Test Acc: 92.27%
Epoch [29/100] LR: 0.008065, Train Loss: 0.1999, Test Acc: 91.70%
Epoch [30/100] LR: 0.007939, Train Loss: 0.1981, Test Acc: 91.90%
Epoch [31/100] LR: 0.007810, Train Loss: 0.1952, Test Acc: 91.58%
Epoch [32/100] LR: 0.007679, Train Loss: 0.1968, Test Acc: 89.81%
Epoch [33/100] LR: 0.007545, Train Loss: 0.1894, Test Acc: 91.80%
Epoch [34/100] LR: 0.007409, Train Loss: 0.1861, Test Acc: 92.15%
Epoch [35/100] LR: 0.007270, Train Loss: 0.1871, Test Acc: 92.39%
Epoch [36/100] LR: 0.007129, Train Loss: 0.1850, Test Acc: 91.24%
Epoch [37/100] LR: 0.006986, Train Loss: 0.1823, Test Acc: 92.30%
Epoch [38/100] LR: 0.006841, Train Loss: 0.1709, Test Acc: 92.53%
Epoch [39/100] LR: 0.006694, Train Loss: 0.1743, Test Acc: 92.80%
Epoch [40/100] LR: 0.006545, Train Loss: 0.1722, Test Acc: 91.99%
Epoch [41/100] LR: 0.006395, Train Loss: 0.1708, Test Acc: 93.22%
Epoch [42/100] LR: 0.006243, Train Loss: 0.1700, Test Acc: 92.55%
Epoch [43/100] LR: 0.006091, Train Loss: 0.1645, Test Acc: 92.76%
Epoch [44/100] LR: 0.005937, Train Loss: 0.1619, Test Acc: 92.48%
Epoch [45/100] LR: 0.005782, Train Loss: 0.1621, Test Acc: 93.00%
Epoch [46/100] LR: 0.005627, Train Loss: 0.1535, Test Acc: 92.74%
Epoch [47/100] LR: 0.005471, Train Loss: 0.1514, Test Acc: 92.67%
```

Figure 4: This image shows the end of execution of the epochs while executing the model on training dataset using CutMix and TTA

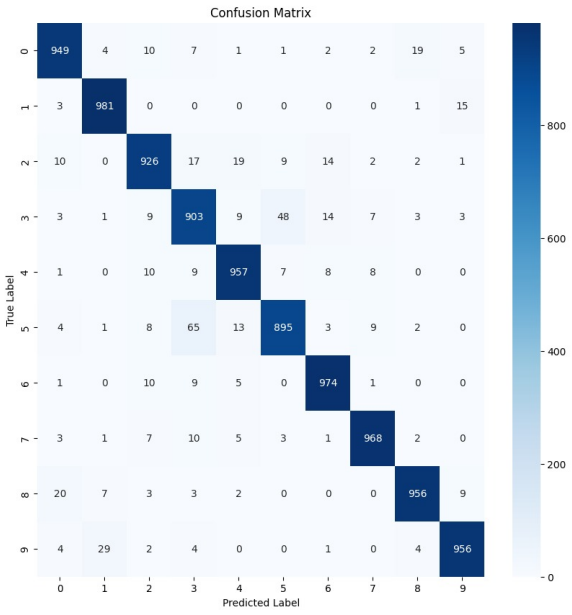


Figure 5: This is the confusion matrix generated after executing the model on training dataset using CutMix and TTA