Assignment 1 Report

Final Report

BARBIER Nathan, ERASMUS STUDENT

I. Introduction

This assignment explores image classification on CIFAR-10 using PyTorch. We implement a full training and evaluation pipeline to compare three models: a custom CNN, ResNet-18, and a Vision Transformer. The study focuses on training strategies (augmentation, regularization, optimization) and compares results to highlight trade-offs between CNNs and Transformers.

II. Methods

A. Dataset and Preprocessing

We used the Python version of CIFAR-10, split into 40,000 training, 10,000 validation, and 10,000 test images. Each image is 32×32 pixels in RGB format, and the dataset is balanced with 6,000 images per class.
Images were loaded using a custom CIFAR10Dataset class. Labels and image shapes were verified according to the assignment instructions. Class order and RGB correctness were also confirmed using the provided viz.py script.

B. Data Augmentation and Transforms

Data augmentation increases the diversity of the training set by applying transformations to input images. This helps prevent overfitting by forcing the model to generalize across variations that do not alter class identity (e.g. flipped or slightly shifted images). We applied random horizontal flips and random resized crops on the training set.

```
train_transform = v2.Compose([
    v2.ToImage(),
    v2.RandomHorizontalFlip(),
    v2.RandomResizedCrop(size=(32, 32), scale=(0.8, 1.0)),
    v2.ToDtype(torch.float32, scale=True),
    v2.Normalize(mean=[0.485, 0.456, 0.406],
                 std=[0.229, 0.224, 0.225]),
])
```

C. Loss and Metrics

The loss function used in this assignment is the **cross-entropy loss**, which is widely applied in classification tasks. Its role is to quantify how well the predicted class scores match the ground-truth labels. Given a vector of logits $s = [s_1, s_2, \dots, s_C]$ and a true label $y \in \{0, \dots, C-1\}$, the cross-entropy is defined as:

$$\mathcal{L} = -\log\left(\frac{e^{s_y}}{\sum_{j=1}^{C} e^{s_j}}\right) = -s_y + \log\left(\sum_{j=1}^{C} e^{s_j}\right)$$

This formulation applies a softmax over the logits to turn them into a probability distribution, and penalizes the log-likelihood of the correct class. To use PyTorch's CrossEntropyLoss, the model must output raw scores (logits) without a softmax layer, and the labels must be integers representing class indices, not one-hot vectors. These conditions are met in our setup: the model ends with a linear layer that outputs logits, and the CIFAR10Dataset returns integer labels.
To evaluate performance, we compute **accuracy** and **per-class accuracy**. Overall accuracy is the ratio of correct predictions:

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\hat{y}_i = y_i]$$

Per-class accuracy, also known as mean class accuracy, calculates accuracy within each class independently and averages the results. While both metrics are similar on CIFAR-10 due to its balanced class distribution, per-class accuracy remains more informative in imbalanced settings.

III. Training and Validation

A. Experimental Setup

The training was conducted using PyTorch, with custom training scripts for each model. Hyperparameters and optimizer choices were adapted to the architecture. We used the AdamW optimizer for most models, configured with a learning rate of 0.001, amsgrad=True, and weight decay either 10^{-4} or 5×10^{-4}. For CNNs and ResNet-18, we also tested SGD with momentum, which can lead to more stable updates in convolutional architectures.
To manage the learning rate during training, we used exponential decay (ExponentialLR) with a decay factor $\gamma = 0.9$, and in some cases, cosine annealing (CosineAnnealingLR) for smoother convergence over

longer runs, especially for the Vision Transformer which required more epochs to train effectively.

Regularization was essential to improve model generalization and prevent overfitting. It was applied through two mechanisms: **weight decay**, which penalizes large weights and encourages simpler models, and **dropout**, which randomly deactivates neurons during training to reduce reliance on specific features. These techniques were particularly important for ViTs, which lack the inductive biases of CNNs and are more prone to overfitting. Models were trained for a sufficient number of epochs to ensure convergence, with the main hyperparameters (dropout rate, weight decay, scheduler, and epochs) summarized in the table below.

Model	Dropout	Weight Decay	Scheduler	Epochs
CNN	0.5	1e-4	ExponentialLR	30
ResNet-18	0.3	5e-4	ExponentialLR	30
ViT	0.1	1e-4	CosineAnnealing	120

B. Experimental Setup

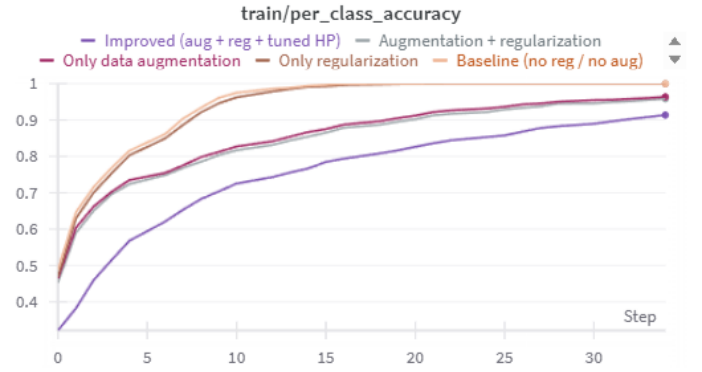
The training and validation process was managed by the ImgClassificationTrainer class, which handled the training loop, evaluation steps, and model checkpointing. Each epoch involved forward and backward passes using cross-entropy loss, with metrics (accuracy and per-class accuracy) computed throughout. Every few epochs, the model was evaluated on the validation set. If its per-class accuracy improved, the model was saved. This ensured the best generalizing model was retained and acted as a simple form of early stopping. Metrics were logged with Weights & Biases for monitoring and analysis.

IV. Experiments and Results

A. Training vs Validation Curves (ResNet-18)

To evaluate the impact of regularization and data augmentation strategies, we trained five variants of the ResNet-18 architecture. These include a baseline model without any regularization or augmentation, a version with dropout and weight decay only, another with augmentation only, one combining both techniques, and a final variant using the same combination with additional hyperparameter tuning (denoted "Improved"). All training runs were logged using Weights & Biases for consistent tracking.

The plots presented below illustrate the evolution of accuracy, loss, and per-class accuracy over 35 epochs, for both training and validation phases.



The trends reveal clear differences in generalization behavior and overfitting depending on the strategy used.

Per-Class Accuracy Analysis and Overfitting Behavior

The evolution of per-class accuracy offers a nuanced view of each model's ability to generalize across all CIFAR-10 classes. When comparing training and validation performance, the degree of **overfitting** becomes evident through the gap between the two curves and the plateauing behavior observed on validation data.

1. Baseline model (no augmentation / no regularization)
This model exhibits the most pronounced overfitting. Its training per-class accuracy rises rapidly and exceeds 95%, indicating that the network fits the training data very effectively. However, the validation curve flattens early, around 74–75%, and does not improve beyond that, despite continued training. The widening gap between training and validation accuracy is a textbook symptom of overfitting: the model learns to memorize the training set, including noise or specific patterns, but fails to generalize to unseen samples.

2. Only regularization (dropout + weight decay)
The addition of regularization improves the situation slightly. The training accuracy curve is still steep, reaching over 95%, but the validation curve climbs higher than the baseline and levels off around 76–77%. While a gap remains, it is somewhat reduced. This shows that regularization slows down the learning process and penalizes complex models, forcing the network to find simpler, more generalizable patterns. Nevertheless, the effect is modest — regularization alone mitigates overfitting partially but not completely.

3. Only data augmentation (random crop + flip)
Here, the pattern shifts more significantly. The training curve grows more slowly and stabilizes around 92%, which is lower than previous models, indicating that augmentation acts as a form of input-level noise — making it harder for the model to overfit specific samples. The validation curve, by contrast, improves consistently throughout training and reaches the highest value among all models (just above 80%). This smaller train/val gap suggests much better generalization, as the model is exposed to a more diverse set of training variations that improve its robustness.

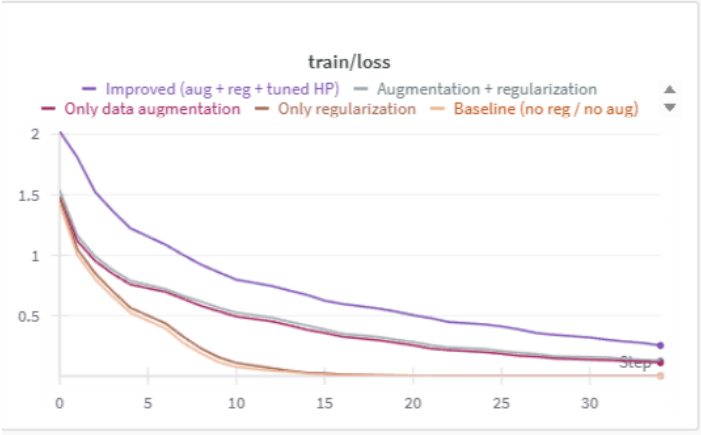
4. Data augmentation + regularization
Combining the two strategies yields a well-balanced training dynamic. The training accuracy grows moderately (up to 93–94%), and the validation curve steadily increases with minimal divergence. The overfitting is very limited in this setup, with the best compromise between learning capacity and generalization. The model avoids memorizing training examples thanks to both noise in the input and constraints in the weights.

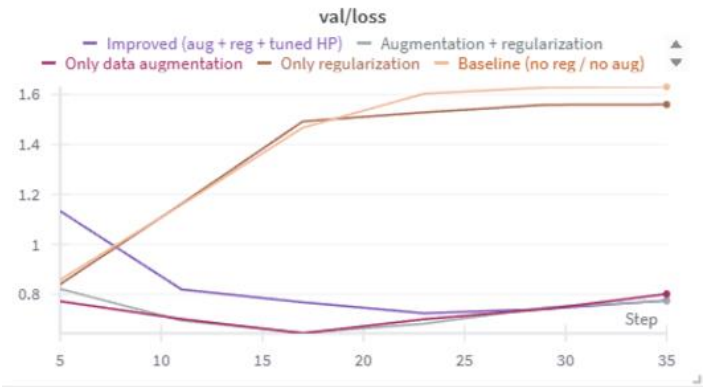
5. Improved (augmentation + regularization + tuned hyperparameters)
The improved variant, with additional tuning (likely adjusted learning rate, scheduler, or batch size), starts off slower than the others but stabilizes with a validation per-class accuracy close to the previous combined model. The slight underperformance in the early epochs suggests more conservative learning, which can be beneficial long-term. The final train/val gap remains small, indicating minimal overfitting and stable generalization. This configuration is likely the most robust and consistent across runs.

In a supervised learning setup, **the training set** is used to fit the model, while the validation set helps tune hyperparameters and detect overfitting during training. The test set, on the other hand, is held out until the very end and serves as the only true measure of generalization to unseen data. All test evaluations below were performed on the best model selected using validation per-class accuracy.
The table below summarizes the final test accuracy for each ResNet-18 variant:

Model	Test per class Accuracy
Baseline (no reg / no aug)	0.761
Only regularization	0.7534
Only data augmentation	0.7912
Augmentation + regularization	0.7955
Improved (aug + reg + tuned HP)	0.7976

We observe that data augmentation alone improves test accuracy more than regularization alone. However, the combination of both techniques leads to a larger performance gain, confirming the results seen on the validation set. The best-performing model is the improved version, which adds tuned hyperparameters (e.g., scheduler, dropout, learning rate), suggesting that careful optimization can bring incremental improvements even on top of well-regularized models.
The difference between validation and test accuracy remains relatively small for all variants, confirming that the validation set provided a good estimate of generalization performance.





All models reduce training loss steadily, but the baseline and regularization-only variants show clear overfitting, with validation loss increasing after epoch 10. Data augmentation helps flatten the validation loss, while the

combination of augmentation and regularization maintains the lowest and most stable curve. The improved model has slightly higher training loss but better generalization, as shown by its consistently low validation loss.

V. Vision Transformers vs Convolutional Neural Networks

Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs) adopt fundamentally different principles for visual recognition. CNNs rely on local connectivity and shared filters to extract spatial hierarchies of features from input images. Their architecture embeds strong inductive biases such as locality, translation equivariance, and compositionality, which makes them data-efficient and highly effective on tasks like CIFAR-10. Our CNN implementation (in cnn.py) follows a VGG-like design composed of three convolutional blocks with increasing depth and max pooling for spatial reduction. Each block includes two convolutional layers with batch normalization and ReLU activation:

```
nn.Conv2d( in_channels: 32,  out_channels: 64, kernel_size=3, padding=1),
nn.BatchNorm2d(64),
nn.ReLU(inplace=True),
nn.Conv2d( in_channels: 64,  out_channels: 64, kernel_size=3, padding=1),
nn.BatchNorm2d(64),
nn.ReLU(inplace=True),
nn.MaxPool2d(kernel_size=2, stride=2),  # 16x16 -> 8x8
```

After the last pooling layer (output shape 4×4×128), the features are flattened and passed through a fully connected classifier with two dropout layers:

```
self.classifier = nn.Sequential(
    nn.Dropout(p=0.5),
    nn.Linear(128 * 4 * 4,  out_features: 256),
    nn.ReLU(inplace=True),
    nn.Dropout(p=0.5),
    nn.Linear( in_features: 256, num_classes)
```

This model achieves strong early convergence and is resistant to overfitting due to its limited capacity and aggressive dropout.

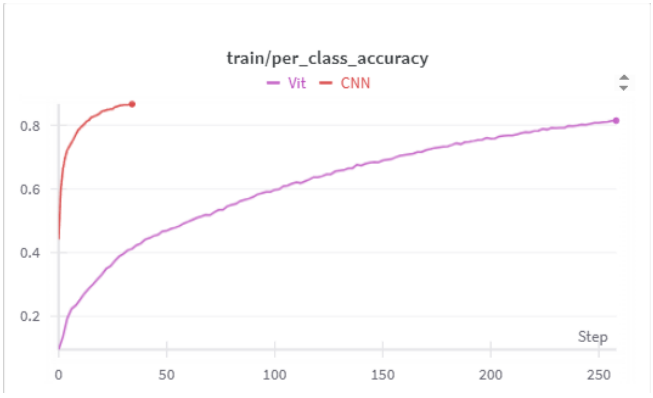
In contrast, our ViT implementation (vit.py) replaces convolutions with patch embeddings and a transformer encoder. The input image is split into fixed-size patches via a convolution with stride equal to patch size.

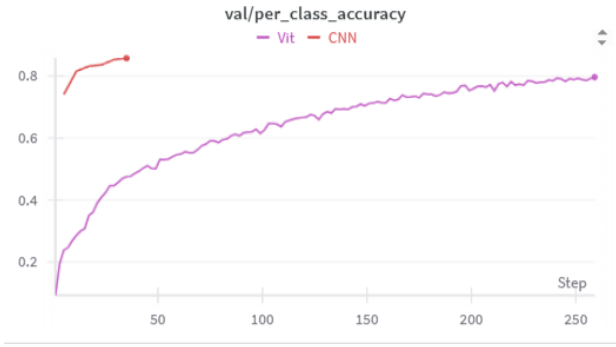
Each patch becomes a token, and a [CLS] token is prepended. Positional encodings are added, and the sequence is processed by multi-head self-attention layers. The final prediction is based only on the output of the [CLS] token:

```
cls_output = x[:, 0]
logits = self.mlp_head(cls_output)
```

CNNs converge quickly thanks to built-in spatial priors, performing well with limited tuning. ViTs, while slower to train and more data-dependent, can outperform CNNs when trained longer with proper regularization, as confirmed in our experiments.

The plots below compare the training and validation per-class accuracy of the CNN and Vision Transformer models, highlighting their respective convergence behaviors and generalization dynamics.





The CNN model quickly reaches high per-class accuracy on both training and validation sets within 30 epochs, showing fast convergence and limited overfitting. In contrast, the Vision Transformer requires many more epochs (250) to reach comparable validation accuracy. Initial experiments showed that using a high learning rate caused unstable training for the ViT, preventing convergence. To address this, we reduced the learning rate and compensated with a longer training schedule. As seen in the plots, the ViT improves steadily over time and ultimately matches the CNN's generalization performance, despite slower learning dynamics.

VI. Conclusion

This project provided hands-on experience with training and evaluating deep learning models for image classification. By comparing CNNs, ResNet-18, and Vision Transformers, we explored the effects of architecture, regularization, and training strategies. The results highlighted the strengths of each approach and the importance of tailored optimization for model performance.