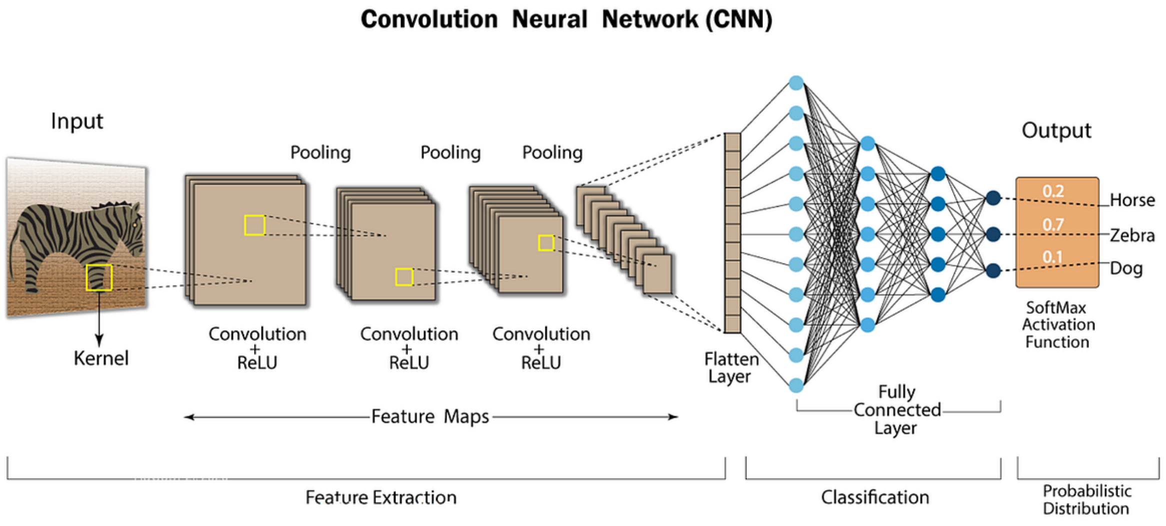
Computer Vision 1
Final Laboratory Part 2 Report

University of Amsterdam

Student Name	**Student ID**
Ippokratis Pantelidis	16124006
Robert-Ştefan Sofroni	16300645
Andor Károly Bodgál	14339617

October 27, 2025



Contents

1 Introduction

This second part of the laboratory project focuses on designing an image classification system using *Convolutional Neural Networks* (CNNs), building upon the first part that employed a classical Bag of Visual Words (BoVW) approach.

Experiments are conducted on the **CIFAR-100** dataset, containing 60,000 color images (32×32 pixels) across 100 categories, and later on a subset of the higher-resolution **STL-10** dataset. Two architectures are implemented:

- **TwoLayerNet** – a simple fully connected network with ReLU activation.
- **ConvNet** – a LeNet-5–inspired CNN adapted for RGB images.

Both models are trained from scratch on CIFAR-100 using PyTorch, with extensive hyperparameter tuning (learning rate, batch size, weight decay, optimizer) to improve accuracy and generalization. Techniques such as Dropout and Batch Normalization are also tested to stabilize training and reduce overfitting.

Finally, the pre-trained ConvNet is **fine-tuned on a five-class subset of STL-10** (*bird, deer, dog, horse, monkey*) by replacing the final layer and retraining on the new data. Performance is assessed through accuracy metrics and feature visualization using t-SNE.

2 Methodology

2.1 Data Preparation

The experiments use the **CIFAR-100** dataset, consisting of 60,000 color images (32×32 pixels) across 100 classes grouped into 20 *superclasses*, each containing five related subclasses (e.g., *large carnivores*, *vehicles*, *trees*). The dataset is split into 50,000 training and 10,000 test images.

Data loading and preprocessing were handled using PyTorch utilities. Images were converted to tensors, normalized, and organized into training and test loaders for efficient batching during training and evaluation.

To illustrate the dataset's structure, a visualization of three superclasses and their corresponding five subclasses was created (Figure 1). This helps highlight the intra-class variability and visual diversity present in CIFAR-100.

2.2 Network Architectures

Two architectures were implemented to compare dense and convolutional representations on CIFAR-100: a fully connected **TwoLayerNet** and a LeNet-5–inspired **ConvNet**. Both were trained under identical settings for fair evaluation.

2.2.1 TwoLayerNet

The **TwoLayerNet** is a baseline fully connected model operating on flattened image vectors of size $32 \times 32 \times 3 = 3072$. It consists of two linear layers with a ReLU activation

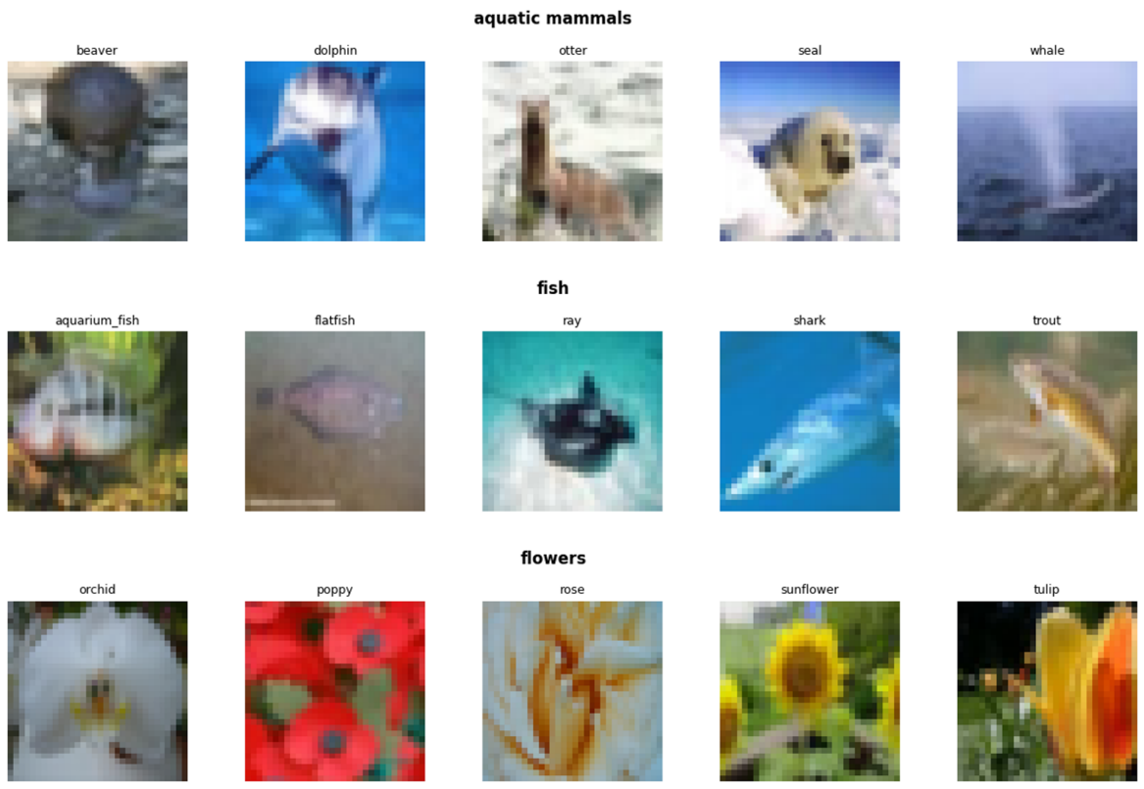


Figure 1: Sample visualization of three CIFAR-100 superclasses and their five subclasses. Each row corresponds to one superclass, and each column shows an example from its subclasses.

in between:

$$3072 \xrightarrow{\text{ReLU}} H \rightarrow 100.$$

The model ignores spatial structure, treating pixels independently, and thus serves as a benchmark for assessing the benefit of convolutional feature extraction. A summary of its architecture is shown in Figure 2.

```
TwoLayerNet(
  (fc1): Linear(in_features=3072, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=100, bias=True)
)
```

Figure 2: Architecture of the **TwoLayerNet**.

2.2.2 ConvNet

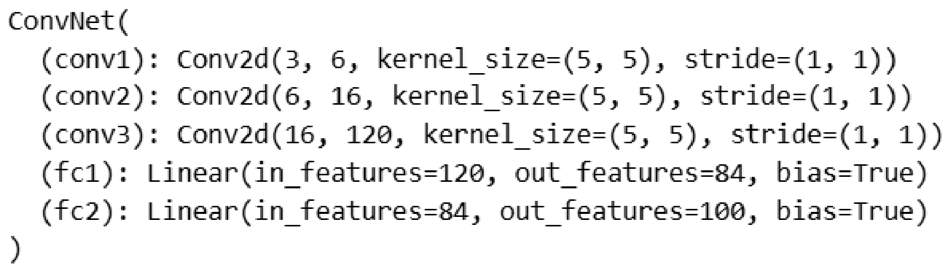
The **ConvNet** follows the classical *LeNet-5* design, adapted for CIFAR-100. It uses three convolutional layers with **Tanh** activations and 2×2 **average pooling**, followed by two fully connected layers. The data flow is summarized as:

$$3 \times 32 \times 32 \rightarrow 6 \times 14 \times 14 \rightarrow 16 \times 5 \times 5 \rightarrow 120 \times 1 \times 1.$$

The flattened output passes through two dense layers ($120 \rightarrow 84 \rightarrow 100$), producing class logits. The final layer (**F6**) contains:

$$\text{Parameters} = (84 \times 100) + 100 = 8{,}500.$$

By leveraging local connectivity and weight sharing, the ConvNet captures spatial patterns efficiently while using relatively few parameters. Its layer summary is shown in Figure 3.



```
ConvNet(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(16, 120, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=120, out_features=84, bias=True)
  (fc2): Linear(in_features=84, out_features=100, bias=True)
)
```

Figure 3: Architecture of the **ConvNet** with Tanh activations and average pooling.

2.3 Training Setup

All experiments were implemented in PyTorch using the **CIFAR-100** dataset, split into 90% training, 10% validation, and the standard test set for evaluation.

2.3.1 Data Preprocessing and Augmentation

To improve generalization, training images were augmented with random horizontal flips ($p = 0.5$), random crops (32×32 with 4-pixel padding), and mild color jittering (brightness, contrast, saturation ± 0.2). All images were converted to tensors and normalized per channel using mean and standard deviation $(0.5, 0.5, 0.5)$:

$$\text{Normalize}(x) = \frac{x - 0.5}{0.5}.$$

Validation and test images were only normalized. All transformations used `torchvision.transforms`.

2.3.2 Training Configuration

Both **TwoLayerNet** and **ConvNet** were trained using the **Adam optimizer** (learning rate 10^{-3}, weight decay 10^{-4}) with **cross-entropy loss**. Training ran for up to 50 epochs with a batch size of 64 and **early stopping** after five stagnant validation epochs. The best-performing model was saved automatically. Training for the **TwoLayerNet** ended after six epochs, showing it quickly reached its capacity and could not further improve.

2.3.3 Evaluation Protocol

The best model was evaluated on the test set using overall and class-wise accuracy:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Samples}} \times 100\%.$$

3 Experiments and Results

3.1 Hyperparameter Tuning

Hyperparameter optimization was performed using the **Optuna** framework, which applies a sequential model-based search and prunes low-performing trials early based on validation accuracy.

3.1.1 Optimization Procedure

For each architecture, Optuna sampled hyperparameter combinations from predefined ranges and trained the model for a fixed number of epochs. Underperforming trials were pruned using the *MedianPruner* after three warm-up epochs to save computation. Each trial trained a model, evaluated its validation accuracy, and reported results back to Optuna to guide the search. A total of 50 trials were conducted per model, aiming to maximize validation accuracy on CIFAR-100.

3.1.2 Search Space

Both training and augmentation parameters were explored, along with architectural hyperparameters for the convolutional model. Table 1 summarizes the main hyperparameters and their ranges.

Parameter	**Search Range**
Learning rate (η)	{1e-4, 5e-4, 1e-3, 5e-3}
Weight decay	{1e-6, 1e-5, 1e-4, 5e-4}
Batch size	{64, 128}
Optimizer	{Adam, SGD}
Epochs	{15–40}
Hidden size (TwoLayerNet)	{256, 512, 1024}
Brightness / Contrast / Saturation	{0.1–0.4}
Horizontal flip probability	{0.3–0.7}
ConvNet filters (C_1, C_2, C_3)	{(6–16), (16–48), (64–120)}
ConvNet kernel sizes (K_1, K_2, K_3)	{3, 5}

Table 1: Search ranges for Optuna hyperparameter tuning.

3.1.3 Best Hyperparameters

After 50 trials, the best **TwoLayerNet** achieved a validation accuracy of **17.50%** with:

$$\eta = 1{\times}10^{-4}, \quad \text{Weight decay} = 1{\times}10^{-5}, \quad \text{Batch size} = 128, \quad \text{Optimizer} = \text{Adam}, \quad H = 1024.$$

For the **ConvNet**, the best configuration (validation accuracy **29.48%**) was obtained with:

$$\eta = 1 \times 10^{-3}, \quad \text{Weight decay} = 1 \times 10^{-4}, \quad \text{Batch size} = 64, \quad \text{Optimizer} = \text{Adan},$$

$$(C_1, C_2, C_3) = (8, 32, 96), \quad (K_1, K_2, K_3) = (3, 5, 1).$$

The most effective augmentation for both models used moderate color jittering (brightness 0.3, contrast 0.2, saturation 0.2) and a horizontal flip probability of 0.3. The final models were retrained using these parameters for consistent and reproducible results.

3.2 Improved Architectures

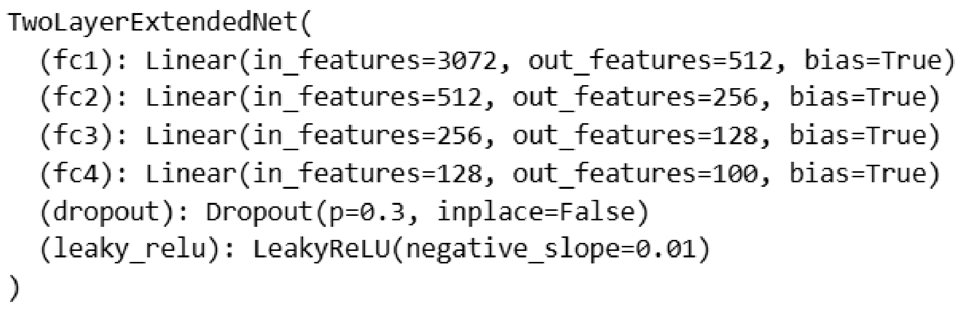
To extend the baseline models, deeper variants were implemented with additional layers, normalization, and regularization to improve feature learning and generalization.

3.2.1 TwoLayerExtendedNet

The **TwoLayerExtendedNet** expands the baseline fully connected model to four layers with progressively smaller hidden dimensions:

$$3072 \rightarrow H \rightarrow \frac{H}{2} \rightarrow \frac{H}{4} \rightarrow 100.$$

Each hidden layer uses a **LeakyReLU** activation to avoid dead neurons, followed by **Dropout** ($p = 0.3$) for regularization. This deeper design captures more complex non-linear patterns in the flattened image representation. The architecture is shown in Figure 4.



```
TwoLayerExtendedNet(
  (fc1): Linear(in_features=3072, out_features=512, bias=True)
  (fc2): Linear(in_features=512, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=128, bias=True)
  (fc4): Linear(in_features=128, out_features=100, bias=True)
  (dropout): Dropout(p=0.3, inplace=False)
  (leaky_relu): LeakyReLU(negative_slope=0.01)
)
```

Figure 4: Architecture of the **TwoLayerExtendedNet** with four fully connected layers and dropout regularization.

3.2.2 ConvExtendedNet

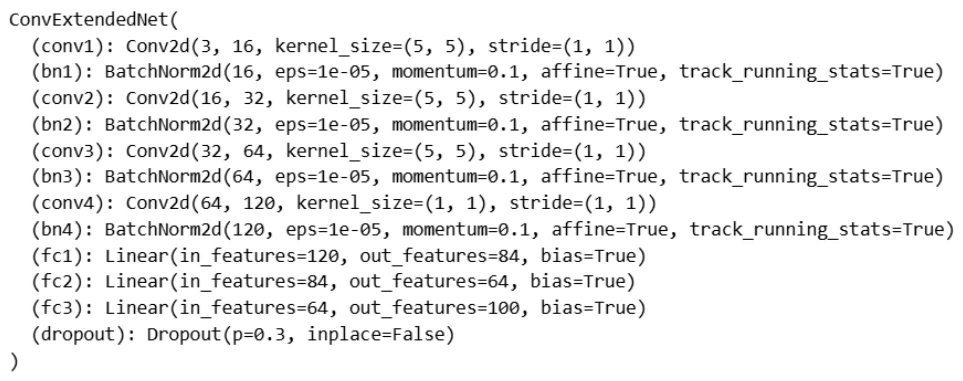
The **ConvExtendedNet** adds one convolutional block and a deeper classifier to the baseline LeNet-style model. It includes four convolutional layers and three dense layers, each convolutional layer followed by **BatchNorm**, **ReLU**, and 2×2 **max-pooling**. The final block uses a 1×1 convolution to increase feature depth without expanding spatial size:

$$3 \times 32 \times 32 \rightarrow 16 \times 14 \times 14 \rightarrow 32 \times 5 \times 5 \rightarrow 64 \times 1 \times 1 \rightarrow 120 \times 1 \times 1.$$

The fully connected classifier ($120 \rightarrow 84 \rightarrow 64 \rightarrow 100$) applies dropout ($p = 0.3$) between layers. These additions enhance representational capacity while maintaining parameter efficiency through convolutional weight sharing. Figure 5 illustrates the structure.

3.3 Results on CIFAR-100

All models were evaluated on the CIFAR-100 test set using identical preprocessing and normalization settings. Performance was measured by overall and per-superclass accuracy across the 20 predefined superclasses (e.g., *aquatic mammals*, *flowers*, *vehicles 1*). The goal was to assess the effect of convolutional feature extraction, hyperparameter tuning, and architectural extensions.



```
ConvExtendedNet(
  (conv1): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1))
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1))
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
  (bn3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv4): Conv2d(64, 120, kernel_size=(1, 1), stride=(1, 1))
  (bn4): BatchNorm2d(120, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=120, out_features=84, bias=True)
  (fc2): Linear(in_features=84, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=100, bias=True)
  (dropout): Dropout(p=0.3, inplace=False)
)
```

Figure 5: Architecture of the **ConvExtendedNet** with added convolutional and dense layers, batch normalization, and dropout.

3.3.1 Overall Performance

Table 2 reports the test accuracies for all models. Convolutional architectures clearly outperform fully connected ones, confirming the benefits of spatial feature learning. Further gains were achieved after hyperparameter tuning and architectural extensions, with the extended ConvNet reaching the best performance.

Model	**Test Accuracy (%)**
TwoLayerNet	10.41
ConvNet	28.42
TwoLayerNetFt	14.97
ConvNetFt	29.52
TwoLayerExtendedNet	17.43
ConvExtendedNet	**37.63**

Table 2: Overall CIFAR-100 test accuracy (%) for all models.

3.3.2 Per-Superclass Analysis

Table 3 shows accuracies across the 20 CIFAR-100 superclasses. Convolutional networks consistently achieve higher scores, especially for visually consistent classes such as animals and vehicles. Fully connected models perform relatively better on texture-dominated classes (e.g., *flowers* or *food containers*) but fail to capture complex spatial relationships.

3.4 Fine-Tuning on STL-10

To evaluate the transferability of learned features, the pretrained **ConvNet** (trained on CIFAR-100) was fine-tuned on a five-class subset of **STL-10**: *bird*, *deer*, *dog*, *horse*, and *monkey*. Figure 6 shows representative samples.

3.4.1 Dataset Preparation

STL-10 images (96×96) were resized to 32×32 for compatibility with the pretrained ConvNet. Training images were augmented with random flips, cropping (padding=4),

Superclass	TwoLayerNet	ConvNet	TwoLayerNetFt	ConvNetFt	TwoLayerExtendedNet	ConvExtendedNet
Aquatic mammals	02.60	15.80	06.80	19.60	14.60	**22.80**
Fish	18.80	31.40	22.80	29.20	21.40	**40.60**
Flowers	24.60	42.40	30.80	41.00	34.60	**52.40**
Food containers	11.20	25.80	12.20	32.40	20.00	**45.20**
Fruit and vegetables	25.40	33.80	22.80	40.00	26.00	**41.60**
Household electrical devices	02.80	06.20	25.80	24.00	08.00	**32.40**
Household furniture	14.60	38.00	08.60	34.00	17.20	**41.40**
Insects	04.80	31.60	17.00	30.00	25.60	**39.20**
Large carnivores	00.80	22.60	08.00	25.00	09.40	**36.20**
Large man-made outdoor things	22.40	45.40	28.20	47.60	31.60	**54.20**
Large natural outdoor scenes	12.80	44.80	27.20	47.40	33.80	**60.20**
Large omnivores and herbivores	06.80	27.40	13.40	32.40	18.40	**33.80**
Medium-sized mammals	05.40	22.40	13.40	22.80	12.00	**33.00**
Non-insect invertebrates	04.00	14.40	13.80	13.40	04.00	**20.00**
People	03.20	14.80	09.20	**15.40**	06.80	14.20
Reptiles	00.40	16.00	05.40	11.00	03.60	**22.60**
Small mammals	02.40	14.00	04.60	11.40	05.00	**15.20**
Trees	23.00	36.60	30.40	38.60	25.60	**45.20**
Vehicles 1	06.80	28.80	06.80	33.00	12.20	**45.40**
Vehicles 2	15.40	39.60	11.80	42.20	18.80	**57.00**

Table 3: Per-superclass accuracy (%) on CIFAR-100 test set.

and mild color jittering, while validation and test sets were only normalized. A 90/10 train–validation split was used.

3.4.2 Transfer Learning Setup

The final layer of the CIFAR-100 **ConvNet** was replaced with a new fully connected layer ($84 \to 5$). All other weights were fine-tuned using the **Adam optimizer** ($\eta = 10^{-4}$, weight decay 10^{-5}) for up to 30 epochs with early stopping (patience=5).

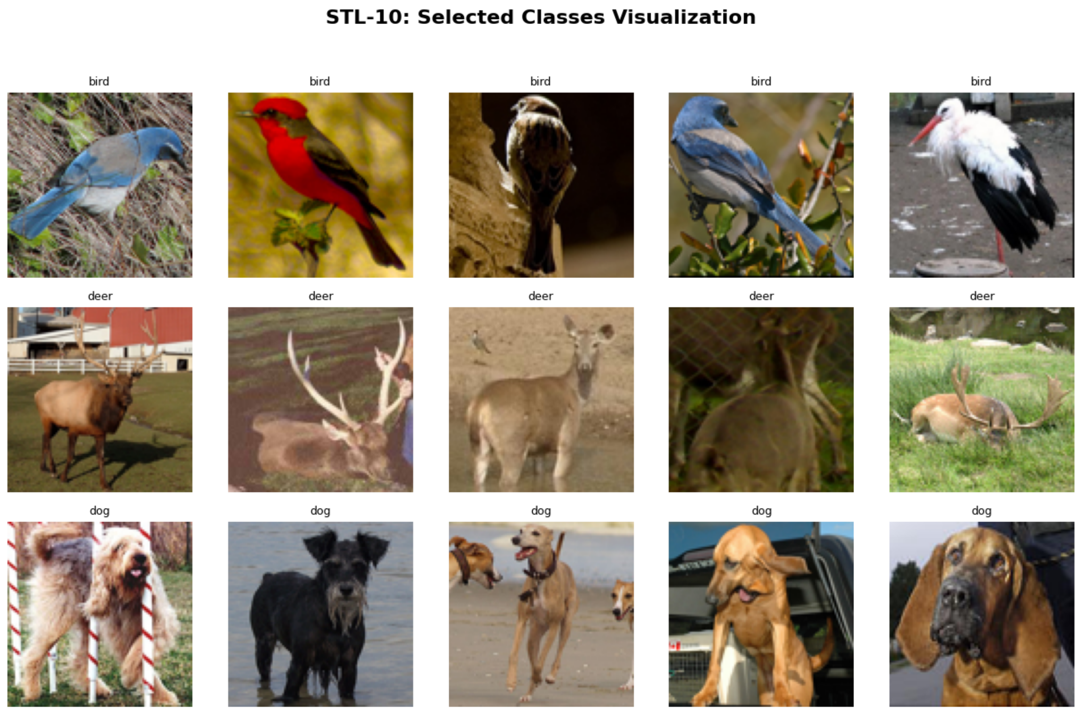


Figure 6: Examples from the five STL-10 classes used for fine-tuning.

3.4.3 Training Performance

Figure 7 shows training and validation curves. The model converged smoothly, plateauing after about 20 epochs, and achieved a test accuracy of **46.92%**.

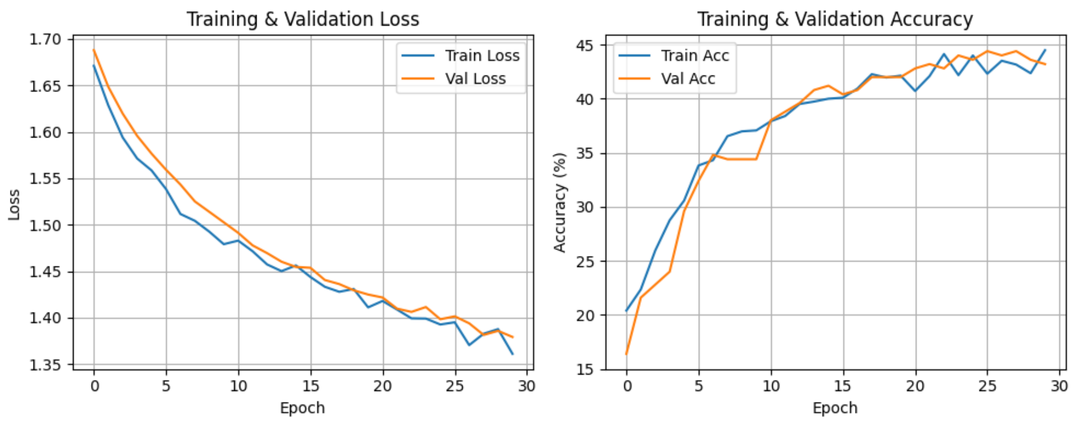


Figure 7: Training and validation loss curves for the fine-tuned ConvNet.

3.4.4 Feature Visualization with t-SNE

A 2D **t-SNE** embedding was generated from the penultimate layer (`fc1`) activations, after PCA reduction to 50 components and standardization. A subset of 2000 test samples was visualized (Figure 8). Class clusters are generally well separated, with partial overlap among visually similar animal classes, reflecting shared textures and shapes.

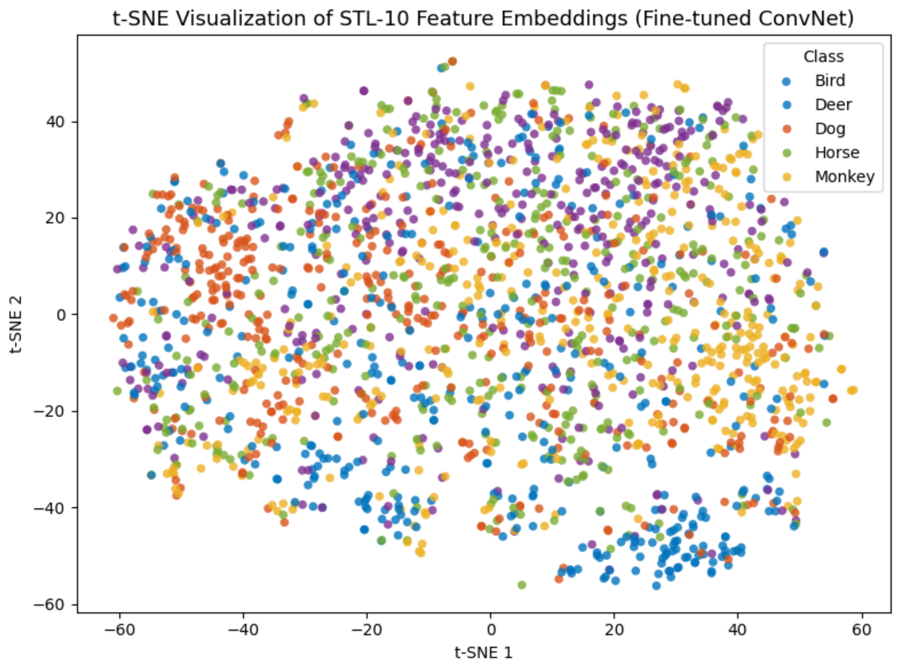


Figure 8: t-SNE visualization of feature embeddings from the fine-tuned ConvNet on the STL-10 subset.