UNIVERSITEIT VAN AMSTERDAM

COMPUTER VISION 1

Lab Project Part 2

Authors:
Jarno BALK
Adit MARADA
Camille NIESSINK

TA's:
A. ERDELEZ
W. TREJTER

October 27, 2025

Contents

1 Introduction

This document explains how we looked into the ability of two different types of neural network architecture to perform image classification.

1.1 CIFAR-100 Dataset

We used the CIFAR-100 dataset for this task, a benchmark dataset widely used in image classification tasks (Krizhevsky et al., 2009). This dataset consists of 100 object categories, each containing 600 images with a dimension of $32 \times 32 \times 3$. The superclasses and example images of their corresponding subclasses are shown in the Appendix (Figure 5 and 6).

1.2 STL-10 Dataset

After training a CNN on the CIFAR-100 dataset described above, we used it in a transfer learning setting, finetuning the model on a novel dataset. This is the STL-10 dataset (Coates et al., 2011), which was designed to encourage learning from unlabeled data. However, in this research, we did not use the unlabeled part of the dataset. We used five of the ten classes; *bird*, *deer*, *dog*, *horse*, *monkey*. Appendix (Figure 7) shows five example images for each of the classes.

2 Implementation Details

This section introduces and describes two neural network architectures: `TwoLayerNet` and `ConvNet`, both implemented with PyTorch and designed to classify images from the CIFAR-100 dataset.

2.1 TwolayerNet

The `TwoLayerNet` is an architecture representing a two-layer neural network. This network inherits from the Pytorch superclass `nn.Module` [1], the fundamental class for all neural network modules. The architecture is initialized with two fully connected layers with a ReLu activation in between. The sizes of these layers are determined by input size, hidden layer size, and number of classes. The number of classes in the CIFAR-100 dataset is 100, and the input has a dimensionality of $32 \times 32 \times 3 = 3072$.

The forward pass of the network first flattens the input tensor and then passes it through the first fully connected layer with a Rectified Linear Unit (ReLu) activation. The output is then fed into the second fully connected layer to produce the final score for classification. The implementation of ReLu introduces non-linearity to the model, allowing it to capture complicated relationships within the data.

[1] https://pytorch.org/docs/master/generated/torch.nn.Module.html#torch.nn.Module

2.2 ConvNet

The second architecture that we implemented is a convolution neural network (`ConvNet`), based on the LeNet architecture (LeCun et al., 1998). Several modifications were introduced compared to the original implementation. First, instead of using grayscale images like LeCun et al., we employed color images, leading to an additional dimension in the input. Consequently, the input gets an extra dimension, and the first convolutional layer takes inputs that have size $3 \times 32 \times 32$. Also, in our implementation, all 16 output layers of C3 draw from each of the 5 output layers of S2. This decision was made because of our reduced computational constraints and simplified implementation.

Another deviation from LeNet lies in the use of average pooling instead of trainable subsampling parameters. Additionally, LeCun et al. use radial basis function units and calculate the Minimum Square Error for generating outputs and loss. Due to the unavailability of RBF units in PyTorch, we used a fully connected layer with outputs matching the number of image classes (utilizing Cross Entropy Loss). While still using the non-linear activation function, we chose it over the more common ReLu.

2.3 Architectural Improvements for Tuning

Using the above architectures on baseline, we made several changes to enhance the performance of our model.

2.3.1 From TwoLayerNet to FourLayerNet

The primary limitation of the TwoLayerNet architecture is its shallow depth. We experimented by just increasing the depth of the model by two layers hoping it would help it learn some complex features even though these were not passed through the convolution layer.

Design Choices:

- **Increased Depth:** As per the instructions, we added two fully-connected hidden layers, creating a hierarchy with neuron counts of 1024, 512, and 256. The aim was to give the model some higher capacity to learn more complex decision boundaries. The final structure used is:

$$\begin{aligned}\text{Input} &\to \text{Linear}(1024) \to \text{ReLU} \to \text{Linear}(512) \to \text{ReLU} \\ &\to \text{Linear}(256) \to \text{ReLU} \to \text{Linear}(100)\end{aligned}$$

- **Regularization:** To prevent the model from overfitting we introduced `Dropout` layers (with a dropout probability of 0.3) after each `ReLU` activation. This was done to push the network to learn more redundant features by preventing from co-adapting.

2.3.2 From ConvNet to ImprovedConvNet

The baseline ConvNet model was also extended by adding two additional layers.

Design Choices: A balanced approach was taken to add the two required layers: one was added to the convolutional base to enhance feature extraction, and the other to the classifier head to improve classification capability.

- **Layer Addition**
  1. An extra **convolutional layer** was added inside a new third convolutional block. This allows the network to build a deeper hierarchy of visual features, moving from simple edges (first block) to complex textures (second block) and finally to abstract object parts (third block).
  2. An extra **fully-connected layer** was added to the classifier head. This improved the model's capacity to learn the complex, non-linear relationships between the high-level features detected by the convolutional base and the final 100 classes.
- **Deviations from Baseline**
  - **Activation and Normalization:** The original scaled `tanh` activation was replaced with the standard `ReLU` for better gradient flow and faster training. We also added `Batch Normalization` after every convolutional layer. This normalizes the activations of each layer, which we helped in stabilizing and accelerating the training of the network.
  - **Pooling & Regularization:** The `AvgPool` layers were replaced with `MaxPool`, which is more effective at identifying more subtle features within a receptive field. Again, to prevent overfitting we introduced `Dropout` (p=0.5) between the fully connected layers.

Tabel 1: Table Comparing the Various Neural Nets

Network	**Type**	**Layers**	**Comparison to Baseline**
TwoLayerNet	Baseline FCN	2 Linear	-
FourLayerNet	Enhanced FCN	4 Linear	**+2 Linear layers**, Added Dropout for regularization.
ConvNet	Baseline CNN	3 Conv, 2 Linear	LeNet-style with custom tanh and AvgPool.
ImprovedConvNet	Enhanced CNN	4 Conv, 3 Linear	**+1 Conv, +1 Linear**, Replaced components with ReLU, BatchNorm, MaxPool, and Dropout.

3 Experiments and Results

This section shows the different experiments taken to explore the impact of hyperparameters on the obtained accuracy. We first evaluated the baseline models and then conducted a hyperparameter search using the improved architectures. At last, a transfer learning experiment was conducted on the smaller STL-10 dataset.

3.1 Baseline Model Performance

The initial models (TwoLayerNet and ConvNet) were trained with an Adam optimizer and a learning rate of 0.001. This provided us with a performance baseline.

- The `TwoLayerNet`, trained for 30 epochs, achieved a peak validation accuracy of **23.51%** peaking at around 11th epoch.
- The `ConvNet`, trained for 25 epochs, achieved a peak validation accuracy of **25.27%** and peaking at around the 15th epoch.

While `ConvNet` did perform slightly better, both showed limited success. The `ConvNet`'s performance plateaued and began to decline, ultimately resulting in the need for a different architecture and training strategy.

3.2 Hyperparameter Tuning and Architectural Refinement

We tried various hyperparameters and refined the architectures to maximize performance. This involved adding data augmentation (random cropping and horizontal flipping), regularization techniques (Dropout, weight decay), and advanced training strategies (learning rate schedulers). Note: A deeper ablation study to isolate impact of various strategies was not performed.

3.2.1 FourLayerNet Performance

The deeper fully-connected network was trained with the settings detailed in Table 2. Despite adding depth and regularization, the model performed poorly, achieving a best validation accuracy of only **16.20%**. The learning curves in Figure 1 show that while the training loss decreased, the validation accuracy stagnated at a very low level. This confirms a critical limitation: for image data, simply increasing the depth of a fully-connected network is ineffective. By flattening the image, the model loses the spatial inductive bias necessary to learn visual features efficiently.

Tabel 2: Hyperparameters for FourLayerNet.

Hyperparameter	**Value**
Optimizer	Adam
Learning Rate	0.001
Epochs	50
Batch Size	128
Weight Decay	1e-4
Data Augmentation	RandomCrop, RandomHorizontalFlip
Regularization	Dropout (p=0.3)

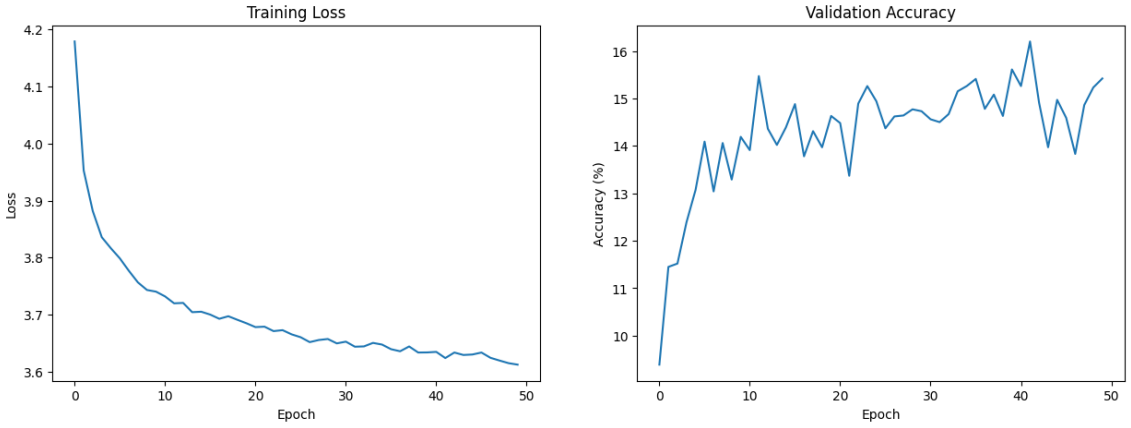

Figure 1: Training loss and validation accuracy for the FourLayerNet

3.2.2 ImprovedConvNet Performance

The `ImprovedConvNet` was trained with a different strategy, detailed in Table 3. This gave us a large improvement over the baseline, achieving a best validation accuracy of **66.29%**. As per the learning curves in Figure 2, we see a steady decrease in training loss corresponding to a stable increase in the validation accuracy. This is a direct effect of using Batch Normalisation with approriate depth along with data augmentation, and learning rate scheduling.

Tabel 3: Hyperparameters for ImprovedConvNet.

Hyperparameter	**Value**
Optimizer	SGD with Momentum (0.9)
Learning Rate	Initial: 0.01, with CosineAnnealingLR
Epochs	100
Batch Size	128
Weight Decay	5e-4
Data Augmentation	RandomCrop, RandomHorizontalFlip
Regularization	Dropout (p=0.5)

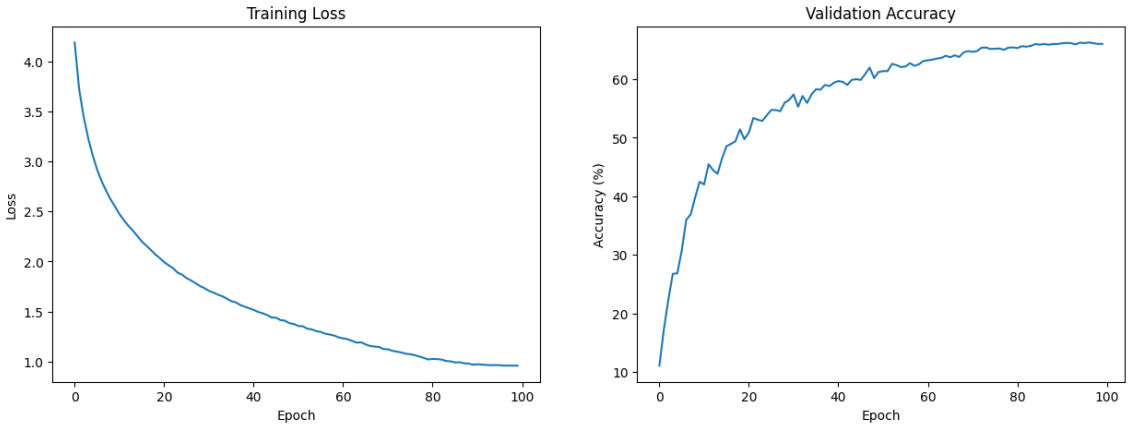

Figure 2: Training loss and validation accuracy for the ImprovedConvNet.

3.3 Model Comparison

There is a big performance gap between `FourLayerNet(16.2%)` and the enhanced `ImprovedConvNet` (66.29%). The primary reason is `ImprovedConvNet` effectively learns spatial inductive bias through its convolution layers allowing it to efficiently learn a hierarchy of visual patterns like edges to textures to object parts. On the other hand, a fully connected feed forward network like `FourLayerNet` esentially discards the spatial information, making it difficult to learn meaningful visual features from image data.

3.4 Transfer learning on STL-10

We used the weights from the best `ImprovedConvNet` pre-trained on CIFAR-100 for the transfer learning task on the 5-class STL-10 subset.

3.4.1 Fine-tuning Methodology

We took the following steps:

1. Load the state dictionary of the `ImprovedConvNet` trained on CIFAR-100.
2. **Freeze the convolutional base:** The weights of both convolutional blocks were frozen, as these layers learned general-purpose features (like edges and textures) that are expected to be useful for the new task.
3. **Replace the classifier head:** The final fully-connected layer, which was specific to the 100 CIFAR-100 classes, was replaced with a new one designed for the 5 STL-10 classes.
4. **Train the new head:** We trained only the parameters of the new classifier head on the STL-10 training data using an Adam optimizer with a learning rate of 0.0005 for 50 epochs. STL-10 images were resized to 32×32 to match the model's input dimensions.

3.4.2 Performance Analysis

The fine-tuned model achieved a best validation accuracy of **76.05%** on our **validation** set. The learning curves are shown in Figure 3. The model learns quickly, achieving a high accuracy which shows that the pretraining provided a substantial starting point. However, we also see signs of overfitting, the training loss drops rapidly while validation accuracy becomes slightly unstable. We suspect this is because of the high capacity of the classifier heads from training on CIFAR-100 which leads to the model memorizing that training data. Despite this we achieve an accuracy score on **73.65%** on the **test** set.

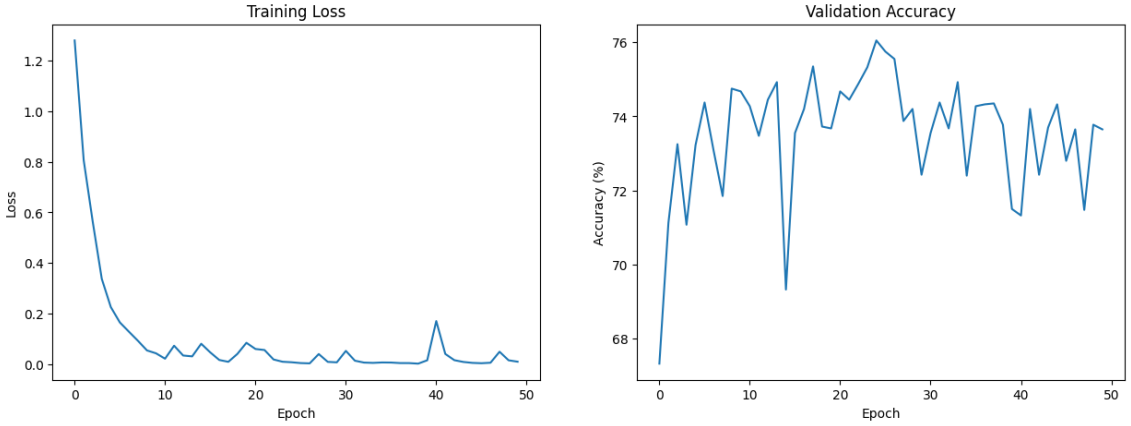

Figure 3: Training loss and validation accuracy for fine-tuning on STL-10.

3.4.3 Feature Space Visualization with t-SNE

To better understand how the model separates these new classes, we extracted the last layer's features and visualized it using t-SNE (Figure 4). The plot shows that the model learned a partially structured feature space. We can observe clusters corresponding to the five classes. We also see significant overlap especially between similar animals like 'horse', 'deer' and 'dog' (all 4 legged animals). This helped us understand the model's confusion explaining the source of errors in its predictions.

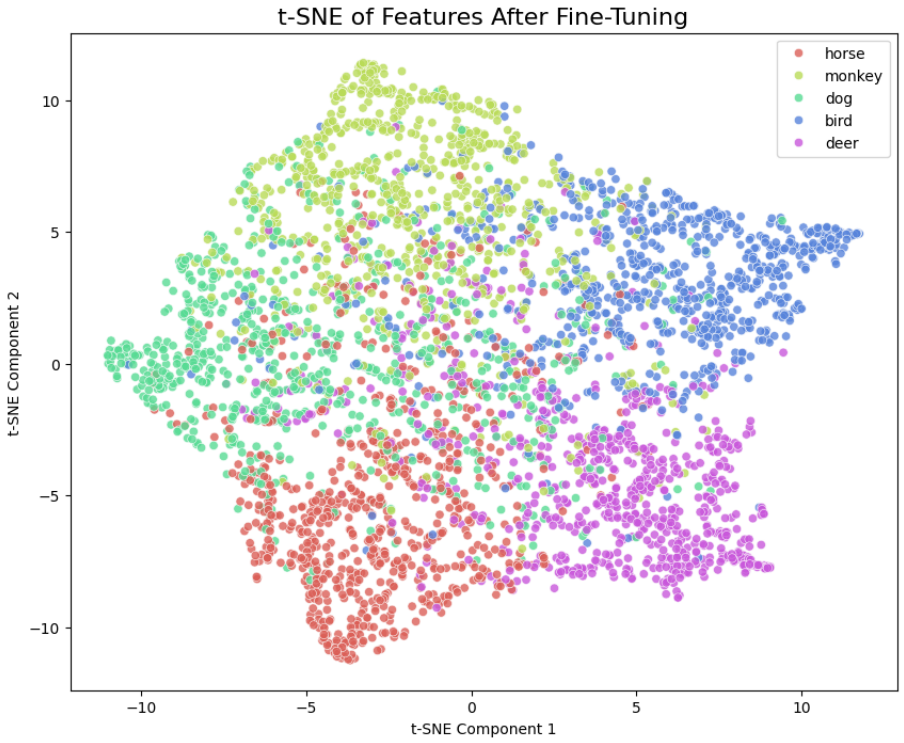

Figure 4: t-SNE visualization of the STL-10 test set features extracted from the fine-tuned model.

3.4.4 Final Performance on Test Sets

A final unbiased evaluation was performed on the held out test set as well. We evaluated our best-performing models on their respective test sets to measure their true generalization capability.

ImprovedConvNet on CIFAR-100 Test Set Our final `ImprovedConvNet` model, after 100 epochs of training, achieved a final accuracy of **66.05%** on the CIFAR-100 test set. This result is consistent with its peak validation accuracy, confirming that the model generalized well and was not overfit to the validation data. The per-class accuracies varied, with visually distinct classes like ‘motorcycle‘ (92%) and ‘orange‘ (91%) achieving high scores, while classes like ‘otter‘ (26%) and ‘girl‘ (32%) proved more challenging.

Fine-Tuned Model on STL-10 Test Set The fine-tuned model was evaluated on the 4,000-image STL-10 test subset, achieving a final accuracy of **73.65%**. The per-class results, shown in Table 4, indicate strong performance across all five classes, with ‘bird‘ being the easiest to classify (79.88%) and ‘dog‘ being the most challenging (66.88%), likely due to high intra-class variance.

Tabel 4: Per-class accuracy of the fine-tuned model on the STL-10 test set.

Class	Test Accuracy (%)
Bird	79.88
Deer	78.62
Dog	66.88
Horse	70.12
Monkey	72.75

4 Conclusion

The performance progression of our models, from simple baselines to a fine-tuned convolutional architecture, is summarized in Table 5.

Tabel 5: Final Summary of Model Performance across all Experiments.

Model	Task / Dataset	Peak Val. Acc. (%)	Final Test Acc. (%)	Key Architectural Features
TwoLayerNet	CIFAR-100	23.51	23.51*	Baseline fully-connected network (2 layers).
ConvNet	CIFAR-100	25.27	25.27*	Baseline CNN (LeNet-style).
FourLayerNet	CIFAR-100	16.20	16.20*	Deeper FCN (4 layers) with Dropout.
ImprovedConvNet	**CIFAR-100**	**66.29**	**66.05**	**Deeper CNN with BatchNorm, ReLU, MaxPool, Dropout.**
ImprovedConvNet	**STL-10 (Fine-Tuned)**	**76.05**	**73.65**	**Pre-trained on CIFAR-100, classifier head replaced.**

For baseline models, validation accuracy was measured on the test set at the end of each epoch; a separate final test run was not performed. The value shown is the peak accuracy achieved during training.

This project demonstrated the strength of Convolutional Neural Networks (CNNs) in handling image classification tasks. We see that CNNs outperform traditional fully connected networks mainly because of their ability to capture spatial relationships within the image. We also observed that having a good architecture alone isn't enough. The training strategy plays an equally important role. Using batch normalization, data augmentation and LR scheduling stabilized training and prevented overfitting.

We were also able to achieve strong results on the STL-10 dataset with relatively little training data. This highlights how features learned on one dataset can effectively generalize to another, making transfer learning a powerful and data-efficient approach. Finally, visualization techniques such as t-SNE helped us understand what the model had learned. We were able to visually assess how well the network separated different classes and identify where it struggled offering a clearer picture of both its strengths and limitations.

References

Coates, A., Ng, A., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings.

Krizhevsky, A. et al. (2009). Learning multiple layers of features from tiny images.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

A Appendix

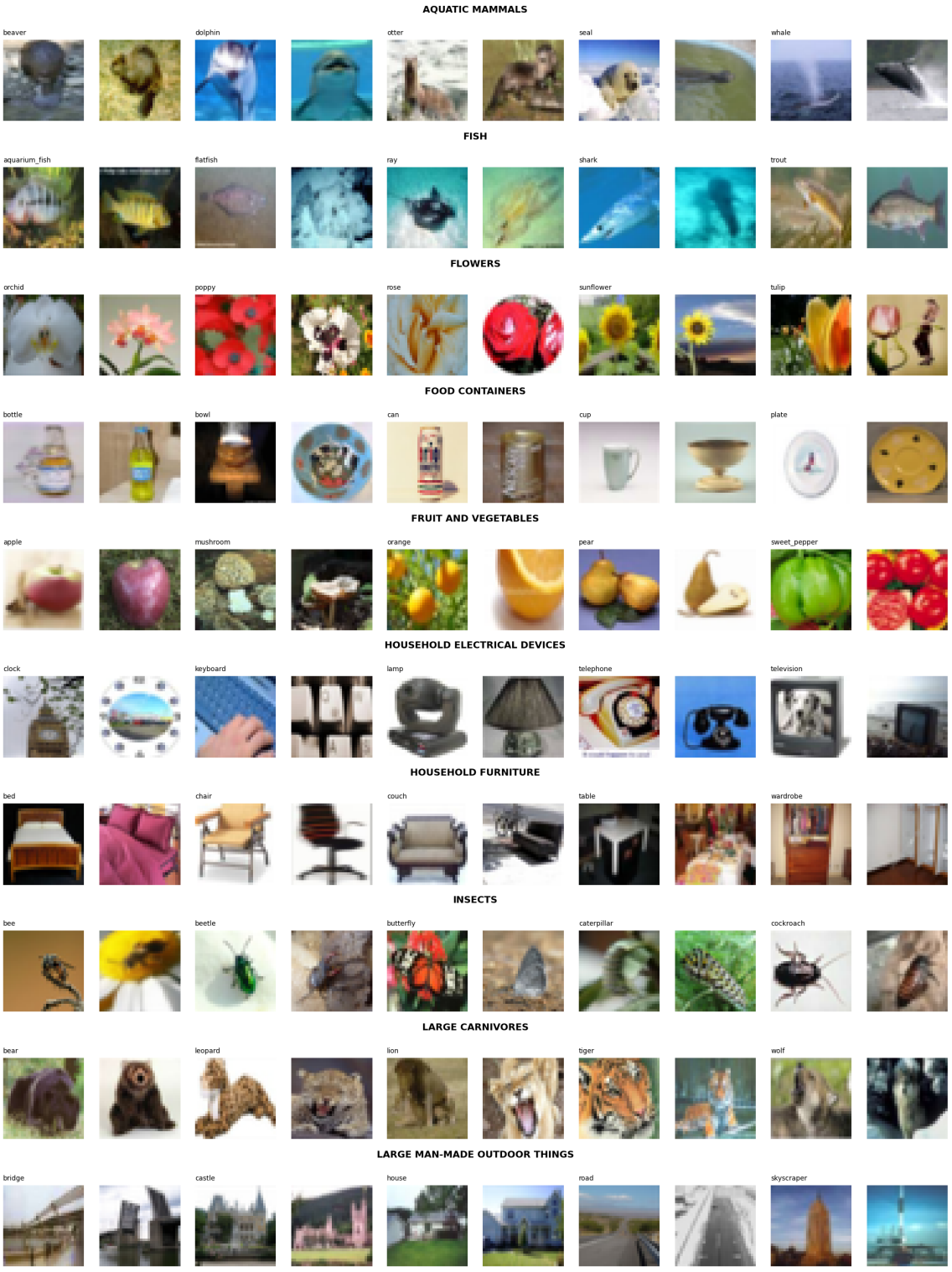

Figure 5: CIFAR-100 superclasses (aquatic mammals, fish, flowers, food containers, fruit and vegetables, household electrical devices, household furniture, insects, large carnivors, large man made outdoor things) with two example images of their corresponding subclasses.

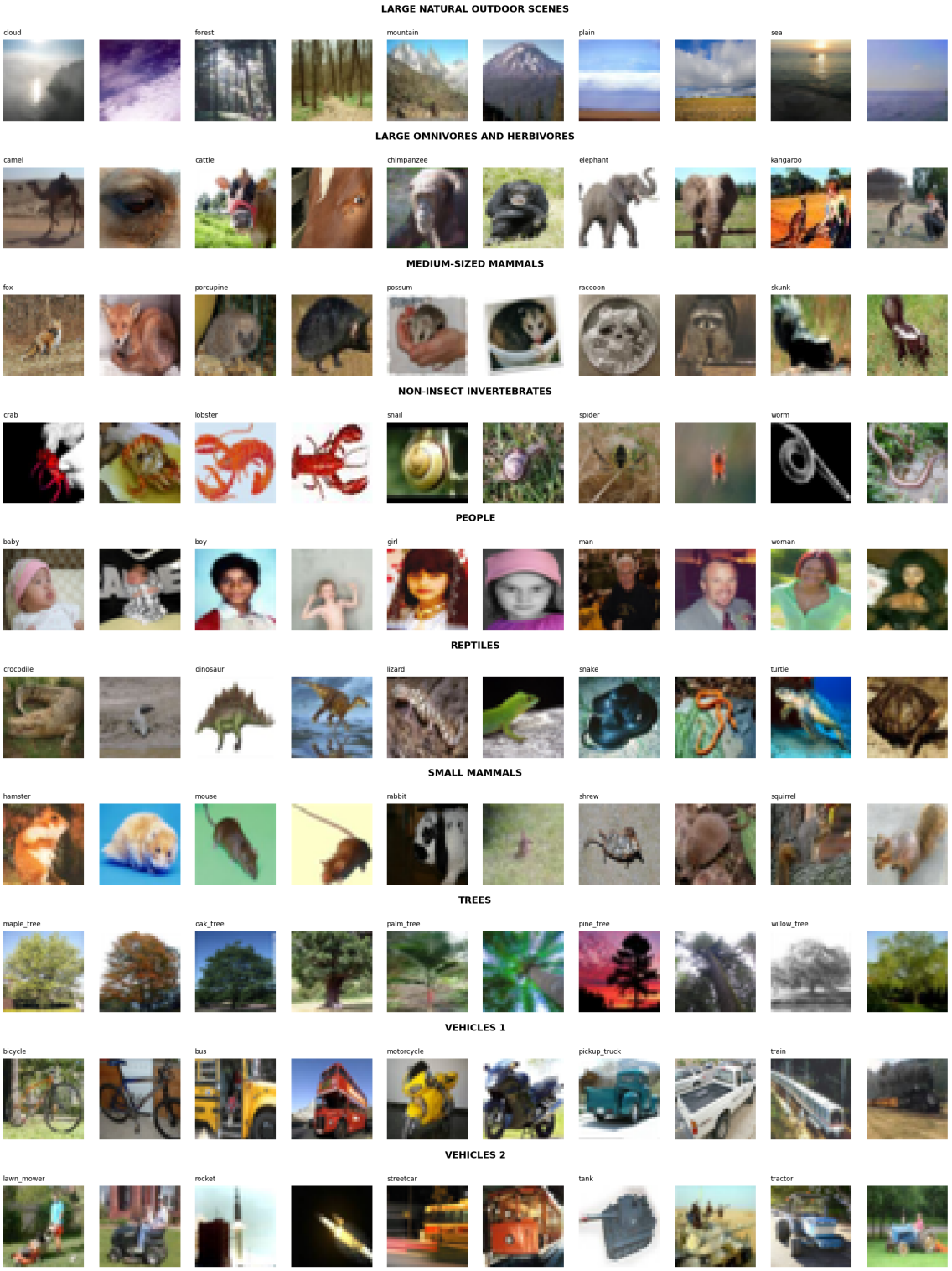

Figure 6: CIFAR-100 superclasses (large natural outdoor scenes, large omnivors and herbivores, medium-sized mammals, non-insect invertebrates, people, reptiles, small mammals, trees, vehicles 1, vehicles 2) with two example images of their corresponding subclasses.

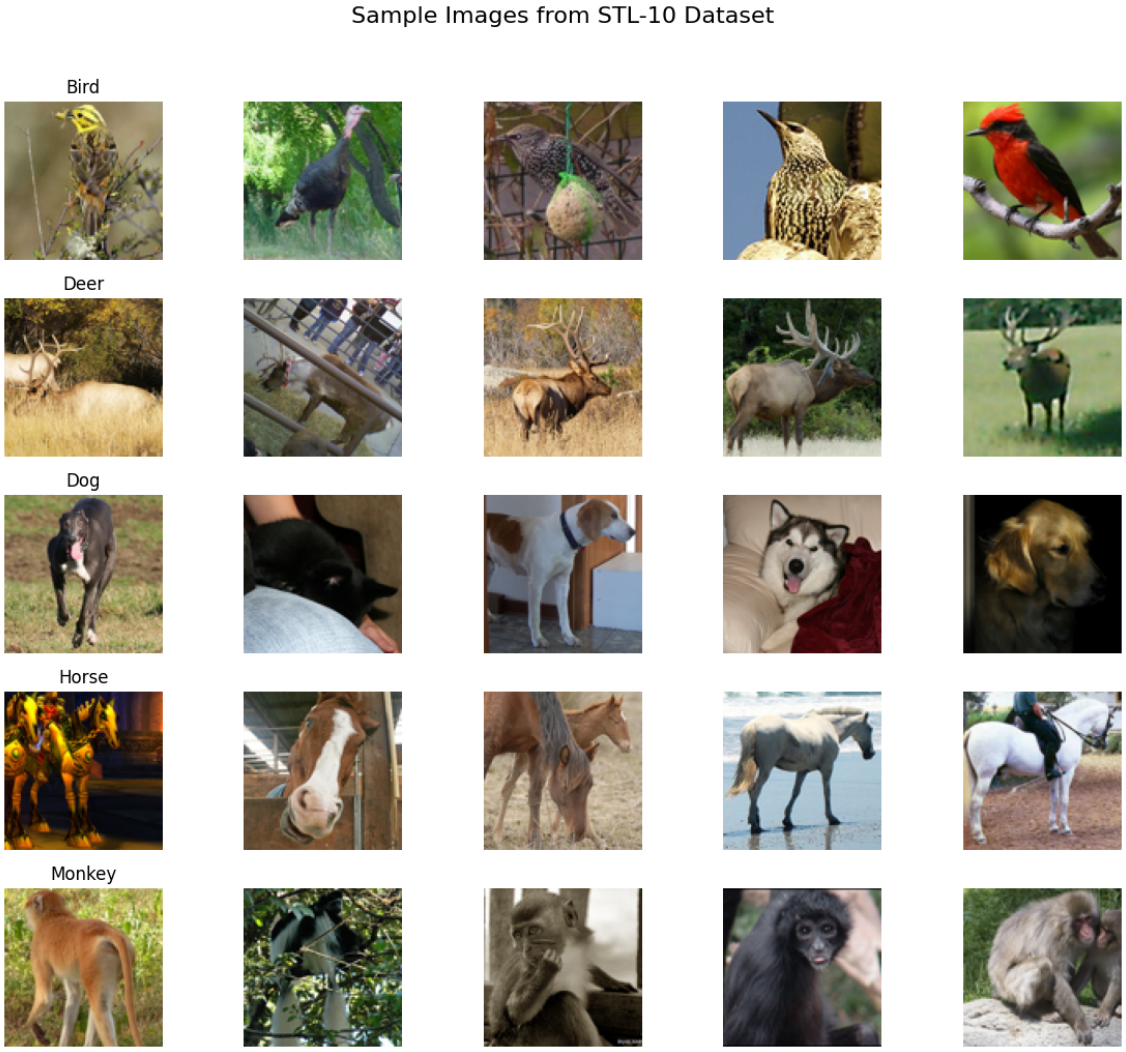

Figure 7: STL-10 Dataset