UNIVERSITY OF AMSTERDAM

COMPUTER VISION FINAL LAB REPORT 2

Image Classification with Convolutional Neural Networks

18 October 2024

Students:
Emma Kasteleyn

Jasper van der Valk

Platon Karageorgis

Tutor:
Martin Oswald, Arnoud Visser

Course:
Computer Vision 1

1 Introduction

Convolutional Neural Networks (CNNs) have become the cornerstone of modern computer vision tasks due to their ability to automatically learn hierarchical feature representations from raw pixel data. Their success spans across various applications, including image classification, object detection, and segmentation. In this project, we explore different CNN architectures and techniques to improve image classification performance on challenging datasets.

2 Previous Work

The work in image classification and image processing in general with convolutional neural networks has been revolutionary in the past decades and naturally it's impossible to cover all of the previous work. However, our aim is to mention a few that constitute the foundation in this branch.

Yann LeCun's LeNet-5 architecture [5] was one of the earliest successful applications of CNNs in character recognition. LeNet-5 introduced key concepts such as convolutional layers, pooling layers, and fully connected layers, forming the basis for future CNN architectures. We implement the baseline ConvNet using the exact parameters from LeNet-5 to evaluate its performance on modern datasets.

Further advancements were made with architectures like AlexNet [4] and VGGNet [6], which demonstrated that increasing depth and using smaller convolutional kernels could significantly improve performance. While these networks are deeper and more complex, the fundamental principles remain consistent with LeNet-5. Our work builds upon these ideas by experimenting with network depth and architectural modifications.

Finally, a significant contribution to CNN architectures came with ResNet [2], which tackled the issue of vanishing gradients in deep networks by introducing residual connections.

3 Datasets

3.1 CIFAR-100

The CIFAR-100 (PLOT HERE) dataset consists of 60,000 color images of size 32 × 32 pixels, categorized into 100 classes (Krizhevsky et al. [3]). Each class contains 600 images, making it a challenging dataset due to the high number of classes and limited image resolution. The dataset is split into 50,000 training images and 10,000 test images. A preview of the CIFAR-100 dataset is available in the appendix.

3.2 STL-10

The STL-10 dataset contains 13,000 labeled images from 10 classes, with higher resolution images of size 96×96 pixels (Coates et al. [1]). For this project, we focus on a subset of five classes: bird, deer, dog, horse, and monkey. This subset allows us to test our models on higher-resolution images and assess their ability to generalize to different datasets. A preview of the STL-10 dataset is available in the appendix.

4 Methodology

4.1 Data Preparation - CIFAR-100

Before training our models, we performed data preprocessing and augmentation on the CIFAR-100 dataset to enhance model performance and generalization. Initially, we normalized the images using the mean and standard deviation of the dataset to ensure that the input features are on a similar scale, which aids in faster convergence during training (Le cun et. al [1]).

Moreover, we applied data augmentation techniques to increase the diversity of the training set. Specifically, we used random horizontal flips and random crops with padding. We expect these techniques to be very useful, as they expand the dataset using synthetic data, which are generated through modified versions of the original images. These transformations can involve flipping, rotating and other similar techniques.

Finally, we split the original training set into a new training set and a validation set, allocating 80% of the data for training and 20% for validation. The validation set was used to monitor the model's performance during training and to implement early stopping. For the early stopping, we enforced a patience of 5 epochs.

4.2 Baseline Models

4.2.1 Two-Layer Neural Network

Our first baseline is a simple two-layer neural network. The architecture consists of an input layer that flattens the image pixels, a hidden layer with a specified number of neurons, and an output layer corresponding to the number of classes. After the hidden layer we activate ReLU. This is a very straightforward approach to image classification, as the architecture is basic.

4.2.2 Baseline Convolutional Neural Network

The second baseline is a ConvNet modelled exactly after LeNet-5 [5]. It includes three convolutional layers followed by two fully connected layers. The convolutional layers use kernel sizes and feature maps as specified in the original paper. We do not use batch normalization or any other sophisticated techniques at this stage to maintain consistency with the original implementation.

To assess the model's complexity, we specifically examine the number of trainable parameters within its fully connected layers, starting with the "F6" layer. The "F6" layer in the ConvNet architecture is a fully connected layer that receives 120 input features and produces 120 output features. We consider both the weights and biases and thus we determine the total number of trainable parameters.

Specifically, each of the 120 input features is connected to each of the 120 output features, resulting in $120 \times 120 = 14{,}400$ weight parameters. Also, there are 120 bias parameters, one for each output feature. Therefore, the total number of trainable parameters in the "F6" layer is $14{,}400 + 120 = 14{,}520$.

4.2.3 Training

Both models were trained on the CIFAR-100 dataset using standard training procedures. We used the training set for learning the model parameters and the validation set for hyperparameter tuning and monitoring overfitting.

The training process included the following steps:

- **Loss Function:** We used the cross-entropy loss function, which is suitable for multi-class classification problems.
- **Optimizer:** The Adam optimizer was employed with an initial learning rate of 0.001.
- **Learning Rate Scheduler:** A learning rate scheduler was used to reduce the learning rate by a factor of 0.1 every 30 epochs, helping the model converge to a minimum by fine-tuning the weights.
- **Early Stopping:** We implemented early stopping to prevent overfitting by monitoring the validation loss. If the validation loss did not improve for five consecutive epochs, the training was stopped.
- **Epochs:** The models were trained for up to 50 epochs unless early stopping criteria were met.

The two-layer neural network achieved a test accuracy of 21.6%, while the baseline ConvNet reached 27.5%. The modest performance of our models is likely due to the difficulty of the dataset. As mentioned earlier, CIFAR-100 has 100 classes, and all of the images are in low resolution (32×32), which makes it challenging for simple architectures to capture the necessary features for accurate classification.

Moreover, we underline the significantly better performance of ConvNet compared to the two-layer network. However, this performance gap is natural due to the simplicity of the latter model and should be no surprise. The models are very basic in their design and as stated before, they are expected to act mostly as baselines.

4.3 Hyperparameter Tuning

Subsequently, we aim to tune a subset of the hyperparameters that our models use in order to investigate their effect and seek potential improvements. We experimented with various hyperparameters for both models, including learning rate, optimizer choice operating with various gamma variables, data augmentation techniques such as rotations and flips of images, and batch normalization. Note that we do not tune the number of epochs as we use early stopping, which makes the tuning of the epochs redundant. Below, we present briefly each one of these techniques.

4.3.1 Hyperparameters Explored

- **Learning Rate:** A critical hyperparameter in deep learning which affects convergence speed and model performance.
- **Optimizers:** Compared SGD and Adam optimizers to evaluate their impact on training dynamics.

- **Data Augmentation:** Applied random horizontal flips and rotations to increase dataset diversity.
- **Batch Normalization:** Incorporated batch normalization layers to stabilize and accelerate training.
- **Learning Rate Scheduler:** Used a step decay scheduler to adjust the learning rate during training.
- **Gamma:** The factor by which the learning rate is multiplied at each step of the scheduler. A lower gamma value, such as 0.1, results in a more significant reduction in the learning rate, helping the model converge slower and fine-tune towards the end of training. Note that this was tuned only for the two layer network due to time constraints.

Finally, it is important to note that the tuning process can be characterized as a "full sweep" as we investigate all of the possible combinations of our hyperparameters.

4.3.2 Best Hyperparameter Combination

Overall, both methods preferred Adam instead of SGD, which was the most significant difference. Also, the gamma value which was tuned only for the two layer network did not seem to have a significant impact. We list specifically the hyperparameter choices of the top 5 runs in Section 5.3.

4.4 Extended Architectures

In the subsequent section we discuss more complex models, as we proceed to add two more layers to each of our baseline models.

4.4.1 Four-Layer Neural Network

We extended the two-layer network by adding two additional hidden layers, resulting in a four-layer network. The hidden layer sizes were set to 512, 256, and 128 neurons, respectively. The additional layers allow the network to learn more complex and abstract representations, potentially improving classification accuracy.

4.4.2 Modified ConvNet with Additional Layers

Similarly, we extended the baseline ConvNet by adding two extra layers—both convolutional and fully connected layers. The extra convolutional layers were added after the original convolutional layers to capture higher-level features. Specifically, we introduced a third convolutional layer with 32 output channels and a kernel size of 3, followed by a fourth convolutional layer with 120 output channels. On the other hand, the fully connected layers were also expanded by adding an extra layer with 84 neurons before the final output layer.

We placed the extra layers in this manner to progressively increase the network's depth and capacity, allowing it to learn more intricate patterns. Network architectures with hidden layer sizes similar to 512-256-128 have empirically performed well across various classification tasks, in image recognition as well as in natural language processing (NLP). The idea of having more neurons in earlier layers and fewer neurons in deeper layers aligns with best practices from studies like those of He et al. [2] and LeCun et al. [5], where progressively smaller layers help balance learning capacity and overfitting risk. This is a natural implementation as if the first layer has more neurons, this allows for a larger feature space, capturing more diverse patterns in the data. In addition, regarding the inner layers, as the features become more abstract, the subsequent layers (256 and 128 neurons) provide a more focused representation of the data, potentially filtering out less important features and refining the learning of complex patterns.

4.5 Model Comparison

The four-layer neural network and the modified ConvNet outperformed the baseline models. This improvement aligns with findings in the literature, where deeper networks tend to perform better due to their ability to model complex functions [6].

Hyperparameter tuning managed to improve a lot the baseline results however it wasn't enough to beat the models with more layers. The additional layers were key to capturing the most intricate patterns in the dataset and that's why the prevailed.

We can see the results of the models in terms of test and validation accuracy in the table below.

Model	Test Accuracy (%)	Validation Accuracy (%)
Two-Layer Net	22.06	22.35
ConvNet	28.51	28.73
Four-Layer Net	27.34	27.25
ConvNet with Extra Layers	30.64	30.79
Two-Layer Net Tuned	23.84	24.45
ConvNet Tuned	30.57	30.88

Table 1: Comparison of Test and Validation Accuracy for Different Models

4.6 Fine-Tuning on STL-10

After we performed a full comparison of our models in CIFAR-100, we continue to fine-tune the best model on the STL-10 dataset. The best model in the previous section was naturally the convolutional neural network with four layers, due to its complexity and of course the increased number of layers that allow the model to perceive the patterns behind the data more easily.

However, even though we choose this model, we do not select it with the previous settings. As we aim to maximize our results and get the best accuracy possible, we perform further modifications in the architecture that we analyse in section Section 4.6.2. Note that this model achieves 32% accuracy in CIFAR-100 proving that it is the best-performing model.

4.6.1 Data Preprocessing

Data preprocessing is crucial in deep learning and this is why we prepare the dataset with a few adjustments before fine-tuning our model. Overall, we resized the images from 96×96 to 32×32 pixels to match our models' input dimensions. Also, data augmentation techniques, such as random crops and horizontal flips, were applied to enhance the training data and prevent overfitting.

4.6.2 ModConvNet_2 Architecture

We introduced the ModConvNet_2 architecture for fine-tuning on STL-10, which is a more sophisticated version of the four layer convolutional neural network of the previous dataset. This architecture has five convolutional layers with increasing feature map sizes: 32, 64, 128, 256, and 512. We added two extra convolutional layers and two additional fully connected layers. Finally, we added dropout layers after the fully connected layers to deal with overfitting.

We designed the architecture to incrementally increase the depth and capacity of the network. The extra layers enable the model to learn high-level features necessary for distinguishing between the classes in STL-10, which are more complex than those in CIFAR-100. Also, we loaded the pre-trained weights from the ModConvNet_2 trained on CIFAR-100 and modified the output layer to have five neurons corresponding to the five classes in STL-10. Fine-tuning was performed with a lower learning rate of 1×10^{-4} to adjust the pre-trained weights to the new dataset.

4.6.3 Discussion

The fine-tuned model achieved a test accuracy of 75% on the STL-10 dataset. Fine-tuning leverages the learned features from the source dataset (CIFAR-100) and adapts them to the target dataset (STL-10), resulting in improved performance compared to training from scratch, which was proved to be true in this case.

We attempted further hyperparameter tuning, adjusting the learning rate scheduler and optimizer settings, to enhance the model's performance. However, significant improvements were limited, suggesting that the model architecture is more crucial than hyperparameter adjustments at this stage.

In the next section, we will visualize our results hoping to get insights behind the performance of our models.

5 Visualization

In this section, we will add some plots and tables that we generated throughout this project. These plots aim to explain our results and give feedback that could raise questions about further improvements.

5.1 Training and Validation Curves

Below we present the training and validation curves for all of our models, including the baseline models, the ones with 2 extra layers, as well as the tuned baseline models.

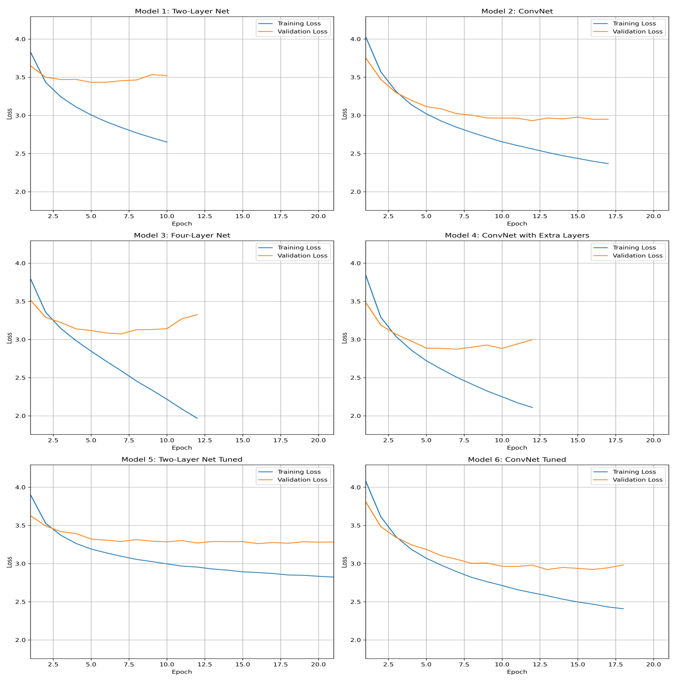

Figure 1: A full comparison of the training and validation losses of our models.

It is apparent that the models with the more complicated architecture are better as they achieve better validation loss. Another key note is that the gap between training and validation losses is smaller for the models with enhanced architectures and tuned hyperparameters. This means that these models are more generalized to unseen data, as they are not simply memorizing the training data but they learn underlying patterns.

5.2 Modified Convolutional Neural Network

In this project, we ran multiple models with various settings, but the best model was the most intricate one, the one we used to fine-tune STL-10. This model was also the one with the most convolution layers, as it was designed to have 5. Below we present the learning curves of this model, which is pre-trained on CIFAR-100 and fine-tuned on the STL-10 dataset.

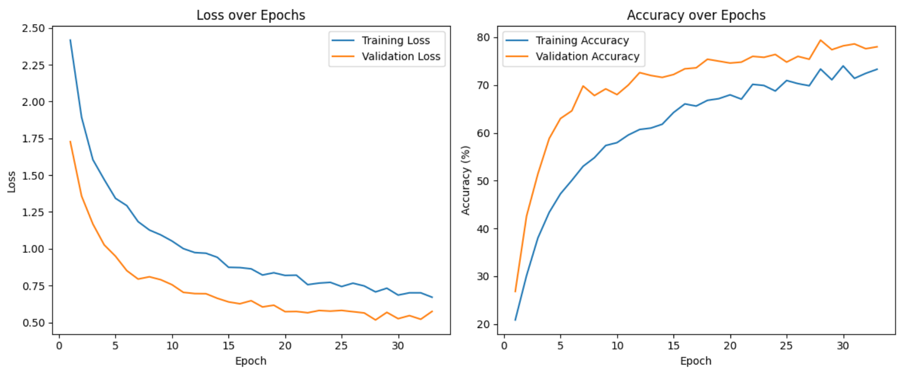

Figure 2: The accuracy curve as well as the learning curves for train and validation set, for the best model overall which is the convolutional neural network with 5 convolution layers.

We can see that we get a very good accuracy equal to 75 % and the loss curves depict a model that learns the patterns behind the data very well. The curves show a consistent decline over the epochs, while the validation loss closely follows the training loss, which suggests that the model is not overfitting and is generalizing well to unseen data.

5.3 Hyperparameter Results

The top hyperparameters for the two layer network:

Name	lr	batch_size	weight_decay	optimizer	val_acc	val_loss
wise-sweep-129	0.001	64	0.001	Adam	28.13	3.704
splendid-sweep-95	0.001	128	0.0001	Adam	27.76	3.668
major-sweep-128	0.001	32	0.001	Adam	27.50	3.589
trim-sweep-150	0.001	128	0.001	Adam	26.90	3.591
comfy-sweep-117	0.001	64	0.0001	Adam	26.68	3.667

Table 2: This table shows the top 5 unique model runs based on validation accuracy. "lr" stands for learning rate, "batch_size" is the number of samples per batch, "weight_decay" is the regularization parameter, "optimizer" indicates the optimizer used (Adam in this case), "val_acc" is the validation accuracy, and "val_loss" is the validation loss.

The top hyperparameters for the convolutional neural network with two layers are:

5.4 t-SNE Visualization

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a powerful dimensionality reduction technique particularly suited for visualizing high-dimensional data in two or three dimensions. It works by converting the similarities between data points into joint probabilities and then minimizes the Kullback-Leibler divergence between these probabilities in the high-dimensional and low-dimensional spaces. This approach effectively preserves the local structure of the data, making it easier to identify patterns and clusters.

Name	lr	batch_size	weight_decay	optimizer	val_acc	val_loss
wise-sweep-129	0.001	64	0.001	Adam	28.13	3.704
splendid-sweep-95	0.001	128	0.0001	Adam	27.76	3.668
major-sweep-128	0.001	32	0.001	Adam	27.50	3.589
radiant-sweep-21	0.001	64	0.001	Adam	27.24	3.630
trim-sweep-150	0.001	128	0.001	Adam	26.90	3.591

Table 3: This table shows the top 5 unique model runs for ConvNet based on validation accuracy. ”lr” stands for learning rate, ”batch_size” is the number of samples per batch, ”weight_decay” is the regularization parameter, ”optimizer” indicates the optimizer used (Adam in this case), ”val_acc” is the validation accuracy, and ”val_loss” is the validation loss.

In our project, we apply t-SNE to the feature representations learned by our fine-tuned model on the STL-10 dataset. We do this in order to visualize how well the model can distinguish between different classes at a feature level. In our case, the separation is quite good, except the distinction of deer and horses that are overlapping. This is not surprising as in comparison with the rest of the classes they look similar, which makes them more challenging to distinguish. This overlap suggests that the model may require additional features or data augmentation focused on these classes to improve discrimination.

We present our t-SNE plot below:

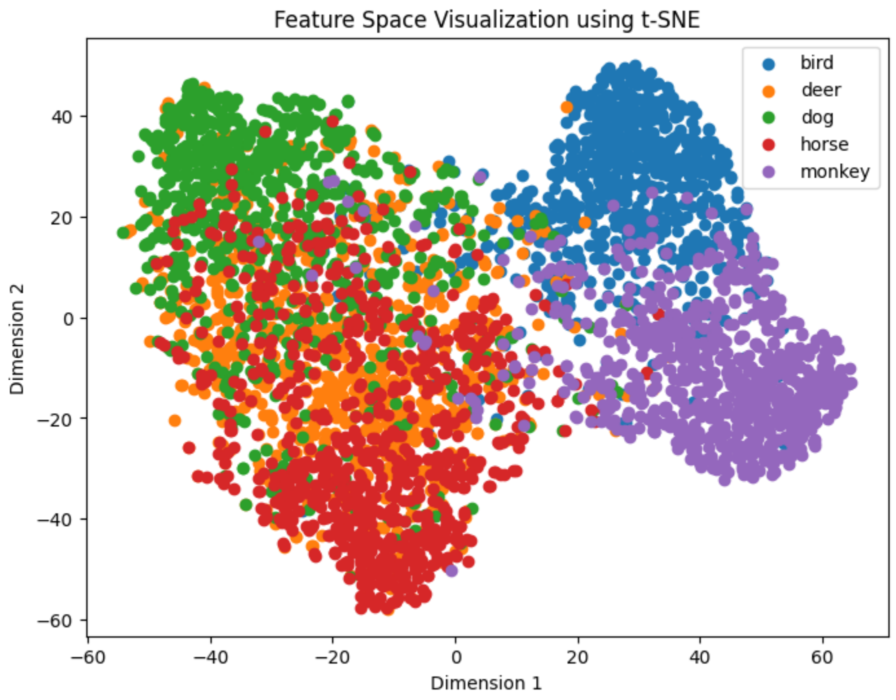

Figure 3: t-SNE visualization of feature embeddings from the fine-tuned model on STL-10. Each color represents a different class. Most classes are well-separated, except for deer and horse, which show significant overlap.

6 Conclusion

Our experiments demonstrate that increasing network depth and carefully designing the architecture significantly improve image classification performance on complex datasets like CIFAR-

100 and STL-10. While hyperparameter tuning enhances model performance, it cannot compensate for the limitations of shallow architectures. Fine-tuning pre-trained models on new datasets leverages learned representations and accelerates convergence, leading to better results.

Last but not least, it is important to mention that these are basic methods of computer vision and deep learning and in the modern era there are far more sophisticated techniques that will prevail over these methods. However, by investigating them we set good foundation for the more complex methods.

7 Appendix

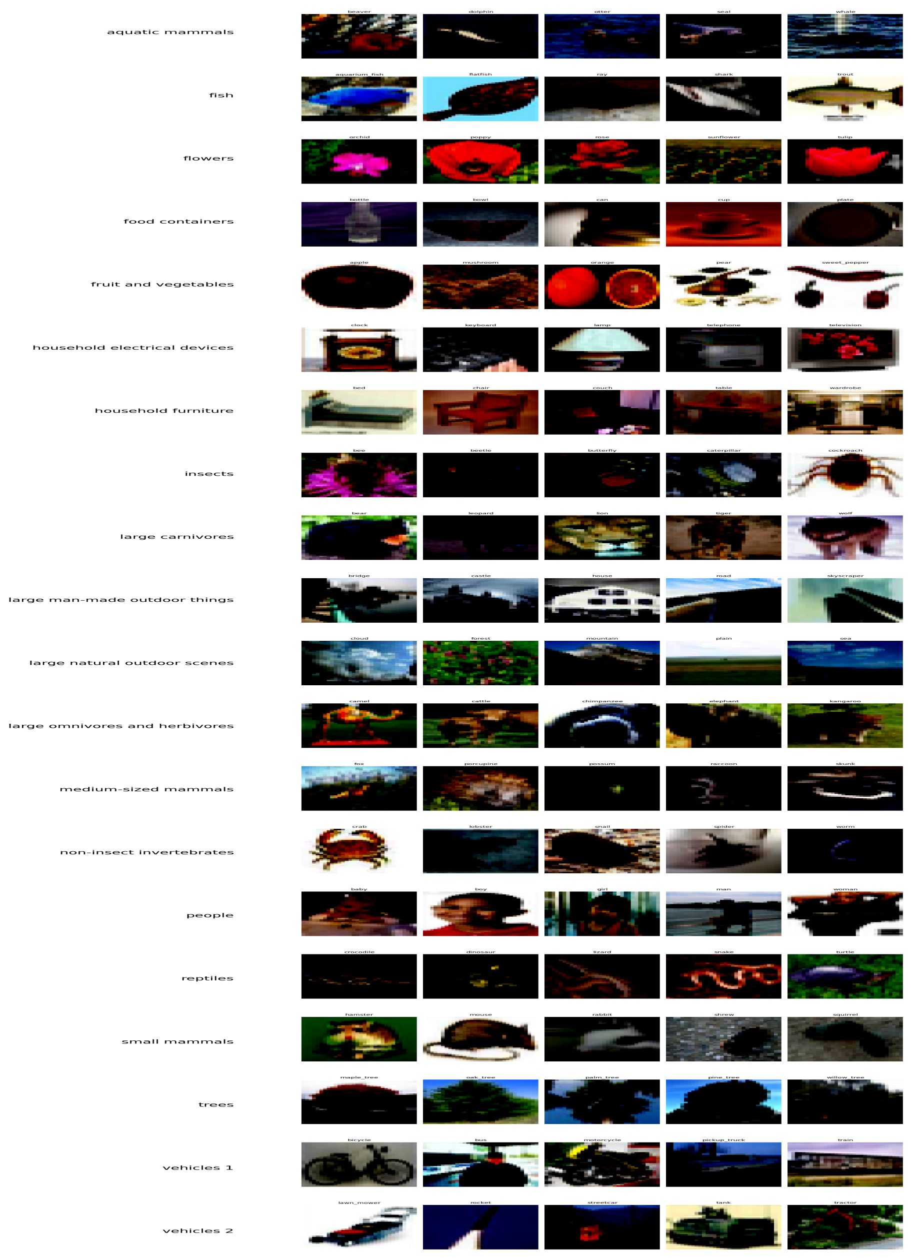

Figure 4: A preview of the CIFAR-100 dataset.

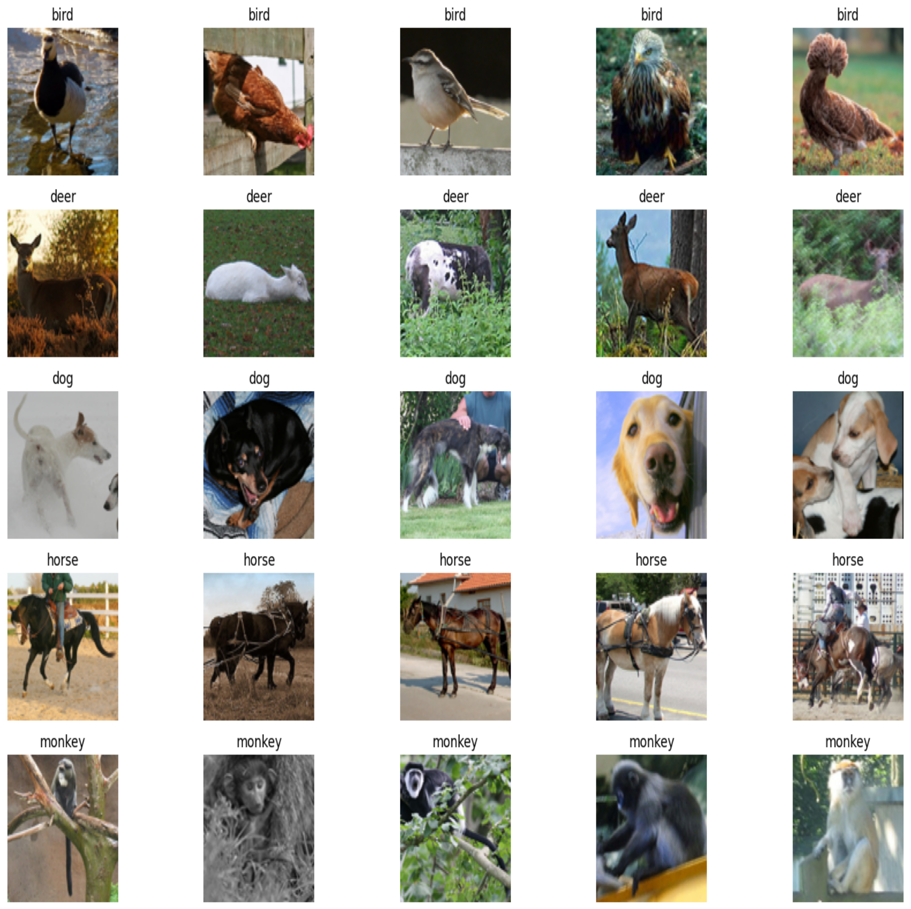

Figure 5: A preview of the STL-10 dataset.

References

[1] Adam Coates, Andrew Y Ng, and Honglak Lee. “An analysis of single-layer networks in unsupervised feature learning”. In: *Proceedings of the 14th international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 215–223.

[2] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[3] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. 2009. URL: `https://www.cs.toronto.edu/~kriz/cifar.html`.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[5] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[6] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).