UNIVERSITY OF AMSTERDAM

FINAL LAB PART 2

Image Classification using Convolutional Neural Networks

October 18, 2024

Students:
Lisanne Wallaard
15735664

Jose L Garcia
15388867

Julio Smidi
11307943

Group:
Group 9

Lecturer:
Martin Oswald and Arnoud Visser

Course:
Computer Vision 1

1 Introduction

In computer vision, one of the most fundamental challenges is accurately classifying objects in images. Solving this issue opens doors to many different types of applications, such as autonomous driving, medical diagnostics, and monitoring a diverse range of objects in different environments. On top of being a challenging task, achieving high accuracies requires a lot of effort, with systems that include effective feature extraction and learning mechanisms.

In this project, we aimed to address the challenge of image classification on a complex and diverse dataset like CIFAR-100, which, as it name suggests, contains 100 objects of different categories. To do this, we implemented and compared two neural networks (NN) architectures: a simple two-layer fully connected network (TwoLayerNet) and a convolutional neural network (CNN) (ConvNet) inspired by the LeNet-5 architecture. Our goal was to obtain a high accuracy on the tested CIFAR-100 dataset while also understanding the limitations and advantages of each model. Additionally, the project explores the possibility of transfer-learning by using the learnt weights and biases of the trained ConvNet to fine-tune it for a subset of the STL-10 dataset.

The project involved multiple steps including visualizing the datasets, and, with PyTorch, building different sets of both models architectures, building custom data loaders, performing data augmentation, training the models with the best possible optimizer, and finding the best hyperparameters to improve the performance of each network.

With the selected approach, the project aims to provide a deeper understanding of the pipelines that must be followed to design and train a neural network for complex image classification tasks.

2 Methods

2.1 CIFAR-100 Dataset

The first task was to use an appropriate dataset for complex image classification. Since on the previous part of the laboratory we used the CIFAR-10 dataset, for this project the chosen dataset was the CIFAR-100 dataset, which consists of 60.000 mutually exclusive 32x32 colour images divided across 100 labelled classes. Each class consists of 600 images and the dataset is pre-divided into 50.000 training images, with 500 randomly selected images from each class, and 10.000 testing images, with 100 images per class. Since the dataset is very diversified, it is also divided into

superclasses and their respective subclasses. 1.

Superclass	Classes
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Figure 1: CIFAR-100 dataset superclasses and subclasses

On the appendix, we included a figure 10 which shows a sample from each of the subclasses present in the superclasses of the dataset.

Having downloaded the dataset, we proceeded to create the architectures that we would use to try and classify this complex dataset.

2.2 Implementing the TwoLayerNet and ConvNet Architectures

Both networks were built using the Pytorch Python library, and aim to take input images of size 32x32x3, corresponding to the processed CIFAR-100 images. Its implementation involved the definition of a Class for each architecture and defining the __init__ methods, where the layers and elements are defined, and a forward methods, to control how the data is processed through those layers and the model trained.

- The TwoLayerNet consisted of a fully connected neural network with, as its name suggests, two dense layers with a ReLU activation function between them. The networks take the input images and process them through a hidden layer. Finally, the output is a probability score for each of the 100 images. This model is quite simple and served as a baseline to compare the more complex ConvNet model.
- The initial ConvNet design was based on the LeNet-5 architecture (Lecun et al. 1998), one of the first and most important models for image classification. The network consisted of two convolutional layers for feature map extraction, followed by three fully connected layers to create the classification score. Our design used several conventions also used on the original LeNet-5 architecture, such as a tanh activation function, an average pooling layer and the same sizes for the fully connected layers (400, 120 and 84, respectively).

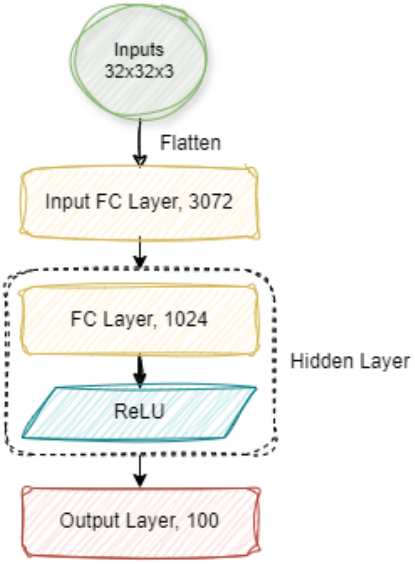

Figure 2: TwoLayerNet Architecture

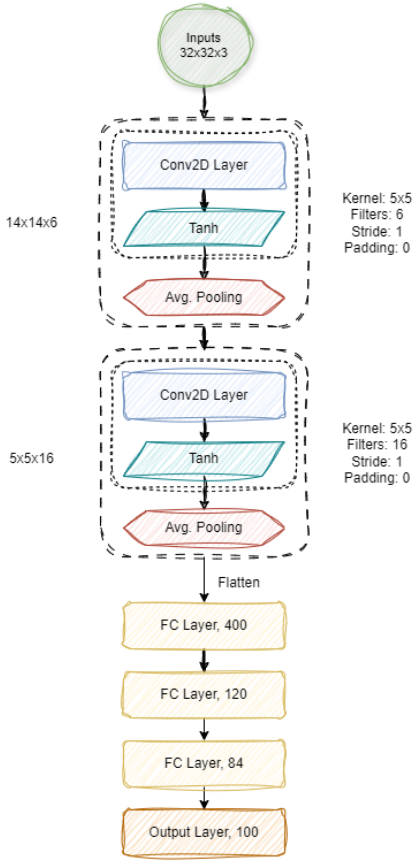

Figure 3: ConvNet Architecture

Using PyTorch's integrated utilities, we calculated that for the ConvNet architecture, the number of trainable parameters on the last fully connected layer was 10.164, while the total number of trainable parameters was 69.656.

2.3 Data Preparation and Augmentation

To prepare the data for training, we implemented a custom dataset class using PyTorch's DataLoader to make the dataset iterable. The custom class was called CIFAR100_loader.

We also created a transformer that implemented data augmentation techniques for the training dataset such as horizontal flipping and random rotations of 10 degrees. This would increase the diversity of the training samples and make sure the models would generalize better, preventing overfitting. Normalization was also applied to each image to improve training efficiency, helping the network converge faster.

Finally, we defined an optimizer to adjust the model's parameters during training, enabling the network to learn from the augmented data. The algorithm we used for this task was the Adaptive Moment Estimation (Adam) optimizer. Adam is an algorithm commonly used for training CNNs that adjusts the learning rate for each parameter, leveraging estimates of the gradients' mean and variance to improve parameter updates.

The method uses the following formulas to update the learning weights and biases:

$$\begin{aligned} x_t &= \beta_1 \cdot x_{t-1} + (1 - \beta_1) \cdot g_t \\ y_t &= \beta_2 \cdot y_{t-1} + (1 - \beta_2) \cdot g_t^2 \\ \Delta\omega_t &= -\eta \cdot \frac{x_t}{\sqrt{y_t} + \epsilon} \\ \omega_{t+1} &= \omega_t + \Delta\omega_t \end{aligned}$$

Where:

- η is the learning rate, controlling the step size during training. It is the main parameter that typically needs manual tuning.
- g_t is the gradient of the loss function with respect to the parameter ω_j at time step t. It indicates the direction and rate of change needed to minimize the loss.
- x_t is the exponential moving average of the gradient along ω_j at time t, which introduces a form of momentum.
- y_t is the exponential moving average of the squared gradient along ω_j at time t, which helps to adapt the learning rate by accounting for the magnitude of the gradient over time.
- β_1, β_2 are decay rates, typically set to 0.9 and 0.999, respectively, based on empirical results. They control how much of the past gradients are retained in the moving averages.
- ϵ is a small constant (e.g., 10^{-8}) added for numerical stability.

The Adam algorithm results in a faster, more efficient convergence and stability during training.

2.4 Training the Models

We trained both the TwoLayerNet and ConvNet models using PyTorch's implemented Cross Entropy loss function, which measures the dissimilarity between the true labels and predicted probabilities. It follows the following formula:

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c})$$

Where:

- N: The total number of samples in the batch.
- C: The total number of classes.
- $y_{i,c}$: The ground truth label for class c of sample i
- $\hat{y}_{i,c}$: The predicted probability that sample i belongs to class c.

Additionally, we implemented data loaders, the Adam optimizer and the augmentation strategies previously mentioned. The Adam optimizer algorithm also lets us select the desired value for the initial learning rate α, which we set to $\alpha = 0.001$, a common initial value for training CNNs.

The training process involved iterating over the training dataset for a specified number of epochs, which we initially set to 15. Since we were implementing the Stochastic Gradient Descent (SGD) algorithm, we calculated the loss and updated the model's parameters using backpropagation for each individual training image.

2.5 Hyperparameter Tuning and Model Improvement

In order to further improve both models' performances, we created two new models that added batch normalization layers to the original architectures. We called these new models TwoLayerNetWithBN and ConvNetWithBN respectively. Batch Normalization (BN) is a technique used to accelerate deep learning processes to reduce the variance between the input values of a network. This means that the inputs are linearly transformed to have zero mean and a unit variance. By doing this, we can make the network converge faster.

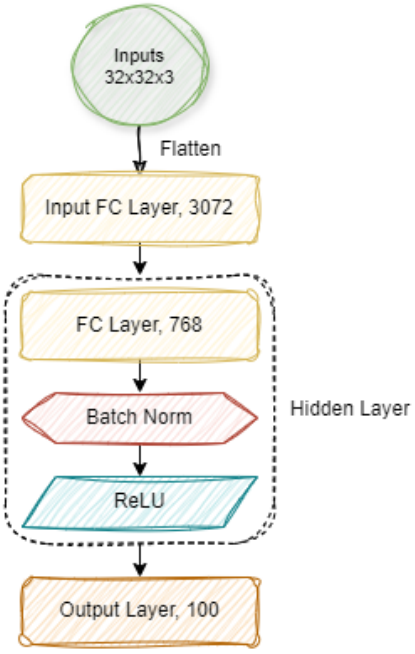

Figure 4: TwoLayerNet with Batch Normalization Architecture

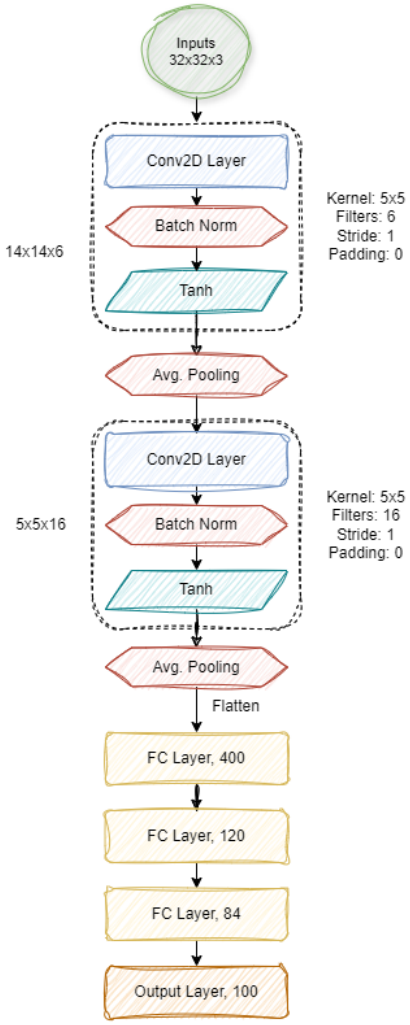

Figure 5: ConvNet with Batch Normalization Architecture

Using these new models, we conducted an extensive hyperparameter tuning using the Optuna package in order to obtain the best results possible.

The search process involved tuning parameters such as learning rate, batch size, number of epochs, optimizer choice and the chosen optimizer's parameters. Additionally, we added weight decay or L2 regularization to our selected optimizer in order to further prevent overfitting and improve the network's generalization. The weight decay constant was tuned. Finally, we also added a learning rate scheduler to improve the results of our network as each training epoch passed. Our aim with this was to make the network converge faster when the loss function was getting stuck when oscillating around a minimum. The scheduler choice was also tuned along with its parameters. We experimented with PyTorch's StepLR and ReduceLROnPlateau schedulers.

We also added model-specific changes: for the TwoLayerNetWithBN model, we also tuned the size of the hidden layer, while for the ConvNetWithBN model, we changed the activation function to a more optimal one: ReLU.

Finally, before starting to train these new models, we added a validation set in order to analyze the network's accuracy performance after each epoch. Once the hypermeters were tuned, we trained the new models to see how much they had improved.

To end this section, we also experimented with improving the constructed models by adding additional layers to the original architectures. We named these improved versions TwoLayerNetImproved and ConvNetImproved. These improved, more complex networks added two fully connected and three convolutional layers respectively, along with batch normalization layers, max pooling layers, dropout layers, skip connections and had a better activation function (LeakyReLU). Additionally, we changed the optimizer to AdamW, a slight variation of Adam which adds a different approach to weight decay, applying it directly to the weights before updating them, and thus, leading to better convergence and generalization.

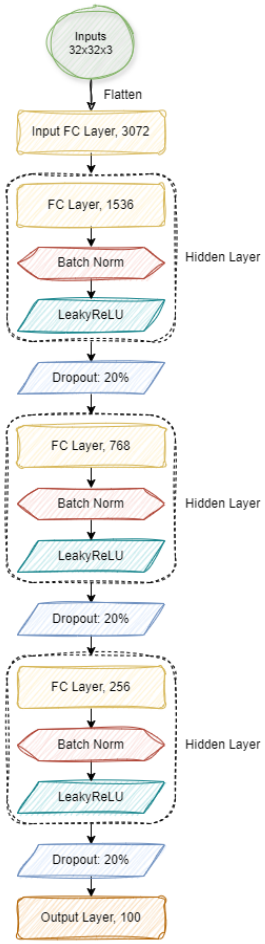

Figure 6: Improved TwoLayerNet Architecture

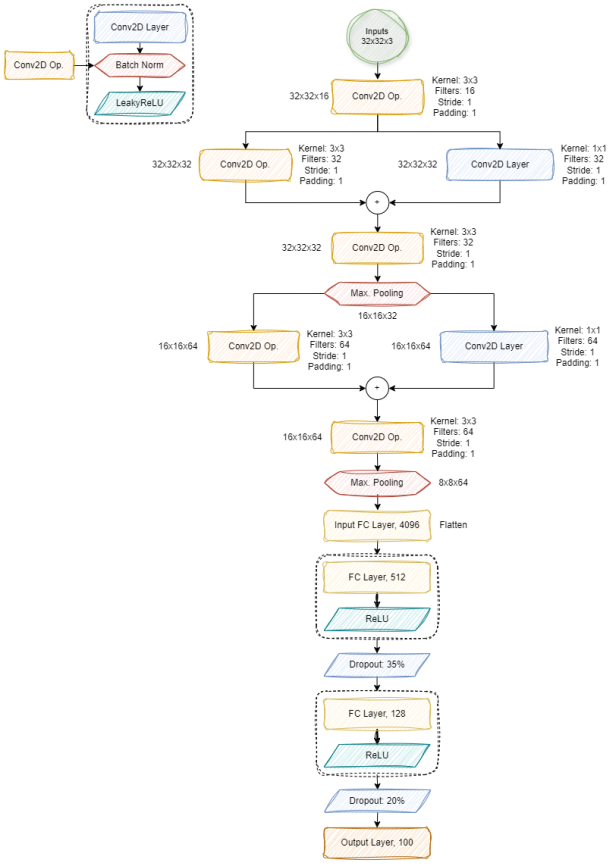

Figure 7: Improved ConvNet Architecture

These improved models were trained using similar hyperparameters to the ones found for the previous models.

2.6 STL-10 Dataset

Next, we move on to the STL-10 dataset. This dataset has a higher resolution than the CIFAR-100 dataset as the images consist of 96x96 pixels. Furthermore, it contains different classes: {airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck }. The dataset has 5000 labeled training images and 8000 labeled test images, which gives us per class 500 labeled training images and 800 labeled test images. There are also 100000 unlabeled images that may not exactly belong to the 10 classes. We only use a subset of the STL-10 dataset, namely the labeled data belonging to the following classes: {bird, deer, dog, horse, monkey }. Figure 8 visualizes 5 images per class in this subset of the STL-10 dataset.

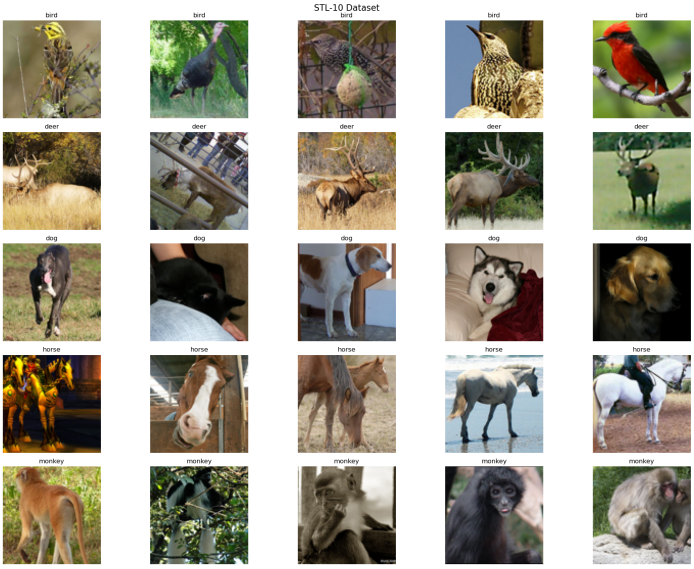

Figure 8: Visualization of 5 sample images for the classses {bird, deer, dog, horse, monkey} in the STL-10 dataset.

2.7 Fine-tuning on the STL-10 Dataset

We fine-tuned on the subset of the STL-10 dataset using the ConvNetImproved model, using the model parameters that were found optimal when performing hyperparameter tuning. The output layer of the model was adjusted to be able to deal with the lower number of classes for this dataset.

3 Results

3.1 TwoLaterNet and ConvNet

Our base models were trained with the following configurations 1:

Hyperparameters	**TwoLayerNetWithBN**	**ConvNetWithBN**
Learning Rate	0.001	0.001
Batch Size	64	64
Epochs	15	15
Hidden Layer Size	1024	-

Table 1: Comparison of Hyperparameters for TwoLayerNetWithBN and ConvNetWithBN models

Our results using the base models were as follows 2:

Model Accuracy	Training (%)	Test (%)
TwoLayerNet	29.82	22.49
ConvNet	27.08	26.14

Table 2: Comparison of Training and Test Accuracies for TwoLayerNet and ConvNet models

3.2 TwoLaterNetWithBN and ConvNetWithBN

As mentioned before, in order to see how much these results would improve by adding Batch Normalization layers, we also applied hyperparameters tuning with the optuna package. After testing, the final configurations for each model were the following 3:

Hyperparameters	TwoLayerNetWithBN	ConvNetWithBN
Learning Rate	0.001	0.001
Batch Size	64	64
Epochs	20	25
Hidden Layer Size	768	-
Weight Decay Constant	1×10^{-5}	1×10^{-4}
Scheduler	ReduceLROnPlateau	ReduceLROnPlateau
Patience	5	5
Learning Rate Decay Factor	0.30	0.29

Table 3: Comparison of Hyperparameters for TwoLayerNetWithBN and ConvNetWithBN models

With these configurations, the results for each model were as follows 4:

Model Accuracy	Training (%)	Validation (%)	Test (%)
TwoLayerNetWithBN	39.12	28.90	30.06
ConvNetWithBN	34.26	33.16	33.01

Table 4: Comparison of Training, Validation, and Test Accuracies for TwoLayerNetWithBN and ConvNetWithBN models

3.3 TwoLaterNetImproved and ConvNetImproved

Finally, in order to make our results as good as possible, we trained the Improved models with the following configurations 5:

Hyperparameters	TwoLayerNetImproved	ConvNetImproved
Learning Rate	0.001	0.001
Batch Size	64	128
Epochs	40	50
Hidden Layer 1 Size	1536	-
Hidden Layer 2 Size	768	-
Hidden Layer 3 Size	256	-
Weight Decay Constant	1×10^{-5}	1×10^{-4}
Scheduler	ReduceLROnPlateau	ReduceLROnPlateau
Patience	5	5
Learning Rate Decay Factor	0.30	0.29

Table 5: Comparison of Hyperparameters for TwoLayerNetImproved and ConvNetImproved models

With these configurations, the results for each model were as follows 6:

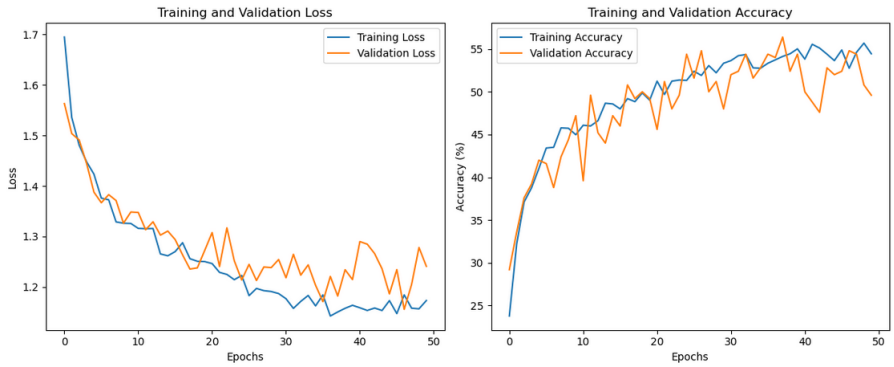

Figure 9: Training and validation accuracy and loss curves for the ConvNetImproved model finetuned on the STL-10.

Model Accuracy	Training (%)	Validation (%)	Test (%)
TwoLayerNetImproved	48.43	34.76	35.88
ConvNetImproved	76.59	57.06	57.19

Table 6: Comparison of Training, Validation, and Test Accuracies for TwoLayerNetImproved and ConvNetImproved models

3.4 Fine-Tuning on STL-10

Fine-tuning the ConvNetImproved model on the STL-10 dataset, we obtain a 56.80% accuracy score on the test set. The training and validation accuracy and loss curves are shown in Figure 9. The model displays quite a good performance on the STL-10 set, especially given its initial performance on the CIFAR-100.

4 Discussion

Models Accuracy	Training (%)	Validation (%)	Test (%)
TwoLayerNet	29.82	-	22.49
ConvNet	27.08	-	26.14
TwoLayerNetWithBN	39.12	28.90	30.06
ConvNetWithBN	34.26	33.16	33.01
TwoLayerNetImproved	48.43	34.76	35.88
ConvNetImproved	76.59	57.06	57.19

Table 7: Comparison of Training, Validation, and Test Accuracies for Different Models on the CIFAR-100 Dataset

4.1 TwoLaterNet and ConvNet

While these results are very poor for both models, they were also expected given the simplicity of the used architectures and the complexity of the dataset. Both models lack the capacity to capture the dataset complexities accurately.

The TwoLayerNet struggles with the spatial nature of these images, resulting in its poor accuracy. The ConvNet, although not significantly better, still benefits from its convolutional layers, which enable it to create better feature maps, demonstrating its advantage over the fully connected model.

4.2 TwoLaterNetWithBN and ConvNetWithBN

While these results are still poor, we demonstrated that adding batch normalization layers and tuning the hyperparameters improves the training process and the testing results. We obtained better accuracies for both models. These changes benefited the TwoLayerNet model the most, resulting in an 8% increase in accuracy on the testing dataset. For the ConvNet model, improvements were not as significant since the architecture is still not sophisticated enough. However, the ConvNet model still proved to be the better option by achieving better testing results.

4.3 TwoLaterNetImproved and ConvNetImproved

While still relatively low, these results showed a significant improvement over our original ones.

We demonstrated how the use of three hidden layers, batch normalization and dropout on the TwoLayerNetImproved model can help the network learn more complex patterns and generalize them. However, fully connected models are still limited in their ability to capture the intricate features of complex images. Adding improvements can certainly enhance results but only up to a point, which remains far from ideal.

On the other hand, the complex CNN model ConvNetImproved proved to be much more capable of learning these complex patters and scaling its features. By only adding three convolutional layers, batch normalization, dropout and skip connections, results were improved by almost 30% on the testing dataset, a significant boost.

Ultimately, the convolutional models proved to be the better option, consistently achieving better performance than the fully connected models. Moreover, by using deeper and more complex architectures, these results can still be further improved.

4.4 Fine-Tuning on STL-10

The fine-tuning results show that the features learned in the initial training are an informative basis for fine-tuning on the STL-10 dataset with new classes. Both the CIFAR-100 and the STL-10 contain images of natural scenes and animals, which might be one of the reasons for this success. This shows that our convolutional model, and convolutional models in general are flexible models for visual object recognition.

5 Conclusions

- Convolutional Neural Networks are better for classifying data from images. Our results show how the convolutional architectures consistently outperform the fully connected models. This happens due to their ability to create feature maps that capture spatial hierarchies and patterns from the images, which the fully connected networks struggle with. Even at the most simple architectures, the convolutional models always proved to be the better option.
- Batch normalization improves the training and generalization of any network: Models that incorporated batch normalization layers demonstrated to have better training processes (with lower loss function values) and improved generalization.
- Model complexity helps handle complex datasets: our results prove the accuracy constantly improves the deeper and more sophisticated the network is.
- Using residual connections, dropout, weight decay, schedulers for learning rate decay, better activation functions and optimizers does improve performance.
- Convolutional Neural Networks are well capable of being fine-tuned on a new data set, after having been trained on another.

References

Lecun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791.

A Appendix LaTeX code

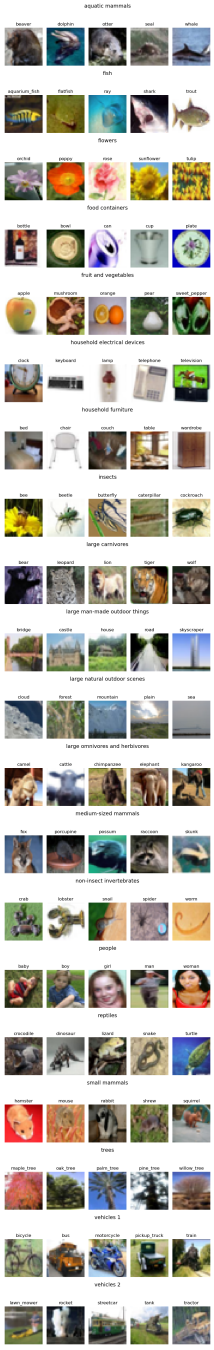

Figure 10: Samples from each subclasses and superclass of the CIFAR-100 Dataset.