UNIVERSITY OF AMSTERDAM

PART 2

CNNs for Image Classification

October 24, 2021

Students:

Piyush Bagad
13677640

Mark Alence
13771272

Ankit
13608568

Group:
Group 38

Course:
Computer Vision 1

Abstract

In contrast to traditional handcrafted features, advent of CNNs have made end-to-end feature learning from data ubiquitous. The features are learnt from data rather than handcrafting and the bulk of innovation has shifted from handcrafting features to designing robust architectures. In this project, we consider two simple architectures and experiment thouroughly on two datasets: CIFAR-10 and STL-10. We study the impact of various hyperparameters and the effectiveness of transfer learning on the task of multi-label classification.

1 Introduction

In this assignment, we study, implement and experiment with two networks: fully-connected layers (FCN) and convolutional-layer based LeNet architecture. We perform extensive experiments on training-from-scratch and fine-tuning on two popular datasets: CIFAR-10 [2] and STL-10 [1].

2 Section 1: Image Classification on CIFAR-10

In this section, we focus on image classification (training-from-scratch) on CIFAR-10 dataset.

2.1 CIFAR-10 Dataset

The CIFAR-10 dataset [2] consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images and the splits are pre-defined by the authors. In Figure 1, we visualize sample images with at least one image from each class.

2.2 Architectures

As described in the assignment, we use two architectures: `TwoLayerNet` (two fully-connected layers) and `ConvNet` using the LeNet architecture.

1. The concerned networks are implemented in the notebook.
2. For the first convolutional layer in `ConvNet`, the kernel size is $\times 6 \times 3 \times (5 \times 5)$ i.e. spatially it is 5×5 for each of the 3 RGB channels and there are 6 such kernel blocks. The `F6` layer in `ConvNet` fully connects a layer with $L = 120$ neurons and one with $M = 84$ neurons. Thus, the number of parameters is $M \times L = 120 \times 84 = 10080$ parameters.

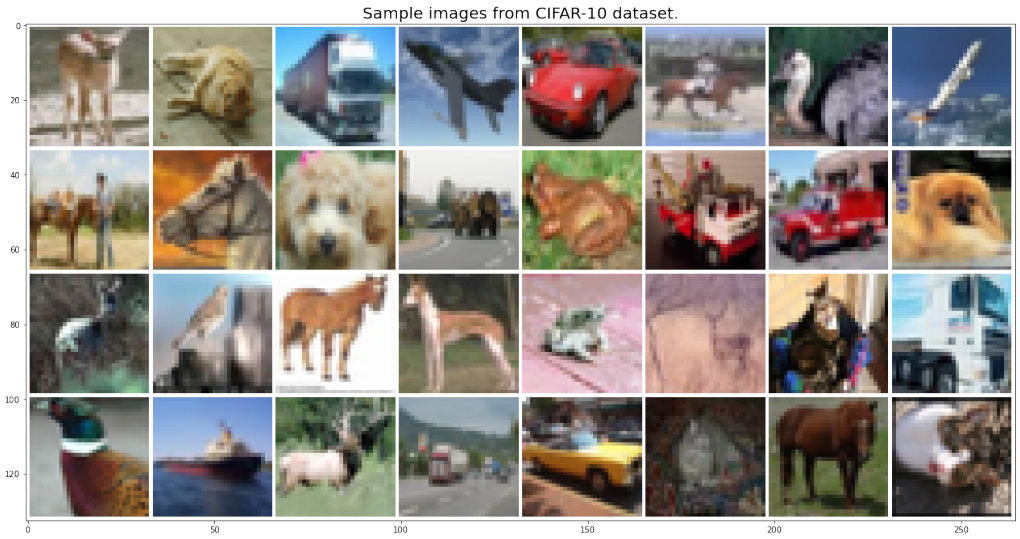
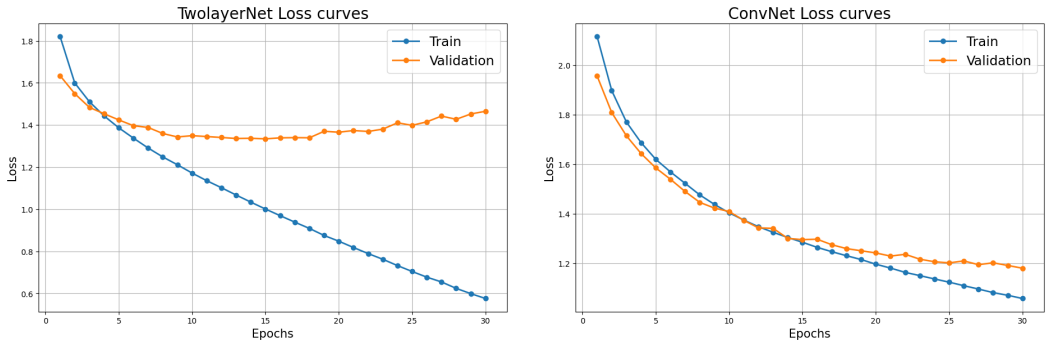


Figure 1: Sample images from all 10 classes in CIFAR-10 dataset.



(a) `TwoLayerNet` (b) `ConvNet`

Figure 2: Training and validation loss curves for the given architectures.

2.3 Training and Evaluation

1. The dataset object for CIFAR-10 has been implemented which takes in custom transforms to be able to use data augmentations.

2. Transforms and optimizer (as well as scheduler) are implemented as required.

3. We train both the architectures with default hyperparameters (`lr = 0.001, num_epochs = 30`). The loss plots for training and validation sets are shown in Figure 2. We split the overall training data into 40000:10000 deterministic split to create train and validation sets. Notice that with the default hyperparameters, there is severe overfitting in `TwoLayerNet`. `ConvNet` trains slightly slowly but seems to start overfitting towards the end.

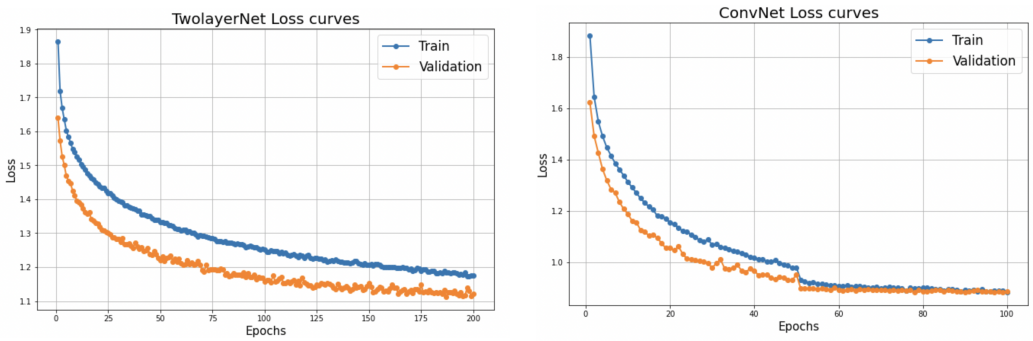
2.4 Setting up the hyperparameters

Hyperparameter Optimization and Evaluation

1. From the previous subsection, it is clear that the networks are overfitting to the training set since the gap between train and validation starts increasing after a point. To counter overfitting, we experiment with varying hyperparameters like epochs, batch size, optimizers (and learning rate schedulers), activation functions.

Net	Epochs	Batch	Optimizer (LR)	Scheduler	Augmentations	Accuracy
TLN	200	128	`Adam` ($1e^{-4}$)	MS (50, 100, 150)	A/F/GB	0.6026
CNN	100	128	`Adam` ($1e^{-3}$)	MS (50)	A	0.6716

Table 1: Optimal hyperparameters for the two networks on CIFAR-10 validation set. Here, augmentations A = Affine, GB = Gaussian Blur and F = Flips (horizontal), LR = learning rate and scheduler MS = Multi-step. Although we optimize performance on validation set, here, we report test set performance for the best hyperparameter setting.



(a) `TwoLayerNet` (b) `ConvNet`

Figure 3: Training and validation loss curves for the given architectures with best found hyperparameter setting.

2. Impact of additional layers in the architecture: As suggested we try adding two FC layers in TLN and two convolutional layers in CNN. Without data augmentations, it hurts to add two more layers since we are worsening the overfitting problem. With strong data augmentation, it does benefit slightly from overparametrization. We add two convolutional layers with CNN and achieve 0.7032 accuracy on test set. For TLN, we add two fully connected layers making the hidden layers sizes as [1024, 512, 128] and achieve 0.6078 accuracy on test set.

3. We report the best set of hyperparameters found for each of the two networks in Table 1. Some key observations are as follows:

   - `ReLU` activation works much better for `ConvNet` instead of the originally proposed `TanH` most-likely because our network outputs logits and softmax is applied within the loss function and thus there is no need to restrict output values within a small range.
   - Data augmentations like random affine really help in preventing overfitting. Without that, both the networks (especially TLN) overfit very quickly.
   - The loss plots for both networks with the best hyperparameters are shown in Figure 3.

4. Comparison of `TwoLayerNet` (TLN) and `ConvNet` (CNN)

   - The TLN does not exploit the spatial structure in images and flattens the image out to pass into FCNs. The CNN, in contrast, exploits the spatial structure and does weight sharing which ensures local translation invariance.
   - The total number of parameters in TLN is about 1.5 million whereas in CNN it is around 60k only. Given the relatively small dataset, TLN overfits much more since it has many more parameters.
   - In terms of performance, the CNN outperforms TLN on the test set (Table 1) and needs a higher learning rate than the latter.

Figure 4: Sample images from (given) 5 classes in STL-10 dataset.

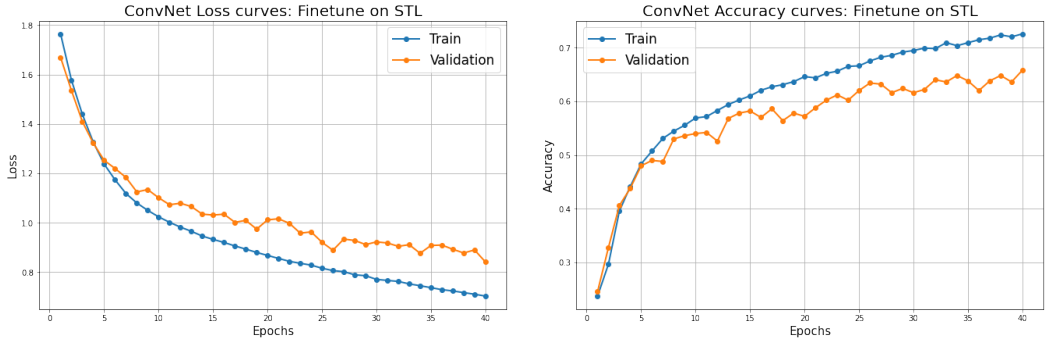


Figure 5: Training and validation loss curves for `ConvNet` finetuning on STL-10. We randomly split the train set into 2000-sized training set and 500-sized validation set.

3 Section 2: Fine-tuning the ConvNet

3.1 STL-10 Dataset

In this section, we use the STL-10 dataset [1] that was proposed in 2011. It has 10 classes with annotated images chosen from ImageNet-1000. It has 500 samples (size: $96 \times 96 \times 3$) in each class in the training set while 800 per class in the test set. For the purpose of this project, we only consider the given five classes: `airplanes, birds, ships, cats, dogs`. A randomly selected subset of samples is shown in Figure 4 consisting of two images per class.

3.2 Fine-tuning ConvNet

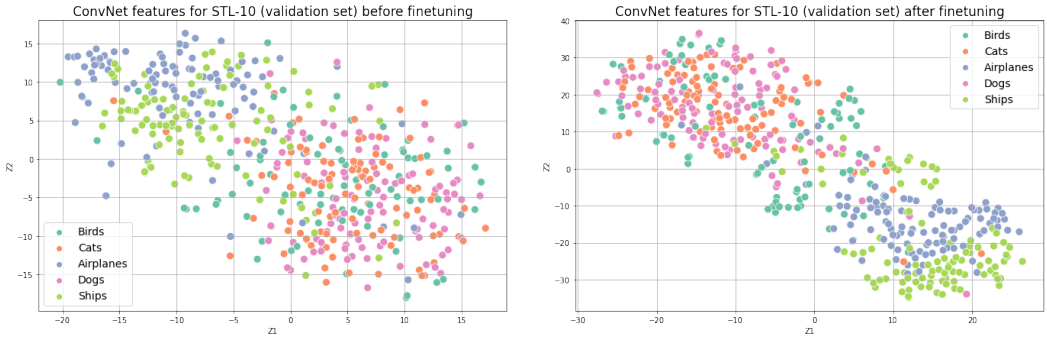
1. The relevant code has been added to the `ConvNet` class with a function `init_network()` to initialize the network weights from a checkpoint pre-trained on CIFAR-10.

2. **Training details**: First, we resize images to 32 x 32 to make training compatible. The default hyperparameter setting we use is: epochs 40, batch size 128, optimizer `Adam` with `lr` $1e^{-4}$, no scheduler, no data augmentations. The train and validation loss curves are shown in Figure 5 . On test set, we get an accuracy of **0.6675**.

3. **Visualizing feature space**: We also visualize TSNE-embeddings of the features produced by the network before and after fine-tuning (Figure 6). Note that classes like birds, dogs and cats are still hard to distinguish between after finetuning.

3.3 Performance Improvement

We try a bunch of techniques to prevent overfitting and get better performance on finetuning of STL-10 dataset. Our best performing model achieves accuracy of **0.6990** on the test set. The



(a) Before finetuning (b) After finetuning

Figure 6: Visualizing features of penultimate layer for`ConvNet` before and after fine-tuning on STL-10 (validation set).

Paradigm	Network	Frozen layers	Augmentation	Scheduler	Accuracy
Linear probing	FCN(84,5)	-	None	None	0.6740
Linear probing	FCN(84,5)	-	Affine	None	0.6820
Linear probing	FCN(84,32,5)	-	Affine	None	0.6680
Finetuning	CNN	None	None	None	0.6880
Finetuning	CNN	None	Affine	None	0.6990
Finetuning	CNN	None	Affine	MS([50])	0.6807
Finetuning	CNN	All conv	Affine	None	0.6870
Finetuning	CNN	All except final	Affine	None	0.6445

Table 2: Albation experiments on STL-10 finetuning. The best performing setting is marked in red.

key observations are listed below and ablations are reported in Table 2. We fix the optimizer `Adam(lr=0.0001)`, number of epochs to be 100 and batch size as 128.

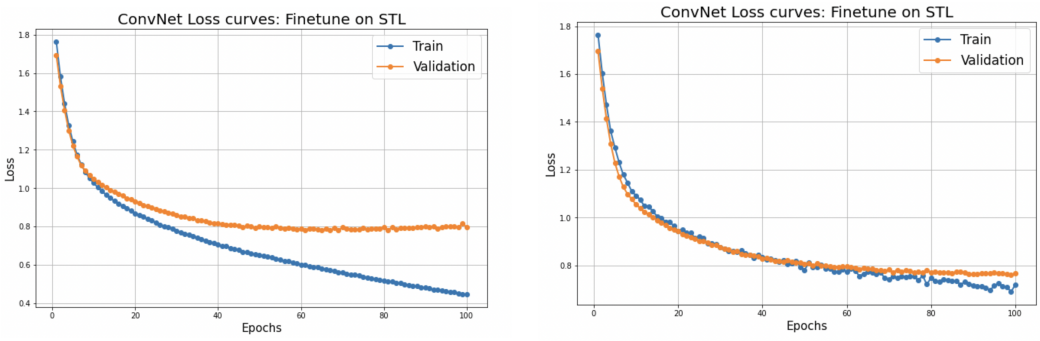
1. **Data augmentations**: We found data augmentations does help to imrpove performance marginally. But the gap between train and validation losses is far lesser with data augmentations as shown in Figure 7.

2. **Freezing layers**: Surprisingly, freezing layers gives slightly less performance than finetuning the entire model. This may be because the pre-training itself could have been better.

3. **Linear probing**: We also tried simply taking features from the penultimate layer and use fully-connected layers on top of them. This performs well but does not beat full finetuning.

4 Conclusion

In this project, we implemented two kinds of network architectures and tried training from scratch on CIFAR-10 dataset and transfer learning on STL-10 dataset. We learnt a lot of tricks (e.g. data augmentations) that could help prevent overfitting in these overparametrized networks. We also studied the effect of variety of other factors like learning rate, schedulers, batch size t



(a) Without data augmentation

(b) With data augmentation

Figure 7: `ConvNet` full-finetuning with and without data augmentations (affine).

References

[1] Adam Coates, Andrew Ng, and Honglak Lee. "An Analysis of Single-Layer Networks in Unsupervised Feature Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pp. 215–223. URL: https://proceedings.mlr.press/v15/coates11a.html.

[2] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *CIFAR-10 (Canadian Institute for Advanced Research)*. URL: http://www.cs.toronto.edu/~kriz/cifar.html.