Technical report on CNN Backbone with CIFAR-10

Mohammed Ataaur Rahaman (SID: 220843052)
m.a.rahaman@se22.qmul.ac.uk

Abstract

This is a classification task on an image dataset, CIFAR-10[1]. Here we use a custom model called Backbone, which majorly has Convolution layers stacked in a particular structure. We explore various aspects of model development and optimization, such as hyperparameter tuning, optimizer and scheduler selection, and overfitting prevention techniques such as data augmentation, batch normalization, dropout, and regularization. Our aim is to achieve the highest possible accuracy with the Backbone architecture. We report a maximum test accuracy of 90.51% on the CIFAR-10 dataset.

Introduction

We train a classifier model (Backbone) on the CIFAR-10 dataset[1], a collection of 60,000 color images in 10 classes. We perform various ablation studies to find the optimal hyperparameters for the model. This technical report consists of five main sections: Dataset, Model Architecture, Model Optimization, Model Training, and Results. We conclude with a discussion of future work and implications.

1 Dataset

1.1 CIFAR-10 Dataset

The CIFAR-10 dataset[1] contains 60,000 color images of 10 different objects, such as airplane, bird, cat, dog, etc. Each image has a size of 32x32 pixels and belongs to one of the 10 classes. The dataset is split into 50,000 training samples and 10,000 test samples. We download the dataset from torchvision.dataset.

1.2 Data Augmentation

We apply data augmentation to the training data to introduce variations in the dataset every epoch, which helps the model learn new features and prevent overfitting. We use torchvision transforms[2] to randomly perform operations such as horizontal flipping, cropping, and rotating the image. We normalize both the training and test sets.

1.3 Data Loader

We use pyTorch's DataLoader module to load the dataset with a batch size of 512. We follow the code from pytorch's tutorial[4] for most of the implementation here.

2 Model Architecture

2.1 Model

The model consists of N Backbones stacked sequentially, which helps to extract the features.The final backbone output is passed to a classifier (see Figure 1). Weights of linear and conv layers are initialized using Xavier Uniform.

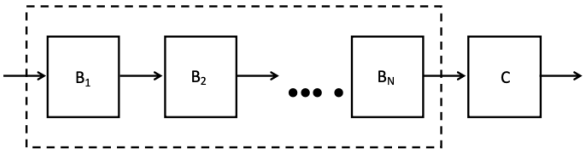

Figure 1: Model Architecture.

2.2 Backbone

Each Backbone consists of K Conv blocks whose weighted sum is calculated. The weights are calculated by a linear transformation of the Spatial Average Pool of the Backbone's input (X).

$$a = g(SpatialAveragePool(X)W)$$
$$O = BatchNorm2d(a_1 Conv_1(X) + .. + a_K Conv_K(X))$$

Input to the Backbone is X and the output is O. g, is a nonlinear sigmoid activation. Batch normalization is performed on the final O.

2.3 Classifier

The classifier does spatial average pooling to the last Backbone's output, and then passes to a Linear classifier with 10 classes output.

2.4 Conv Block

The Convolution block is defined as M ConvLayers which is a sequential layer having Conv2d, BatchNorm2d, ReLU, MaxPool2d, and Dropout.

3 Model Optimization

3.1 Loss Function

As this is a classification task, Cross entropy loss criterion is used to measure the loss of the model during the training phase.

3.2 AdamW Optimizer

The improvised version of Adam, that is AdamW[3] is used to perform the weight updates. Weight decay of 1e-4 is used.

3.3 LR Scheduler

We experimented with various LR Schedulers, such as Constant, Linear, Polynomial, Exponential, Step, Cyclic, OneCycle, ReduceLROnPlateau, and SequentialLR. We found that ReduceLROnPlateau performed the best, with the criterion of minimum validation loss for detecting a plateau.

4 Model Training

4.1 Trainer

The trainer class is implemented to help in training which has the train loops and the validation loops. This helps to track the performance of the model every epoch, and also plot training and validation curves.

4.2 Hyperparameters

Various Hyperparameters are listed in Table 1 & Table 2. Detailed ablation of hyperparameter experimentation conducted is in Appendix A.

Parameter	Value
Conv Kernel	3
Channels	[[3, 128, 256], [256, 512, 1028]]
K	[3, 3]
N	2
Conv Dropout	0.2
grad_clip	0.1

Table 1: Model hyper parameters

Parameter	Value
Loss Func	CrossEntropyLoss
Optimizer	AdamW(weight_decay=1e-4)
LR	1e-3
LR Scheduler	ReduceLROnPlateau

Table 2: Training hyper parameters

5 Results

5.1 Training Curves

Following training curves are given, Learning rate (Figure 2) and Loss curve (Figure 3).

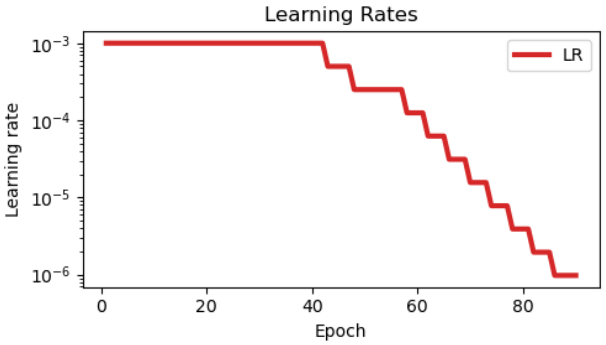

Figure 2: Learning rate curve.

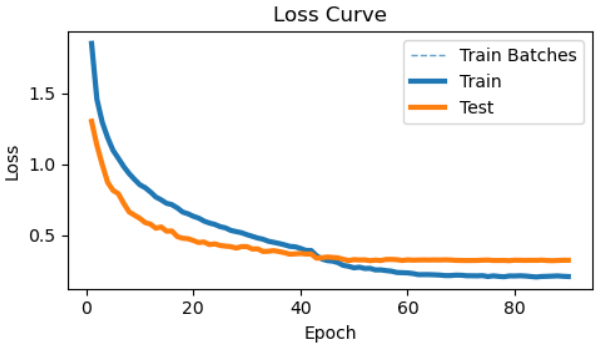

Figure 3: Train and test loss curve.

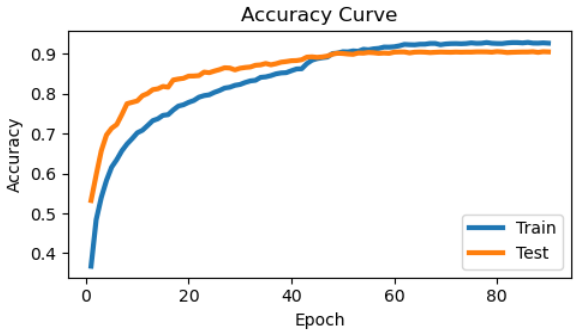

Figure 4: Train and test accuracy curve.

5.2 Test Accuracy

After 90 Epochs it is observed that the train Accuracy is at **92.68**%, and the test Accuracy has reached **90.51**%. (See Figure 4)

6 Conclusions

6.1 Conclusion and next steps

The CNN Backbone architecture achieved a maximum accuracy of 90.51% on the validation set. The model showed signs of overfitting in the later stages, and we attempted to use skip connections to mitigate this issue. However, further hyperparameter tuning with Optuna, RayTune, etc. is required.

6.2 Acknowledgement

I would like to express my gratitude to professor Dr. Yorgos Tzimiropoulos for providing me with this opportunity to learn and explore convolutional neural networks and their behavior by allowing me to experiment with different parameters and settings. I also thank Ioannis Maniadis (Course Demonstrator) for his guidance and feedback on my work. I appreciate the pytorch[5] documentation for helping me understand various components of nn.

7 References

[1] CIFAR-10 Dataset
https://www.cs.toronto.edu/~kriz/cifar.html

[2] Torchvision Transforms
https://pytorch.org/vision/main/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py

[3] AdamW https://arxiv.org/abs/1711.05101v3

[4] Pytorch Tutorial CIFAR
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

[5] Pytorch documentation
https://pytorch.org/docs/stable/index.html

Appendix A

			Data		Model					Trainer					Results				
S No.	Experiment Name	Version		Data Agumentation		K	N	Conv Channels	Conv Kernel	Image Sizes		LR	Optimizer	Scheduler	Batch Size		Epochs	Train Loss	Test Loss	Train Accuracy	Test Accuracy
1	Xavier Init	v0.2.2		Normalization		5	4	[3, 16, 32, 64, 128]	5	[32, 28, 24, 20, 16]		1.00E-04	Adam	-	4		50	1.41	1.47	-	48.7
2	AdamW optimizer	v0.3.0		"		5	4	[3, 4, 8, 16, 32]	5	[32, 28, 24, 20, 16]		1.00E-03	AdamW	-	64		40	1.47	1.52	-	47.1
3	Linear channels	v0.4.0		"		5	4	[3, 5, 7, 9, 11]	7	[32, 26, 20, 14, 8]		1.00E-03	AdamW	-	64		40	1.44	1.48	-	48.19
4	Conv + ReLU + MaxPool	v0.5.0		"		3	5	[3, 32, 64, 128, 256, 512]	3	[32, 29, 26, 23, 20, 17]		1.00E-04	AdamW		64		40	0.37	0.69	-	77.7
5	Add BatchNorm2D	v0.5.1		"		3	4	[3, 64, 128, 256, 512]	3	[32, 29, 26, 23, 20]		1.00E-03	AdamW	-	64		20	0.24	0.52	85.3	82.76
6	Add Transformations + Dropout + 3Layer MLP Classifer	v0.5.4		Add [resize, horizonatal Flip, random Affine, Color Jitter]		3	4	[3, 128, 256, 512, 1028]	3	[32, 16, 8, 4, 2]		1.00E-04	AdamW	-	128		40	0.11	0.78	92.42	80.75
7	Add StepLR Scheduler	v0.5.5		"		3	4	[3, 64, 128, 256, 512]	3	[32, 16, 8, 4, 2]		1.00E-03	AdamW	StepLR	256		40	0.32	0.41	89.04	86.52
8	CycleLR Scheduler	v0.5.6		[Horizonatal Flip, Crop, Rotate, normalize]		3	4	[3, 64, 128, 256, 512]	3	[32, 16, 8, 4, 2]		1.00E-03	AdamW	CycleLR	256		41	-	-	-	88.6
9	OneCycleLR Scheduler	v0.5.7		"		3	5	[3, 64, 128, 256, 512, 1028]	3	[32, 16, 8, 4, 2, 1]		1.00E-03	AdamW	OneCycleLR	256		40	0.21	0.41	92.62	87.41
10	ReduceLROnPlateau Scheduler	v0.5.8		"		3	5	[3, 64, 128, 256, 512, 1028]	3	[32, 16, 8, 4, 2, 1]		1.00E-03	AdamW	ReduceLR OnPlateau	256		40	0.17	0.37	94.06	88.73
11	PolyLR + CycleLR	v0.5.9		"		3	5	[3, 64, 128, 256, 512, 1028]	3	[32, 16, 8, 4, 2, 1]		1.00E-03	AdamW	PolyLR + CycleLR	256		52	0.32	0.49	88.71	84.73
12	ReduceLROnPlateau LR	v0.5.10		"		3	4	[3, 64, 128, 256, 512]	3	[32, 16, 8, 4, 2]		1.00E-03	AdamW	ReduceLR OnPlateau	256		62	0.25	0.32	91.19	89.46
13	ReduceLROnPlateau LR	v0.5.11		"		3	5	[3, 64, 128, 256, 512, 1028]	3	[32, 16, 8, 4, 2, 1]		1.00E-03	AdamW	ReduceLR OnPlateau	256		100	0.25	0.32	91.2	89.47
14	Deeper ConvBlock	v0.6.0		"		3	2	[[3, 16, 32], [32, 64, 128]]	3	[32, 8, 2]		1.00E-03	AdamW	ReduceLR OnPlateau	256		60	0.33	0.42	88.66	86.03
15	Deeper ConvBlock II	v0.6.1				3	2	[[3, 64], [64, 128, 512]]	3	[[32, 15], [15, 6, 2]]		1.00E-03	AdamW	ReduceLR OnPlateau	256		80	0.36	0.38	87.5	87.59
16	Resnet	v0.6.2		"		3	1	[[3, 64, 128, 256, 512]]	3	[[32, 32, 16, 16, 8, 4, 4, 2]]		1.00E-02	AdamW	OneCycleLR	256		10	0.91	1.27	68.27	58.46
17	Deeper ConvBlock III	v0.6.3		"		3	2	[[3, 128, 256], [256, 512, 1028]]	3	[[32, 16, 8], [8, 4, 2]]		1.00E-03	AdamW	ReduceLR OnPlateau	256		60	0.12	0.30	95.8	90.35
18	Deeper ConvBlock IV	v0.6.5		"		3	2	[[3, 128, 256], [256, 512, 1028]]	2	[[32, 16, 8], [8, 4, 2]]		1.00E-03	AdamW	ReduceLR OnPlateau	256		200	0.14	0.33	95.148	91.38
19	Deeper ConvBlock V	v0.6.6		"		3	2	[[3, 128, 256], [256, 512, 1028]]	3	[[32, 16, 8], [8, 4, 2]]		1.00E-03	AdamW	ReduceLR OnPlateau	256		90	0.21	0.32	92.686	90.51

Table 3: Experiments performed Experiments