Report Experiments Training Pipeline

Nechifor Alexandru - IAO1

November 2024

1 Introduction

The objective of the homework was to create a training pipeline that will ease the use of different models and parameters without needing to change the code every time. In order to accomplish this, I opted for JSON configuration files, which include the model used, the optimizer, the hyperparameters and different data augmentation options.

2 Points

The assignment required developing a training pipeline that enables training a variety of models using different optimizers. These models should be trained on specified datasets and support a range of data augmentation schemes for flexible experimentation. The datasets include a caching mechanism, while the training function allows early stopping, which can be modified from the configuration file. Beside those requirements, it is also required to run at least 8 experiments, with a diverse combination of models, optimizers and data augmentation schemes and, based on the results, to write a report which showcases the observations made.
The above-mentioned requirements are met, thus obtaining the score of 16 out of 20 points. The other 4 points are for obtaining a testing accuracy of 0.78 on **CIFAR100**, which I could not obtain on the predefined models, but also to ensure that the training pipeline is efficient, which I do not know if I could achieve the goal.

3 Configuration file

The format of the configuration file is as follows:

- device: can be ”cuda” or ”cpu”;
- dataset: can be ”MNIST”, ”CIFAR10” or ”CIFAR100”;
- model: contains information about the model:

  - model-defined: specify if the model chosen is defined or a custom one; this is only for future modifications (the pipeline is not configured to accept custom models at the moment);
  - model-name: specify the name of the model; it can be "resnet18-cifar10", "resnet18-cifar100", "PreActResNet18-CIFAR10", "PreActResNet18-CIFAR100", "MLP", "LeNet"

- batch-size: specify the batch size for both training and testing (in separate fields)
- shuffle: specify if the DataLoader should shuffle the dataset or not
- epochs: number of epochs for training
- transform: contains the data augmentation schema for the configuration. It contains diverse transformation that can be applied. For each transformation, the parameters can be given as a key-value pairs. The parameters names should be the same as the one given in the declaration
- optimizer: details about the optimizer. The parameters names should be the same as the one given in the declaration
- scheduler: details about the scheduler. The parameters names should be the same as the one given in the declaration
- loss: The loss function used
- logging: the logging directory for the TensorBoard summary
- early-stopping: configurations for the early stopping mechanism. The most important one is "patience"
- use-amp: true if the model should use autocast
- use-tta: true if Test-Time Augmentation is used
- tta-repeats: the number of repetaed augmentation for the testing data

4 Efficiency

The caching and early stopping mechanisms are part of the requirements. The training pipeline also allows the usage of half-precision tensors, by using autocast. The operations are in batches. Also, a GradScaler is used.

5 Results

In order to understand the following tables, I would denote the configuration options below:
Models:

- **resnet18**: The model used is **Resnet18** for **CIFAR100**, imported from the *timm* package.
- **PreActResNet-18**: The model used is **PreActResNet-18** for **CIFAR100**, whose implementation is taken from *Laboratory 2*.

Optimizers:

- **SGD**: The optimizer is **SGD** with a learning rate of **0.1**, momentum of **0.9**, and a weight decay of **0.0005**. For this optimizer, the **CosineAnnealingLR** scheduler is used with **T_max** of **200**.
- **AdamW**: The optimizer is **AdamW** with a learning rate of **0.001** and a weight decay of **0.0001**. For this optimizer, the **ReduceLROnPlateau** scheduler is used with the following parameters:
  - mode: min
  - factor: 0.1
  - patience: 15

Data augmentation schemes:
All the images are transformed to tensors and normalized. Because this is an expected behavior, it is not added into the configuration files. I will denote with **D1** and **D2** the two data augmentation schemes used.

- **D1** consists of a *random crop* and a *random horizontal flip*
- **D2** consist of a *random crop*, a *random vertical flip*, a *random 15 degrees rotation*, a *gaussian blur* and a *random erase*

Observation: The first configuration (Resnet + SGD + D1) has a rather large patience for the early stopping (150 epochs). This is due to the fact that I wanted to see the results presented by *HuggingFace*, the site on which the model is from. All the other configurations have a patience of 60 epochs.

The tables showcase the best training and testing accuracy for each configuration (the results are percentages):

	SGD		AdamW	
	D1	D2	D1	D2
Resnet	100.00	100.00	100.00	100.00
PreActResNet	89.33	86.86	100.00	99.85

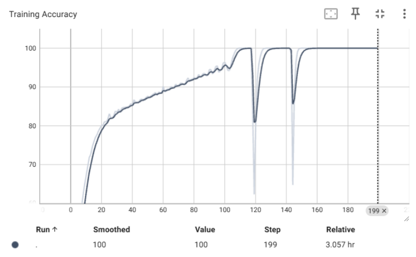
Table 1: Training accuracies

	SGD		AdamW	
	D1	D2	D1	D2
Resnet	62.30	41.69	54.49	38.36
PreActResNet	48.03	35.99	55.12	37.34

Table 2: Testing accuracies

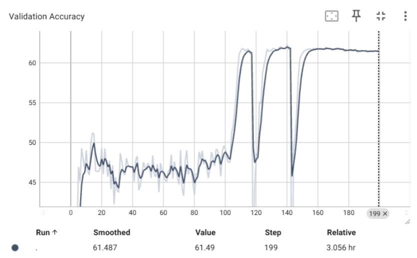
As it can be observed from the table, the models are overfitting on the training data. The convergence for the testing data is slow and often times can not be achieved if an early stopping mechanism is used with a low patience. Due to the fact that the training accuracy converges too fast, the gradient vanishes, thus any improvement to the test accuracy is due to the randomness of the neuronal network.
A possible improvement is to use test-time augmentation, but from previous experiments that I did not include in the above results, I found that there is no direct dependency between the testing augmentation and the convergence rate.
Another conclusion at which I arrived is that, with a low patience, training on CIFAR100 can lead to an early stop, before it can lead to better testing results.
The usage of a scheduler seems to greatly benefit the gradient, but it needs tunning depending on the model used.
As for the optimizers, **SGD** has a faster convergence rate for training data, while **Adam** and **AdamW** are slower. By finding the right combination of patience for the early stopping mechanism, hyperparameters and the model, a model is trained faster with **SGD**.
Last but not least, I observed that, by having a multitude of transformation, it reduces the training accuracy. One possible reason for that is that the model trains on images that become too different than the original ones, thus not recognizing the training images and classifying them wrong.

Below I will add the images for training and testing accuracies, which were generated by using Tensorboard:



Run ↑	Smoothed	Value	Step	Relative
● .	100	100	199	3.057 hr

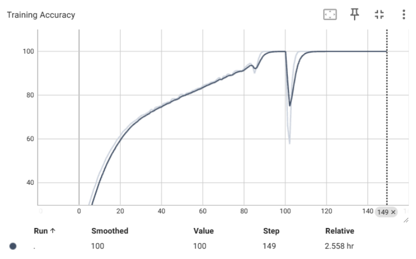
(a) Training config. 1



Run ↑	Smoothed	Value	Step	Relative
● .	61.487	61.49	199	3.056 hr

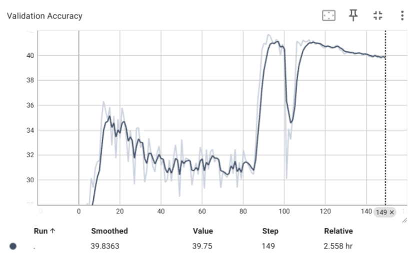
(b) Testing config. 1

Figure 1: Comparison between training and testing accuracy config. 1



Run ↑	Smoothed	Value	Step	Relative
.	100	100	149	2.558 hr

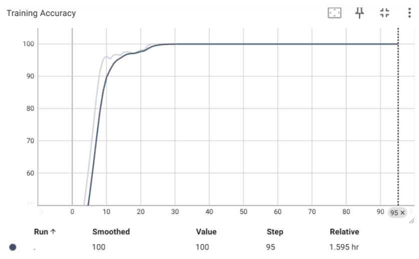
(a) Training config. 2



Run ↑	Smoothed	Value	Step	Relative
.	39.8363	39.75	149	2.558 hr

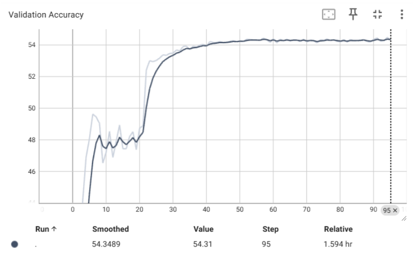
(b) Testing config. 2

Figure 2: Comparison between training and testing accuracy config. 2



Run ↑	Smoothed	Value	Step	Relative
.	100	100	95	1.595 hr

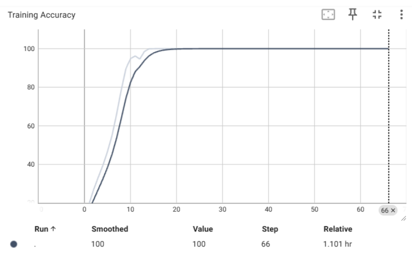
(a) Training config. 3



Run ↑	Smoothed	Value	Step	Relative
.	54.3489	54.31	95	1.594 hr

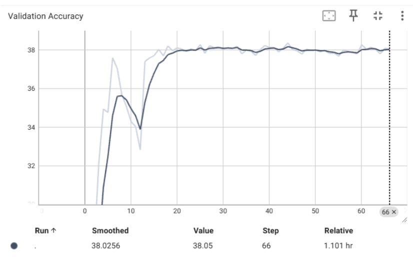
(b) Testing config. 3

Figure 3: Comparison between training and testing accuracy config. 3



Run ↑	Smoothed	Value	Step	Relative
.	100	100	66	1.101 hr

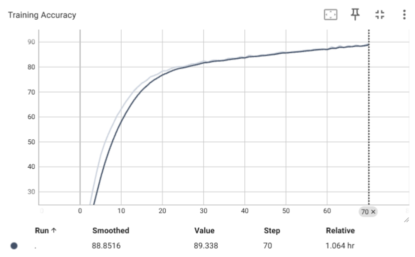
(a) Training config. 4



Run ↑	Smoothed	Value	Step	Relative
.	38.0256	38.05	66	1.101 hr

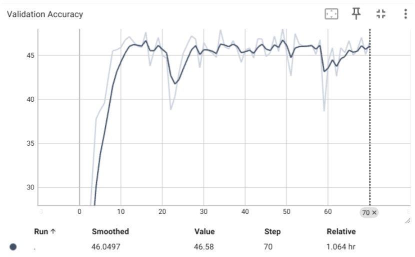
(b) Testing config. 4

Figure 4: Comparison between training and testing accuracy config. 4



Run ↑	Smoothed	Value	Step	Relative
.	88.8516	89.338	70	1.064 hr

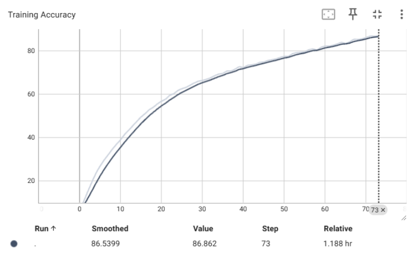
(a) Training config. 5



Run ↑	Smoothed	Value	Step	Relative
.	46.0497	46.58	70	1.064 hr

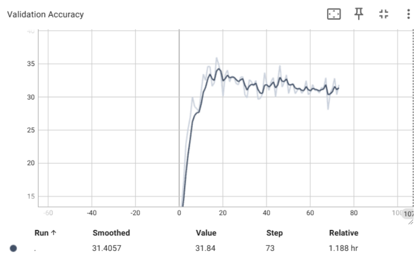
(b) Testing config. 5

Figure 5: Comparison between training and testing accuracy config. 5



Run ↑	Smoothed	Value	Step	Relative
.	86.5399	86.862	73	1.188 hr

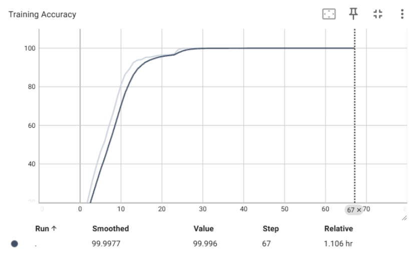
(a) Training config. 6



Run ↑	Smoothed	Value	Step	Relative
.	31.4057	31.84	73	1.188 hr

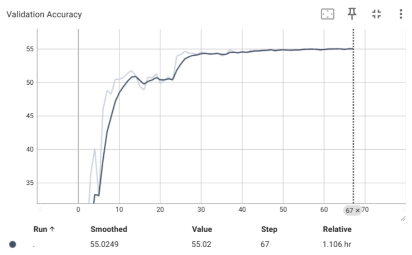
(b) Testing config. 6

Figure 6: Comparison between training and testing accuracy config. 6



Run ↑	Smoothed	Value	Step	Relative
.	99.9977	99.996	67	1.106 hr

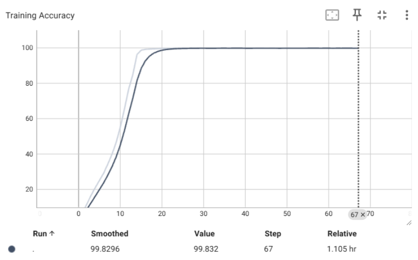
(a) Training config. 7



Run ↑	Smoothed	Value	Step	Relative
.	55.0249	55.02	67	1.106 hr

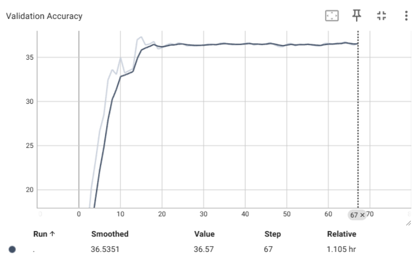
(b) Testing config. 7

Figure 7: Comparison between training and testing accuracy config. 7



Run ↑	Smoothed	Value	Step	Relative
● .	99.8296	99.832	67	1.105 hr

(a) Training config. 8



Run ↑	Smoothed	Value	Step	Relative
● .	36.5351	36.57	67	1.105 hr

(b) Testing config. 8

Figure 8: Comparison between training and testing accuracy config. 8