Deep learning classification using a Convolutional Neural Network (CNN)

Development and finetuning techniques

Saamie Vincken
FHNW Internship S5

Version history

Version	Date	Changes	Author
0.1	01.10.2024	Introduction and context	Saamie Vincken
0.2	01.10.2024	Neural networks and convolutional layers	Saamie Vincken
0.3	05.10.2024	Metrics	Saamie Vincken
0.4	11.10.2024	Finetuning initial setup	Saamie Vincken
0.5	12.10.2024	Finetuning additions	Saamie Vincken
0.6	14.10.2024	Loss function	Saamie Vincken
0.7	16.10.2024	Activation functions	Saamie Vincken
0.8	16.10.2024	Transfer learning	Saamie Vincken
0.9	18.10.2024	N-fold cross validation	Saamie Vincken
0.10	21.10.2024	Technical requirements	Saamie Vincken
1.0	21.10.2024	Final adjustments document	Saamie Vincken

Distribution

Version	Date	Distributed to	Status
1.0	21.10.2024	Canvas	None

Table of Contents

1 Introduction

This document contains research on the development of Convolutional Neural Networks for an image classification task. The following chapters will go into the architecture behind Convolutional Neural Networks, the training process and parameters, finetuning to improve the learning process and how to use existing models for a technique called transfer learning.

1.1 Context

During this research, the CIFAR-10 dataset is used as an example and will be analyzed and implemented in further documentation. The CIFAR-10 dataset provides a clear introduction into the use of a CNN on multi-class image related classification tasks, providing an understandable basis for the development of a CNN.

1.2 Tools

This classification is fully done in a Python project using PyTorch and related libraries. All logging is done using the platform Weights & Biases, offering easy visualization of the run logs and configuration. All used requirements are mentioned in chapter 10: Software requirements.

2 Neural Networks

A neural network is a machine learning model designed based on how the human brain processes information. It is made up of layers and connected neurons, which learn from data and make predictions.

A neural network has three main layers:

1. **Input layer:** Accepts input data (e.g. pixels in an image) for the model.
2. **Hidden layers:** Perform computations and transformations on the input data.
3. **Output (fully connected) layer:** Produces the final predictions.

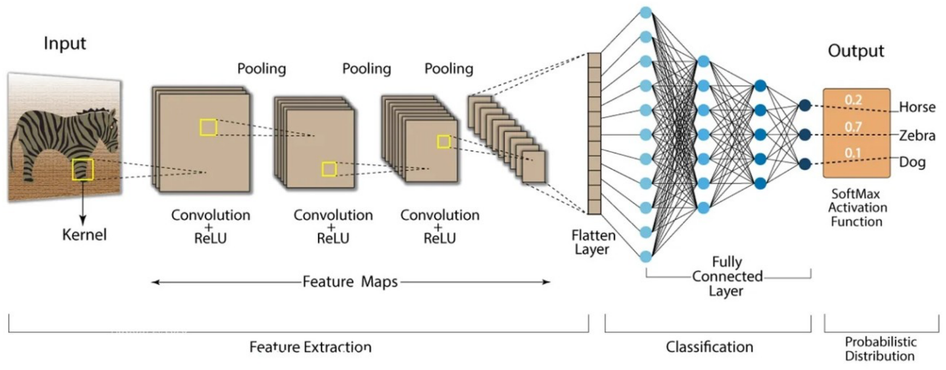
2.1 Parameters

In a neural network, parameters are the values learned by the model during training. The two main types of learnable parameters are **Weights** and **Biases**. Weights control the importance of the connections between neurons, if a feature is important, it has a larger weight than the less important features. Biases allow the model to adjust the predictions of the model, they use the true label in a training set to learn if the output of the layers should be slightly adjusted.

2.2 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (CNN) is a type of neural network specifically for image processing tasks like classification or object detection. The main difference is that a CNN uses additional convolutional layers to capture (image) features from the **input layer**, an activation function, and pooling layers to reduce dimensions and improve computation complexity. Figure 1[1] displays the basic structure of feature extraction in a convolutional neural network.

Figure 1: Convolutional neural network layers visualization



[1] (Ingoampt, sd)

2.2.1 Convolutional layers

The convolutional layers in a CNN apply kernels* to the input image. The kernels slide across the width and height of the input. The convolutions* produce feature maps with features like edges, textures and shapes. The size of the feature maps is smaller than the original image, depending on the stride* and padding. There are multiple levels of convolutional layers, where the low-level layers detect features like edges or textures, and the high-level layers detect more detailed objects or features related to the classes.

The output size of the convolutional layers is calculated using equation 1[2]. This output value is then used for the initialization of the fully connected layers.

Equation 1: Feature map size

$$\mathrm{O} = \frac{(\mathrm{T} - \mathrm{K} + (2P)}{\mathrm{S}} + 1$$

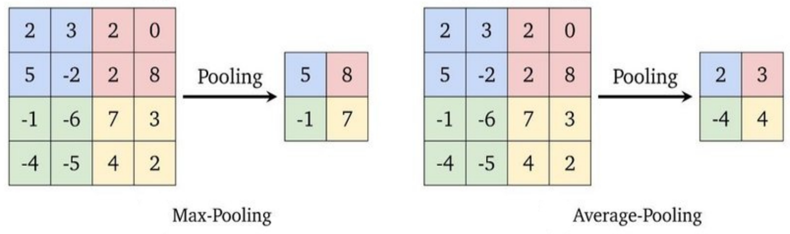
Where:

- **O:** Output size of the feature map
- **T:** Input volume size (height and/or width)
- **K:** Length of the convolution filter, also called kernel size
- **P:** Padding as the number of pixels added to each side of the input
- **S:** Stride as the number of pixels the filter moves at each step

2.2.2 Pooling layers

Pooling layers reduce the dimensions of the feature maps to improve the computation complexity. There are two types of commonly used pooling layers[3]: **Max Pooling** and **Average Pooling.** Max Pooling focuses more on the important features of each region in an image, compared to average pooling which smooths the feature map by taking an average of each region. Max Pooling is most often used for image classification because it captures the most important features. Figure 2[4] displays the difference between using a max-pooling layer and an average pooling layer.

Figure 2: Max pooling and Average pooling comparison



2 (Fleuret, The Little Book of Deep Learning)
3 (Zhao & Zhang, 2024)
4 (Guissous, 2019)
* *Kernel:* Filter to extract features from input data
* *Convolutions:* Operations of sliding a kernel across input data
* *Stride:* Step size per kernel

2.2.3 Fully connected layer parameter calculation example

Below a small example is provided using an image of the CIFAR-10 dataset.

For this example, a simple setup is used with the following layers and parameters:

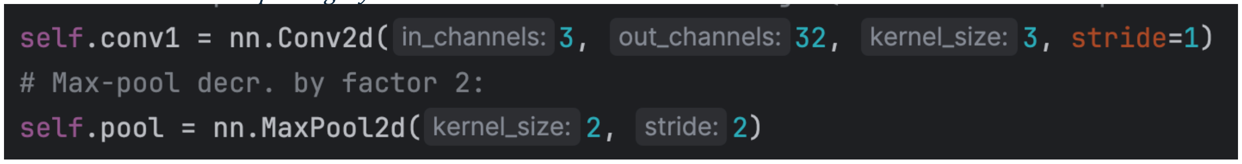
1. Convolutional layer
2. Pooling layer

Where:

- **Image size:** 32x32 pixels
- **Kernel size:** 3
- **Padding:** 0
- **Stride:** 1

This is implemented in code as follows:

Code 1: Convolution and pooling layer



```
self.conv1 = nn.Conv2d( in_channels: 3,  out_channels: 32,  kernel_size: 3, stride=1)
# Max-pool decr. by factor 2:
self.pool = nn.MaxPool2d( kernel_size: 2,  stride: 2)
```

To calculate the size of the output of these first layers, ***Error! Reference source not found.*** is used as follows:

$$Output\ size = (((32 - 3) + (2 * 0)) \div 1) + 1 = \mathbf{30}$$

Then during the max-pooling layer, the size of the feature map is decreased by factor 2:

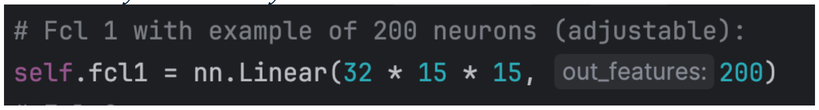
$$Output\ size = 30 \div 2 = \mathbf{15}$$

Then the fully connected layer would have the following parameters:

Input size: 32 (output of previous conv. Layer) * 15 * 15
Output size: Choice of neurons, adjustable for fine tuning of model

This would be implemented into code for a base model as follows:

Code 2: Fully connected layer



```
# Fcl 1 with example of 200 neurons (adjustable):
self.fcl1 = nn.Linear(32 * 15 * 15,  out_features: 200)
```

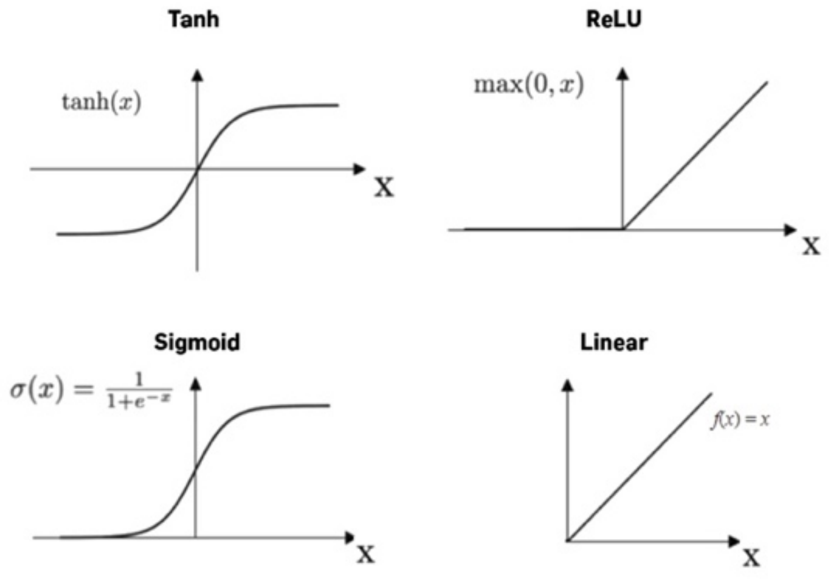
2.2.4 Activation functions

After each convolution, an activation function is applied to add non-linearity* to the network. Without activation functions, the entire neural network would only be linear operations and have too many average outputs.

Figure 3[5] displays various activation functions in comparison to a linear function. **ReLU** is used for this classification task, because it is designed for multi-class classification tasks. It outputs the input when the result is positive, and outputs zero when negative, making it fast and avoiding vanishing gradients*.

Sigmoid and **Tanh** are useful for binary classification but can face vanishing gradients*, which is not optimal for image classification.

Figure 3: Activation functions



[5] (Activation Function)
* *Gradients:* Measure of weight adjustment for learning
* *Vanishing gradient problem:* The calculated gradients are too small for effective learning
* *Non-linearity:* A relationship between inputs and outputs is not a straight line, allowing more complex patterns in data

3 Metrics

Because the goal of this assignment is to classify images in the CIFAR-10 dataset, multiple metrics will be used for evaluation.

3.1.1 Accuracy

The accuracy of the model is the most important metric for this assignment because it gives insight into how correct the prediction of the model are. Equation 2[6] displays the equation for getting an accuracy value between 0 and 1.

Equation 2: Accuracy evaluation

$$\text{Accuracy} = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

3.1.2 Precision

The precision metric measures the proportion of positive outputs that are actually correct (true positives). Equation 3[7] displays the equation for getting a precision value between 0 and 1.

Equation 3: Precision evaluation

$$\text{Precision} = \frac{True\ positive}{True\ positive + False\ positive}$$

3.1.3 Recall

The recall metric measures the proportion of true positives that were correctly identified. Equation 4[8] displays the equation for getting a recall value between 0 and 1.

Equation 4: Recall evaluation

$$\text{Recall} = \frac{True\ positive}{True\ positive + False\ negative}$$

3.1.4 F1-Score

The F1 score is an average of the recall and precision metrics. Equation 5[9] displays the equation for getting an F1 score between 0 and 1.

Equation 5: F1 evaluation

$$\text{F1 Score} = 2\ x\ \frac{recall\ x\ precision}{recall + precision}$$

[6] (Sanchinsoni, 2023)
[7] (Google developers, sd)
[8] (Google developers, sd)
[9] (Google developers, sd)

3.1.5 Loss function

In addition to performance metrics, the loss function is measured and used for training the model. A loss function calculates how far the predicted output is from the true label. This value is then used during training for adjusting the weights.

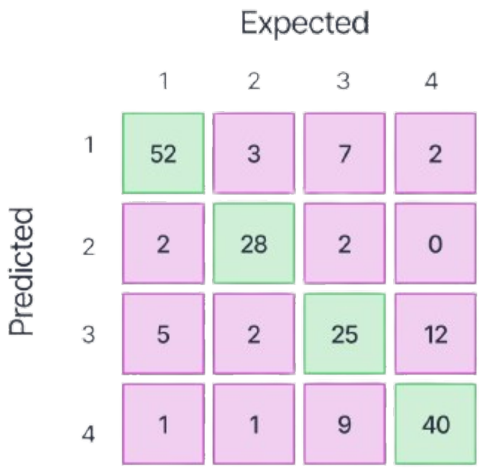
There are various ways to calculate the loss[10] depending on the use case, these can all be explained using different equation. However, for the classification of CIFAR-10 images, the cross-entropy loss function[11] is commonly used because it is designed for multi-class classification. This function is built into many tools for deep learning tasks, including in PyTorch[12], which is why further explanation of the equation is in this case not relevant.

3.1.6 Confusion matrix

A confusion matrix is used to evaluate the performance of a multi-class model, like CIFAR-10. It provides an overview on all correct and incorrect predictions by class. This offers an easy insight into which classes get mixed up or have a low performance.

Figure 4[13] displays an example of a confusion matrix where you can clearly see the distribution of results per class.

Figure 4: Confusion matrix example



Predicted \ Expected	1	2	3	4
1	52	3	7	2
2	2	28	2	0
3	5	2	25	12
4	1	1	9	40

3.1.7 Weights and Biases (W&B or Wandb)

Weights and Biases is a platform for visualizing metrics during training of a model. The tracking of metrics using this tool are done in real-time, making it easier to spot issues during the training process. From the project's codebase, the metrics can be initialized and logged to wandb. In the wandb dashboard these logs can then be used to create diagrams or matrices for all metrics.

[10] (Juan Terven, 2023)
[11] (Loss functions, 2017)
[12] (PyTorch, CrossEntropyLoss, sd)
[13] (Rohit Kundu, 2022)

4 Training

Training a CNN is divided up into multiple steps where the pre-processed images are used to do multiple iterations of training to teach the model how to recognize patterns in the data. A basic training loop[14] [15] is built up out of the following components:

1. **Model initialization**: Defining the architecture of the CNN and what layers should be used for training.
2. **Loading the dataset**: Loading the dataset as a test and validation set.
3. **Model to device:** Adding the model and its parameters to the correct device[16] (GPU, CPU or MPS).
4. **Loss function initialization**: Defining a loss function to measure the difference between the predicted and actual labels.
5. **Metrics initialization:** Defining and resetting metrics for each iteration.
6. **Forward pass:** Input data is processed and passed to the next layer.
7. **Backward pass (Backpropagation):** The results of the loss are used to go back through the network and update the parameters.
8. **Epochs and iterations:** Repeating the forward and backward passes for multiple epochs (multiple loops over the whole dataset, 1 epoch = 1 whole dataset).
9. **Validation set:** Using a separate subset of the data (validation set) is used to assess the performance on new data after training.
10. **Logging metrics:** After each epoch or batch metrics are logged for visualization.

5 Finetuning

Finetuning the model is essential to reach much higher results in accuracy and reduce overfitting while training the model.

5.1 Learning rate

The learning rate[17] determines the step size during the updating of parameters. It controls how much the model's weights are adjusted. A higher learning rate ensures faster learning but may cause the model to miss important parameter updates and reduce the overall accuracy. A lower learning rate ensures higher precision but may slow down training.

5.2 Momentum

Momentum[18] helps to speed up training and smooth the overall learning iteration by using previous steps (direction of the gradients) for the next step. A low momentum is more sensitive to small changes in the learning process making updates slower. A high momentum ensures faster learning by using previous changes but has a risk of skipping important changes.

[14] (TensorFlow, sd)
[15] (Rosebrock, 2021)
[16] (PyTorch, 2024)
[17] (Bhattbhatt, 2024)
[18] (Bhattbhatt, 2024)

5.3 Optimization algorithms

An optimizer is one of the most important functions to use while finetuning a training process, it has a large influence on the performance of the model. The optimizer determines how the model's parameters are updated during training, which is called a gradient descent algorithm[19]. Two commonly used optimizer functions for a classification task are the **SGD** and **Adam** optimizers[20].

5.3.1 SGD optimizer

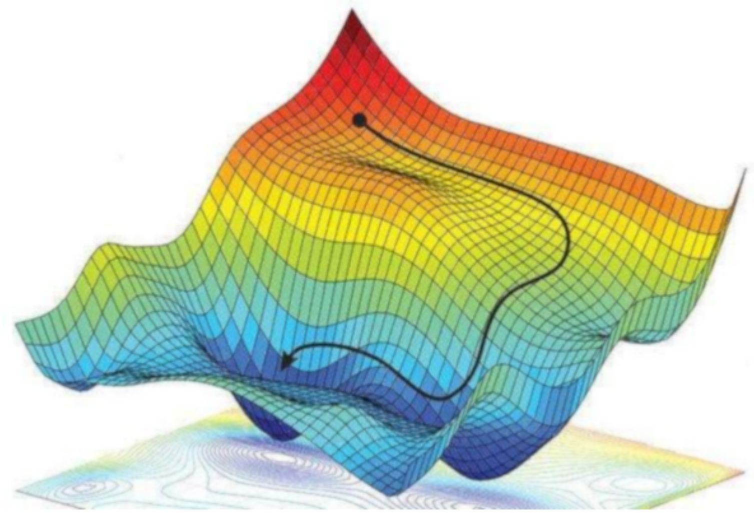
The Stochastic Gradient Descent (SGD)[21] is a basic optimizer that updates model parameters based on a batch of training data, ensuring faster training compared to updates on the entire dataset. In the SGD optimizer the learning rate and optionally the momentum are fixed values and remail consistent during the training process.

5.3.2 Adam optimizer

The Adaptive Momentum Estimation (Adam)[22] is a more advanced optimizer algorithm. It ensures a more effective use of the learning rate and momentum because it dynamically adjust these values during training, and does not require manual tuning as much as SGD.

Figure 5[23] provides a visualization of gradient descend, where the model's parameters are adjusted based on the loss function. The surface represents the loss function landscape, and the black line represents the path taken by the optimizer to minimize the loss.

Figure 5: Gradient descent in loss function landscape



[19] (Kwiatkowski, 2021)
[20] (Park, 2021)
[21] (PyTorch, SGD, sd)
[22] (PyTorch, Adam, sd)
[23] (Mishra, 2023)

5.4 Learning rate scheduler

A learning rate scheduler is a tool that is used to adjust the learning rate during training according to a predefined schedule[24]. The following are commonly used learning rate schedulers:

- **Step decay:** Reduces the learning rate by a factor per specified epoch. [25]
- **Exponential decay**: Continuously decreases the learning rate during training. [26]
- **Reduce on plateau**: Reduces the learning rate when the validation loss stops improving. [27]

5.5 Regularization

Regularization is a technique to prevent overfitting by preventing the model to become too focused on specific details in the training data[28]. It discourages the model from assigning too much importance to features or weights by setting a constraint.

5.5.1 L1 regularization (Lasso)

L1 regularization[29] adds a constraint to the loss function based on the absolute values of the weights. This means that some weights are set to zero, making the model ignore some features completely and only use the important ones.

5.5.2 L2 regularization (Ridge)

L2 regularization[30] adds a constraint to the loss function based on the squared value of weights. This means that the model 'prefers' smaller weights overall instead of allowing a single weight to become too large.

5.6 Weights initialization

The initialization of weights can ensure better overall performance as the weights are initialized before the training begins. There are multiple options for initialization of the weights including a default option offered by PyTorch.

Common activation functions are **Kaiming (he)** and **Xavier** initializations[31]. However, for this specific classification task the only relevant algorithm is the Kaiming initialization because it is designed to work with the ReLu activation function which was explained in chapter 2.2.4: Activation functions.

[24] (Lau, 2017)
[25] (PyTorch, StepLR, sd)
[26] (PyTorch, ExponentialLR, sd)
[27] (PyTorch, ReduceLROnPlateau, sd)
[28] (Nagpal, 2017)
[29] (Fordjour, 2024)
[30] (Fordjour, 2024)
[31] (Ouannes, 2019)

5.6.1 Kaiming uniform initialization (He)

The Kaiming uniform initialization (also called the ‘He’ initialization) is often used in combination with ReLu-based activation functions, like how it’s done for this specific classification task. When using this algorithm, the weights are randomly initialized from a range of values between -k to +k. The value of k depends on the number of neurons and is explained in equation 6[32].

Equation 6: Kaiming uniform initialization

$$k = \frac{1}{\sqrt{fan\ in}}$$

Where:

- **K** is the range for the weights initialization.
- **Fan in** is the number of input units (neurons).
- The **square root** is used to prevent values from becoming too large or small.

By default, PyTorch initializes the weights using this method, because it is suitable for many types of models.

5.7 Batch size

The batch size is the number of samples processed before the model’s parameters are updated. A smaller batch size (e.g. batch size 16) may improve generalization but can increase training time. A larger batch size (e.g. batch size 128) can speed up training (depending on the available hardware) but requires more memory and could reduce generalization.[33]

5.8 Number of epochs

The number of epochs is the number of times the training process goes over the entire dataset[34]. To get good results with a model it is necessary to go over the dataset multiple times so the model can learn from the data. With too few epochs the model might not learn enough from the data (underfitting), while with too many epochs the model may start to memorize the training data while not getting the same results on validation (overfitting).

5.9 Early stopping

Early stopping can be used to stop the training process when the models’ performance on the validation set stops improving. This prevents overfitting when there are too many epochs, and the model starts to memorize the training data. [35]

[32] (PyTorch, torch.nn.init, sd)
[33] (Devansh, 2022)
[34] (Rsvmukhesh, 2023)
[35] (paperswithcode.com, sd)

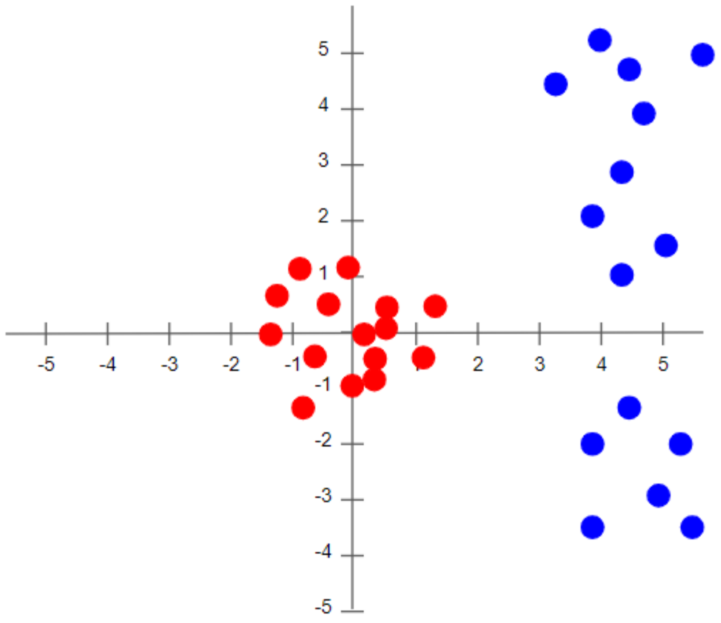
5.10 Dropout

Dropout is a technique where a fraction of the neurons is randomly set to zero during training.[36] A neuron can become dependent on specific patterns from other neurons, and adding dropout can make sure that the neurons do not rely on each other too much. It helps finds a balance between underfitting and overfitting. A low dropout (e.g. dropout of 0.1 where 10% of neurons are dropped) reduces overfitting slightly, while a high dropout rate (e.g. dropout of 0.5 where 50% of neurons are dropped) reduces overfitting significantly but there is a risk of underfitting if too many neurons are dropped.[37]

5.11 Batch normalization

Batch normalization is used to normalize the input of each layer to have a mean of zero and a standard deviation of one.[38] This helps the model learn faster by keeping the data on a more even scale. Batchnorm is usually added after convolutional layers and before activation functions. In figure 6[39], a result of batch normalization is visualized, where the blue dots represent the input before normalization, and the red dots represent the data after normalization.

Figure 6: Batch normalization visualization



[36] (PyTorch, Dropout, sd)
[37] (Brownlee, 2019)
[38] (Sadananda, 2021)
[39] (Doshi, 2021)

6 N fold cross validation

N-fold (also called K-fold) cross validation is a method to look at the generalization of a model[40]. The dataset is divided into N folds, which are equally sized subsets of the data. The model is then trained on each different fold except for one ($N - 1$), which is used for validation. This process is repeated N times, so that each fold is used as the test set at least once. The result of all folds can be viewed as the model's average performance. This helps to ensure that the model's performance is not bias on a specific training set. [41]

7 Transfer learning

Transfer learning is a technique where an existing pre-trained model is used and finetuned for a different but related task and dataset. Pre-trained models have already learned many generic features which can be applied to the new task. A pre-trained model architecture is loaded and then the final fully connected layer is adjusted so it matches the number of classes for the specific task (e.g. a pretrained ImageNet model has 1000 classes while CIFAR-10 only has 10 classes).[42]

7.1 Finetuning in transfer learning

When finetuning a pre-trained model, only certain layers of the model are updated during training. Often the low-level layers (e.g. features like edges and textures) remain the same while the high-level layers (e.g. class specific features like faces or object parts) are adjusted based on the new dataset. During finetuning it is common to use a lower learning rate because the pre-trained weights should not be adjusted too much.

7.2 Freezing layers

The layers in the transfer model can be frozen or finetuned[43]. Freezing layers means preventing their weights from being updated during training. The low-level layers often capture features such as edges, which do not have to be trained all over again and can be frozen so only the more specific features in the high-level layers are used for training on the specific dataset.

7.2.1 Gradual unfreezing

Another approach to freezing layers is to gradually unfreeze layers, meaning that it starts with only the final layers being unfrozen, and overtime during the training process more layers are unfrozen. It helps with finding a balance in learning from new data while also preventing changing too many of the pre-trained weights.

[40] (3.1. Cross-validation: evaluating estimator performance, sd)
[41] (Ayush Chaurasia & Wandb.ai, 2024)
[42] (TensorFlow, Transfer learning and finetuning, sd)
[43] (Fakhar, 2023)

8 Software requirements

PyTorch and related libraries:

- *torch==2.4.1*: Core PyTorch library.
- *torchvision==0.19.0:* Contains datasets, model architectures, and common image transformations for PyTorch.
- *torchmetrics==1.4.1:* For evaluation metrics.

WandB (Weights & Biases):

- *wandb==0.17.9:* For experiment tracking and visualization.

Data Handling and processing:

- *numpy==1.26.4:* For numerical operations and array manipulations.
- *matplotlib==3.9.2:* For plotting and visualization.
- *scikit-learn==1.5.2*: K-Fold cross-validation.

9 Table of figures

10 Bibliography

3.1. Cross-validation: evaluating estimator performance. (n.d.). Retrieved from scikit-learn.org: https://scikit-learn.org/stable/modules/cross_validation.html

Activation Function. (n.d.). Retrieved from https://machine-learning.paperspace.com/wiki/activation-function

Ayush Chaurasia, & Wandb.ai. (2024, July 15). *Using K-Fold Cross-Validation To Improve Your Machine Learning Models*. Retrieved from wandb.ai: https://wandb.ai/wandb_fc/kaggle_tutorials/reports/Using-K-Fold-Cross-Validation-To-Improve-Your-Machine-Learning-Models--VmlldzoyMTY0MjM2

Bhattbhatt, V. (2024, January 24). *Learning Rate and Its Strategies in Neural Network Training*. Retrieved from medium.com: https://medium.com/thedeephub/learning-rate-and-its-strategies-in-neural-network-training-270a91ea0e5c

Brownlee, J. (2019, August 6). *A gentle introduction to dropout for regularizing deep neural networkds*. Retrieved from machinelearningmastery.com: https://www.machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Devansh. (2022, January 17). *How does Batch Size impact your model learning*. Retrieved from medium.com: https://medium.com/geekculture/how-does-batch-size-impact-your-model-learning-2dd34d9fb1fa

Doshi, K. (2021, May 18). *Batch Norm Explained Visually — How it works, and why neural networks need it*. Retrieved from towardsdatascience.com: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

Fakhar, M. (2023, July 14). *Unfreezing and transfer learning in deep learning*. Retrieved from medium.com: https://medium.com/@fakhar3534/unfreezing-and-transfer-learning-in-deep-learning-a31ef2ad9e8c

Fleuret, F. (n.d.). *The Little Book of Deep Learning.* Retrieved from fleuret.org: https://fleuret.org/public/lbdl.pdf

Fleuret, F., & Genève, U. d. (n.d.). *Deep learning 1.4 Tensor basics and linear regression.* Retrieved from https://fleuret.org/dlc/: https://fleuret.org/dlc/materials/dlc-handout-1-4-tensors-and-linear-regression.pdf

Fleuret, F., & Université de Genève. (n.d.). *Deep learning 1.5 High dimention tensors.* Retrieved from fleuret.org: https://fleuret.org/dlc/materials/dlc-slides-1-5-high-dimension-tensors.pdf

Fordjour, H. A. (2024, September 13). *PyTorch loss functions*. Retrieved from digitialocean.com: https://www.digitalocean.com/community/tutorials/pytorch-loss-functions

Google developers. (n.d.). *Classification: Accuracy, recall, precision, and related metrics*. Retrieved from developers.google.com: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall

Guissous, A. E. (2019, November). *Skin Lesion Classification Using Deep Neural Network.* Retrieved from researchgate.net: https://www.researchgate.net/figure/Example-for-the-max-pooling-and-the-average-pooling-with-a-filter-size-of-22-and-a_fig15_337336341

Helmholtz. (2023). *Introduction to grad-CAM*. Retrieved from xai tutorials: https://xai-tutorials.readthedocs.io/en/latest/_model_specific_xai/Grad-CAM.html

Ingoampt. (n.d.). *CNN- Convolutional Neural Networks - Day 53*. Retrieved from ingoampt.com: https://ingoampt.com/cnn-convolutional-neural-networks-day-53/

Juan Terven, D. M.-E.-P.-U.-G. (2023, July 5). *Loss functions and metrics in deep learning.* Retrieved from https://arxiv.org/pdf/2307.02694

Kwiatkowski, R. (2021, May 22). *Gradient Descent Algorithm — a deep dive*. Retrieved from towardsdatascience.com: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21

Lau, S. (2017, July 29). *Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning*. Retrieved from towardsdatascience.com: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1

Loss functions. (2017). Retrieved from ml-cheatsheets.io: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

Mishra, M. (2023, June 5). *Stochastic Gradient Descent: A Basic Explanation*. Retrieved from https://mohitmishra786687.medium.com/stochastic-gradient-descent-a-basic-explanation-cbddc63f08e0

Nagpal, A. (2017, October 13). *L1 and L2 Regularization Methods*. Retrieved from towardsdatascience.com: https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

Ouannes, P. (2019, March 22). *How to initialize deep neural networks? Xavier and Kaiming initialization*. Retrieved from pouannes.github.io: https://pouannes.github.io/blog/initialization/

paperswithcode.com. (n.d.). *Early stopping*. Retrieved from paperswithcode.com: https://paperswithcode.com/method/early-stopping

Park, S. (2021, June 21). *A 2021 Guide to improving CNNs-Optimizers: Adam vs SGD*. Retrieved from medium.com: https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008

PyTorch. (2024). *Saving and loading models across devices in PyTorch*. Retrieved from pytorch.org: https://pytorch.org/tutorials/recipes/recipes/save_load_across_devices.html

PyTorch. (n.d.). *Adam*. Retrieved from pytorch.org: https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

PyTorch. (n.d.). *CrossEntropyLoss*. Retrieved from https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

PyTorch. (n.d.). *Dropout*. Retrieved from pytorch.org: https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html

PyTorch. (n.d.). *ExponentialLR*. Retrieved from pytorch.org: https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ExponentialLR.html

PyTorch. (n.d.). *ReduceLROnPlateau*. Retrieved from pytorch.org: https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html

PyTorch. (n.d.). *SGD*. Retrieved from pytroch.org: https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

PyTorch. (n.d.). *StepLR*. Retrieved from pytorch.org: https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.StepLR.html

PyTorch. (n.d.). *torch.nn.init*. Retrieved from pytorch.org: https://pytorch.org/docs/stable/nn.init.html#torch.nn.init.kaiming_uniform_

Rohit Kundu. (2022, September 13). *Confusion Matrix: How To Use It & Interpret Results*. Retrieved from v7labs: https://www.v7labs.com/blog/confusion-matrix-guide

Rosebrock, A. (2021, July 19). *PyTorch: Training your first Convolutional Neural Network (CNN)*. Retrieved from pyimagesearch.com: https://pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-network-cnn/

Rsvmukhesh. (2023, May 19). *Determining the Number of Epochs*. Retrieved from medium.com: https://medium.com/@rsvmukhesh/determining-the-number-of-epochs-d8b3526d8d06

Sadananda, N. (2021, November 3). *Convolutional Neural Network: Data Augmentation and Batch Normalization to improve CIFAR-10 images results.* Retrieved from medium.com: https://medium.com/@nischitasadananda/convolutional-neural-network-data-augmentation-and-batch-normalization-fd9d6237e9e

Sanchinsoni. (2023, June 12). *Different Metrices in Machine Learning for Measuring performance of Classification Algorithms*. Retrieved from medium.com: https://medium.com/@sachinsoni600517/different-metrices-in-machine-learning-for-measuring-performance-of-classification-algorithms-509e55c0a451

TensorFlow. (n.d.). *Convolutional Neural Network (CNN)*. Retrieved from tensorflow.org: https://www.tensorflow.org/tutorials/images/cnn

TensorFlow. (n.d.). *Transfer learning and finetuning*. Retrieved from tensorflow.org: https://www.tensorflow.org/tutorials/images/transfer_learning

Zhao, L., & Zhang, Z. (2024, January 18). *A improved pooling method for convolutional neural networks*. Retrieved from nature.com: https://www.nature.com/articles/s41598-024-51258-6