Deep Learning for Visual Computing

Assignment 1

Authors:
Tibor Čuš (12325298),
Şaban Akay (12045645)
Vienna, May 8, 2024

1 Introduction

The purpose of this assignment is to familiarize ourselves with the basics of deep learning in PyTorch. For this, we have implemented multiple Convolutional Neural Network (CNN) and Vision Transformer (ViT) models for a multi-class image classification problem based on the CIFAR-10 dataset [2].

In addition to implementing the models, we will also explore data preprocessing, model training, evaluation, and fine-tuning. By exploring these aspects, we intend to develop a comprehensive understanding of the end-to-end process involved in deep with PyTorch.

2 Dataset

The CIFAR-10 dataset[2] consists of 60,000 32x32 color images divided into 10 classes, with 6,000 images per class. We split the data into three datasets. The training dataset contains 40,000 images, while both the test and validation datasets contain 10,000 images each. We use the training set to train the model, the validation set to fine-tune hyperparameters and prevent overfitting during training, and the test set to evaluate the model's performance on unseen data, ensuring its generalization ability to real-world scenarios. Each set serves a distinct purpose in ensuring the model's effectiveness, robustness, and reliability.

3 Experiment setup

In the following sections, we will compare different CNN and ViT models against each other and two baseline models. For the baseline models, we used a majority classifier model and a pre-trained ResNet18 [1], which was additionally fit to our data. We will run the following experiments. The models will be evaluated based on Accuracy (ACC) and per-class Accuracy (PACC).

- shallow CNN vs deep CNN
- deep CNN vs deep Normalized CNN
- shallow ViT vs deep ViT
- deep ViT vs deep ViT with residual connections
- ResNet18 with data augmentation vs no data augmentation

To make the comparison of models easier we have fixed the following variables during training.

- optimizer: Adam
- batch size: 64
- learning rate: 0.001
- maximum number of epochs: 10
- training data augmentation: randomly crop or mirror input image
- loss function: categorical cross-entropy

Arguably, the most important experiment variable is the loss function we use for training our models. The loss function defines what we want our model to learn and measures how well the model's predictions align with the data. In our case, we are using the cross-entropy loss function, for which our model needs to return probabilities for each possible data class in our dataset. Cross-entropy measures how much probability we assigned to each correct label and tries to guide our model to assign as much probability as possible to the correct one. Minimizing the cross-entropy loss function corresponds to finding the maximum likelihood estimator of our model based on the provided data.

To discourage our models from overfitting, we decided to augment the training images (except for the ResNet18 experiment) by randomly mirroring or cropping them. This works as a kind of model regularization as it adds noise to the training data. There are other forms of regularization such as dropout, which randomly drops some neurons during training, preventing them from co-adapting too much. This can also be seen as adding noise to the network [3]. Another type of regularization is adding a penalty term to our loss function. Most commonly, we penalized the L2 norm of our model's weights, which prevents them from getting extreme values, or the L1 norm which encourages sparseness. Finally, we can also prevent our model from training for too long by implementing early stopping. A simple version of the algorithm checks the loss function at sequential time steps, and if the loss didn't change enough, we stop training.

4 Models

In our assignment, we were tasked with implementing two different kinds of models: CNN models and ViT models. In the following sections, we will discuss the main differences between the architectures and their underlying concepts. Since the code for all used models is provided, we will skip the exact architecture definitions in this report.

4.1 Convolutional Neural Networks

The basic idea of convolutional neural networks is to define filters (weights) and slide (convolve) them along images and use them to extract certain features from a part of an image. Normally, we stack multiple convolutional layers on top of each other and add non-linearities in between. Stacking two convolutional layers is the same as using a bigger filter, but it's normally more optimal as it requires fewer trainable parameters, which is why it is more common to only use filter sizes of 3x3 or 5x5 and not greater. Having more smaller layers also allows us to include more non-linearities in our model.

As our models get deeper, the gradients start to vanish or explode. There are multiple solutions such as batch normalization, which normalizes the input of a layer, reducing the occurrence of extreme values. Batch normalization is really useful in front of activation functions such as ReLU, which can have a zero gradient at its extreme values. It is also common to include pooling layers between convolutional layers, which downsample filters, making them less sensitive to the position of the filter and more robust to changes. A common implementation of a CNN block can be found in our cnn.py file.

```
def normalized_convolution_block(fan_in: int, fa_out: int, kernel: int, padding, stride: int):
    block = nn.Sequential(
        nn.Conv2d(fan_in, fa_out, kernel, stride, padding, bias=False),
        nn.BatchNorm2d(fa_out),
        nn.ReLU(),
        nn.MaxPool2d(2, 2)
    )
    return block
```

This function creates a convolutional filter which results are normalized and transformed using the Relu function and finally downsampled using MaxPool2D which select the maximum value in a 2x2 square.

4.2 Vision Transformers

Vision transformers work using the attention mechanism[4]. They were initially used for text generation and translation in NLP but were also adapted into the computer vision domain. ViT transformers start by patching the input image into smaller segments (PatchEmbedding class in vit.py), which are then flattened. They represent the embedding of a image patch. To embedings they further add positional information (function get_sinusoid_encoding in vit.py), which provide the model with context about the position of each input pixel and each patch. The embedings can be fixed or learnable. We have decided to use fixed embedings as it creates smaller models and it should not drastically affect the models performance [5].

The embeddings are passed into an attention layer which allows the model to look at all the other embeddings in the sequence and learn relations between them. Each attention layer normally has three learnable parameters, which are linearly transformed with the input embeddings. When we talk about multihead attention we refer to repeating the attention mechanism multiple times. Models of this type tend to get really deep as that allows the model to learn more complex relations, but it can also cause vanishing or exploding gradients. That is why the transformer architecture uses residual connections between attention layers which allow gradients to skip complex computations that may cause them to vanish.

4.3 CNN & ViT comparison

There are some similarities between the two model types. CNN models learn different filters which let them extract information from image segments. This is, in a way, similar to vision transformers' projections learning sets of abstract patterns that represent each patch. Both models also break the image into smaller pieces and try to extract information from that piece. Other than that, they have many differences. For example, the receptive field of CNN filters is small at the start and gets deeper with each added layer. ViT's receptive field already covers the whole image in the first attention layer. This desirable, but it can cause overfitting on small datasets. Transformers also allow you to visualize the attention of the heads, making the models more explainable than CNNs.

5 Result comparison

Models were trained using the Adam optimizer (learning rate of 0.001, weight decay of 0.0001, dropout rate of 0.5) for 10 epochs with a batch size of 64. Loss function was CrossEntropyLoss. Data augmentation (random cropping, left-right mirroring) was applied to the training set. Test and validation sets were normalized with mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225].

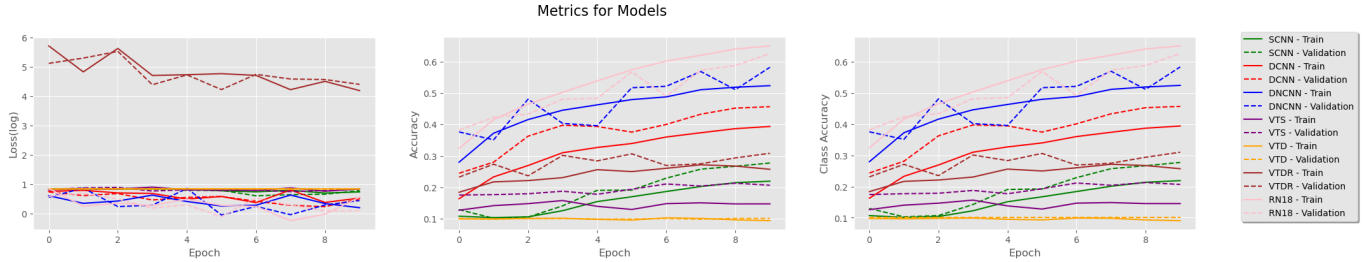

Figure 1: Comparison of model training behaviour

Model	val. ACC	val PACC	test ACC	test PACC	train time(s)	test time(s)
Simple CNN	0.1886	0.1895	0.2730	0.2730	53.46	1.18
Deep CNN	0.3798	0.3797	0.4625	0.4625	108.25	2.62
Deep Normalized CNN	0.4712	0.4707	0.5814	0.5814	113.07	2.80
Shallow ViT	0.1922	0.1922	0.2170	0.2170	89.76	2.38
Deep ViT	0.0994	0.1000	0.1000	0.1000	138.06	3.10
Deep Residual ViT	0.2780	0.2775	0.3106	0.3106	136.58	3.11
ResNet18	0.5051	0.5053	0.6305	0.6305	1643.49	61.08
Majority Model	0.1	0.1	0.1	0.1	-	-

Table 1: Model benchmarks

ResNet18 model outperforms all other models in terms of accuracy and per-class accuracy. Having more layers allows it to capture more information and complex features, but simultaneously slows down inference, which can be a limiting factor in resource-restricted applications.

It is also noteworthy to observe the decrease in performance of the shallow ViT when another layer is added. This is likely due to vanishing gradients, as performance significantly improves when residual connections are added into the deep model. Additionally, we can observe the improvement in CNN performance with the addition of more layers to the shallow model and batch normalization to the deep model. Overall, the results are not too surprising. The only unconventional finding we noticed is that all models perform better on the test set than on the validation set, suggesting that the test set may contain easier examples.

References

[1] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[2] Alex Krizhevsky, Geoffrey Hinton, and Vinod Nair. URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[3] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[4] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[5] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].