Homework Assignment: Convolutional Neural Networks

Santiago Andrés Serrano Vacca[1*]

[1*]Department of Computer Science and Engineering, Tecnológico de Monterrey, Monterrey, NLE, México.

Abstract

Convolutional Neural Networks (CNNs) are commonly used for image classification tasks, due to their ability to automatically learn hierarchical features from raw pixel data [1]. This work explores the impact of different changes to a basic CNN (such as changing architecture, hyperparameter tuning, batch normalisation and data augmentation). Using the CIFAR-10 dataset for testing such modifications, a final accuracy of 0.7244 on the test set was achieved, in comparison to the 0.6342 accuracy of the base model.

Keywords: Convolutional Neural Network, Deep Learning, Artificial Intelligence, Image Classification

1 Introduction

Image classification is a fundamental problem in computer vision, with applications ranging from object recognition to scene understanding [2]. Convolutional Neural Networks (CNNs) have emerged as the dominant approach for this task, demonstrating state-of-the-art performance on benchmark datasets such as CIFAR-10 [3]. However, designing and training effective CNN architectures remains a challenge, requiring careful consideration of factors such as network depth, hyperparameters, and regularization techniques [4].

This study aims to investigate the impact of various architectural and training modifications on the performance of a basic CNN for image classification on the

[0]The code is freely available at https://github.com/santiago-a-serrano/cnn-homework.

CIFAR-10 dataset. By exploring techniques such as changing network depth, hyper-parameter tuning, batch normalization [5], and data augmentation [6], we seek to identify effective strategies for improving classification accuracy.

The remainder of this paper is organized as follows: Section 2 provides a description of the methodology used to achieve incremental advancement on the performance of the neural network. Section 3 describes the experiments performed. Section 4 presents the experimental results and discussion, comparing the performance of the modified models to the baseline. Finally, Section 5 presents a conclusion of this work and its findings.

2 Methodology

As a starting point for our investigation, we designed a simple baseline Convolutional Neural Network (CNN) architecture for image classification on the CIFAR-10 dataset. The baseline model consists of the following layers:

- Convolutional layer with 32 filters, 3×3 kernel size, and Rectified Linear Unit (ReLU) activation.
- Max Pooling layer with a 2×2 pool size to downsample the feature maps.
- Flatten layer to convert the 2D feature maps into a 1D feature vector.
- Dense layer with 128 units and ReLU activation to introduce non-linearity and learn high-level representations.
- Dense output layer with softmax activation to produce class probabilities for the 10 classes in CIFAR-10.

This simple architecture serves as a foundation for exploring various subsequent modifications and improvements (which will be specified in Section 4). The baseline model and its subsequent modifications were implemented using PyTorch [7], in a Google Colab [8] environment.

3 Experiments

3.1 Dataset

All experiments were conducted using the CIFAR-10 dataset, which consists of 60,000 32x32 color images in 10 classes, with 6000 images per class [3]. The dataset was split into 50,000 training (or training and validation, depending on the experiment), and 10,000 test images. For the experiments with a validation set, 10,000 images were used, leaving 40,000 training images.

3.2 Training Setup

The models were trained using the Adam optimizer [9] with various learning rates and batch sizes. Training was performed for 100 or 200 epochs, depending on the specific experiment. The Cross-Entropy Loss function was used as the criterion for all models [10]. A Google Colab environment, with the following specifications, was used:

- PyTorch version: 2.2.1+cu121

- Python version: 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0]
- CUDA version: 12.1
- cuDNN version: 8906
- GPU model: Tesla T4 (NVIDIA)

3.3 Baseline Model Modifications

Modifications to the baseline model can be summarized in 3 different types:

1. Architecture Changes
2. Hyperparameter Tuning
3. Advanced Techniques

For the Architecture Changes (as well as the baseline model), the data was split into only training and test sets. For the Hyperparameter Tuning and Advanced Techniques, a validation set was used in order to tune hyperparameters. Nonetheless, the final comparison will be performed by calculating the accuracy of the models in the test set.

It is also worth mentioning that a unique seed was implemented to guarantee the same random splits of data into batches (in the case of models with the same batch size and modification type), so that each of the different variations of the base model (including the base model) could be compared with fairness.

4 Results and Discussion

4.1 Baseline Model

When trained for 100 epochs and a learning rate of 0.001, with batches of size 128, an accuracy of **0.6342** was achieved on the test set.

4.2 Architecture Changes

ExtraConvModel: The first modification implemented consists on an extra convolutional layer placed after the one from the original model. This new layer receives the 32 feature maps from the previous one, applies a 2x2 kernel, and contains 32 filters too. The ReLU activation from the first layer is then moved to the second layer, so that the activation is done after the second layer has done its job with the direct output (with no activation function) of the first one. After this only modification to the baseline model, an accuracy of **0.6476** was achieved on the test set, a slight improvement.

ExtraDoubleConvModel: The second modification consists on adding two extra convolutional layers (instead of one) to the original model, both after the original convolutional layer. The first one applies a 4x4 kernel and contains 64 filters. The second one goes immediately after the first, and applies a 2x2 kernel with 32 filters. The only ReLU among the convolutional layers goes after the last one. With this modification, an accuracy of **0.5982** was achieved on the test set, a noticeable downgrade even compared to the baseline model.

DropoutModel: This last modification stems from the first one (*ExtraConvModel*), as it achieved better performance than *ExtraDoubleConvModel*. It is the same as *ExtraConvModel*, with the addition of a dropout layer between the two convolutional layers, with $p = 0.5$. An accuracy of **0.6217** was achieved on the test set, making it slightly worse than the baseline model.

4.3 Hyperparameter Tuning

As *ExtraConvModel* was the best model from the previous phase, it will be used in conjunction with the following modifications:

Small Batch Model: A batch size of 64 was used. An accuracy of **0.6196** was achieved on the validation set.

Big Batch Model: A batch size of 256 was used instead of the 128 from the baseline. An accuracy of **0.6344** was achieved on the validation set, which is a very slight improvement compared to the baseline model (with no extra convolution). However, although this score is still lower than *ExtraConvModel* with no hyperparameter modifications, it will be used in further experiments, as it might lead to smoother gradient estimates, which could make a good combination with slightly bigger learning rates, as proposed by *Goyal et al.* [11], who propose that linearly scaling the learning rate proportional to the scaling of the batch is "surprisingly effective". All next experiments will have a batch size of 256.

Big Learning Rate: A learning rate of 0.005 was used instead of the 0.001 from the base model. This model was extremely low-performing, with an accuracy of **0.0971** on the validation set.

Slightly Big Learning Rate: For a learning rate of 0.002, a validation accuracy of **0.6396** was obtained, which shows an improvement of the big batch model. Still not as good as *ExtraConvModel* alone, but this combination of big learning rate and big batch size may prove beneficial as batch normalization (which will be implemented in the final model) benefits from larger batch sizes [12].

Double Epochs: With 200 epochs, identical models but with different learning rates (lr) were tested. The accuracy on the validation sets were the following:

lr=0.0005	lr=0.0008	lr=0.0015
0.6219	0.6290	0.6327

It can be seen that a higher learning rate seems to give better results.

4.4 Advanced Techniques

In order to make a more fair comparison with the baseline model, only 100 epochs were used to train the models that follow. Both stem from *ExtraConvModel* with a learning rate of 0.002.

Batch Normalization: Batch normalization was applied, specifically in the form of two batch normalization layers, one after each convolutional layer. A validation accuracy of **0.6839** was achieved. The advantages of batch normalization are easily notable, as this is the best-performing model yet.

Batch Normalization: Batch normalization was applied, specifically in the form of three batch normalization layers, one after each of the two convolutional layers, and the third one after the dense layer. It should also be noted that after each normalization layer a ReLU layer followed, making the model end up with three ReLU activation functions in total. A validation accuracy of **0.6839** was achieved. The advantages of batch normalization are easily notable, as this is the best-performing model yet.

Batch Normalization + Data Augmentation: Finally, the training data was augmented by flipping horizontally some of the images (selected at random), and cropping them at random locations. This augmentation multiplied the number of training images by five. A validation accuracy of **0.7256** was achieved, making this the best version of the model, and beating the baseline significantly.

4.5 Final results

The following table shows, ranked from best to worst accuracy, the performance of all models mentioned on the test set:

Model	**Accuracy**
BatchNorm+DataAugModel	0.7244
BatchNormalizationModel	0.6791
ExtraConvModel	0.6476
DoubleEpochs, lr=0.0008	0.6389
BaselineModel	0.6342
SlightlyBigLearningRate	0.6325
DoubleEpochs, lr=0.0005	0.6319
SmallBatchModel	0.6312
DoubleEpochs, lr=0.0015	0.6292
DropoutModel	0.6217
BigBatchModel	0.6134
ExtraDoubleConvModel	0.5982
BigLearningRate	0.1000

Table 1 Comparison of model accuracies on the CIFAR-10 test set.

5 Conclusion

In this study, the impact of various architectural and training modifications was explored in a Convolutional Neural Network for image classification using the CIFAR-10 dataset. The baseline CNN model achieved an accuracy of 0.6342 on the test set, serving as a reference point for evaluating the effectiveness of the applied modifications.

Through a series of experiments, we demonstrated that adjusting the network depth, tuning hyperparameters, incorporating batch normalization, and applying data augmentation techniques can significantly improve or worsen the classification accuracy of the CNN. The final model, which incorporated these modifications, achieved an accuracy of 0.7244 on the test set, representing a substantial improvement over the baseline.

In conclusion, this study demonstrates the potential for improving CNN performance in image classification tasks through a combination of architectural and training modifications.

References

[1] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015) https://doi.org/10.1038/nature14539

[2] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., *et al.*: Imagenet large scale visual recognition challenge. International journal of computer vision **115**(3), 211–252 (2015)

[3] Krizhevsky, A.: Learning multiple layers of features from tiny images (2009)

[4] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, 770–778 (2016)

[5] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456 (2015). PMLR

[6] Shorten, C., Khoshgoftaar, T.M.: A survey on image data augmentation for deep learning. Journal of Big Data **6**(1), 1–48 (2019)

[7] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library (2019)

[8] Bisong, E.: Google Colaboratory. Apress (2019). https://doi.org/10.1007/978-1-4842-4470-8_7 . https://doi.org/10.1007/978-1-4842-4470-8_7

[9] Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization (2017)

[10] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning, pp. 222–223. MIT Press, ??? (2016). http://www.deeplearningbook.org

[11] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., He, K.: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour (2018)

[12] Summers, C., Dinneen, M.J.: Four things everyone should know to improve batch normalization. CoRR **abs/1906.03548** (2019) 1906.03548