
Fabian Hieber


Mónia Gomes


Moritz Neujeffski

Deep Learning - Image Classification with CNN

***Convincing CNNs** that yes, this is a pipe!*
– Using the **CIFAR-10 dataset!**

Index

Executive Summary

In this project we developed a Convolutional Neural Network (CNN) model to classify images from the CIFAR-10 dataset. Our custom model consists of 23 layers, including convolutional layers with 64, 128, and 256 filters, BatchNormalization, MaxPooling, Dropout, and GlobalAveragePooling. We compared the performance of our model with the VGG16 architecture and applied transfer learning to improve our model's accuracy. The model was designed for image classification across 10 categories and trained using the CIFAR-10 dataset, achieving competitive results through experimentation and refinement.

Working Environment

To establish an efficient working environment for the project, we started by setting up the necessary tools and resources. Given the relatively small scope of the project, we opted not to implement a Kanban project management tool. Instead, we communicated tasks and ideas directly within the team. For collaboration and version control, we created a GitHub repository to efficiently share our contributions. To meet the computational demands of running our models, we utilized two notebooks on the Paperspace platform, along with an additional Google Colab notebook. We also attempted to deploy the Live Share extension for Visual Studio Code to enable collaborative work within a single Jupyter notebook, but this approach proved unsuccessful.

Dataset and Preprocessing

For data preprocessing, only minimal steps were required, as the CIFAR-10 dataset consists of 32x32 pixel images, eliminating the need for resizing. The class labels were converted into categorical values to facilitate multi-class classification. Furthermore, we normalized the image data by scaling the pixel values to a range between 0 and 1, which is essential for improving the model's performance. This was done by converting the pixel values to float32 and dividing them by 255.0 for both the training and test datasets. In addition, we undertook data augmentation steps along the way when training our model. Based on these data preprocessing steps we advanced to developing our training model.

CNN Model (s) Architecture & Development

Our approach was centered around incrementally increasing the complexity and size of our model architecture to enhance performance. We systematically tested a total of eleven different models, ultimately reaching a maximum of 23 layers (see table 3 in the annex for details on the model architectures). Throughout this process, we observed steady improvements in our model's performance. The decision to scale up the architecture was informed by ongoing evaluation of key metrics, ensuring that increased complexity led to tangible performance gains rather than overfitting or diminishing returns. Our metrics rested on the following key indicators:

- **F1-Score:** This metric provides a balanced assessment of precision and recall, making it particularly useful when dealing with imbalanced datasets.
- **Accuracy:** While simple, accuracy remains a fundamental metric, measuring the overall proportion of correct predictions.
- **Loss Value:** This quantifies the error made by the model, with lower values indicating better performance.
- **Validation Loss (val_loss):** This metric monitors the error on the validation set, helping to ensure that the model generalizes well to unseen data.
- **Validation Accuracy (val_accuracy):** This assesses how well the model performs on the validation set compared to the training set, providing insight into potential overfitting.

Performance Improvement Steps

The initial model's architecture consisted of one convolutional layer (32 filters, ReLU activation), followed by a flattening layer and two dense layers (100 neurons and 10 neurons, respectively). The final layer used softmax for multi-class classification. We used a Stochastic Gradient Descent (SGD) with categorical cross-entropy loss and trained the model for 60 epochs. As shown in Table 1 below, we initially achieved an accuracy rate of 0.62. Gradually we improved the performance by taking various steps such as Adding BatchNormalization and Dropouts (model 2); deepening the network by adding 4 Convolutional Layers (model 3) or replacing the flatten function in the end with GlobalAveragePooling2D (model 4). The GlobalAveragePooling helped by increasing the efficiency without sacrificing accuracy.

Further steps included switching the model optimizer to Adam (model 5); introducing EarlyStopping (model 6), adding two convolutional blocks with 256 filters (model 7) and reducing the batch size from 512 to 128 (model 8). Especially Adam's adaptive learning rates led to quicker training and improved the accuracy significantly. This also held true for adding more layers and working with a smaller batch size, which resulted in more frequent updates to the model weights during training and improved our performance again (see table 1). As our validation loss was still much higher than our loss, we introduced data augmentation techniques to reduce the overfitting of our model (model 9). Specifically, we applied random horizontal shifts and rotations to the input images. This decreased the validation loss to under 50%.

In the last step, now in model 10, we decreased Adam's learning rate to 0.001. This lowered the learning rates and led to finer updates, decreasing overfitting. This resulted in an accuracy rate of 0.91, a loss of 0.26, a validation accuracy of 0.87 and a validation loss of 0.42. Thus, model 10 – the MFM-24 – provided the best results. The MFM-24 starts with two convolutional layers with 64 filters to capture low-level features, followed by BatchNormalization and MaxPooling to normalize activations and reduce spatial dimensions, respectively. As the network deepens, we use higher filter counts (128 and 256) in subsequent layers to capture more complex patterns, applying Dropout to prevent overfitting. The model concludes with a GlobalAveragePooling layer and two dense layers, allowing it to flatten the feature maps and classify images into 10 categories.

Further attempts to, for example, reduce the learning rate only resulted in minor effects on improving the model's performance, the gains became marginal at this stage. Lastly, we reduced the dropout rate, but additional reductions in this hyperparameter also yielded diminishing returns (see model 11 in table 1). We concluded our optimization efforts when the model's performance plateaued, as additional changes no longer led to meaningful improvements.

Model #	Accuracy	Loss	Accuracy validation	Validation Loss
1	0.6239	1.0862	0.5747	1.2166
2	0.8395	0.4586	0.6883	0.9531
3	0.8583	0.4044	0.7274	0.8292
4	0.7159	0.8152	0.6550	0.9520
5	0.9783	0.0625	0.8016	0.9593
6	0.9241	0.2275	0.7781	0.7260
7	0.9447	0.1580	0.8292	0.5854
8	0.9514	0.1378	0.8469	0.5476
9	0.8885	0.3184	0.8415	0.4907
10	**0.9078**	**0.2593**	**0.8678**	**0.4217**
11	0.8127	0.5943	0.7535	0.7780

Table 1- Model´s Performance Evaluation

Transfer Learning

To test our model, we chose three pre-trained models to do transfer learning. These were VGG16, ResNet50 and InceptionV3.

1. **InceptionV3** uses multiple convolutional filter sizes in parallel within the same layer, allowing the model to capture both fine and coarse features, which makes it highly efficient in terms of computational complexity while maintaining high accuracy.
2. **ResNet-50** introduces residual learning by using skip connections to bypass certain layers, enabling the training of much deeper networks by mitigating the vanishing gradient problem.
3. **VGG16,** a deep convolutional network with 16 layers that uses very small 3x3 filters, designed to emphasize depth and simplicity, offering strong performance in image classification at the cost of high computational requirements.

Initially we attempted to run the three pre-trained models on our training dataset with the least modifications possible. At this stage we only undertook modifications to fit the pre-trained models to our pre-processed training datasets. This included the data augmentation process of randomly rotating and flipping images horizontally. As the table below (Table 2) shows, VGG16 revealed the best performance on the CIFAR-10 dataset.

Model	Accuracy	Loss	Accuracy validation	Loss validation
(#15) InceptionV3	0.61	1.08	0.65	1.04
(#14) ResNet-50	0.41	1.64	0.43	1.59
(#12) VGG16	**0.83**	**0.47**	**0.75**	**0.82**

*Table 2 - **Evaluation of the pre-trained models on the CIFAR-10 dataset performance***

Description on our final pre-trained model (vgg16 with unfreezing last four layers)

Our most accurate pre-trained VGG16 architecture model leveraged transfer learning by freezing most of the VGG16 layers while tweaking the last four layers. By unfreezing the final layers (vgg16_base. layers [-4:]), the model is able to retain the feature extraction capabilities of VGG16, which was trained on ImageNet, while adapting to the specific features of the CIFAR-10 dataset. This approach allows the model to improve the deeper, high-level feature representations in the dataset without overfitting too much or losing the powerful generalization ability of the pre-trained layers.

Additionally, the model integrates a GlobalAveragePooling2D layer, which reduces the dimensions of the feature maps, helping to prevent overfitting by reducing the number of parameters while preserving spatial information. The final dense layer uses SoftMax activation to classify the images into 10 CIFAR-10 classes. We keep the other parameters constant (Adam for optimization, learning rate of 0.0001) to ensure a greater consistency with our previous modeling efforts.

This model achieved an accuracy of 0.97, a loss of 0.075, a validation accuracy of 0.88, and a validation loss of 0.43, making it the best-performing model overall. However, the significant gap between the validation loss and training loss suggests strong overfitting. While overfitting remains a challenge, we are confident that with more time, we could implement strategies to reduce it effectively.

Conclusion

From our experience working on CNN model development, we've learned the importance of starting with broad changes, such as modifying batch size, optimizers, layers, and data augmentation techniques. This initial experimentation gave us a solid foundation before moving into more precise adjustments. In conclusion, our work on the CIFAR-10 dataset highlights three key strategies that significantly improved model performance. First, leveraging BatchNormalization and Dropout helped stabilize training and reduce overfitting. Second, switching to Adam optimizer with an adjusted learning rate accelerated convergence and enhanced accuracy. Finally, in transfer learning, unfreezing the last four layers of VGG16 enabled improving accuracy while preserving the pre-trained model's generalization power. These steps, along with minimalistic data preprocessing and systematic architecture adjustments, led to a highly accurate model, achieving 0.97 accuracy, while still facing some overfitting challenges.

ANNEX

Model#	Key Changes	Optimizer	Key Improvements	Implications
Created during model improvement				
1	Baseline CNN, 1 Conv Layer	SGD	Simple architecture. overfitting	Struggles to capture complexity, needs more layers and regularization
2	Added BatchNormalization & Dropout (0.25)	SGD	Stabilized training, reduced overfitting	Regularization helped generalization but still lacked complexity
3	Deepened CNN, more Conv Layers (64/128)	SGD	Better feature extraction due to increased depth	Higher accuracy but higher computational cost
4	GlobalAveragePooling instead of Flatten	SGD	Reduced model size, same performance	Efficient without sacrificing accuracy
5	Switched to Adam Optimizer	Adam	Faster convergence, improved performance	Adam's adaptive learning rates led to quicker training

6	Introduced EarlyStopping	Adam	Reduced overfitting, halted unneeded training	Generalization improved; training time reduced by 15-20%
7	Added more filters (256)	Adam	Increased depth, better accuracy	Higher computational cost, diminishing returns in accuracy improvement
8	Reduced Batch Size to 128	Adam	More precise weight updates	Slight improvement but longer training time
9	Added Data Augmentation	Adam	Reduced overfitting, better generalization	Augmentation improved model robustness
10	**Reduced learning rate to 0.001, Data Aug.**	**Adam**	**Finer updates, improved stability**	**Combination of low learning rate and augmentation helped stability**
11	VGG-replica deep network with Dropout	Adam	High accuracy but computationally expensive	VGG architecture offers good depth, but resource-intensive

Used in transfer learning				
12	VGG16 pre-trained (frozen layers)	Adam	Transfer learning improved accuracy	Efficient training due to pre-trained layers
13	Fine-tuned VGG16	Adam	Fine-tuning boosted performance	Fine-tuning enabled better adaptation to CIFAR-10 dataset
14	ResNet50 (transfer learning, frozen layers)	Adam	Strong feature extraction via ResNet	Skip connections helped deeper training, great performance
15	InceptionV3 (input resized to 75x75)	Adam	Mixed convolutions, strong generalization	InceptionV3 balances complexity and accuracy
16	Fine-tuned VGG16, lower learning rate	Adam	Best performance due to fine-tuning	Fine-tuning with a low learning rate resulted in high accuracy

Table 3 - Details on performance improvement*