# Animals classification project (CNN)

## Introduction

The objective of this report is to present the various Convolutional Neural Network (CNN) architectures explored throughout the project, including both custom and transfer learning models. It also outlines the results obtained from these experiments and discusses the initial challenges encountered during the development process.

## Methods and architecture

After the initial struggle, when the trainings were failing one after another, things finally started to stabilize. Starting from scratch was the key to get the first model which hit almost 0.7 in validation accuracy. That was a real leap after several detraining epochs. After some trial and error, tweaking, and documentation, we finally got something more stable. Additionally, we experimented with reloading our own trained model weights and rerunning the models, which led to some successful results. However, this approach also inadvertently introduced unwanted data leaks, highlighting the need for careful management of training and validation data when reusing model weights.

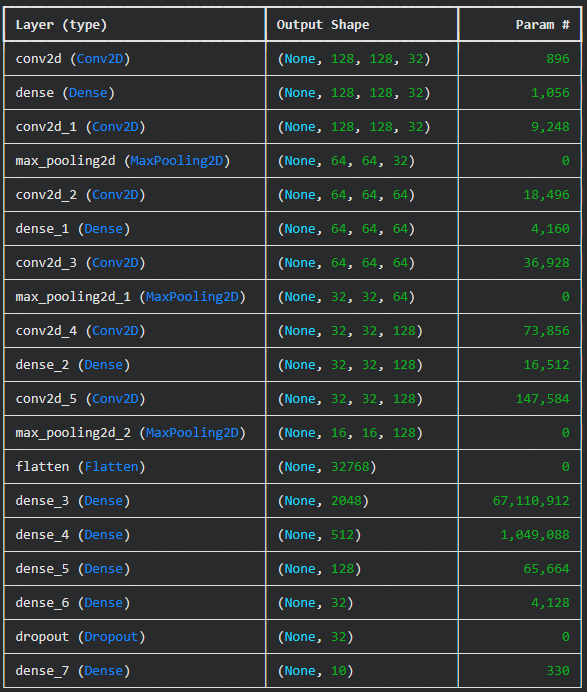
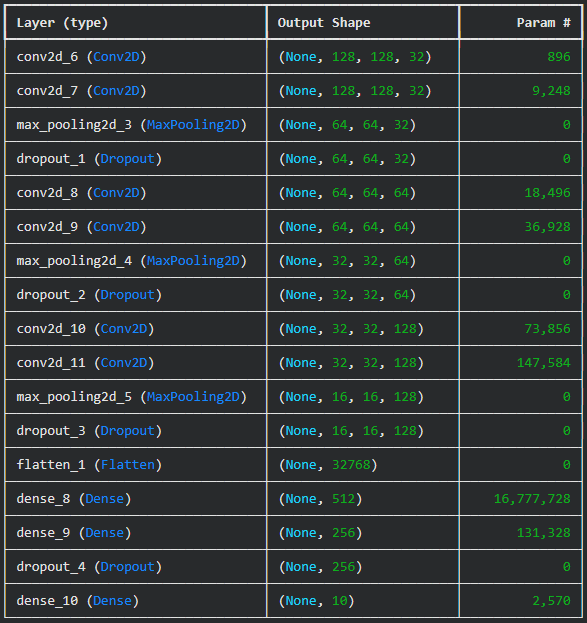
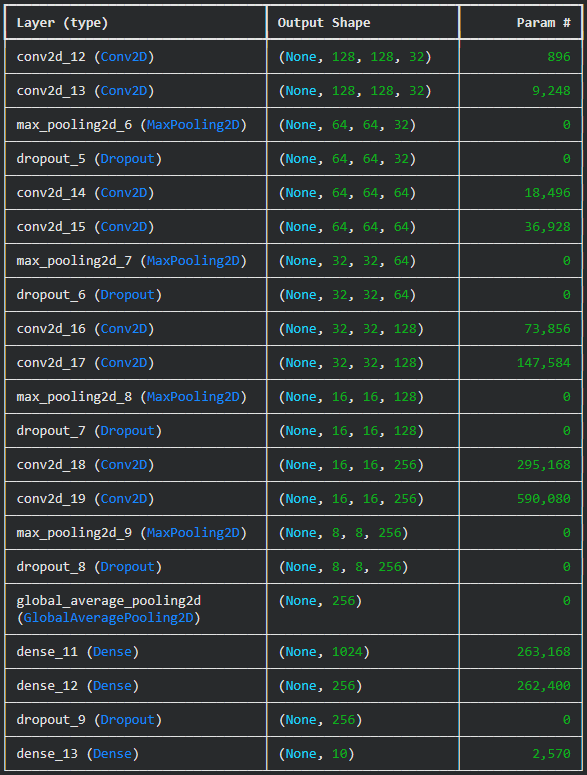
### Custom model

#### Architectures

We started with a simple model (Architecture 1) with 3 blocks of Conv2D and MaxPooling2D, adding Dense layers between them from 32 filters up to 128 and finally increasing the Dense layers to the output strongly as a final block. Notably, we observed a significant improvement in the model's ability to train and generalize when we reduced the input image size to 128 by 128 pixels, which helped stabilize training and achieve better results.

In the second architecture, Dense layers were removed inside the Conv2D blocks, keeping it simpler. Dropouts were added at the end of each Conv2D block in ascending order (from 0.1 to 0.5). The final Dense Layers were simplified as well to reduce model complexity.

The third architecture is similar to the second one but adds another double Conv2D block up to 256 filters to get more details from the images. The Flatten layer was replaced by a GlobalAveragePooling2D, allowing for better spatial information aggregation and improved generalization while reducing the total number of parameters:

From left to right (architecture 1, 2 -center- and 3 -right)

#### Model comparison

Models ummary table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Architecture | Image shape | Epochs | Learning Rate | Optimizer | Final val accuracy | Final training accuracy | Final validation loss |
| 1 | 128x128 | 35 | 0.0002 | Adam | 0.695 | 0.67 | 0.72 |
| 2 | 128x128 | 35 | 0.0002 | Adam | 0.799 | 0.816 | 0.61 |
| 3 | 128x128 | 35 | 0.0002 | Adam | 0.833 | 0.84 | 0.53 |

#### Transfer Learning

**Pre-trained Model Experimentation**: We implemented transfer learning using several pre-trained convolutional neural network architectures, leveraging only their convolutional base layers while freezing the pre-trained weights to expedite training and prevent model degradation. This approach allowed us to benefit from features learned on ImageNet while adapting the models to our specific animal classification task. The frozen weights strategy significantly reduced training time while maintaining the robust feature extraction capabilities developed on millions of images.

**Architecture Performance Comparison**: Initial experiments with VGG16 and ResNet50 architectures yielded disappointing results, with both models struggling to achieve validation accuracies above 0.75 despite various optimization attempts. The breakthrough came when transitioning to the EfficientNet family of models, which immediately demonstrated superior performance with validation accuracies consistently exceeding 0.80. Through systematic experimentation with different EfficientNet versions, we ultimately settled on EfficientNetB7 as our base architecture, which provided the optimal balance between model complexity and performance for our dataset.

**Critical Architecture Optimization**: The most significant performance breakthrough was achieved through careful optimization of the flattening method used to connect the pre-trained convolutional base to our custom classification head. Initial attempts using global average pooling or direct flattening of all feature values resulted in overwhelming information density that our dense layers could not effectively process, limiting performance to modest accuracy levels. By implementing max pooling as the flattening strategy, we dramatically improved the model's ability to focus on the most salient features, ultimately achieving exceptional performance with over 0.96 accuracy on training data and 0.97 accuracy on validation data. This architectural refinement proved crucial in unlocking the full potential of the EfficientNet base model for our animal classification task.

| **Class** | **Precision** | **Recall** | **F1-Score** | **Support** |
| --- | --- | --- | --- | --- |
| cane | 0.98 | 0.97 | 0.97 | 486 |
| cavallo | 0.96 | 0.97 | 0.96 | 262 |
| elefante | 0.99 | 0.99 | 0.99 | 144 |
| farfalla | 0.95 | 0.99 | 0.97 | 211 |
| gallina | 0.97 | 0.99 | 0.98 | 309 |
| gatto | 0.99 | 0.98 | 0.98 | 166 |
| mucca | 0.94 | 0.93 | 0.93 | 186 |
| pecora | 0.95 | 0.94 | 0.94 | 182 |
| ragno | 0.99 | 0.98 | 0.99 | 482 |
| scoiattolo | 0.97 | 0.98 | 0.98 | 186 |
| **Accuracy** |  |  | **0.97** | **2614** |
| **Macro Avg** | **0.97** | **0.97** | **0.97** | **2614** |
| **Weighted Avg** | **0.97** | **0.97** | **0.97** | **2614** |

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Deployment

## Our deployment efforts failed due to significant computational environment incompatibilities between training and inference platforms. In an attempt to minimize resource costs and deployment complexity, we tried to deploy the EfficientNetB7 model in a CPU-only environment, despite having trained it on GPU infrastructure. Unfortunately, this approach revealed a critical flaw in our deployment strategy: the model exhibited considerable degradation in accuracy when transferred from the GPU training environment to a CPU inference environment. This cross-platform incompatibility resulted in completely incorrect predictions, transforming what should have been confident, accurate classifications into unreliable outputs that would be unusable in any production scenario.

## The severity of this GPU-to-CPU transition problem is exemplified by our test results on a simple dog image classification task. When running on GPU, the model performed flawlessly, correctly identifying a dog image with 99.57% confidence, followed by reasonable secondary predictions of cat (0.37%) and cow (0.03%). However, the same model running on CPU infrastructure failed catastrophically, misclassifying the identical dog image as a spider with only 61.15% confidence, followed by butterfly (34.89%) and chicken (3.01%) - a complete failure in both accuracy and confidence. Despite extensive troubleshooting efforts including experimenting with different data types, removing batch normalization layers, and adjusting weight freezing strategies, we were unable to resolve this fundamental compatibility issue. The next logical step would involve retraining the model specifically in a CPU environment or implementing specialized weight adjustment techniques, but these approaches were beyond the scope of our current project timeline and represent areas for future development.

## Conclusions

During the different training phases conducted with the custom CNN architectures and transfer learning models, several insights were obtained regarding model performance, optimization, and computational efficiency:

#### Input image size and performance trade-off

The use of heterogeneous image dimensions, specifically 224 × 224, proved challenging for the custom CNN model. While higher resolutions theoretically provide richer spatial information, in practice, this setup significantly increased training time and computational load. The model converged slowly and required considerable resources to process each epoch, reducing experimental efficiency.

#### Optimizer experimentation

Another optimizer was tested (AdamW), but did not produce a substantial improvement over the baseline Adam configuration. Most adjustments to the learning rate or optimizer type led to marginal or unstable gains, suggesting that the model’s performance limitations were more related to architecture and data variability than to the optimizer itself.

#### Training duration and epochs

The experiments were capped at 35 epochs due to time and resource constraints. However, the model was still improving in both training and validation accuracy near the final epochs. This indicates that additional epochs could have yielded better results (adding dynamic learning rate schedule or fine-tuning phase).

#### Fine-tuning and incremental training

Future improvements could be achieved by leveraging the *initial\_epoch* parameter in Keras. This would allow training to resume from a previously saved checkpoint, effectively enabling a second phase of fine-tuning with a reduced learning rate. This strategy would help the model refine its learned representations without losing previously acquired knowledge.

#### Impact of data augmentation

Although data augmentation techniques were applied (such as rotations, flips, and zooms), they did not lead to a significant boost in validation accuracy. This suggests that the dataset already contained a reasonable amount of variability, or that the augmentation transformations did not effectively represent the intra-class diversity of the dataset.

#### Transfer learning and hardware considerations

Transfer learning models (e.g., based on VGG16, EfficientNetB7) showed clear advantages. But… problems when predciting