An Iterative Approach to Image Classification

Tulceanu Cezar-Alexandru
cezar.tulceanu@gmail.com
github.com/your-username/your-project-repo

October 12, 2025

Abstract

This document outlines the step-by-step process of developing a Convolutional Neural Network for an image classification task. Six distinct attempts were performed, starting from a simple baseline and progressively introducing techniques such as regularization, data augmentation, class weighting, and learning rate scheduling to address challenges like overfitting and training instability. Each step's motivation, implementation, and results are documented, culminating in a final model that achieved a 90.9% accuracy on Kaggle.

1 Model Development and Experiments

1.1 Attempt #1

Code and graphs: Folder ML1

To begin, I studied the images and observed that they mostly contain AI-generated animals performing various activities. For example, there are animals playing chess, riding bicycles, animals (especially zebras) eating, animals washing themselves, etc.

I should mention that I will be using the TensorFlow library on the Google Colab platform.

Important subsequent observation: In this attempt, I made the mistake of working on the Colab CPU instead of the GPU, and for this reason, I had to make the first model quite small.

Initially, I wanted to familiarize myself more with the workspace, so for this attempt, I did not use any form of image augmentation. The only modification to the images was dividing the numbers in the RGB channels by 255 to bring them into the [0,1] range so they could be fed into the model.

Given this information, my first idea was to somehow analyze the small-scale details of the images. For instance, to identify zebras, the model I was about to build should analyze small quadrants of the image and look for the zebra pattern. For this reason, I started with a 2D convolutional layer that applies 32 filters with a 3x3 kernel on the image.

After this step, I also added a max pooling layer to reduce the layer's size by a factor of 0.5 and to discard information that is likely irrelevant.

Then, I repeated the same series of layers 2 more times to process the image details even better. In each subsequent layer, I used more neurons to extract more and more features. The rest of the hyperparameters were set to standard values.

During training, an obvious problem emerged, namely overfitting, as shown in the graph below. It is clear that the validation accuracy peaked at around 79%, when the model reached its maximum performance, after which it began to decline. On the other hand, the training accuracy reached almost 98%, a sign that the model started to memorize the training data perfectly.

Indeed, for this attempt, I did not use any regularization techniques, so it was expected that such a problem would appear. In the following attempts, I will implement strong regularization techniques.

This attempt received a score on the Kaggle test of approximately **78%**. Quite good for the first submission.

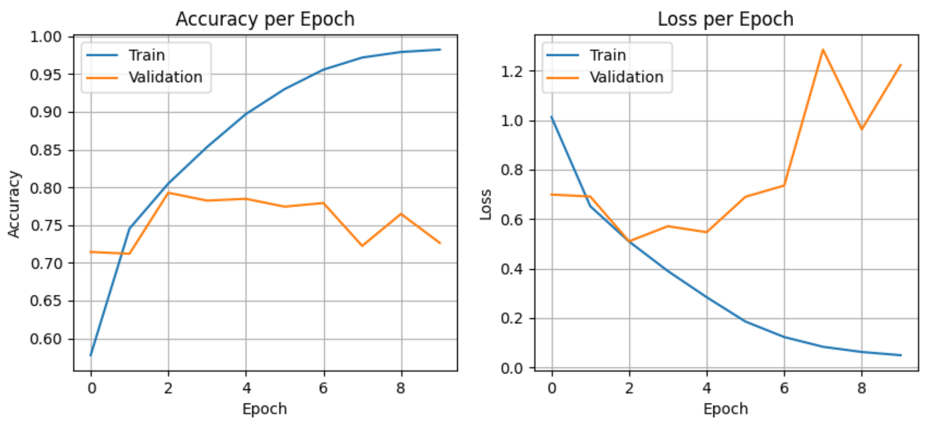


Figure 1: Attempt #1 Training Curves: The high divergence between training and validation accuracy demonstrates clear overfitting.

1.2 Attempt #2

Code and graphs: Folder ML2

This attempt is largely based on the first one but has some minor modifications. This time, I also did not use any augmentation because, first, I want to familiarize myself with the models and get an idea of what might work.

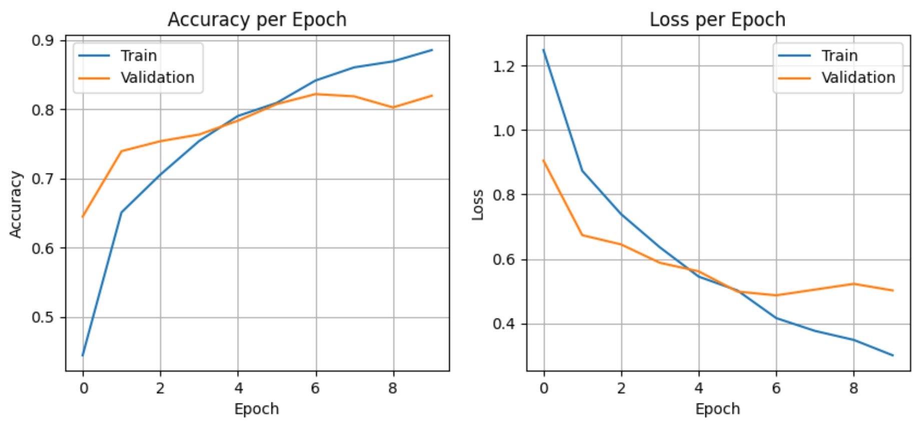
The main modification was changing the model's structure. To start, within the second convolutional layer, I increased the kernel size to 5x5 to facilitate a broader view of the image details for the model. Then, instead of the dense layer with 256 neurons, I added two layers with 128 and 64 neurons, respectively, to facilitate a more organized, step-by-step data processing method.

Then, the most important change by far was the addition of the first regularization methods. First, I added neuron dropouts for each layer. Larger in the central layers and smaller in the layers closer to the input/output to keep the feature extraction and class membership conclusion as accurate as possible (I mean that these network skills could be largely memorized to achieve an optimal result). The central layers of the network have a higher dropout rate to allow the network to ”think” in all sorts of different ways at each iteration.

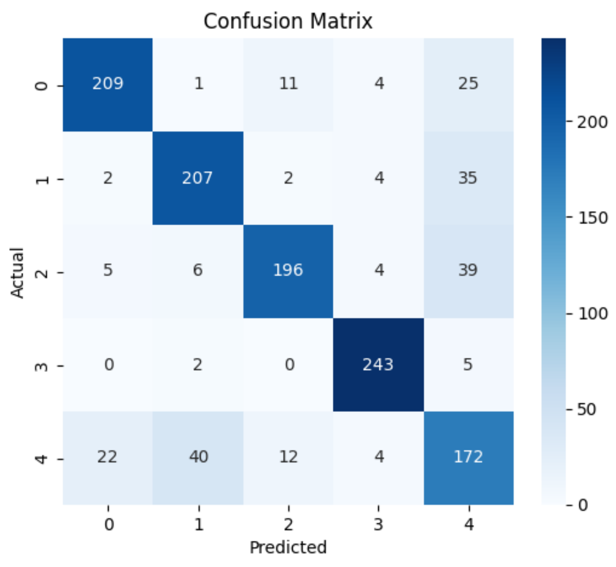
From the first attempt, I noticed that the validation accuracy dropped sharply towards the end of training, and for this reason, in the second attempt, I also implemented early stopping.

At this point, I also introduced the code for generating the confusion matrix on the validation set, as well as for plotting accuracy and loss function graphs across epochs. (I later added these to the first model from Attempt #1 as well).

This attempt eventually obtained an accuracy of approximately **82%**. Better than the first, but not by much.



(a) Training and Validation Accuracy/Loss Curves



(b) Validation Confusion Matrix

Figure 2: Attempt #2 Results: Structural modifications and regularization improved generalization.

1.3 Attempt #3

Code: Folder Merger

This attempt is more about testing whether the idea of an ”ensemble of experts” works. Specifically, I generated several .csv files (about 10) with predictions for submission using the models from the first two attempts and with very small variations in hyperparameters, which I didn't get to document as they weren't very important.

Then I used a merger script to choose the most popular prediction for each picture from the 10 CSVs. To my surprise, although none of the variations based on the first two attempts had exceeded 82.5% on Kaggle, the new .csv document obtained by combining them achieved an accuracy of **84%**.

I will try to apply this technique to future submissions as well.

1.4 Attempt #4

Code and graphs: Folder ML4

In the confusion matrix from Attempt #2, I noticed something very strange, namely that the model was struggling a lot and making quite a few mistakes when identifying images from class 4. It performed considerably better on the other classes. I decided to do something to fix this problem.

At this point, I considered it was time to add some form of data augmentation. I added a little bit of rotation, horizontal and vertical shifting, as well as zoom and horizontal flip. The image generator will apply random transformations from the range described in the code at each epoch for every image in the training dataset.

Finally, the cherry on top is tripling the class 4 images, while the other images are only doubled in number of appearances in the training dataset. This maneuver was done to ensure that the model sees as many class 4 images as possible to repair its deficiency in classifying these images.

Another very important change was adding a secondary branch to the model that takes the input again, applies a convolutional layer and an AVGPooling layer to it, and then merges with the main branch. At the end, I increased the number of neurons in the output layers to facilitate a better and broader passage of the newly increased information flow.

The last major change was to add a decay to the learning rate. I had made the learning rate increase over time because I was getting bored when the validation accuracy started to stagnate. (I later discovered this caused model instability).

This attempt obtained an accuracy of **85.7%** on Kaggle. A new record, but not by much.

Also, a problem I want to fix later is the training instability. The graph clearly shows how both the loss and the accuracy on the validation data vary extremely, making me feel more like I'm at a casino than at university during the model training. I will try to stabilize the model's performance on the validation data in the next attempts.

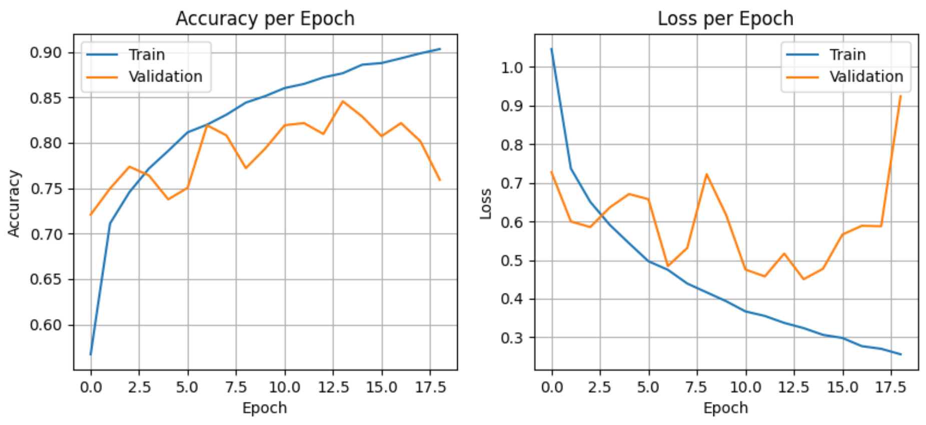


Figure 3: Attempt #4: Validation metrics show high instability despite the improved accuracy.

1.5 Attempt #5

Code and graphs: Folder ML5

This time I didn't make too many big changes. In this attempt, I emphasized data augmentation to allow the model to learn in a much more general way, thus reducing overfitting.

The second very important change was to greatly increase the "thickness" (width) of the model's final layers, to allow it to draw conclusions better. I also thickened the initial layers a bit to let more information pass through the model.

I also added regularizers to several layers to prevent the model from having weights that are too large, which would lead to memorizing certain things and overfitting.

I also increased the patience of the early-stopping mechanism to continue even after an initial drop in accuracy. This step was necessary because the model was still unstable.

This attempt obtained an accuracy of **89%** on Kaggle, far exceeding the other models.

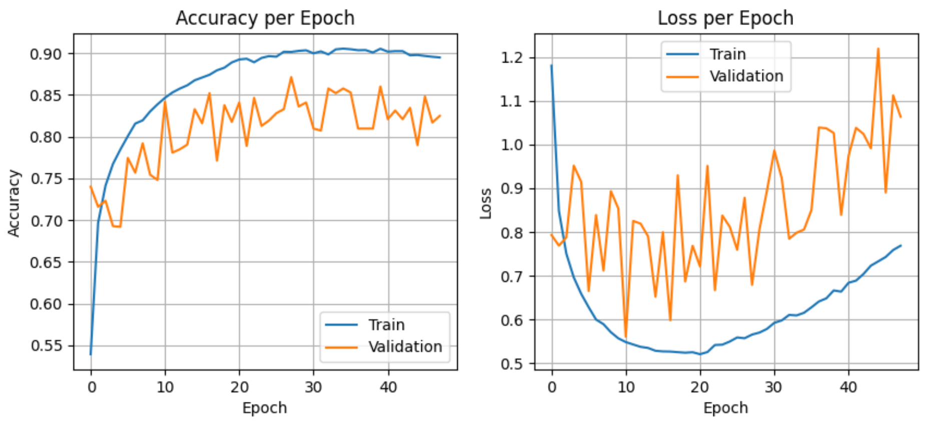


Figure 4: Attempt #5: Increased model capacity and augmented data improved accuracy to 89%.

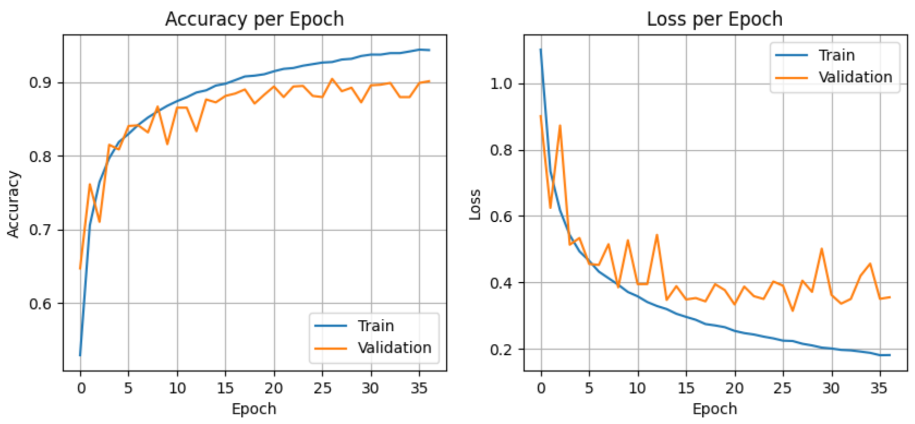
1.6 Attempt #6

Code and graphs: Folder ML6

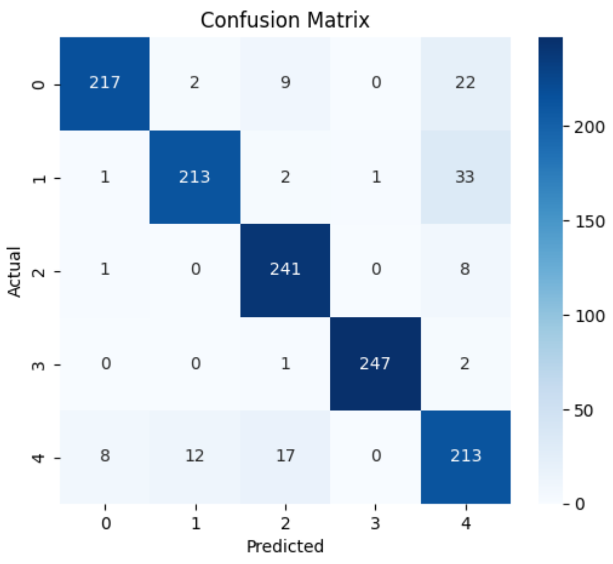
Here, I just made some small adjustments. First, I set the proportions between all images back to equal to see if the idea of showing more class 4 images was actually helpful.

The most important change was setting the learning rate correctly, making it decrease as time goes on. The decay rate was set to 0.95 once every 1000 steps.

The stability of the model is evident from the accuracy and loss graphs. This attempt, combined with some techniques inspired by the concept of an ”ensemble of experts”, obtained an impressive final score of **90.9%** on Kaggle.



(a) Stable training curves



	0	1	2	3	4
0	217	2	9	0	22
1	1	213	2	1	33
2	1	0	241	0	8
3	0	0	1	247	2
4	8	12	17	0	213

(b) Balanced confusion matrix

Figure 5: Attempt #6: Final stable model with smooth convergence.