IMAGE CLASSIFICATION

MSC Artificial Intelligence For Media
2024-2025

Alexander Moed

0.1 Introduction

The task was to develop a classification model for a dataset of animal photographs provided. The dataset contains images from 7 categories: Squirrel, Lion, Horse, Elephant, Chicken, Camel, and Bear. The data was pre-organized into separate test and training folders, with 70 images distributed across 7 categories in the test set and 2,392 images across the same 7 categories in the training set.

0.2 Research

Before starting this project, I wanted to survey the available methods and identify the best for this task, along with their pros and cons. My bias was to explore options beyond convolutional neural networks (CNN) because I had used it before with good results, but I believed there were probably more advanced alternatives available. I determined that transformer-based architecture used in Vision Transformers (ViT) and Convolutional Neural Networks (CNN) are the main models currently being used.

0.3 Transformers

I was interested in Vision Transformers (ViT). What particularly interested me was the fact that ViT looks at the whole image, which allows it to understand the relationship of each part of the image and how each part relates to one another. It does not rely on a kernel size to determine the area that is sampled (Trigka and Dritsas (2025)). This was important because the images are different sizes and sometimes focus on different parts of the animal. Having a complete view of the image would be ideal. However, a concern was the computational expense of transformers (Trigka and Dritsas (2025)). Another concern was that we had a limited amount of data, and ViT is designed for large-scale datasets—sometimes requiring tens of millions of images, which we do not have. Additionally, it is a requirement of this assignment to not to use pre-trained models. I believe it would be unlikely for ViT to work effectively on our relatively small dataset (Mekal (2025)).

0.4 CNN

A positive aspect of CNN is that it works well with limited data. It even outperforms ViT in some cases with small datasets. I could also adjust the kernel size to be slightly larger to capture more features (van der Werff (2024)). Additionally, CNNs are best if pre-trained checkpoints are unavailable, which we intentionally do not have. CNN has also undergone thorough testing and has been utilised in various applications. The negatives of CNN include that it might not capture the larger picture and focuses on smaller pieces because it uses a kernel structure to extract features (van der Werff (2024)). Despite the negatives, it is clear that CNN is the right choice due to data limitations and the requirement not to use pre-trained models.

0.5 Building a CNN image classifier

0.6 Version: 1

I decided to start with a simple image classifier and build up from there. After examining the data, I noticed some small differences between the sizes of the images. This

meant I needed to apply uniform resizing; I initially chose 128x128 dimensions for efficiency. My first approach was a simple fully connected neural network to establish a baseline. The model contained two hidden layers with ReLU activation functions and implemented a moderate dropout rate of 0.1 at the end of the network. I used CrossEntropyLoss as my criterion and Stochastic Gradient Descent (SGD) as my optimizer with a learning rate of 0.01. I placed the dropout only at the end because dropout should not be used in feature extraction layers. This model was very basic without any data augmentation. It got stuck at around 58% accuracy and seemed like it might be overfitting.

```
Epoch 50: Loss=0.0355, Test Accuracy=58.57%, LR=0.00928
```

0.7 Version: 2

I upgraded to a more complex CNN model in this version and increased the input image size to 512×512. I wanted to test whether providing more detail and using larger kernel sizes would improve performance. The results showed this approach did help compared to the 128x128 resolution. I introduced light data augmentations to improve variation without drastically altering the input. Specifically, I added a random horizontal flip, a random rotation of 10 degrees, and a slight colour jitter of brightness 0.1, contrast 0.1, and saturation 0.1, all moderate adjustments to simulate added realistic variation. I excluded vertical flips as they could confuse the model and rarely reflect realistic images. The model begins with a convolutional layer using a kernel size of 4 and padding of 1 to help capture medium-sized features in the image. I kept the stride at 1 to have more overlap and feature extraction. These modifications were intended to test performance with increased resolution and subtle augmentation. This version significantly improved, but the model's learning hit a ceiling early. It broke through 70 per cent accuracy around 30 epochs and peaked at 77 per cent by epoch 41. After that, it got stuck and started getting worse, hovering in the 60-70% range. This pattern made it clear that we needed to try something different with the architecture or the training approach. Overfitting still is happening.

```
Epoch 41: Loss=0.3678, Test Accuracy=77.14\%, LR=0.00820
```

0.8 Version: 3

I focused on finding the correct optimizer in this version, and kept the other settings the same. I succeeded in different projects with the Adam optimizer and switched to it here. I also added a scheduler, which I've seen used in other methods. Having a scheduler decreases the learning rate over time and hopefully allows for more fine-tuning as the model converges. The learning rate for Adam operates on a different scale, so I lowered it from 0.01 to 0.001. The gamma for the scheduler was set at 0.9785. It lowers the learning rate, which seemed moderate. But over time, I realized it might have been too aggressive considering where it started. It was clear that Adam was helping the model push past testing 80% accuracy, but I wanted to go further than 84%. Another thing I attempted was increasing the batch size. Exposing the model to more samples per iteration would lead to faster and better learning. Here is the best epoch:

```
Epoch 72: Loss=0.0212, Test Accuracy=84.29%, LR=0.00026
```

0.9 Version: 4

In this version, I decided to revisit the image resolution and test whether 512×512 was necessary. I scaled it down to 256×256 to see if I could achieve similar or even better performance and I did. This version outperformed the previous ones. My main goal was to capture the general shapes of the animals more efficiently, so I modified the first Conv2d block to have a stride of 2, which helps the model extract broader features earlier and reduces the spatial size upfront. Another change was the addition of batch normalization after each convolutional layer to improve training stability and convergence. I also increased the strength of the colour jitter augmentation from 0.1 to 0.25. The idea was to introduce more variety in training data and reduce overfitting. I also increased dropout to 0.3 at the end. The dropout placement was intentional. As mentioned above I did not want to have dropout in feature extraction. After researching optimizers, I switched to AdamW, which separates weight decay from updating the gradient. This is more effective than using weight decay with Adam, where the penalty is applied inside the gradient and gets distorted by Adam's adaptive updates.(Loshchilov and Hutter (2019)) I also added this weight decay to penalize the model for becoming too confident and to promote better generalization. I set it at 1e-4, which I believe is aggressive but not extremely aggressive. I also experimented with a higher numeric value of 1e-3 and a lower value of 1e-5, but 1e-4 gave me the best results. With these changes, we gained 4% accuracy.

```
Epoch 77: Loss=0.0498, Train accuracy=88.57%, LR=0.00053
```

0.10 Version: 5

In this version, I decided to push the model further and reach the 90 percent range. Looking at earlier epochs, I noticed signs of overfitting. The model kept plateauing, and I realised I needed to compare the training and testing accuracy for each epoch to understand what was happening. I saw that early on, the training accuracy was lower than testing but later, it quickly surpassed the test accuracy. Testing accuracy rose slowly, then plateaued. That led me to believe the model was overfitting and that the transformations needed to be changed.

Another issue was that some training images were naturally cropped, only parts of the animals were visible, and the model was not handling that well. So I added RandomErasing to simulate partial occlusion and help the model learn to recognize animals even if only parts are visible. I also increased the RandomRotation to 25 degrees to introduce more variation. To fight overfitting further, I increased the dropout to 0.5. I also experimented with decreasing the numeric value of the weight decay to 1e-5, intending to penalize the model more harshly when it started to memorize. I did not see an improvement from the weight decay change. I later realised it's because I had it backwards and was making it more lenient (Loshchilov and Hutter (2019)). I also changed the learning rate to 0.001100; I wanted to start slightly higher due to the learning rate scheduler. I did this because I noticed that in the first few epochs, the model made large jumps in performance and then dropped off, which is expected, but I wanted to test if keeping the learning rate higher for longer might help push through that plateau. While it did not break into the 90s, the testing accuracy reached the 70s faster (by epoch 17), but began declining as shown below. The problem is that training accuracy increased, while testing accuracy began to decrease over time and eventually stalled.

```
Epoch 17: Loss=0.4874, Train Acc=83.40%, Test Acc=77.14%, LR=0.00079
Epoch 18: Loss=0.4403, Train Acc=84.41%, Test Acc=71.43%, LR=0.00077
```

```
Epoch 19: Loss=0.4431, Train Acc=83.90%, Test Acc=68.57%, LR=0.00076
```

0.11 Version 6

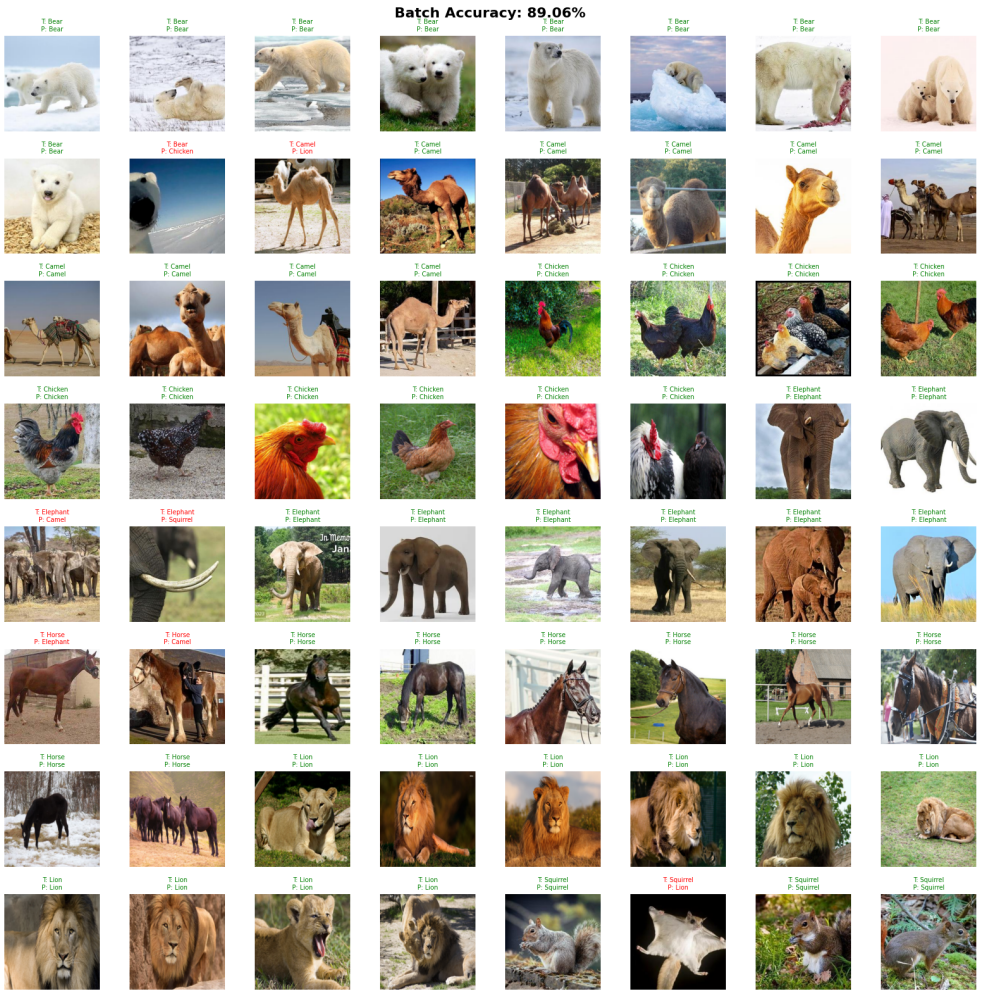
I also experimented with RandAugment as an automated augmentation strategy. This was my main change, using the setting transforms.RandAugment(num_ops=2, magnitude=9). The goal was to let the model see more diverse and randomized transformations without hand-tuning individual augmentation parameters. However, I did not observe a noticeable performance improvement in practice compared to my manually defined augmentations. It also seemed not to help overall. The augmentations selected by RandAugment were either too aggressive or not well-suited to my dataset, which already included transformations like ColorJitter, RandomRotation, and RandomErasing. This version was not working, so I stopped the testing here.

```
Epoch 24: Loss=0.3962, Train Acc=86.25%, Test Acc=74.29%, LR=0.00087
Epoch 25: Loss=0.3592, Train Acc=87.63%, Test Acc=82.86%, LR=0.00087
Epoch 26: Loss=0.3639, Train Acc=86.87%, Test Acc=78.57%, LR=0.00086
Epoch 27: Loss=0.3536, Train Acc=87.54%, Test Acc=82.86%, LR=0.00085
Epoch 28: Loss=0.3197, Train Acc=88.21%, Test Acc=82.86%, LR=0.00084
Epoch 29: Loss=0.3001, Train Acc=89.72%, Test Acc=80.00%, LR=0.00083
Epoch 30: Loss=0.3285, Train Acc=88.13%, Test Acc=72.86%, LR=0.00083
Epoch 31: Loss=0.2671, Train Acc=90.80%, Test Acc=81.43%, LR=0.00082
Epoch 32: Loss=0.3015, Train Acc=90.34%, Test Acc=77.14%, LR=0.00081
Epoch 33: Loss=0.2913, Train Acc=89.76%, Test Acc=71.43%, LR=0.00080
```

0.12 Version: 7

I returned to the settings and configuration from Version 5 since RandAugment in Version 6 did not improve performance. I removed the RandAugment and ran the model for a significantly longer period. This approach was successful, but only after a lengthy training run. Previously I had stopped at 50 epochs, but it took 300 to reach 91.43% test accuracy. The model achieved 90.00% accuracy around epoch 120, which would probably be a more practical stopping point considering the time and computation required.

```
Epoch 290: Loss=0.0165, Train Acc=99.46%, Test Acc=91.43%, LR=0.00007
```



0.13 Conclusion

The model has developed a good understanding of the training data, achieving up to 99 per cent training accuracy and between 89-91.43 per cent testing accuracy on the 64-image batch testing sample. Despite these promising results, there are still a few examples it struggled with.

Looking at the batch display, some interesting misclassifications stand out. Despite reaching an accuracy of 89.06%, specific images fall outside what the model handles well. For instance, classifying the flying squirrel as a lion was unusual. But It is also understandable that it might not be recognized as a squirrel since the other squirrels in the dataset look different I do not understand why it specifically classified it as a lion. Another confusing case was a bear being predicted as a chicken, and an elephant labelled a squirrel. To help mitigate these misclassifications, I introduced RandomErasing, hoping it would encourage the model to focus on individual features and really teach them even in images that do not show full animal or face. Starting with a larger kernel size helped the model capture the overall shape of each animal. This likely contributed to the high accuracy, especially when the whole body was visible. However, the model correctly classified the image even with close-up images, such as a tight crop on a chicken's face. This suggests it recognized both fine and coarse details. The architecture effectively balanced global and local feature extraction in many cases. I could not save a checkpoint at the best epoch, so the final

testing had slightly lower accuracy (89% vs 90%). If I were to do this over again, I would have had a stricter procedure for testing in terms of the number of epochs per test, and I would have made fewer changes per test to pinpoint better what caused the improvements or loss of accuracy.

Bibliography

Loshchilov, I. and Hutter, F., 2019. Decoupled weight decay regularization. URL https://arxiv.org/abs/1711.05101.

Mekal, P. M., 2025. Transformers vs. cnns: The battle for image classification supremacy. medium.com/@pmekal25/transformers-vs-cnns-the-battle-for-image-classification-supremacy. Accessed: 2025-04-16.

Trigka, M. and Dritsas, E., 2025. A comprehensive survey of deep learning approaches in image processing. *Sensors*, 25 (2). URL https://www.mdpi.com/1424-8220/25/2/531.

van der Werff, T., 2024. Cnn vs. vision transformer: A practitioner's guide to selecting the right model. https://tobiasvanderwerff.github.io/2024/05/15/cnn-vs-vit.html. Accessed: 2025-04-16.