Light source presence in images
Experiments

Filip Comșa

One of the biggest challenges that I faced was finding the best model and the best parameters to achieve high accuracy. Through much experimentation I have arrived to a model that is up to 85% accurate. Experiments through which I went trying to find the best model and settings are described below.

1 Existing Architectures - Efficient Net

EfficientNet is a convolutional neural network architecture and scaling method that uniformly scales all dimensions of depth/width/resolution using a compound coefficient.
Through multiple experiments, we tried all EfficientNet BX from B0 to B5 (as much as the hardware could handle).
Model was trained using an M1 Pro processor having 10 core CPU, 16 core GPU and 16 core Neural Engine.
The results were not very good, having reached a maximum of 65% validation accuracy.

2 AutoML - AutoKeras

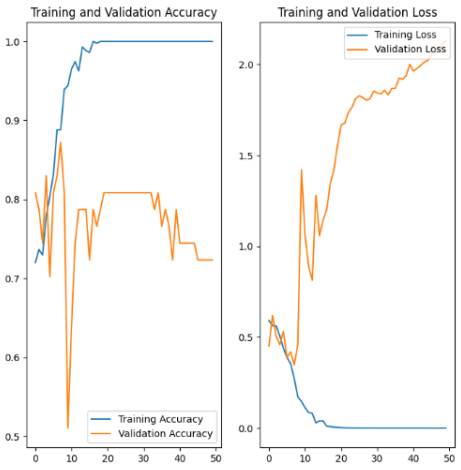
AutoKeras is an open-source library for performing AutoML for deep learning models. The search is performed using so-called Keras models via the TensorFlow tf. keras API.
Automated Machine Learning provides methods and processes to make Machine Learning available for non-Machine Learning experts, to improve efficiency of Machine Learning and to accelerate research on Machine Learning.
AutoKeras was run on Intel Core i7-6700HQ processor, 4 core CPU.
AutoKeras not being available for M1 architecture yet, a lower end computer had to be used, dramatically increasing training times (from minutes to hours), while reaching quite inconclusive results. We found that with the available hardware we could not get AutoKeras to find a good enough model for our use case. Maximum accuracy was under 50%, that after 6 hours of training.

3 My own models

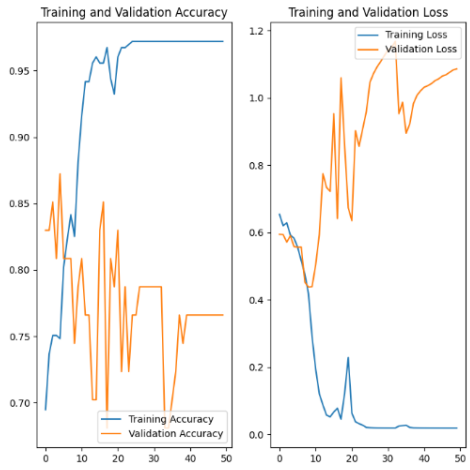
Finding the best augmentation settings: The course of multiple training attempts is presented through this table:

As we can notice from the table and the graphs, it looks like the simple matrix multiplication seems to have the best results, having a power of only 2.

Epochs	Final Validation Accuracy	Epoch with Max Val Acc	Graph	Max Val Acc	Data Augmentation Matrix Multiplication Power
50	0.7234	8	0	0.8723	4
50	0.766	17	1	0.8511	4
50	0.6809	4	2	0.8298	4
50	0.7872	27	-	0.8723	None
50	0.8298	45	3	0.8723	2
50	0.7447	7	4	0.8723	2
50	0.7021	12	5	0.8936	2
50	0.7234	3	6	0.8723	2
50	0.7447	2	7	0.8723	3
50	0.7447	6	8	0.8723	3
50	0.7872	10	9	0.8723	3

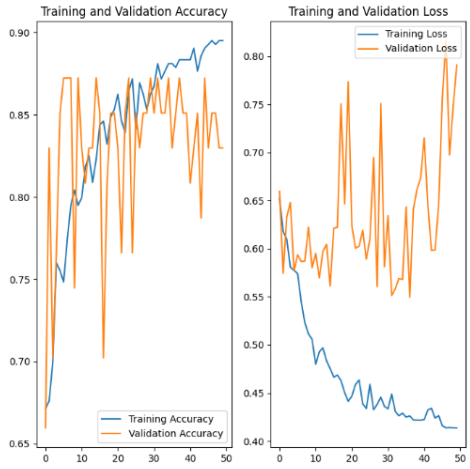


Graph 0

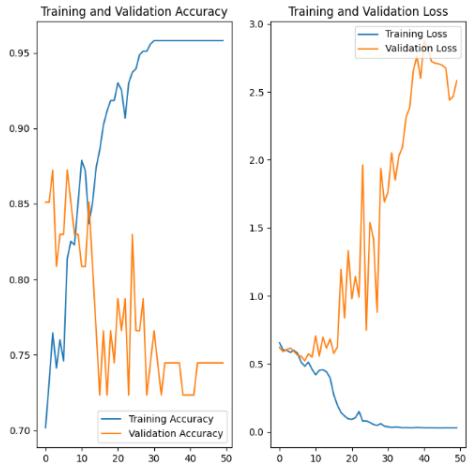


Graph 1

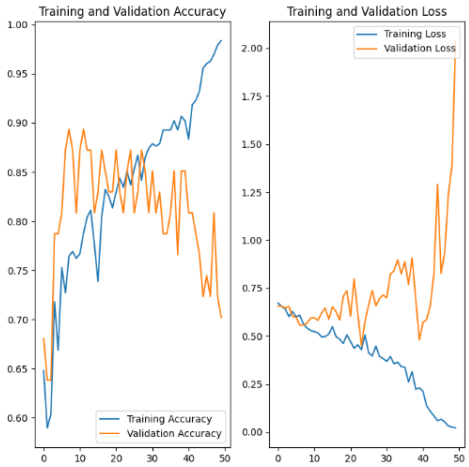
Graph 2



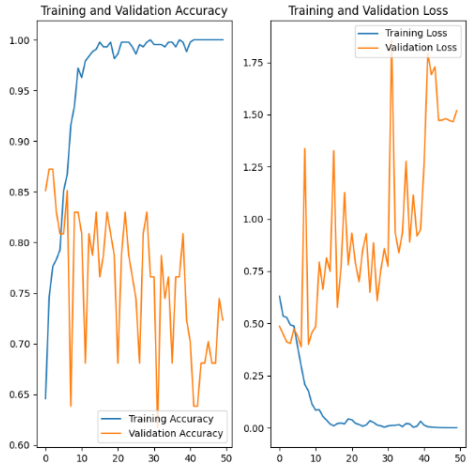
Graph 3



Graph 4

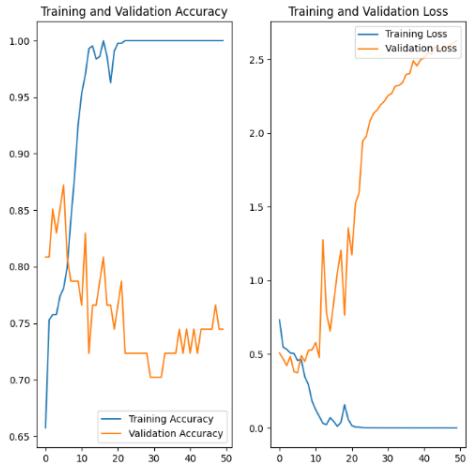


Graph 5

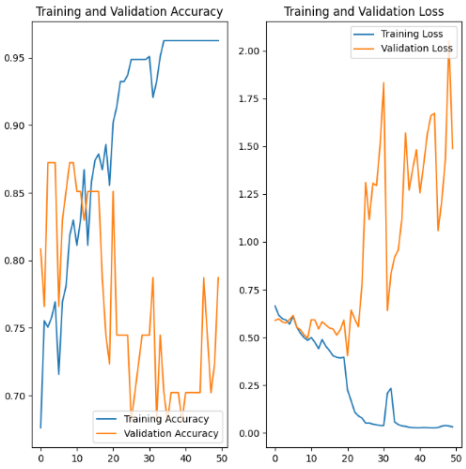


Graph 6

Graph 7

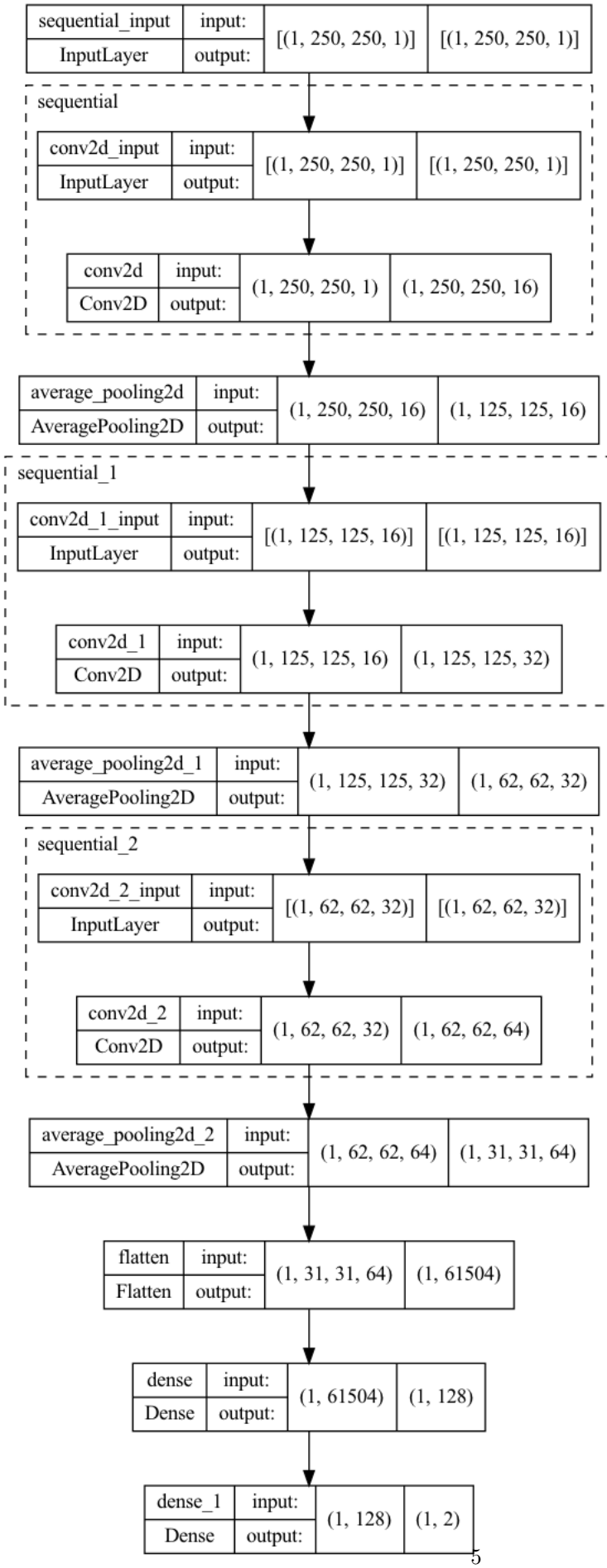


Graph 8



Graph 9

The best model:


sequential_input
InputLayer
input:
output:
[(1, 250, 250, 1)]
[(1, 250, 250, 1)]
sequential
conv2d_input
InputLayer
input:
output:
[(1, 250, 250, 1)]
[(1, 250, 250, 1)]
conv2d
Conv2D
input:
output:
(1, 250, 250, 1)
(1, 250, 250, 16)
average_pooling2d
AveragePooling2D
input:
output:
(1, 250, 250, 16)
(1, 125, 125, 16)
sequential_1
conv2d_1_input
InputLayer
input:
output:
[(1, 125, 125, 16)]
[(1, 125, 125, 16)]
conv2d_1
Conv2D
input:
output:
(1, 125, 125, 16)
(1, 125, 125, 32)
average_pooling2d_1
AveragePooling2D
input:
output:
(1, 125, 125, 32)
(1, 62, 62, 32)
sequential_2
conv2d_2_input
InputLayer
input:
output:
[(1, 62, 62, 32)]
[(1, 62, 62, 32)]
conv2d_2
Conv2D
input:
output:
(1, 62, 62, 32)
(1, 62, 62, 64)
average_pooling2d_2
AveragePooling2D
input:
output:
(1, 62, 62, 64)
(1, 31, 31, 64)
flatten
Flatten
input:
output:
(1, 31, 31, 64)
(1, 61504)
dense
Dense
input:
output:
(1, 61504)
(1, 128)
dense_1
Dense
input:
output:
(1, 128)
(1, 2)

Some other models, based on this one were also tried, but this one had the best results.

Dataset split: 90% of the data was used for training, and 10% for validation. The average accuracy was 73%, the better models and parameters getting an up to 85%, and the worst as low as 54% (though that would be quite rare).