Table of Contents

1 Measurements and results

1.1 Model comparisons

As mentioned in section ?? we used Keras Tuner model to find hyperparameters that would yield the highest accuracy. Instead of hard-coding hyperparameters when building a model with Keras API, we defined a search space of possible values with `HyperParameter` class and used that as a hyperparameter.

We passed the created model to a `RandomSearch` class, with few other parameters such as batch size, number of epochs and maximum number of trials. As we started the hyperparameter search, Keras Tuner started picking a randomly set of hyperparameters, which were used to train a model. This process was repeated for a trial number of times. Used hyperparameters and achieved accuracy on validation set for each trained model were logged in a text file for later use.

After training a number of different models we picked a few and compared them. Comparison of models trained in Edge Impulse Studio was also done.

1.1.1 Hyperparameter search space and results analysis

General structure of CNN model was already described in section ?? and in Figure ??. We decided to search for the following hyperparameters:

- Number of filters in all three convolutional layers (can be different for each layer)
- Size of filters in all three convolutional layers (same for all layers)
- Size of dense layer

- Dropout rate
- Learning rate

Possible values of hyperparameters (also known as hyperparameter search space) are specified in Table 1.1.

Table 1.1: First hyperparameter search space

Hyperparameter	**Set of values**
`FilterNum1`	From 16 to 80, with a step of 8
`FilterNum2`	From 16 to 80, with a step of 8
`FilterNum3`	From 16 to 80, with a step of 8
`FilterSize`	3 x 3 or 3 x 4
`DenseSize`	From 16 to 96, with a step of 8
`DropoutRate`	From 0.2 to 0.5, with a step of 0.05
`LearningRate`	0.0001 or 0.0003
Random search variable	**value**
`EPOCHS`	25
`BATCH_SIZE`	100
`MAX_TRIALS`	300

Search space of `FilterNumX`, `DenseSize` and `DropoutRate` hyperparameters was chosen based on initial training tests conducted on thermal image dataset and other various models that were trained on similar data. Value of `FilterSize` is usually 3 x 3, however most of example ML projects that we could find on the Internet were training on image data of the same dimensions. We wanted to test how would a filter with same ratio of dimensions as image data (3 x 4 and 60 x 80 respectively) perform. Hyperparameter `learning_rate` was chosen heuristically, we saw that 10 times higher values, such as 0.001 or 0.003, would leave model's accuracy stuck at suboptimal optima, from where it could not be improve anymore.

We also had to set 3 variables that directly affected how long will random search last. From initial tests we saw that models usually reached maximum possible accuracy around 20th epoch, to give some headroom we set the number of epochs to 25. We kept batch size relatively small, at 100, which meant that weights would get updated regularly. Hyperparameter `MAX_TRIALS` had the biggest impact on the training time, we set it to 300.

Training lasted for about 12 hours. After it was done we compiled a list of all 300 trained models and their different hyperparameter values, number of parameters and achieved accuracies Part of it can be seen in Table 1.2.

Table 1.2: Partial results of first random search of hyperparameters

Model ID	FilterNum1	FilterNum2	FilterNum3	DenseSize	DropoutRate	FilterSize	LearningRate	Number of parameters	Accuracy[%]
0a	72	80	64	72	0.4	3x4	0.0003	1,514,400	98.35
1a	32	40	72	56	0.35	3x4	0.0001	1,260,332	98.31
2a	40	48	32	64	0.35	3x4	0.0001	656,797	98.31
3a	56	16	48	72	0.4	3x4	0.0001	1,057,924	98.28
4a	80	64	40	96	0.45	3x4	0.0003	1,245,788	98.28
96a	16	32	72	80	0.25	3x4	0.0001	1,762,508	98.00
97a	72	56	40	56	0.45	3x4	0.0003	748,580	98.00
98a	32	24	24	48	0.35	3x3	0.0001	358,308	98.00
99a	48	16	40	40	0.45	3x3	0.0003	493,412	98.00
100a	24	72	64	40	0.45	3x3	0.0003	844,684	98.00
191a	64	56	16	52	0.4	3x3	0.0001	386,996	97.76
192a	48	40	24	24	0.4	3x4	0.0001	208,172	97.73
193a	56	64	72	24	0.25	3x4	0.0003	617,692	97.73
194a	48	72	48	32	0.25	3x4	0.0003	544,652	97.73
295a	48	32	64	16	0.5	3x4	0.0001	351,012	95.87
296a	40	24	56	24	0.5	3x4	0.0001	431,572	95.77
297a	56	16	80	16	0.2	3x4	0.0001	411,020	95.63
298a	24	16	48	24	0.5	3x4	0.0001	359,924	94.46
299a	40	48	56	16	0.35	3x3	0.0003	310,860	82.86

After analyzing results we came to several conclusions:

1. We saw that almost all trained models, except of the last one, achieved accuracy above 90 %. This proved that the general architecture of the model was appropriate for the problem.

2. We could not see any visible correlation between a specific choice of a certain

hyperparameter and accuracy. This showed that selection of hyperparameters is really a non-heuristic task, at least for our particular problem.

3. Filter of size 3 x 4 did not perform significantly better compared to one with size 3 x 3.

4. There is a weak correlation between number of parameters (model's complexity) and accuracy, however, models with small size and great accuracy exist, model *2a* is example of that.

5. First 200 models cover an accuracy range of 0.62 %. However inside of this range model number of parameters varies hugely, for example, model *192a* has more than 8 times less parameters than model than model *96a*, although the difference in accuracy (0.27 %) is negligible.

It is apparent from results that large models are not necessary to achieve high accuracy on our training data, so we decided to run the random search of hyperparameters again.

This time we lowered the maximum and the minimum numbers of filters and size of the dense layer. We decreased all steps from 8 to 2, thus increasing the number of possible configurations. We decided to lower the bottom boundary of `DropoutRate` from 0.2 to 0.0, which means that some models will not be using dropout layer at all. We expected that training without dropout layer would produce suboptimal results, however we wanted to test it. Redefined search space for second random search can be seen in Table 1.3 We increased the number of `MAX_TRIALS` from 300 to 500, as we were expecting that more models will end up underfitting and also because there would be more possible options because of smaller step size. Partial table of results of random hyperparameter search can be seen in Table 1.4.

Table 1.3: Second hyperparameter search space

Hyperparameter	**Set of values**
`FilterNum1`	From 4 to 48, with a step of 2
`FilterNum2`	From 4 to 48, with a step of 2
`FilterNum3`	From 4 to 48, with a step of 2
`FilterSize`	3 x 3 or 3 x 4
`DenseSize`	From 4 to 48, with a step of 2
`DropoutRate`	From 0.0 to 0.5, with a step of 0.05
`LearningRate`	0.0001 or 0.0003
Random search variable	**value**
`EPOCHS`	25
`BATCH_SIZE`	100
`MAX_TRIALS`	500

Table 1.4: Partial results of second random search of hyperparameters

Model ID	FilterNum1	FilterNum2	FilterNum3	DenseSize	DropoutRate	FilterSize	LearningRate	Number of parameters	Accuracy[%]
0b	40	20	20	48	0.25	3x4	0.0001	304,216	98.14
1b	44	10	28	42	0.2	3x4	0.0003	362,264	98.14
2b	18	38	26	38	0.1	3x4	0.0003	316,956	98.11
95b	20	16	34	40	0.3	3x3	0.0003	416,230	97.62
96b	46	42	28	32	0.4	3x3	0.0003	297,466	97.62
97b	30	26	30	34	0.2	3x3	0.0001	320,570	97.59
195b	28	16	40	24	0.1	3x3	0.0001	298,252	97.31
196b	44	30	32	20	0.3	3x4	0.0003	220,098	97.31
197b	46	40	10	40	0.1	3x3	0.0001	140,874	97.31
295b	20	8	34	26	0.3	3x3	0.0003	269,464	96.90
296b	18	16	10	20	0.3	3x4	0.0003	65,740	96.87
297b	8	22	28	16	0.1	3x3	0.0001	141,742	96.87
395b	10	20	12	30	0.0	3x3	0.0001	112,246	96.87
396b	24	24	46	18	0.2	3x3	0.0003	263,924	96.14
397b	6	18	12	24	0.4	3x4	0.0001	90,520	96.11
497b	42	30	22	6	0.4	3x3	0.0003	57,386	82.86
498b	4	4	20	12	0.4	3x3	0.0003	72,992	82.86
499b	32	36	36	4	0.15	3x3	0.0001	65,648	82.86

Some observations:

1. We can see that the accuracy of the best model *0b* compared to the best model *0a* from previous search is only 0.21 % lower, although it has about 5 times less parameters.

2. Although that it might seem that `FilterSize` of 3 x 4 yields best results, we did not saw a strong tendency towards 3 x 3 or 3 x 4 filter size after manually analyzing best 30 models.

3. We can see that the worst three models have the same accuracy of 82.86 %, same as the worst performing model from first random search. There are 82.86 % images of elephants in validation class, which means that model probably assigned all validation images to elephant class and was satisfied with achieved accuracy.

4. We can see that the model *296b* has a quite low number of parameters, only 65,740 when compared to it neighbours.

1.1.2 Comparison of selected, re-trained models

Two random searches gave us a large amount of different models to choose from. In every other ML application where the execution time would not be a constraint, we could simply take the best performing model and be done with it. In our case we had to make a trade off between model's accuracy and execution speed.

For comparison and later on device performance testing we decided to pick and retrain[1]6 models: *0a*, *2a*, *0b*, *172b*, *338b* and *460b*, their properties are listed in Table 1.5.

Chosen models vary greatly in number of parameters. Models *0a*, *2a*, *0b* have high number of parameters but their accuracy is high. Models *172b*, *338b* and *460b* were chosen because of their small size and reasonably good accuracy.

[1]Retraining was required as Keras Tuner module only saved hyperparameter settings during search and not each trained model. As the weights are initially randomized, accuracy of retrained models is going to be similar but not exact when compared to the accuracy returned by random search.

Table 1.5: Selected models

Model ID	FilterNum1	FilterNum2	FilterNum3	DenseSize	DropoutRate	FilterSize	LearningRate	Number of parameters	Accuracy[%]
Model ID									
0a	72	80	64	72	0.4	3x4	0.0003	1,514,400	98.35
2a	40	48	32	64	0.35	3x4	0.0001	656,797	98.31
0b	40	20	20	48	0.25	3x4	0.0001	304,216	98.14
172b	42	44	8	14	0.1	3x4	0.0001	60,672	97.38
338b	4	18	6	10	0.05	3x4	0.0003	20,290	96.63
460b	6	28	4	8	0.1	3x4	0.0003	13,114	93.60

As we are dealing with imbalanced dataset, where 82.86 % of our validation data consists of elephant images, accuracy is not the best metric to use when comparing models. Simply classifying all images into elephant class would yield 82.86 % accuracy, which sounds high, although it would not solve our problem.

When analyzing performance of a model on a imbalanced dataset is more appropriate to use precision and recall metrics[2]. They can give us a better idea how well the model is performing on specific classes. Calculated metrics can bee seen in Table 1.6, we abbreviated precision to PR and recall to RE for brevity.

30, 222, 27

As we can see all six models are generally classifying elephants correctly, both precision and recall of elephant class are high, above 97 %, which is important. Precision and recall values of other classes are generally lower, especially for nature/random. We can see that top three models *0a*, *2a* and *0b* are quite similar in terms of precision and recall, which means that we can easily prefer *0b*, without sacrificing accuracy. Models *172b* and *338b* perform a bit worse when compared to top three models, however they have low number of parameters which should translate to lower inference time. Last model, *460b*, performs the worst and it should generally not be used.

[2]Precision tells us what percentage of data points in a specific predicted class actually fall into that class. Recall tells us what percentage of data points inside a certain class were actually predicted correctly [1].

Table 1.6: Second hyperparameter search space

Model ID	0a	2a	0b	172b	338b	460b
Metrics						
accuracy[%]	[redacted]	98.04	98.04	96.80	96.28	93.4
number of parameters	1,515,404	656,797	304,216	60,672	20,290	13,114
PR of elephant class[%]	99.22	99.46	99.25	99.29	98.80	97.80
PR of human class[%]	96.92	95.38	95.38	92.00	91.69	80.31
PR of cow class[%]	90.99	93.69	90.09	84.68	75.68	69.37
PR of nature/random class[%]	77.42	64.52	79.03	46.77	59.68	40.32
RE of elephant class[%]	99.29	98.80	98.84	97.87	98.43	97.39
RE of human class[%]	93.20	94.51	95.09	91.44	89.22	85.57
RE of cow class[%]	94.39	92.04	96.15	89.52	84.00	81.91
RE of nature/random class[%]	87.27	97.56	84.48	93.55	67.27	28.09

Another way to compare models performance is to look at confusion matrix. Figure 1.1 shows comparison between confusion matrices of *0a* model on the left and *460b* model on the right. In case of *0a* 19 elephant images were not classified correctly, and 17 images were wrongly classified as elephants. This is not ideal, however is much better compared to performance of *460b*, where 53 elephants were wrongly classified and 63 of images classified as elephants were not actually elephants.

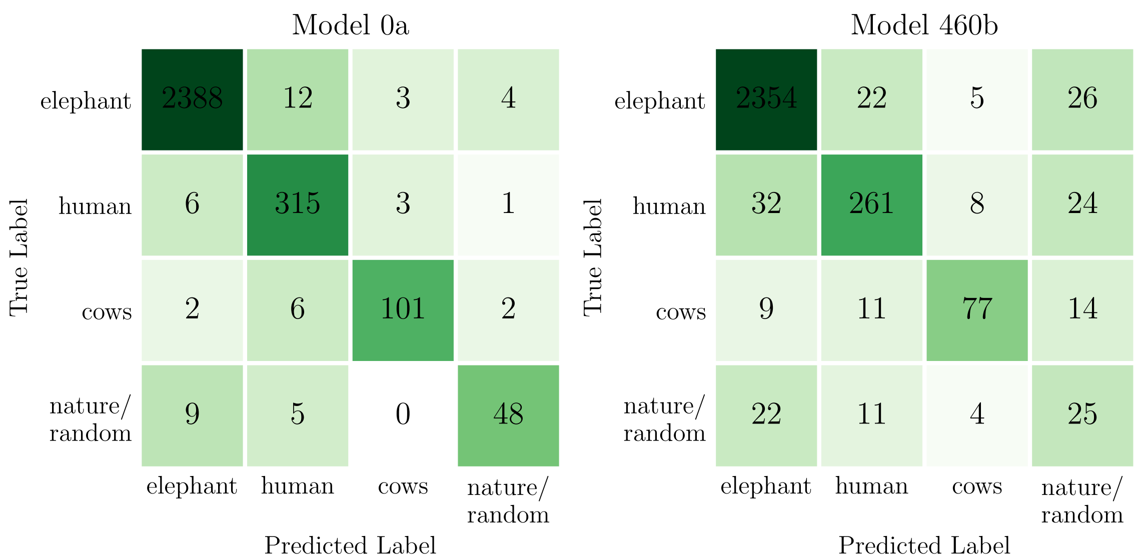

Figure 1.1: Confusion matrices of *0a* model (left) and *460b* model (right).

1.1.3 Comparison with Edge Impulse models

We wanted to compare 6 models that we selected from random searches against 6 models created in Edge Impulse Studio, using the same hyperparameters. However, at the time of writing this thesis Edge Impulse supported only model training on images of same dimensions. Images with different dimensions could either be cropped or scaled to fit 60 x 60 ratio. Using the same hyperparameters in Edge Impulse Studio, that we used in our model, would always create a model with smaller number of parameters. Smaller image creates a smaller network when compared to a bigger image if architecture does not change. That mean that we could not make a direct comparison with our hyperparameters.

We could not also perform random search of hyperparameters in Edge Impulse, as this feature was not fully supported at the time of writing this thesis.

We decided to train a few differently sized models, using the same general CNN architecture as before. We also trained a few models with Transfer Learning technique.

Epoch number was 15 to avoid long training times. Edge impulse model data

Table 1.7: Selected models

Model ID	FilterNum1	FilterNum2	FilterNum3	DenseSize	DropoutRate	FilterSize	LearningRate	Number of parameters	Accuracy[%]
Model ID									
0e	72	80	64	72	0.4	3x4	0.0003	1,168,804	95.8
1e	40	48	32	64	0.35	3x4	0.0001	503,196	94.8
2e	64	48	40	48	0.35	3x4	0.0001	493,068	96.0
3e	N/A	N/A	N/A	16	N/A	N/A	0.000045	430,676	98.5
4e	40	20	20	48	0.25	3x4	0.0001	231,204	95.0
5e	42	44	8	14	0.1	3x4	0.0003	52,272	96.6

Precision and recall table

Choose one, or few model configurations and make a model in EI.

Table 1.8: Second hyperparameter search space

Model ID	0a	2a	0b	172b	338b	460b
Metrics						
accuracy[%]	98.18	98.04	98.04	96.80	96.28	93.4
number of parameters	1,515,404	656,797	304,216	60,672	20,290	13,114
PR of elephant class[%]	99.22	99.46	99.25	99.29	98.80	97.80
PR of human class[%]	96.92	95.38	95.38	92.00	91.69	80.31
PR of cow class[%]	90.99	93.69	90.09	84.68	75.68	69.37
PR of nature/random class[%]	77.42	64.52	79.03	46.77	59.68	40.32
RE of elephant class[%]	99.29	98.80	98.84	97.87	98.43	97.39
RE of human class[%]	93.20	94.51	95.09	91.44	89.22	85.57
RE of cow class[%]	94.39	92.04	96.15	89.52	84.00	81.91
RE of nature/random class[%]	87.27	97.56	84.48	93.55	67.27	28.09

1.2 On device performance testing

Describe setup

To profile execution of our code we first wrote a timer driver based on a Arm's systick timer, however we later decided to use data watch trigger (DWT). DWT does not use interrupts, therefore it does not introduce overhead of calling interrupt routines like systick timer does.

how are you timing,

1.2.1 Comparison of different optimisation options

0b —- 0s 11295 ms 03 4117 ms CMSIS-NN 1023 ms I and D Cache, Flash prefetch and ART 228ms

1.2.2 Comparison of performance of selected models

Transfer learning could be interesting here, bigger number of parameters and takes less time.

1.3 Power profiling of an embedded early warning system

To write this section you need to: - wirte uart communication channel for zephyr and your system, has to be simple dont lose time on this. - Basic parsing of results and packing them into lora payload - Zephyr, put system to sleep and wake it up with pir, turn on fet. - Make an image with flir and do inference on it, (subtract mean image, you could test this on real dataset, without subtracted mean)

- [ ] Power consumption test of whole setup, PIR wakes up wisent, wisent turns on stm32f7 and flir, which makes a picture, does inference, reports result and wisent sends the result. Otii image of consumption with marked sections.(shouldnt be hard)

1.3.1 Battery life estimations

Based on numbers and different scenarios estimate how long would this last with different batteries.

Bibliography

[1] Geron, A. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems, 2nd edition.* O'Reilly Media, Sebastopol, CA, 2019.