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 then passed the created model to a `RandomSearch` class, with few other parameters such as batch size, number of epochs and maximum number of trials. We then started hyperparameter search, which means that Keras Tuner was randomly picking a set of hyperparameters and training a model with them. This process was repeated for a trial number of times. Accuracy and hyperparameters that were used while training every module were saved to a log file for later analysis.

After training a number of different models we handpicked a few of them and compared them. Comparison of equivalent model 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 following hyperparameters: number of filters in all three convolutional layers (can be different for each layer), filter size in all three convolutional layers (same for all layers), size of dense layer, dropout rate and learning rate. Possible values of hyperparameters (also known as hyperparameter search space) are specified in table 1.1.

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

Table 1.1: First hyperparameter search space

Search space of `filter_numX`, `dense_size` and `dropout_rate` hyperparameters was chosen based on initial training tests and various models that were trained on similar data. Value of `filter_size` is usually 3 x 3, however all example ML projects were training on image date of 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 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 accuracies, part of it can be seen in Table 1.2.

After analyzing results we came to several conclusions:

Hyperparameter	FilterNum1	FilterNum2	FilterNum3	DenseSize	DropoutRate	FilterSize	LearningRate	Number of parameters	Accuracy[%]
Model ID									
0	72	80	64	72	0.4	3x4	0.0003	1.514,400	98.35
1	32	40	72	56	0.35	3x4	0.0001	1.260,332	98.31
2	40	48	32	64	0.35	3x4	0.0001	656,797	98.31
3	56	16	48	72	0.4	3x4	0.0001	1,057,924	98.28
4	80	64	40	96	0.45	3x4	0.0003	1,245,788	98.28
5	64	24	72	88	0.45	3x3	0.0001	1,931,356	98.28
6	64	56	40	80	0.35	3x3	0.0001	1,013,556	98.24
7	16	40	64	88	0.35	3x3	0.0003	1,719,108	98.24
8	48	64	32	64	0.35	3x4	0.0003	676,884	98.24
9	72	48	56	80	0.3	3x4	0.0001	1,419,172	98.24
91	72	48	56	40	0.35	3x3	0.0003	728,324	98.00
92	48	24	64	64	0.35	3x4	0.0001	1,262,092	98.00
93	24	48	32	72	0.4	3x3	0.0001	716,076	98.00
94	32	48	72	32	0.25	3x3	0.0001	736,732	98.00
95	64	24	64	48	0.3	3x4	0.0001	959,628	98.00
96	16	32	72	80	0.25	3x4	0.0001	1,762,508	98.00
97	72	56	40	56	0.45	3x4	0.0003	748,580	98.00
98	32	24	24	48	0.35	3x3	0.0001	358,308	98.00
99	48	16	40	40	0.45	3x3	0.0003	493,412	98.00
100	24	72	64	40	0.45	3x3	0.0003	844,684	98.00
191	64	56	16	52	0.4	3x3	0.0001	386,996	97.76
192	48	40	24	24	0.4	3x4	0.0001	208,172	97.73
193	56	64	72	24	0.25	3x4	0.0003	617,692	97.73
194	48	72	48	32	0.25	3x4	0.0003	544,652	97.73
195	72	56	24	56	0.25	3x4	0.0003	469,012	97.73
196	72	48	72	40	0.3	3x3	0.0003	927,252	97.73
197	80	16	32	80	0.25	3x3	0.0001	785,380	97.73
198	56	24	16	88	0.25	3x3	0.0001	438,996	97.73
199	56	24	16	88	0.25	3x3	0.0001	438,996	97.73
295	48	32	64	16	0.5	3x4	0.0001	351,012	95.87
296	40	24	56	24	0.5	3x4	0.0001	431,572	95.77
297	56	16	80	16	0.2	3x4	0.0001	411,020	95.63
298	24	16	48	24	0.5	3x4	0.0001	359,924	94.46
299	40	48	56	16	0.35	3x3	0.0003	310,860	82.86

Table 1.2: Partial results of first random search of hyperparameters

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 shows that selection of hyperparameters is really a non-heuristic task.

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. Although eight of top ten models have more then 1 million parameters, models with IDs 2 and 8 have almost half of the parameters, but still perform well.

5. First 200 models cover a accuracy range of 0.62 %. However we can there models of very different size, for example model with ID 192 has more than 8 times less of parameters than model than model with ID 96, altough the difference in accuracy (0.27 %) is neglible.

As we realized that we do not need complex models to achieve high accuracy on our training data, we decided to run the random search of hyperparameters again. We decided to lower the maximum and minimum numbers of filters and size of dense layer. We also decreased the step from 8 to 2. We decided to lower the bottom boundry of `DropoutRate` from 0.2 to 0.0, which means that some models will not be using dropoout layer at all. We expect that training without dropout layer will produce suboptimal results, however we would like to test this. Redefined search space for second radnom search can be seen in Table 1.3 We increased the number of `MAX_TRIALS` from 300 to 500, as we are expecting that more models will end up underfitting and also because there will be more possible options because of smaller step size.

By providing accuracy metric, Keras tuner automatically sorts trained models in descending accuracy. This is in many use cases sufficient, however in our case, we

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 1.3: Second hyperparameter search space

were also interested in best performing models that had small size.

- [ ] Do model comparisons, analyze them and present them. This means that you need to describe the setup, what you want to get from this subchapter, hyperparameter search space, list of models,

1.1.2 Comparison of selected, re-trained models

how you will select them, an image of history (accu and loss), confusion matrix, precision and recall

1.1.3 Comparison with Edge Impulse model

Describe how did you build Edge Impulse model, maybe mention transfer learning (might be extra work, you did not write about theory)

1.2 On device performance testing

Describe setup how are you timing,

1.2.1 Comparison of different optimization options

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

- [ ] 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