SNR classes project - birds species recognition using deep neural networks - second stage report

Michał Sypetkowski, Marcin Lew

June 13, 2018

1 General information

Git repository:
`https://github.com/msypetkowski/SNR-proj.git`.
Previous stage report:
`https://github.com/msypetkowski/SNR-proj/blob/master/doc/main.pdf`.

In all experiments we use Adam optimizer and we exponentially decrease learning rate by 50% for 1000 iterations. We use batch size of 128 and initial learning rate of 0.03.

This document is a sequence of sections corresponding to various experiments. Accuracy curves for the model for current experiment and for the best model from previous experiments, are always shown in a graph (if there are accuracy curves at all). An exception is section 2 where only current results are shown, because there was no preceding experiments.

2 Establishing Layers count

First, we experimented with layers count. We trained 3 different models (they have 11, 8 and 6 layers respectively). Detailed layers descriptions for each model are shown in tables 1, 2 and 3 respectively. Rounded total trainable parameters count is respectively: 2.0M, 1.7M and 1.5M.

Batch normalization with relu activation layers are used in all these models, where the horizontal lines occur in the tables.

2.1 Results

We used cross entropy as a loss function. Accuracy curves are shown in figure 1.

Training accuracy increases faster for models with fewer layers. Smaller networks naturally learn faster, but they have fewer degrees of freedom and may get worse results in the end. 11-convolutional-layers network architecture shown to be too complicated for our dataset and/or training method. Best accuracy was achieved by medium – 8-convolutional-layers network. We used this architecture for further experiments.

Table 1: 11-convolutional-layers convolutional NN architecture.

Layer	kernel/window	strides	output shape
Conv1	(5, 5)	(1, 1)	224x224x64
MaxPool1	(2, 2)	(2, 2)	112x112x64
Conv2	(5, 5)	(1, 1)	112x112x64
MaxPool2	(2, 2)	(2, 2)	56x56x64
Conv3	(5, 5)	(1, 1)	56x56x64
MaxPool3	(2, 2)	(2, 2)	28x28x64
Conv4	(5, 5)	(1, 1)	28x28x64
MaxPool4	(2, 2)	(2, 2)	14x14x64
Conv5	(5, 5)	(1, 1)	14x14x64
MaxPool5	(2, 2)	(1, 1)	14x14x64
Conv6	(5, 5)	(1, 1)	14x14x64
MaxPool6	(2, 2)	(2, 2)	7x7x64
Conv7	(5, 5)	(1, 1)	7x7x64
MaxPool7	(2, 2)	(1, 1)	7x7x64
Conv8	(3, 3)	(1, 1)	7x7x128
MaxPool8	(2, 2)	(2, 2)	4x4x128
Conv9	(2, 2)	(1, 1)	4x4x128)
MaxPool9	(2, 2)	(1, 1)	4x4x128
Conv10	(2, 2)	(1, 1)	4x4x128)
MaxPool10	(2, 2)	(1, 1)	4x4x128
Conv11	(3, 2)	(1, 1)	4x4x128)
MaxPool11	(2, 2)	(1, 1)	4x4x128
Flatten	-	-	2048
Dense1	-	-	512
Output	-	-	50
Softmax	-	-	50

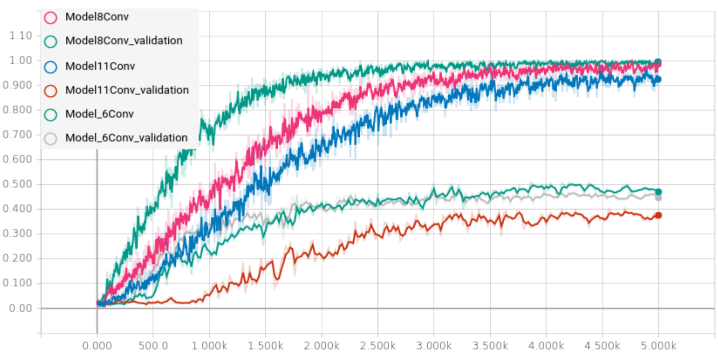

Figure 1: Accuracy curves for models with different layers count.

3 Experiments with size of the last convolutional layer

Our 8-layer network from section 2, outputs feature vectors of size 2048 from the last convolutional layer (after flattening, see Conv11 in table 2). We tried models with 1024 and 4096 sizes of these vectors. The results are shown in figure 2. These dimensions significantly affect total trainable parameters count (around 1.1M in case of 1024 dimensions and around 2.8M in case of 4096 dimensions).

Decreasing or increasing this parameter caused only loss in accuracy (models had in the end not enough or too many parameters for our dataset).

Table 2: 8-convolutional-layers convolutional NN architecture (layers names correspond to some layers names in 11-convolutional-layers architecture 1).

Layer	kernel/window	strides	output shape
Conv1	(5, 5)	(1, 1)	224x224x64
MaxPool1	(2, 2)	(2, 2)	112x112x64
Conv2	(5, 5)	(1, 1)	112x112x64
MaxPool2	(2, 2)	(2, 2)	56x56x64
Conv3	(5, 5)	(1, 1)	56x56x64
MaxPool3	(2, 2)	(2, 2)	28x28x64
Conv4	(5, 5)	(1, 1)	28x28x64
MaxPool4	(2, 2)	(2, 2)	14x14x64
Conv6	(5, 5)	(1, 1)	14x14x64
MaxPool6	(2, 2)	(2, 2)	7x7x64
Conv8	(3, 3)	(1, 1)	7x7x128
MaxPool8	(2, 2)	(2, 2)	4x4x128
Conv9	(2, 2)	(1, 1)	4x4x128)
MaxPool9	(2, 2)	(1, 1)	4x4x128
Conv11	(3, 2)	(1, 1)	4x4x128)
MaxPool11	(2, 2)	(1, 1)	4x4x128
Flatten	-	-	2048
Dense1	-	-	512
Output	-	-	50
Softmax	-	-	50

Table 3: 6-convolutional-layers convolutional NN architecture (layers names correspond to some layers names in 11-convolutional-layers architecture 1).

Layer	kernel/window	strides	output shape
Conv1	(5, 5)	(1, 1)	224x224x64
MaxPool1	(2, 2)	(2, 2)	112x112x64
Conv2	(5, 5)	(1, 1)	112x112x64
MaxPool2	(2, 2)	(2, 2)	56x56x64
Conv3	(5, 5)	(1, 1)	56x56x64
MaxPool3	(4, 4)	(4, 4)	28x28x64
Conv6	(5, 5)	(1, 1)	14x14x64
MaxPool6	(2, 2)	(2, 2)	7x7x64
Conv8	(3, 3)	(1, 1)	7x7x128
MaxPool8	(2, 2)	(2, 2)	4x4x128
Conv11	(3, 2)	(1, 1)	4x4x128)
MaxPool11	(2, 2)	(1, 1)	4x4x128
Flatten	-	-	2048
Dense1	-	-	512
Output	-	-	50
Softmax	-	-	50

4 Experiments with activation function

We tested sigmoid activation function using 8-convolutional-layers network from section 2. We trained the sigmoid model variant for 5000 iterations longer.

As we can see in figure 3, experimental model learns significantly slower, but achieves only slightly worse result in the end.

Training accuracy grows relatively fast – test accuracy starts giving better answers than random choice (above 2%), when the training accuracy is above 90%. Experimental model learns more memory-like rules for first 3k iterations. When

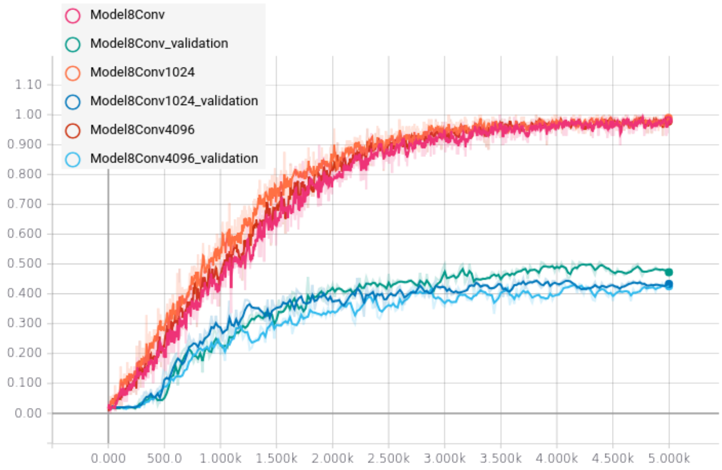

Figure 2: Accuracy curves for models with different last convolutional layer size.

this strategy achieves its limit, the generic rules emerge. In the end, validation accuracy has sigmoid-like shape.

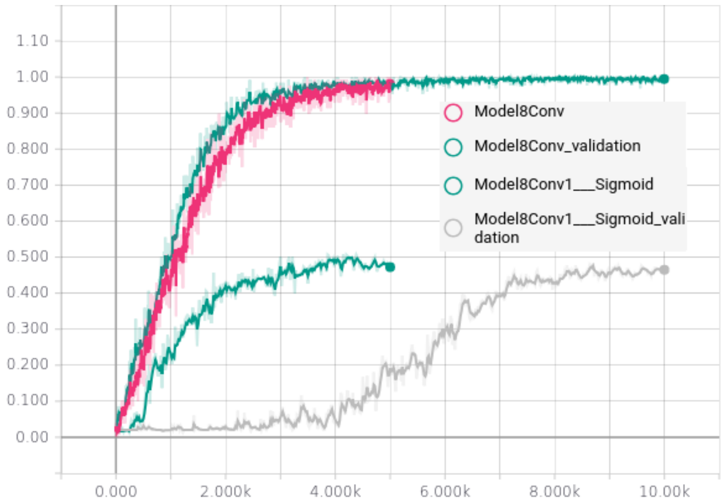

Figure 3: Accuracy curves for model with sigmoid activation function.

5 Experiments with loss function

We tested MSE (Mean Squared Error) as activation function instead of cross entropy. Nowadays, it is a common knowledge that using cross entropy in classification models works much better than instead of MSE.

In our experiment, MSE model accuracy is worse during the whole training. Accuracy and training curves are shown in figure 4.

6 Experiments with SVMs

We experimented with using features from batch normalization layer after last dense layer and last convolutional layer (to be exact – from flattening layer). For training each SVM, we used 20k feature vectors. We used various kernels in this experiment. Last dense layer had 512 output features and last convolutional layer had 2048. Results of this experiments are shown in table 4.

Most surprising fact is that linear kernel after last convolutional gives better results than variant without SVM. In

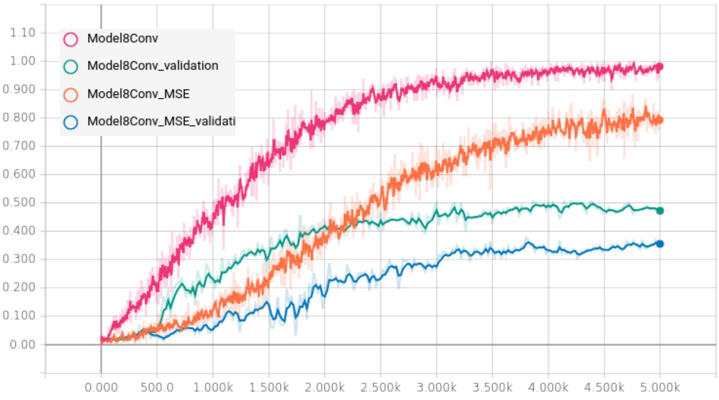

Figure 4: Accuracy curves for model with MSE loss function.

cases, where we trained SVMs after last convolutional layers we used in fact a model with significantly fewer parameters (we had around 0.7M instead of 1.7M). Inspired by this observation we did experiment described in section 7.

Table 4: Accuracy for various SVMs usages.

method	accuracy
noSVM	46.3%
RBF after last dense	49.3%
RBF after last convolutional	49.6%
linear after last dense	47.6%
linear after last convolutional	48.0%
polynomial (degree=3) after last dense	49.6%
polynomial (degree=3) after last convolutional	51.3%
sigmoid after last dense	46.0%
sigmoid after last convolutional	37.0%

7 Experiment with removing dense layer

Inspired by results achieved in previous section, we trained model without last dense layer. New model had 0.7M trainable parameters instead of 1.7M. Accuracy curves are shown in figure 5, and SVM experiments results – in table 5.

As we can see, training SVMs after flattened last convolutional layer output gives only worse results in this case.

This experiment showed that sometimes training model with a certain amount of trainable parameters, then removing more than a half of them (in our case decreasing from 1.7M to 0.7M) and adding SVM, may actually give significantly better results than training model with lower parameters count from the beginning and then adding SVM.

Table 5: Accuracy for various SVMs usages (on model without last dense layer).

method	accuracy
noSVM	40.0%
RBF after last convolutional	39.6%
polynomial (degree=3) after last convolutional	6%

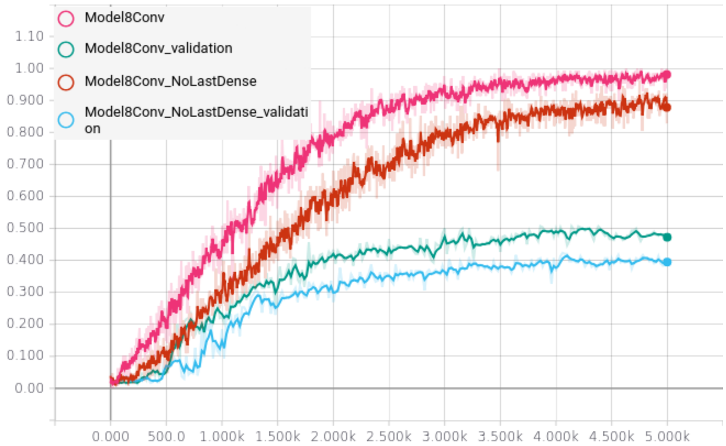

Figure 5: Accuracy curves for 8-convolutional-convolutional-layers model without dense layer.

8 Summary

In this stage we did six experiments. Each experiment concerns a particular element of neural network architecture. The results are summarized in table 6. In the end, best accuracy (around 51%) is achieved by model with:

- 8 convolutional layers
- relu activation function
- cross entropy loss function
- SVM with polynomial kernel (degree=3) after last convolutional layer

Table 6: Accuracies for various models from experiments.

model	experiment parameters	accuracy
Multilayer Perceptron	sigmoid activation function	21%
	relu activation function	24%
VGG16		67%
Own Convolutional Neural Network	6 layers	45%
	8 layers	47%
	11 layers	38%
	1024 output size	44%
	2048 output size	47%
	4096 output size	43%
	relu activation function	47%
	sigmoid activation function	46%
	MSE loss function	36%
	cross entropy	47%
SVM	noSVM	46.3%
	RBF after last dense	49.3%
	RBF after last convolutional	49.6%
	linear after last dense	47.6%
	linear after last convolutional	48.0%
	polynomial (degree=3) after last dense	49.6%
	polynomial (degree=3) after last convolutional	51.3%
	sigmoid after last dense	46.0%
	sigmoid after last convolutional	37.0%
SVM without last dense layer	noSVM	40.0%
	RBF after last convolutional	39.6%
	polynomial (degree=3) after last convolutional	6%