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

Michał Sypetkowski, Marcin Lew

May 31, 2018

1 General information

Git repository `https://github.com/msypetkowski/SNR-proj.git`. We use the following tools:

- **Python**[4]
- **OpenCV**[3]
- **NumPy**[2]
- **TensorFlow**[1]

We also use Tensorboard for model and training progress visualization. The project is tested to run in Linux environment.

2 Multilayer Perceptron

2.1 Data

Dataset consists of 50 subsets – types of bird species. Each subset is a set of 60 different pictures. Altogether it gives us a data set of 3000 pictures.

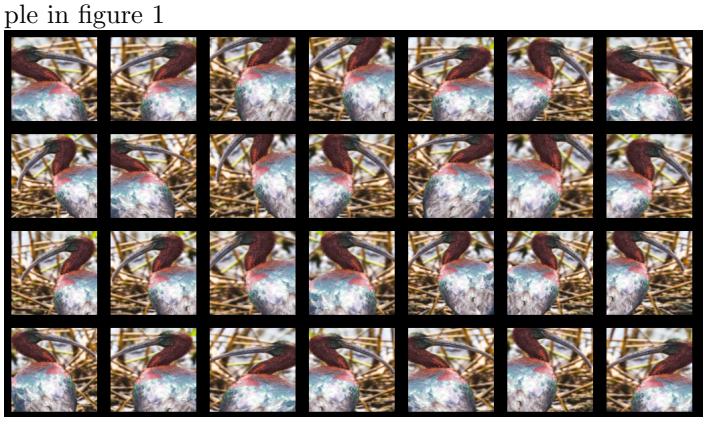
We divided this dataset, so that there are 300 examples in test set and 2700 examples in training set. Number of examples per each class is equal in both training and test set (54 examples per class in training set and 6 in test set). Additionally, training set is augmented during training (see section 2.2).

2.2 Data augmentation

Every training image is randomly rotated, flipped or cropped. Augmentation is done online, so the number of training examples is roughly infinite. We skip the distributions of the transformations in this document. Instead we attach an example visualization. Raw – unaugmented example (center crop) is shown in figure 1. Corresponding augmented examples are shown in figure 2.

Figure 1: Not augmented example

Figure 2: Augmented examples corresponding to raw example in figure 1



2.3 HOG features

To get HOG features, we first resize an image to 128x64x3. Then we calculate the features using OpenCV library:

```
hog = cv2.HOGDescriptor()
fd = hog.compute(img).flatten()
```

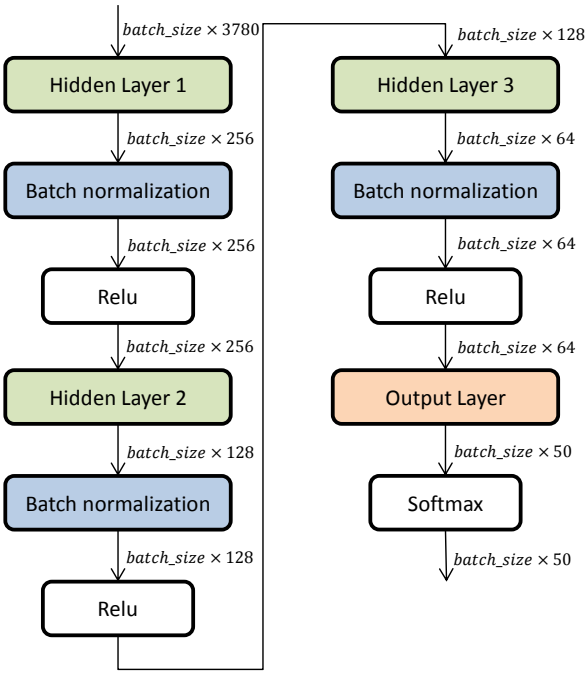
Image is first divided into 8x8 cells (in result, we get 16 * 8 cells). We use gradient orientation bins of size 9. Then the algorithm takes 16x16 blocks (4 cells) – there are 7 * 15 such blocks. Histograms of each of 4 cells in block are concatenated and L2 normalized. In the end, all blocks histograms are concatenated giving final feature vecor of length (7 * 15) * 36 = 3780. We directly feed such vectors into our network.

2.4 Model architecture

We selected the architecture by experimenting. Our network has 3 hidden dense layers of sizes 256, 128 and 64. Since we have small dataset, larger models tended to overfit (despite of data augmentation). Each of these layers have corresponding batch normalization[7] layer. After the last layer there is a softmax function. We use cross entropy as a loss function. We tested 2 models with different activation functions – relu and sigmoid.

Visualization of the model (with relu activation) is shown in figure 3.

Figure 3: Visualization of multilayer perceptron architecture (relu activation variant)



2.5 Training and results

Training and validation curves are shown in figures 4 and 5. For each model, they diverge from the begining because of small dataset (one epoch is around 20 iterations). Because of batch normalization[7] layers, validation accuracy increases along with training accuracy (but slower). Full results on testset for the model using relu are shown in figure 8.

Final accuracy of relu model is around 24%. Sigmoid model achieves slightly lower accuracy (around 21%). Nowadays, it is a common knowledge that using relu as activation function usually gives faster and better results i. a. for classification tasks; hence the result is not surprising.

3 Convolutional neural networks

3.1 "Ground truth" model

We trained for 1500 iterations 8-th layer of VGG16[6] pretrained on ImageNet[5] dataset. Then we finetuned it for 1000 iterations achieving final accuracy of 67%. Training and validation curves are shown in figure 6. Full results on testset shown in figure 7. The ckpt file is downloaded from `http://download.tensorflow.org/models/vgg_16_2016_08_28.tar.gz`.

3.2 Next stage plans

We are planning to experiment with convolutional neural network architectures without using pretrained models. i.a. with:

- convolutional layers count
- loss function
- activation function
- using SVM (with various exponential kernels) after the last network layer

Figure 4: Accuracy graph for a multi-layer perceptron.

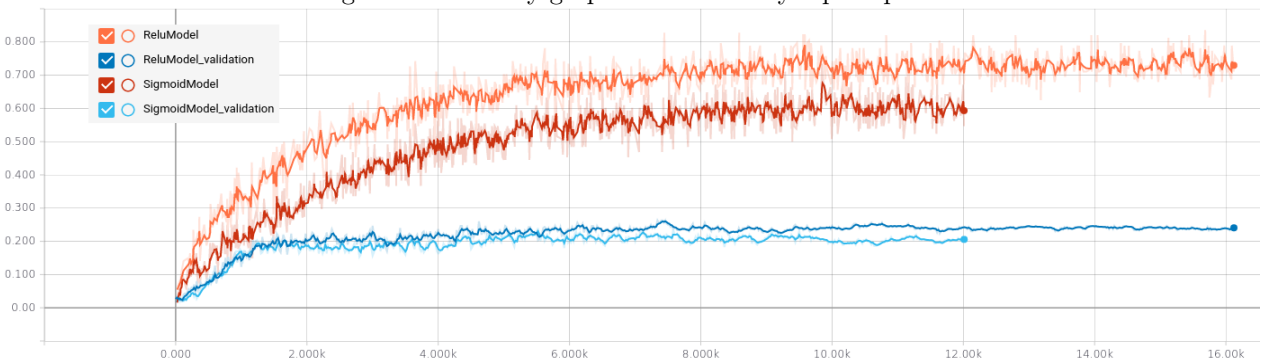


Figure 5: Loss graph for a multi-layer perceptron.

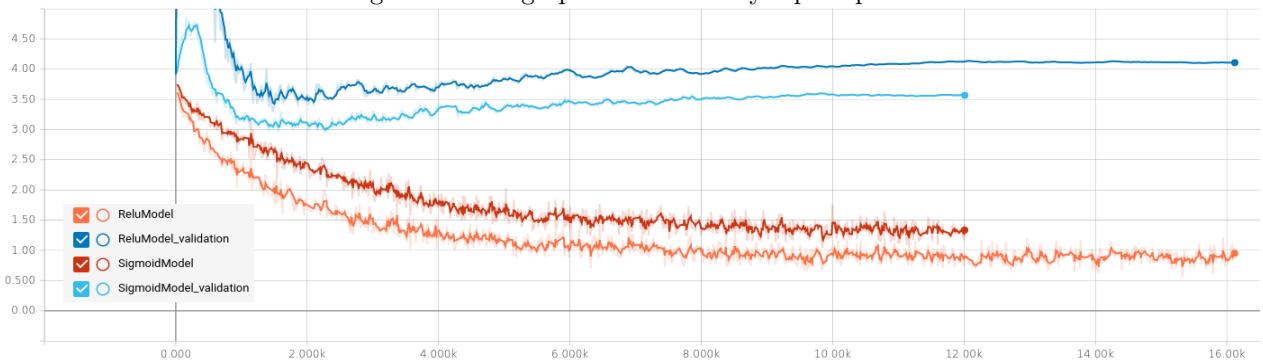


Figure 6: Training and validation accuracy curves for the VGG16 pretrained model (orange line is for validation)

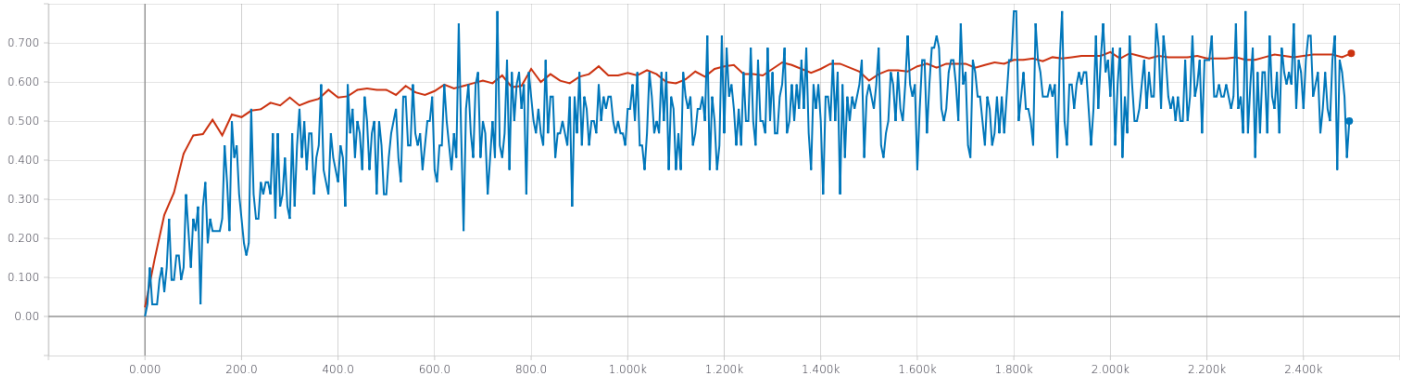


Figure 7: Results on testset of VGG16 finetuned model (blue - correct answers).

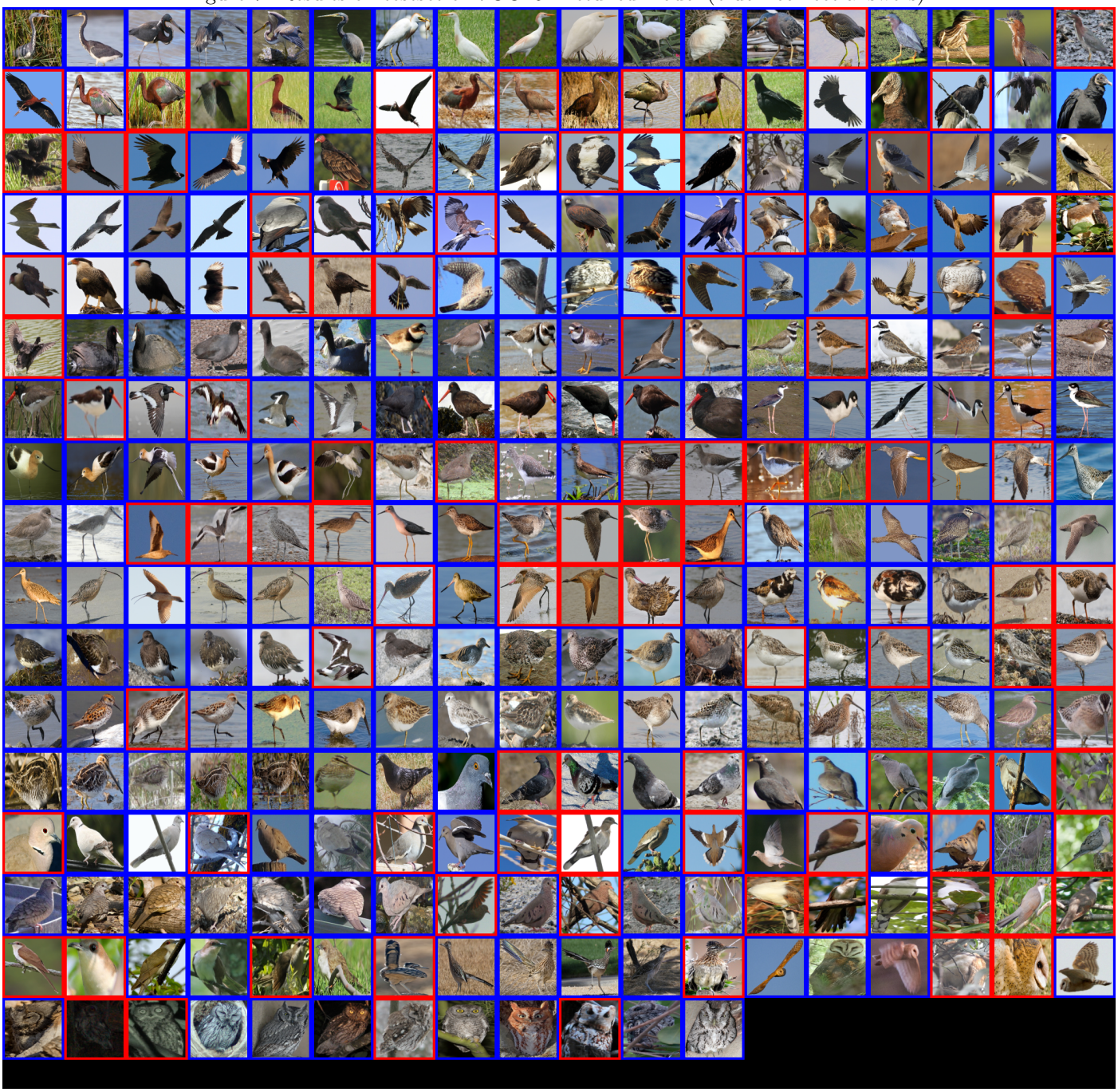
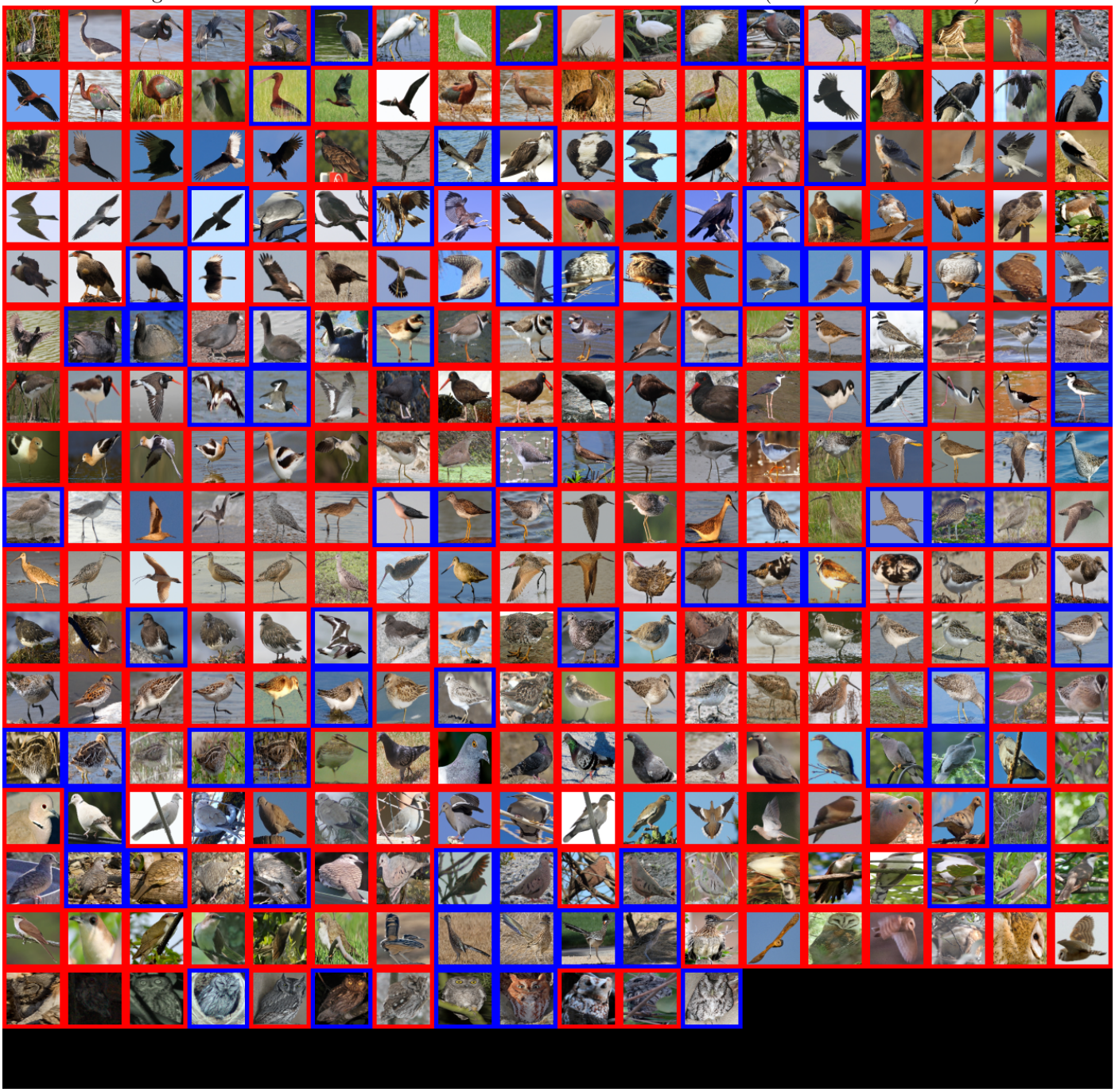


Figure 8: Results on testset of small model that uses HOG features (blue - correct answers)



References

[1] Google Brain team. https://www.tensorflow.org/, (09.05.2018).

[2] NumPy developers. http://www.numpy.org/, (09.05.2018).

[3] OpenCV team. https://opencv.org/, (09.05.2018).

[4] Python Software Foundation. https://www.python.org/, (09.05.2018).

[5] Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z., Karpathy A., Khosla A., Bernstein M., Berg A. C., Fei-Fei L.: *ImageNet Large Scale Visual Recognition Challenge*, 2014. https://arxiv.org/abs/1409.0575, (09.05.2018).

[6] Simonyan K., Zisserman A.: *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2014. https://arxiv.org/abs/1409.1556, (09.05.2018).

[7] Sergey Ioffe and Christian Szegedy: *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015. http://arxiv.org/abs/1502.03167, (10.05.2018).