Final Project Report - Dog Breed Classification

Denis Kim

***Abstract*—The dog breed classification is one of the popular computer vision problems. In this project three main methods were examined. The first is an implementation of well-known convolutional neural network (CNN) architectures such as AlexNet, and VGG16 with slight modifications of last fully connected layers, and training them from scratch. The second is an implementation of my own CNN architectures with an aim to reduce the number of learnable parameters. The last is an application of transfer learning with fixed weights of convolutional layers and updated weights of fully connected layers. Furthermore, performance of SGD, Adam, and RMSprop optimizers were compared. The results of experiments showed that the reduction of learnable parameters improves performance for learning from scratch. However, the most complex network showed the best accuracy for transfer learning. The dataset used in this project was taken from the the Kaggle competition. It comprises 120 breeds of dogs, and is divided into 10.2k labeled trained images and 10.4k unlabeled test images. The data was augmented using random rotation and horizontal flip. Pretrained ResNet152 with RMSprop optimizer was selected as the final model. It achieved an 83% accuracy for the dataset. This is a satisfied result considering very limited amount of data. Finally, a simple graphical user interface (GUI) was developed for convinient testing of images.**

I. Introduction

There are hundreds of different dog breeds in the world, and most people suffer to recognize the name of the breed of a randomly presented dog. However, there is a technical solution inspired by deep learning that can help. The aim of this project is to provide software that can get a picture of a dog and output the name of its breed. This will be achieved by training a neural network on a sufficient amount of data with Pytorch and creating a GUI with the Tkinter. The project includes implementation from scratch of two famous CNN architectures: AlexNet and VGG16, three original architectures with a fewer number of learnable parameters, and transfer learning of pretrained VGG16, ResNet18, ResNet50, and ResNet152. Furthermore, the comparative analysis of SGD, Adam, and RMS prop optimizers will be presented.

II. Methods

A. Dataset

The dataset for this project was taken from the Kaggle competition which was held 3 years ago. It comprises 120 different dog breeds and contains 10.2k training images, 10.4 testing images, and labels of the training images in CSV format, where the first column represents the name of the image, and the second column represents the name of a breed.

Initially, a dictionary of classes was created in order to easily convert the name of the breed into the corresponding index. After that, data from the CSV file was loaded into two lists, the 'images' for names of the images, and the 'labels' for the names of the breed of corresponding dog. The next step was to implement a class that extends the torch Dataset class. Three main functions were defined: init(), len(), and getitem(). The init() initializer takes four parameters: images, labels, root_dir, and transforms. Images and labels correspond to the lists that were defined in the second step, root_dir is a path to the directory with images, and transforms is a set of transforms that are used for the data. For this project, the data was first resized to 256 pixels, then it was cropped in the center to obtain 224 by 224 pixels input. After that, the data was augmented with random rotation on a 30-degree angle and random horizontal flip. Finally, it was transformed into the Tensors and normalized. After the creation of a dataset, it was split into train and validation, of sizes 9000 and 1222, respectively.

B. Training from Scratch

Five different architectures were implemented for evaluation. Number of learnable parameters for each model was calculated using the following formulas:

$$n = (k * k * in\_ch + 1) * out\_ch$$

$$n = (in\_ch * out\_ch) + 1 * out\_ch$$

Where n is the number of learnable parameter in a particular layer, k is the kernel size of convolutional filter, in_ch is the number of input channels and out_ch is the number of output channels. First equation is used for calculating number of learnable parameters in convolutional layers, and the second equation is used for calculating number of learnable parameters in fully connected layers.

First two networks had already proven their performance on ImageNet competitions, AlexNet and VGG16. The architecture of AlexNet consists of 8 layers: 5 convolutional and 3 fully-connected, carrying the ReLU activation function in the first 7 layers and softmax function for the last layer for classification [1]. Exactly the same architecture that was described in [1] was implemented with only modification in the last layer, particularly number of output channel was changed from 1000 to 120. It will be called in this article as MyAlexNet, and the details of MyAlexNet can be found in Table I. Originally AlexNet has 60M parameters [1], but since the last layer was reduced, MyAlexNet has 47M parameters.

VGG16 is twice deeper than AlexNet and contains 13 convolutional and 3 fully-connected layers with the same activation functions as AlexNet [2]. Similarly the VGG16 architecture was implemented according to [2] with the last layer modified. It will be called in this article as MyVGG16, and the details of MyVGG16 can be found in Table II. Originally VGG16 has 138M parameters [2], but since the last layer was reduced, MyVGG16 has 132M parameters.

TABLE I
MYALEXNET ARCHITECTURE

Layer	Kernel Size	Stride	Padding	Input Channel	Output Channel
ConvLayer + Relu	11	4	0	3	96
MaxPool	3	2	0	96	96
ConvLayer + Relu	5	1	2	96	256
MaxPool	3	2	0	256	256
ConvLayer + Relu	3	1	1	256	384
ConvLayer + Relu	3	1	1	384	384
ConvLayer + Relu	3	1	1	384	256
MaxPool	3	2	0	256	256
Fully Connected Layer				256*5*5	4096
Fully Connected Layer				4096	4096
Fully Connected Layer				4096	120

Optimizer	Adam
Learning Rate	0.001
Batch Size	90
Epochs	30
Total number of Learnable Parameters	47M

TABLE II
MYVGG16NET ARCHITECTURE

Layer	Kernel Size	Stride	Padding	Input Channel	Output Channel
ConvLayer + Relu	3	1	1	3	64
ConvLayer + Relu	3	1	1	64	64
MaxPool	2	2	0	64	64
ConvLayer + Relu	3	1	1	64	128
ConvLayer + Relu	3	1	1	128	128
MaxPool	2	2	0	128	128
ConvLayer + Relu	3	1	1	128	256
ConvLayer + Relu	3	1	1	256	256
ConvLayer + Relu	3	1	1	256	256
MaxPool	2	2	0	256	256
ConvLayer + Relu	3	1	1	256	512
ConvLayer + Relu	3	1	1	512	512
ConvLayer + Relu	3	1	1	512	512
MaxPool	2	2	0	512	512
ConvLayer + Relu	3	1	1	512	512
ConvLayer + Relu	3	1	1	512	512
ConvLayer + Relu	3	1	1	512	512
MaxPool	2	2	0	512	512
Fully Connected Layer				512*7*7	4096
Fully Connected Layer				4096	4096
Fully Connected Layer				4096	120

Optimizer	Adam
Learning Rate	0.001
Batch Size	90
Epochs	30
Total number of Learnable Parameters	132M

After that three simple convolutional networks were implemented from scratch: MySimpleNet, MySimpleNet1, and MySimpleNet2. MySimpleNet has four convolutional layers and two fully connected layers, with a total number of learnable parameters equal to 17 016 824. MySimpleNet1 is simpler, it has three convolutional layers and two fully connected layers, as a result, the network has only 2 668 792 learnable parameters. And MySimpleNet2 has more convolutional layers than MySimpleNet and less learnable parameters than MySimpleNet1. It consists of six convolutional layers and two fully connected layers with 2 421 368 parameters. All three networks are summarized in Tables III, IV and V, respectively.

Performance of all listed models were compared in terms of accuracy and loss.

C. Transfer Learning

Due to insufficient amount of data, it was decided to apply a transfer learning. Four different pretrained models were used: VGG16, ResNet18, ResNet50, and ResNet152. For VGG16 all weights of convolutional layers were fixed and only weights of fully connected layers were updating. For all three ResNets the weights were updating only in the last layer. All models were trained on 25 epochs with Adam optimizer and batch size equal to 90. Additionally, ResNet152 was trained on 64 batch size with Adam optimizer and RMSprop optimizer. The scheduler was used in order to decay the learning rate.

Performance of all pretrained models were compared in terms of accuracy and loss.

III. RESULTS

A. Training from Scratch

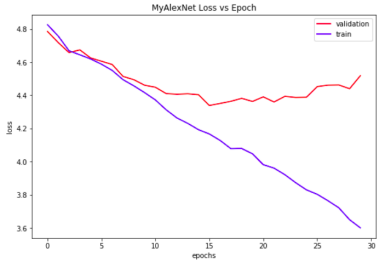

Fig. 1. MyAlexNet trained on 30 epochs with Adam optimizer, accuracy after 30 epochs: 5.07%

TABLE III
MYSIMPLENET ARCHITECTURE

Layer	Kernel Size	Stride	Padding	Input Channel	Output Channel
ConvLayer + Relu	5	1	0	3	64
MaxPool	2	2	0	64	64
ConvLayer + Relu	5	1	0	64	128
MaxPool	2	2	0	128	128
ConvLayer + Relu	3	1	0	128	256
MaxPool	2	2	0	256	256
ConvLayer + Relu	3	1	0	256	256
MaxPool	2	2	0	256	256
Fully Connected Layer				256*11*11	512
Fully Connected Layer				512	120

Optimizer1	Adam
Optimizer2	SGD
Learning Rate	0.001
Batch Size	90
Epochs	30
Total number of Learnable Parameters	17M

TABLE IV
MYSIMPLENET1 ARCHITECTURE

Layer	Kernel Size	Stride	Padding	Input Channel	Output Channel
ConvLayer + Relu	7	4	0	3	64
MaxPool	2	2	0	64	64
ConvLayer + Relu	5	1	0	64	128
MaxPool	2	2	0	128	128
ConvLayer + Relu	3	1	0	128	256
MaxPool	2	2	0	256	256
Fully Connected Layer				256*4*4	512
Fully Connected Layer				512	120

Optimizer1	Adam
Learning Rate	0.001
Batch Size	90
Epochs	30
Total number of Learnable Parameters	2.6M

TABLE V
MYSIMPLENET2 ARCHITECTURE

Layer	Kernel Size	Stride	Padding	Input Channel	Output Channel
ConvLayer + Relu	5	2	0	3	32
MaxPool	2	2	0	32	32
ConvLayer + Relu	3	1	1	32	32
ConvLayer + Relu	1	1	0	32	64
MaxPool	2	2	0	64	64
ConvLayer + Relu	3	1	1	64	64
ConvLayer + Relu	1	1	0	64	128
MaxPool	2	2	0	128	128
ConvLayer + Relu	3	1	1	128	128
MaxPool	2	2	0	128	128
Fully Connected Layer				128*6*6	512
Fully Connected Layer				512	120

Optimizer1	Adam
Learning Rate	0.001
Batch Size	90
Epochs	30
Total number of Learnable Parameters	2.4M

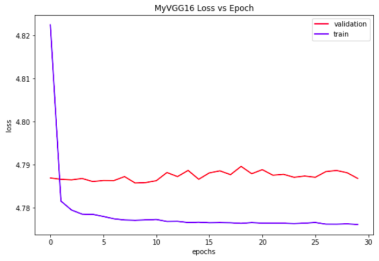

Fig. 2. MyVGG16 trained on 30 epochs with Adam optimizer, accuracy after 30 epochs: 0.98%

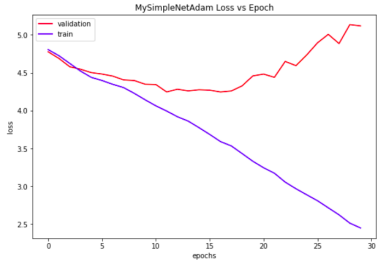

Fig. 3. MySimpleNet trained on 30 epochs with Adam optimizer, accuracy after 30 epochs: 8.01%

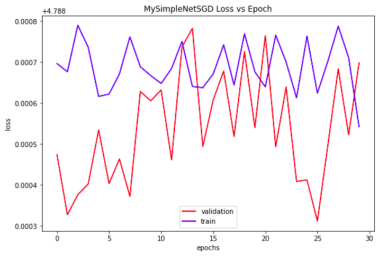

Fig. 4. MySimpleNet trained on 30 epochs with SGD optimizer, accuracy after 30 epochs: 0.73%

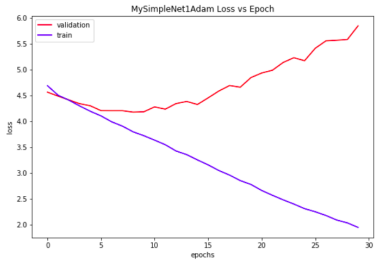

Fig. 5. MySimpleNet1 trained on 30 epochs with Adam optimizer, accuracy after 30 epochs: 7.52%

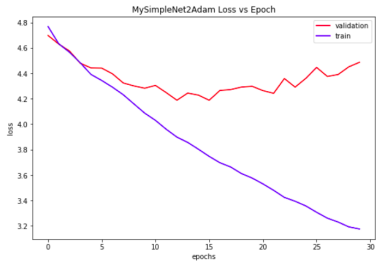

Fig. 6. MySimpleNet2 trained on 30 epochs with Adam optimizer, accuracy after 30 epochs: 8.91%

It can be seen from the results presented above that all models have an overfitting problem. And the most simple network with less learnable parameters showed the most accurate result. Also, it can be observed that the SGD optimizer does not efficiently optimize the weights, compared to Adam optimizer. It can be seen that the reduction of complexity helped to improve accuracy for this dataset. But the reduction of convolutional layers can lead to negative results. For example, MySimpleNet1 has worse results, compared to MySimpleNet which has more learnable parameters and convolutional layers. And MySimpleNet2 having more convolutional layers and fewer parameters than MySimpleNet showed better results.

B. Transfer Learning

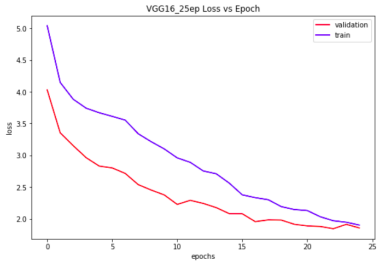

Fig. 7. VGG16 trained on 25 epochs with Adam optimizer, accuracy after 25 epochs: 52.04%

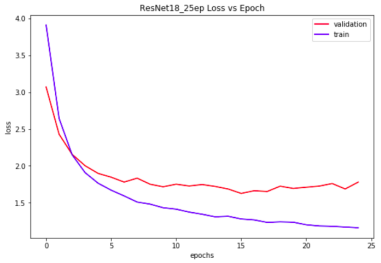

Fig. 8. ResNet18 trained on 25 epochs with Adam optimizer, accuracy after 25 epochs: 51.47%

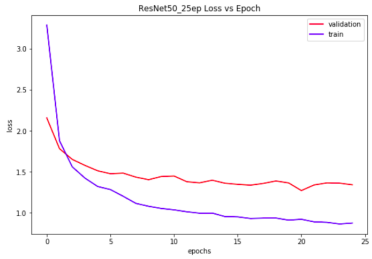

Fig. 9. ResNet50 trained on 25 epochs with Adam optimizer, accuracy after 25 epochs: 63.58%

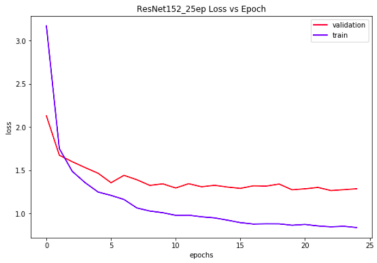

Fig. 10. ResNet152 trained on 25 epochs with Adam optimizer, accuracy after 25 epochs: 66.20%

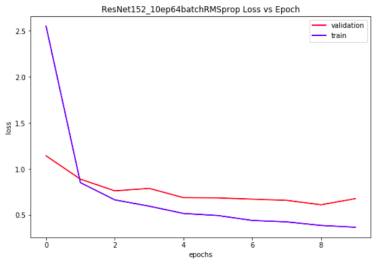

Fig. 11. ResNet152 trained on 10 epochs with RMSprop optimizer, accuracy after 10 epochs: 83.22%

It can be seen from the results presented above that for transfer learning accuracy improved dramatically, and overfitting problem decreased. For VGG16 there is even no overfitting problem for 25 epochs. ResNet50 showed the best performance among all tested models. However, it could identify a breed of the dog only with 66.2% accuracy when was trained with Adam optimizer. Therefore, the RMSprop optimizer was tested, which showed 17% accuracy improvement and reached 83.22%.

IV. Conclusion

The software that can classify a breed of the dog by its picture with an accuracy of 83.22% was developed in this project. The results of the experiments showed that for the dataset of size 10.2k images with 120 different classes, over-complex networks do not work accurately and lead to significant overfitting when training from scratch. However, transfer learning can handle it, and more complex pre-trained models show better results. Furthermore, three optimizers were compared. And results showed that Adam optimizer works better than SGD optimizer, whereas RMSprop showed better results than Adam for this particular problem.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.