Summer 2019

Deep learning for image classification on very small datasets using transfer learning

Mengying Shu

Deep Learning for Image Classification on Very Small Datasets Using Transfer Learning

by

Mengying Shu

A report submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Joseph Zambreno, Major Professor

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation/thesis. The Graduate College will ensure this dissertation/thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

DEDICATION

I would like to dedicate this creative component to Dr. Zambreno without whose support I would not have been able to complete this work. To my friends who have been affected in every way possible for giving me inspiration, courage, and confidence.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

First and foremost, I would like to take this opportunity to express my thanks to Dr. Zambreno for his support and encouragement throughout this journey. Not only I have given the inspiration for completing my graduate education, but also the patience and support through the tough road. His feedback plays a significant role in this journey. Also, I would like to thank Department of Electrical and Computer Engineering and Iowa State University for helping me get through every moment that I have struggled.

ABSTRACT

Since the ImageNet Large Scale Visual Recognition Challenge has been run annually from 2010 to present, researchers have designed lots of brilliant deep convolutional neural networks(D-CNNs). However, most of the existing deep convolutional neural networks are trained with large datasets. It is rare for small datasets to take advantage of deep convolutional neural networks because of overfitting when implementing those models. In this report, I propose a modified deep neural network and use this model to fit a small size dataset. The goal of my work is to show that a proper modified very deep model pre-trained on ImageNet for image classification can be used to fit very small dataset without severe overfitting.

CHAPTER 1. OVERVIEW

It is seen that deep learning has a huge impact for well-defined kinds of perception and classification problems. Huge amounts of data like millions of images are required for the neural networks of deep learning models to learn a task. It is obvious that large amount of training data plays a leading role in making the Deep learning models successful. Over the last few years, state-of-the-art models have shown superhuman performance at recognizing objects like Alexa, Google Home, and Baidu voice's assistant. Those models shows the maturity of the deep learning which ensures the spread adoption. In reality, there exists task to achieve great performance when given small size data. When given the task, people use shallow models that is a waste of time and achieves low performance with huge overfitting. However, models trained on tasks with proper modifications can be reused for different problems in the same domain that can achieve good performance without severe overfitting.

1.1 Introduction

In deep learning, a convolutional neural network (CNN) plays a leading role, mostly used to analyze visual images. CNN was invented in 1980s [1], thanks to the popularity of GPU [2], it embraced the breakthrough in 2000. In 2011, Dan presented high-performance GPU-based CNN variants to achieve a recognition test error rate of 0.35%, 2.53% and 19.51% for digit recognition (MNIST), 3D object recognition (NORB), and natural images (CIFAR10) seeming to be the best performance [3]. In 2012, Krizhevsky won the ImageNet Large Scale Visual Recognition Challenge [4] [5]. Krizhevsky's model was trained on the 1.2 million high-resolution images, with 8 weight layers(5 convolution layers and 3 fully-connected layers) and ReLU activation function [6]. Since the excellent success of CNN in AlexNet, ZFNet was proposed by tweaking the hyper-parameters of AlexNet while maintaining the same structure [7], which performed better than AlexNet in 2013. In

2014, GoogleNet won the ImageNet Large Scale Visual Recognition Challenge [8]. Its architecture consisted of a 22 layer deep CNN but reduced the number of parameters from 60 million (AlexNet) to 4 million. In 2015, ResNet introduced skip connection (or shortcut connection) to fit the input from the previous layer to the next layer without any modification of the input with very deep network of up to 152 layers and won the ImageNet Large Scale Visual Recognition Challenge [9]. The above architectures confirmed the significance of depth can lead to excellent performance of CNNs.

However, VGG, GoogleNet and ResNet using CNNs are both trained on the giant ImageNet datasets. In reality, most of time there are no available giant size data like ImageNet datasets. Collecting a huge size dataset can be expensive for a specific task. When it comes to a smaller dataset, making technology that can work with deep network is efficient and can achieve high performance. Then transfer learning is introduced to train the small size data. Transfer learning means using the learning from one task to another task without learning from scratch. Deep models like VGG, GoogleNet, ResNet trained on ImageNet datasets learn to identify images and more complicated structure can be learned by later layers building on top of these features. The last layer is classifier to determine the images into different categories. In that case, I can use the transfer learning on the small size data such that the optimization process is fast because the lower layer feature still remain relevant and overfitting caused due to small number of data can be reduced with modifications like data augmentation and dropout.

CHAPTER 2. BACKGROUND

In this chapter, I would like to introduce the overview of machine learning, different models of CNNs, and transfer learning.

2.1 Overview of Machine Learning

Machine learning, deep learning, and AI has been shown in countless articles in recent years. Born in 1950, artificial intelligence was camp up by people from the field of computer science by asking whether computers could be made to think. In general, artificial intelligence is a field including machine learning and deep learning, but also other learning. Machine learning came from the question that whether a computer could perform a task on its own. In classical programming, answers can be produced by applying rules and data. With machine learning, rules can be produced by applying data and answers. Then those rules can be used to new data to produce new answers. Overall, machine learning system is trained by presented with examples related to the task and finding the rules to automate the task. Deep learning is a sub-field of machine learning with successive layers of the increasingly representations. Guided by experimental findings instead of theory, algorithmic advance can be achieved with appropriate data and hardware.

2.2 Overview of CNNs

A CNN architecture is composed of convolutional layer, pooling layer, reLU layer, fully connected layer, and loss layer. Convolutonal layer is the core block of a CNN consisting of filters (or kernels) to detect different types of features from the input and pass them forward. Pooling layer is in-between successive convolutional layers to reduce the parameters and computers in the network. ReLu layer is the activation function that sets negative values to zero. Fully connected layer has full connections to all activations in previous layer to flatten the output that represents high-level

features in the data. Loss layer is used to show the deviation between predicted (output) and true labels.

2.3 Deep Models Matter

2.3.1 VGGNet

VGGNet from [10] scored the second place in ImageNet Large Scale Visual Recognition Challenge that is after GoogLeNet and the first place in image localization. Compared with VGG16, VGG19 is slightly better but requests more memory. VGG16 model is composed of convolutions layers, max pooling layers, and fully connected layers. The total is 16 layers with 5 blocks and each block with a max pooling layer. Similar to VGG16, VGG19 has 19 layers with extra convolution layers in the last three blocks. With deep layers, both VGG16 and VGG19 achieve great performance in the image competition.

2.3.2 GoogLeNet

GoogLeNet, winner of ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014, contains 22 layers with inception network added [8]. It also emphasizes that depth matters. However, GoogLeNet uses 12 times fewer parameters than AlexNet with better performance that is close to human performance. The highlight for GoogLeNet is parallel combination of 1*1, 3*3, and 5*5 convolutional filters and 1*1 convolutional kernels to reduce computation. Moreover, GoogLeNet continues the research by presenting Inception V2. In Inception V2, researchers compute the mean and standard-deviation of all feature maps at the output of a layer, and normalize the responses with these values. Later, Inception V3 is created by carefully constructing networks and using 3×3 and 1×1 filters instead of 5×5 and 7×7 filters. GoogLeNet goes deeper to make better performance and proves the powerful of very deep models.

2.3.3 InceptionResNet

Inspired by the performance of ResNet [9], Google presents InceptionResNet. Residual is added to the output of the convolution operation of the inception module [11]. After convolution, the depth is increased and this model achieves top-5 error on the ImageNet classification. InceptionResNet takes the idea of inception network and deep residual network. It accelerates the speed of training and improve the accuracy with about 467 layer in total. It shows the power of deep layers as well.

CHAPTER 3. RELATED WORK

My method is related to many works like imagine classification, transfer learning, and deep learning, which I briefly discuss below.

3.1 Image Classification

Effective use of multiple features of data and the selection of a suitable classification method [12] are significant for image classification. Similar to my work, [13] performs neural-network feature extraction for face recognition and wood defect detection by using clustering low-level features. Differently to [13], I pre-train the model based on large scale datasets and apply modified these models on small size datasets with great performance.

3.2 Transfer Learning

In reality, very few people would train the entire model from the beginning since huge size dataset rarely exists. However, many researchers prefer to pretrain the model on a huge dataset like ImageNet and then extract features from it. Then transfer learning is introduced. It is an optimization to save time and achieve better performance at the same time. [14] presents knowledge transfer would improve the performance of learning and reduce the effort to recollect the data. [14] illustrates three issues that "what to transfer", "how to transfer", and "when to transfer". "What to transfer" means the specific part to be transferred across domains corresponding to the "how to transfer" issue. "When to transfer" means transfer learning to be done in specific situation. The above point provides me the idea how to think about transfer learning according to my project. Also [15] shows the challenges of transfer learning. As [15] presents, the goal of transfer learning is to improve the learning skills in the target task given the knowledge from source-task. One of the challenges in transfer learning is to produce positive transfer while avoiding negative transfer

among related tasks. The reason why negative transfer exists is that the relationship between source task and target task is weak. Thus it is significant to recognize and reject bad information while learning the target task. In that case, the performance with transfer learning is at least no worse than without it. [15] presents me the idea of consider transfer learning carefully and avoid negative transfer.

Transfer learning is the improvement of learning in a new task based on the transfer of knowledge from a related task that has already been learned [15]. In computer vision, examples of transfer learning including [16] [17] lead to significantly improved results for object classification. However, training a new classifier with a small number of samples would dramatically increase the risk of overfitting the new data, leading to poor generalization [18]. Fortunately, some methods include dropout in [19] and data augmentation in [20], [21], [22] which try to prevent neural network from overfitting.

Similar to my work, [23], [9], and [6] use deep learning on image classification based on large scale datasets. Differently to [23], [9], and [6], I pre-train and modify the model to fit small size datasets without severe overfitting. Transfer learning with deep neural network has been explored in medical image classification in [24] and [25] in a manner related to my work. Other efforts done in parallel propose the transferring representations learned from large image classification datasets using the convolutional neural network architecture with dropout and fine-tuning strategy in [26]. However, they explore the transfer learning in person re-identification in large datasets like [27], [28], and [29]. In this report, I focus on deep network in a very small dataset.

3.3 Deep Learning

A growing number of work on triggers the interest multilayer neural networks in deep learning, either implementing supervised methods, as in [6] [30], or implementing unsupervised methods, as in [31] [32].

Deep learning allows models that are composed of multiple layers to learn data in [33]. AlexNet from [6], VGG net from [10], GoogleNet from [34], ResNet from [9], ResNeXt from [35], RCNN (Region Based CNN) from [36], and YOLO (You Only Look Once) from [37], are advanced models for deep learning. But in this report, I mainly focus on the VGG, GoogleNet, and ResNet.

While it is shown that all of the models work well in practice, it is unclear that those models perform well when modified and used to fit the small datasets. This question raised considerable interest. A very deep learning architecture called VGG-16 is modified for image classification and fit the small datasets without [38]. The goal of this work is to show that a very deep convolutonal neural network can be used to fit small dataset like CIFAR-10 with simple but proper modifications [38]. Similar to my work, [38] uses Batch Normalization or strong dropout setting on the deep model to achieve better performance. Differently to [38], I add the data-augmentation transformations to the data preprocessing configuration.

CHAPTER 4. METHODS AND PROCEDURES

I use a very deep CNN to fit a very small size dataset from kaggle. The overall architecture and the features of the model are described in the following section. I focus on classifying images as dogs or cats in a dataset containing 6,000 pictures of cats and dogs. 3,000 pictures are used for training where 2,000 for validation, and 1,000 for testing. In my report, the best accuracy is around 96%, even though I train the model less than 20% of the data.

4.1 Overall Architecture

The models I used are pre-trained on large scale dataset called ImageNet that most of the classes are animals and daily objects. I modify these models to recognize different images. These learned features of deep models make very deep network effective for small size datasets problems. VGG16, VGG19, Inception V3, and Inception-ResNetV2 are the models I used on the very small size dataset.

4.2 Deep neural network

The first model that I created is baseline, then I apply this model on my very small size dataset. The reason why I include the baseline is that I can compare pre-trained very deep models with it and prove that very deep models can achieve great performance. For these very deep models like VGG16, VGG19, Inception V3, and InceptionResNet V2, I download weights trained on ImageNet. For each model, I print the summary to see the layers and the final feature map. After that I add the classifier on the top of these layers and use binary classifier as the optimizer. At last, I plot the training accuracy and validation accuracy.

4.2.1 Reduce overfitting

Even though I apply feature extraction with each model and the performance is better than the baseline, there still exists severe overfitting due to lack of large number of data. In order to reduce it, I implement data augmentation and fine-tune with strong dropout to reduce the severe overfitting.

4.2.1.1 Data Augmentation

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=45,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True)
```

From the above code, I use the ImageDataGenerator from tensorflow.keras.preprocessing.image package to increase the diversity of data for training models like rotating, shifting, shearing, zooming, and flipping images. Then I add 50% dropout to the classifier to reduce overfitting more.

4.2.1.2 Fine-tuning

```
pre_trained_model.trainable = True
set_trainable = False
for layer in pre_trained_model.layers:
    if layer.name == 'layer_name':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
```

```
else:
    layer.trainable = False
```

For each model, its summary is displayed and one layer is selected to unfreeze. Once the layer is selected, set the layer_name such that this layer to the classifer layer can be trained and the other layers are frozen. The layer that I choose is randomly from the top since earlier layers in the model are more generic. For example, block4_conv1 layer is chosen for VGG16 and VGG19. Mixed9 layer is chosen for Inception V3 and block8_10_mixed is chosen for InceptionResNet V2. All the layers are from the top layers. Partially models layers' summary can be found in the following.

4.3 Models summary

block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

Figure 4.1: VGG16

block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv4 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv4 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv4 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

Figure 4.2: VGG19

concatenate (Concatenate)	(None, 3, 3, 768)	0	activation_82[0][0] activation_83[0][0]
activation_84 (Activation)	(None, 3, 3, 192)	0	batch_normalization_84[0][0]
mixed9 (Concatenate)	(None, 3, 3, 2048)	0	activation_76[0][0] mixed9_0[0][0] concatenate[0][0] activation_84[0][0]
conv2d_89 (Conv2D)	(None, 3, 3, 448)	917504	mixed9[0][0]
batch_normalization_89 (BatchNo	(None, 3, 3, 448)	1344	conv2d_89[0][0]
activation_89 (Activation)	(None, 3, 3, 448)	0	batch_normalization_89[0][0]
conv2d_86 (Conv2D)	(None, 3, 3, 384)	786432	mixed9[0][0]
conv2d_90 (Conv2D)	(None, 3, 3, 384)	1548288	activation_89[0][0]
batch_normalization_86 (BatchNo	(None, 3, 3, 384)	1152	conv2d_86[0][0]
batch_normalization_90 (BatchNo	(None, 3, 3, 384)	1152	conv2d_90[0][0]
activation_86 (Activation)	(None, 3, 3, 384)	0	batch_normalization_86[0][0]
activation_90 (Activation)	(None, 3, 3, 384)	0	batch_normalization_90[0][0]
conv2d_87 (Conv2D)	(None, 3, 3, 384)	442368	activation_86[0][0]
conv2d_88 (Conv2D)	(None, 3, 3, 384)	442368	activation_86[0][0]
conv2d_91 (Conv2D)	(None, 3, 3, 384)	442368	activation_90[0][0]

Figure 4.3: Inception V3

conv2d_201 (Conv2D)	(None, 3, 3, 224)	129024	activation_200[0][0]
batch_normalization_201 (BatchN	(None, 3, 3, 224)	672	conv2d_201[0][0]
activation_201 (Activation)	(None, 3, 3, 224)	0	batch_normalization_201[0][0]
conv2d_199 (Conv2D)	(None, 3, 3, 192)	399360	block8_9_ac[0][0]
conv2d_202 (Conv2D)	(None, 3, 3, 256)	172032	activation_201[0][0]
batch_normalization_199 (BatchN	(None, 3, 3, 192)	576	conv2d_199[0][0]
batch_normalization_202 (BatchN	(None, 3, 3, 256)	768	conv2d_202[0][0]
activation_199 (Activation)	(None, 3, 3, 192)	0	batch_normalization_199[0][0]
activation_202 (Activation)	(None, 3, 3, 256)	0	batch_normalization_202[0][0]
block8_10_mixed (Concatenate)	(None, 3, 3, 448)	0	activation_199[0][0] activation_202[0][0]
block8_10_conv (Conv2D)	(None, 3, 3, 2080)	933920	block8_10_mixed[0][0]
block8_10 (Lambda)	(None, 3, 3, 2080)	0	block8_9_ac[0][0] block8_10_conv[0][0]
conv_7b (Conv2D)	(None, 3, 3, 1536)	3194880	block8_10[0][0]
conv_7b_bn (BatchNormalization)	(None, 3, 3, 1536)	4608	conv_7b[0][0]
conv_7b_ac (Activation)	(None, 3, 3, 1536)	0	conv_7b_bn[0][0]

Figure 4.4: InceptionResNet V2

CHAPTER 5. MEASUREMENTS AND RESULTS

I train models on a very small size dataset consisting of 3,000 training images, 2,000 validation images, and 1,000 testing images. There are 6,000 images in total. The number of images is very small, for a very deep CNN.

5.1 Measurements

The experiments I work on are on Google Colab Notebooks with virtual GPU. First I download the data from Kaggle. Then I create training, validation, testing directories. In each directory, I create 2 directories cat and dog with corresponding images in it. After that I upload these images to Google Drive and use it when I run the models. Since all the images are JPEG files, the first thing that I work is to decode those files to RGB grid pixels by using tensorflow keras ImageDataGenerator instance. There are 5 models that I use for training including baseline, VGG16, VGG19, Inception V3, and InceptionResNet V2. For the last 4 models, feature extraction, data augmentation, and fine-tuning are used. I use accuracy and loss to decide the performance and overfitting.

5.1.1 Baseline

For the baseline, I directly run the constructed 5-layer CNNs with binary classifier as optimizer without any dropout or data augmentation to be compared with following models as shallow model. As can be seen from figure 5.1, the baseline perform not well that has only about 73% test accuracy and there exists string overfitting since training accuracy is much better than validation accuracy and validation loss value is greater than training loss value.

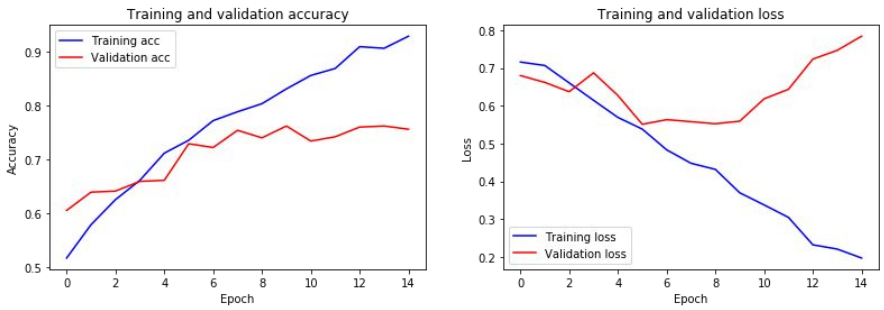

Figure 5.1: Training on baseline

5.1.2 Feature extraction with models

Since I need to utilize the pre-trained models, I download their weights online. Then I add a densely connected classifier layer with dropout on top of the final feature map that is (4, 4, 512) for VGG16 and VGG19. For the other two models Inception V3 and InceptionResNet V2, the final feature map is (3, 3, 2048) and (3, 3, 1536). These feature maps are used as input size in the applied layers training process. Also since I run these convolution base on the very small dataset and apply the extracted features as input, it is very fast to work. The performance for each model is better than baseline, but strong overfitting isn't reduced. Thus data augmentation is necessary for these models.

5.1.3 Data augmentation with models

After working on feature extraction, I find there exists severe overfitting even though the accuracy improves a lot compared with shallow network. Then I decide to run with data augmentation. Similar to the feature extraction, but training data is augmented with these models. Also I freeze the convolution base to prevent the weights from updated. The running time is like 20 minutes

with virtual GPU on Google Colab Notebooks for each model. However, the severe overfitting is reduced and the performance is pretty well. For VGG16 and VGG19, validation accuracy is nearly 90% and overfitting can be barely seen but the loss is high that means performance can be better with modification. For Inception V3 and InceptionResNet V2, the validation accuracy is around 95% but the validation loss increases a little. However, these two models both perform better than VGG16 and VGG19 with higher accuracy and lower loss value.

5.1.4 Fine-tuning with models

In order to make better performance, I fine-tune these models as well. For VGG16 and VGG19, I unfreeze the last two blocks containing convolution layers and max pooling layers used for feature extraction. Both these layers and classifier layer with dropout are trained. For Inception V3, mixed9 layer is chosen with following many layers. For InceptionResNet V2, block8_10_mixed is chosen. It takes around 45 minutes for all the training with virtual GPU on Google Colab Notebooks for each model. However, the performance is greater and overfitting is reduced a lot. For VGG16 and VGG19, the validation is higher and the loss value is lower but the loss value increases a little that these models can be modified in the future. For Inception V3 and InceptionResNet V2, the accuracy is even higher, but the validation loss increases a lot that can be figured out in the future.

5.2 Results

Figure 5.1 to figure 5.13 shows the experiment results.

I find that, when applying featura extraction with pre-trained very deep models on very small dataset, these models achieve higher accuracy than the baseline model and faster even on CPU mode. This shows that when high accuracy is necessary and time is pressed, very deep models are better option than the shallow one.

When using data augmentation and strong dropout on very deep model, the model achieves very high accuracy. From my point of view, depth provides the opportunity to gain high accuracy, data

augmentation and dropout control overfitting significantly. This shows that a very small dataset can utilize the power of depth.

5.3 Training on baseline

From the figure 5.1, it is pretty fast to get the result. However, the performance of this model is around 75% for validation and 86% for training. The training accuracy is much greater than validation accuracy and training loss value is much less than validation loss value. It is obvious that there exists strong overfitting since our dataset size is very small. Thus this baseline model does not achieve a great performance and I decide to find an existing deep model to make a better performance and avoid the severe overfitting.

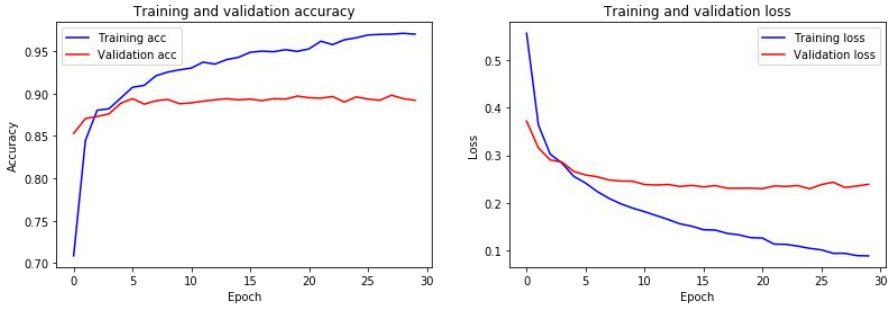

Figure 5.2: Training on VGG16

5.4 Training on VGG16 and VGG19

From the figure 5.2 and figure 5.3, the performance improves a lot compared with baseline. However, strong overfitting still exists since the training accuracy is still greater than validation

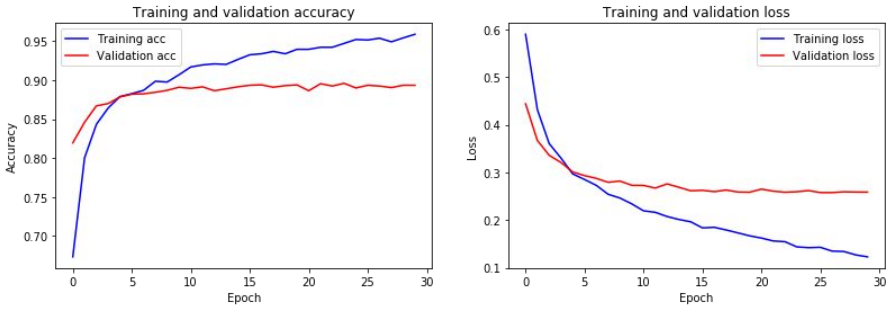

Figure 5.3: Training on VGG19

accuracy and training loss value is less than validation loss value. The test accuracy is 83% for both VGG16 and VGG19.

5.5 Training on VGG16 and VGG19 with data augmentation

The training process lasts some time and the test accuracy is around 88.1% for VGG16 and 85.2% for VGG19. However, there exists no more overfitting when data augmentation applied. As can be seen from the figure 5.4 and figure 5.5, the performance between VGG16 and VGG19 is similar. The training accuracy is a little bit less than validation accuracy and the training loss value is a little bit greater than validation value. Apparently data augmentation reduces the overfitting dramatically. However, in order to perform better with higher accuracy, fine tuning is needed to fit the model.

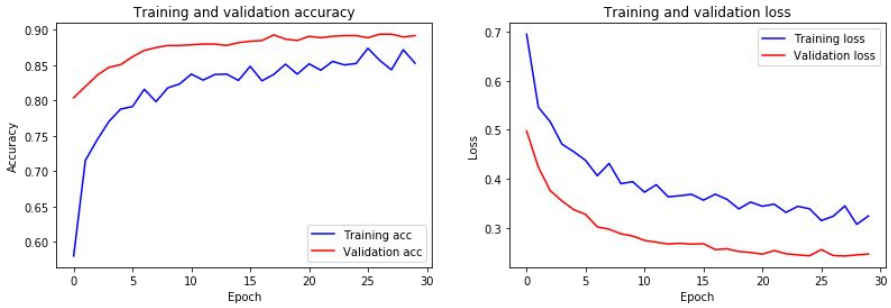

Figure 5.4: Training on VGG16 with Data Augmentation

5.6 Training on VGG16 and VGG19 with fine-tuning

As can be seen from the figure 5.6 and figure 5.7, the validation accuracy is greater than training accuracy at first, but later training accuracy is a little greater than validation accuracy. Also the training loss value is greater than validation value at first, then training loss value is less than validation value. Even though there is a little overfitting later, the performance of both models is great. What's more, VGG16 performs better than VGG19 with higher accuracy and lower loss value. Also the performance is better than above, but there still exists a little bit overfitting for the last half of training process. Compared with previous training, it improves a lot. The test accuracy is 95% and 95.3% for VGG16 and VGG19 respectively.

5.7 Training on Inception V3 and InceptionResNet V2

From the figure 5.8 and figure 5.11, the training accuracy is much greater than validation accuracy and the training loss value is much less than validation loss value. Validation accuracy and training accuracy on both models are high. However, there exists severe overfitting the same

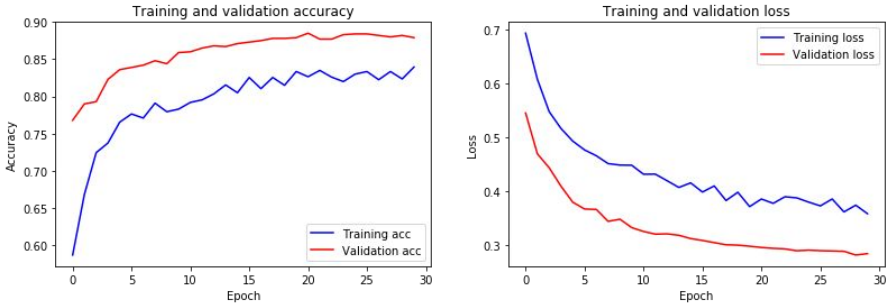

Figure 5.5: Training on VGG19 with Data Augmentation

as VGG 16 and VGG19. But Inception V3 and InceptionResNet V2 gain great test accuracy with 93% and 95% respectively.

5.8 Training on Inception V3 and InceptionResNet V2 with data augmentation

According to figure 5.9 and figure 5.12, the training accuracy is less than validation accuracy and the training loss value is greater than validation loss value. Similar to VGG16 and VGG19, I apply data augmentation on these models as well. Fortunately validation accuracy is higher than training accuracy. However, the loss value is very high. Further, enhancing the performance is necessary. The test accuracy is 94.7% and 95.6% for Inception V3 and InceptionResNet V2 respectively.

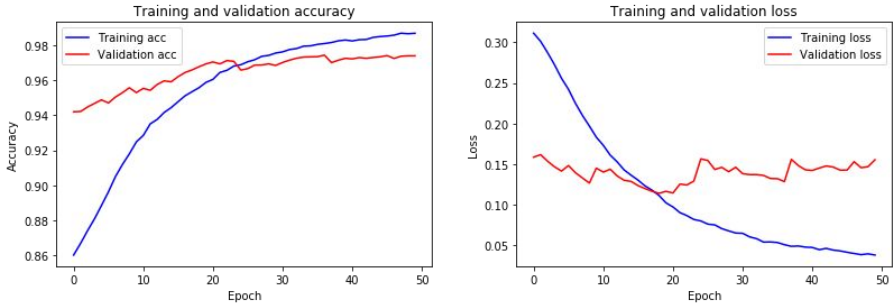

Figure 5.6: Training on VGG16 with Fine-tuning

5.9 Training on Inception V3 and InceptionResNet V2 with fine-tuning

From figure 5.10 and figure 5.13, the training accuracy is less than validation accuracy. For figure 5.10, training loss value is much less than validation loss value that means there still exists overfitting in the model. For figure 5.13, the training loss value is greater than validation loss value at first but less than validation value later. Both models achieve better performance with previous ones. InceptionResNet V2 achieves 96% testing accuracy while Inception V3 achieves 95%. However validation loss value for both models increases a lot that need to figure out in the future work.

5.10 Discussion

The following figures present the performance of accuracy and loss of training and validation sets. If the training accuracy is much greater than validation accuracy, then that is overfitting. If the training accuracy is much less than validation accuracy, then that is underfitting. The job of loss value is to determine how well the model has done on the data by computing the score of

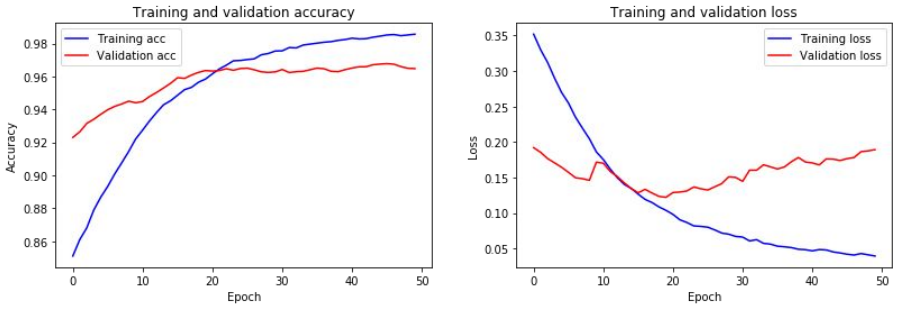

Figure 5.7: Training on VGG19 with Fine-tuning

predictions and true target. If the training loss is much greater than validation loss, then that is underfitting. If the training loss is much less than validation loss, then that is overfitting. Figure 5.1, figure 5.2, figure 5.3, figure 5.8, and figure 5.11 show the severe overfitting due to very small data. However, figure 5.4, and figure 5.5 show that overfitting is reduced a lot with modifications. Figure 5.9, and figure 5.12, show a little bit underfitting but the performance of both models is great. After applying the fine-tuning, the performance of figure 5.6, and figure 5.7 look great. However, there exists a little bit undefitting in figure 5.10, and figure 5.13. In that case, dropping too many weights may be the reason during training process since I add 50% dropout that is a lot.

5.11 Analysis

The best test accuracy is 96% so far. Around 40 images are incorrect labels. As can be seen from figure 5.14, some of the images are blurry and scaled up. Some of dogs and cats are facing away from the image and blend in with their body that makes identify details hard. Some of dogs' and cats' eyes are closed. What's more, there are fences around dogs and cats that make it even

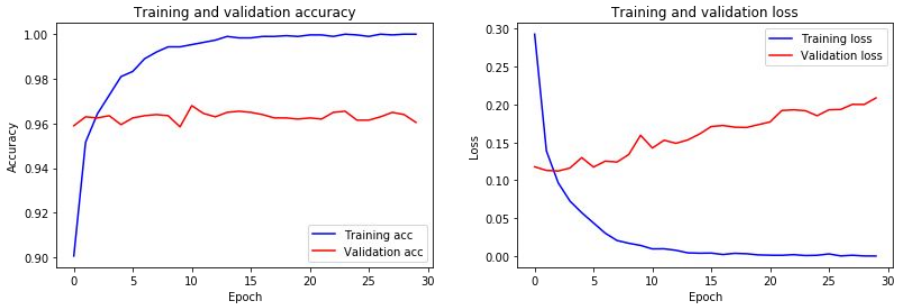

Figure 5.8: Training on Inception V3

harder to recognize. Some dogs and cats are with human beings that occupy a little portion of images. Overall, there still some noises that make it hard to recognize the cats and dogs.

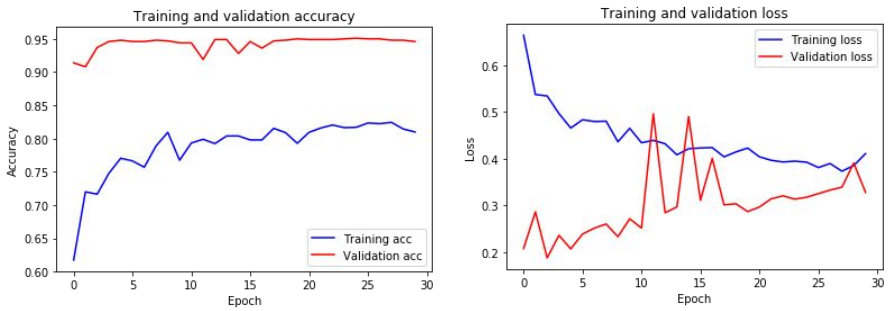

Figure 5.9: Training on Inception V3 with Data Augmentation

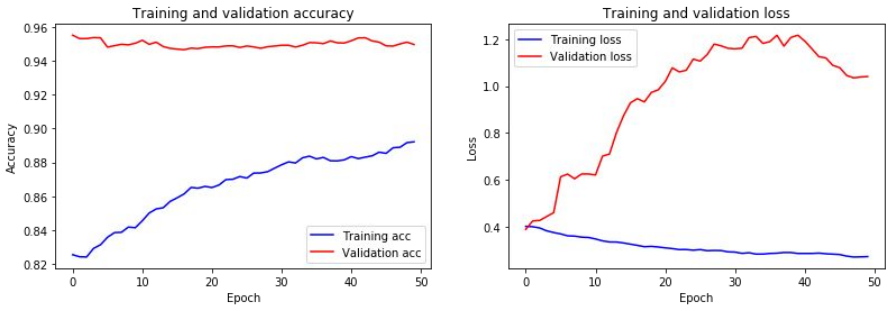

Figure 5.10: Training based on Inception V3 with Fine-tuning

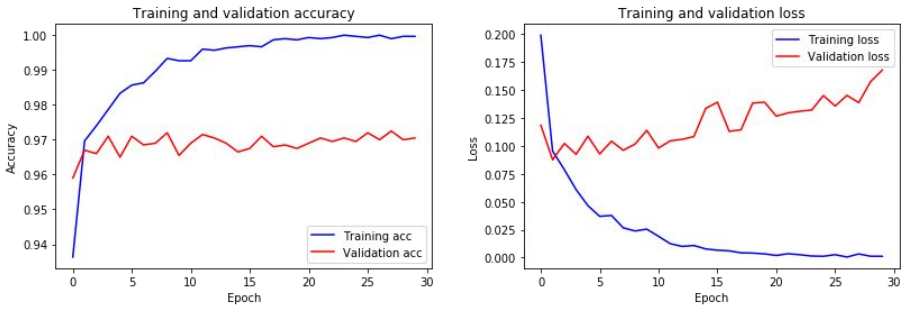

Figure 5.11: Training based on InceptionResNet V2

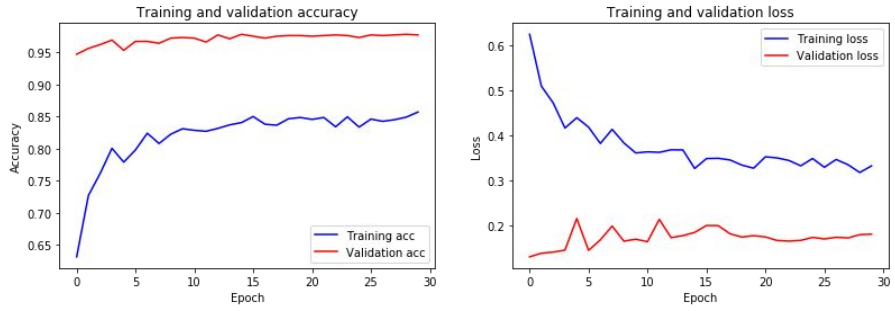

Figure 5.12: Training on InceptionResNet V2 with Data Augmentation

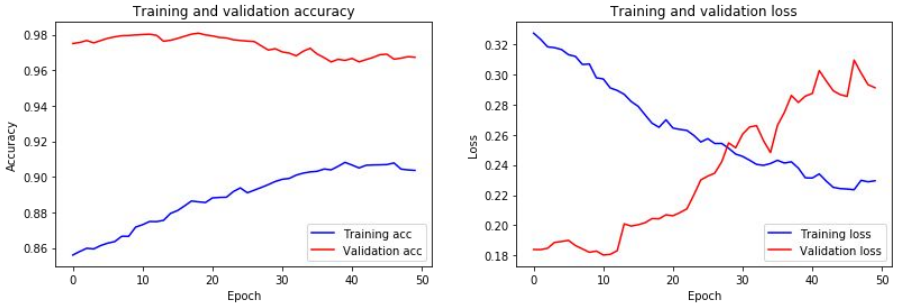

Figure 5.13: Training on InceptionResNet V2 with Fine-tuning

Figure 5.14: Analysis

CHAPTER 6. CONCLUSION

The objective of this project is to show how deep models like VGG16, VGG19, Inception V3, InceptionResNet V2 can be used on very small size data with total 6,000 images including 3000 training data, 2,000 validation data, and 1,000 testing data without severe overfitting and with great performance. All of the models are pre-trained on ImageNet. According to the experiments, all of the models perform well. Specifically, I apply data augmentation, dropout, and fine-tune the these models. It turns out it barely has any overfitting. As proved from experiments, very deep models can be used to fit a very small dataset as long as the good model is chosen and proper modifications are applied. The key to adopt the deep models on very small dataset is data augmentation with dropout and fine-tuning. Choosing proper deep models to fit in very small datasets is crucial to determine the performance. However, overfitting is reduced a lot during the experiments process, underfitting occurs for some models. The reason why this happens may be that too many weights are dropped. Overall, the experiments prove that deep models can be used to fit in very small datasets with proper modifications without severe overfitting.

6.1 Future work

Experiments on different models with proper modifications can be carried on in the future. For example, the issue like why fine-tuning these models leads to the increase of validation loss and how underfitting happens in the models. Whether deep models such as RCNN (Region Based CNN) from [36], and YOLO (You Only Look Once) from [37] can be fit a very small dataset or not. Modifying more models to better fit in very small datasets and comparing them are meaningful to work on in the future.

Table 6.1: Results on different models

	Training Accuracy	**Validation accuracy**	**Testing accuracy**
Baseline	86%	75%	73.8%
VGG16	95%	89%	83%
VGG16 with Data Augmenation	83%	89%	88.1%
VGG16 with FineTuning	97%	96%	95%
VGG19	95%	89%	83%
VGG19 with Data Augmenation	83%	88%	85.2%
VGG19 with FineTuning	98%	96%	95.3%
Inception V3	99%	96%	93%
Inception V3 with Data Augmenation	80%	95%	94.7%
Inception V3 with FineTuning	88%	95%	95%
InceptionResNet V2	99%	97%	95%
InceptionResNet V2 with Data Augmenation	82%	95%	95.6%
InceptionResNet V2 with FineTuning	90%	97%	96%

BIBLIOGRAPHY

[1] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[2] C. McClanahan, "History and evolution of gpu architecture," *A Survey Paper*, p. 9, 2010.

[3] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[5] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[7] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[11] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[12] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," *International journal of Remote sensing*, vol. 28, no. 5, pp. 823–870, 2007.

[13] J. Lampinen and E. Oja, "Distortion tolerant pattern recognition based on self-organizing feature extraction," *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 539–547, 1995.

[14] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[15] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI Global, 2010, pp. 242–264.

[16] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1717–1724.

[17] Q. You, J. Luo, H. Jin, and J. Yang, "Robust image sentiment analysis using progressively trained and domain transferred deep networks," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[18] Y. Yao and G. Doretto, "Boosting for transfer learning with multiple sources," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 1855–1862.

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[20] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[21] S. Xie, T. Yang, X. Wang, and Y. Lin, "Hyper-class augmented and regularized deep learning for fine-grained image classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 2645–2654.

[22] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding data augmentation for classification: when to warp?" in *2016 international conference on digital image computing: techniques and applications (DICTA)*. IEEE, 2016, pp. 1–6.

[23] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "Pcanet: A simple deep learning baseline for image classification?" *IEEE transactions on image processing*, vol. 24, no. 12, pp. 5017–5032, 2015.

[24] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[25] B. Q. Huynh, H. Li, and M. L. Giger, "Digital mammographic tumor classification using transfer learning from deep convolutional neural networks," *Journal of Medical Imaging*, vol. 3, no. 3, p. 034501, 2016.

[26] M. Geng, Y. Wang, T. Xiang, and Y. Tian, "Deep transfer learning for person re-identification," *arXiv preprint arXiv:1611.05244*, 2016.

[27] A. Kumar, J. Kim, D. Lyndon, M. Fulham, and D. Feng, "An ensemble of fine-tuned convolutional neural networks for medical image classification," *IEEE journal of biomedical and health informatics*, vol. 21, no. 1, pp. 31–40, 2017.

[28] K. Yanai and Y. Kawano, "Food image recognition using deep convolutional network with pre-training and fine-tuning," in *2015 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE, 2015, pp. 1–6.

[29] A. K. Reyes, J. C. Caicedo, and J. E. Camargo, "Fine-tuning deep convolutional networks for plant recognition." *CLEF (Working Notes)*, vol. 1391, 2015.

[30] G. Carneiro and N. Vasconcelos, "Formulating semantic image annotation as a supervised learning problem," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2. IEEE, 2005, pp. 163–168.

[31] R. Fergus, P. Perona, A. Zisserman *et al.*, "Object class recognition by unsupervised scale-invariant learning," in *CVPR (2)*, 2003, pp. 264–271.

[32] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS*, 2011.

[33] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[35] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[36] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[38] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *2015 3rd IAPR Asian conference on pattern recognition (ACPR)*. IEEE, 2015, pp. 730–734.