Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Bachelor Thesis Informatics

Fine-Grained Plant Image Classification with Deep Neural Network

Dingfan Chen

24.08.2017

Supervisor

Prof. Dr. Andreas Schilling
Wilhelm-Schickard-Institut für Informatik
University of Tübingen

Chen, Dingfan :
Fine-Grained Plant Image Classification with Deep Neural Network
Bachelor Thesis Informatics
Eberhard Karls Universität Tübingen
Period: from 31.4.2017 till 31.8.2017

Abstract

Fine-grained plant image classification aims to classify similar images of plant organs into different categories (i.e. plant species). In this thesis, we first study two well-known deep convolutional neural network models that are pre-trained on ImageNet dataset [JWR+09], namely Inception-V2 [IS15] and Inception-Resnet-V2 [SIVA16]. Subsequently, we introduce a fine-tuning scheme that is carried out on the basis of the PlantCLEF 2015 [GBJ16] training dataset. Furthermore, we investigate several different data-processing mechanisms and provide evaluations of their model performances in terms of classification results. Finally, we employ the visualization techniques including Class Activation Map [ZKL+16], Gradient-Class Activation Map [SCD+16], Guided Backpropagation [SDBR14] and Guided Gradient-Class Activation Map [SCD+16] into our fine-tuned network models, which help us interpret what the deep neural network models can learn and understand how these models categorize plant species.

Acknowledgements

I wish to express my sincere gratitude to Prof. Dr. Andreas Schilling and Prof. Dr. Hanspeter A. Mallot, for providing me all support and guidance which made me complete the project duly.

I am also greatly thankful to my project guide Gerrit Ecke, who took keen interest on my bachelor thesis and guided me all along till the completion of my thesis.

Contents

List of Figures

List of Tables

List of Abbreviations

CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit
BP	Back propagation
SGD	Stochastic Gradient Descent
Adam	Adaptive moment estimation
BN	Batch Normalization
CAM	Class Activation Map
Grad-CAM	Gradient-weighted Class Activation Map

Chapter 1

Introduction

In recent years, an increasing interest in image-based plant identification task can be seen both in the field of biological taxonomy and computer vision. However, this kind of fine-grained image classification task is challenging due to the small inter-class difference, which makes it extremely difficult even for professional plant taxonomists. So far, the most promising solution towards this problem is the deep learning approach proposed by computer vision scientists. For such an image classification problem, the basic idea of the deep learning approach is to transform this issue into an optimization problem and train the convolutional neural networks (CNN) to find a good parameter set which minimizes the total error on the training dataset.

Over the past years, deep CNN has gained popularity because of its tremendous success in various image retrieval tasks including the Imagenet *large scale visual recognition challenge (LSVRC)* [JWR+09] and the *LifeCLEF Plant Identification Task (PlantCLEF)* [GBJ16], and several network structures have been designed, e.g. the Alex-net [KSH12], VGG-net [SZ15], Googlenet [SLJ+14], Residual net [HZRS15] etc. Most recent work exploiting deep CNN for image classification task adopts a simple strategy: pre-train a deep CNN on a large-scale external dataset (e.g. ImageNet) and fine-tune the pre-trained models on new dataset to fit the specific classification task. Following this idea, we utilized two widely used models, i.e. Inception-V2 and Inception-Resnet-V2, which are pre-trained and achieved high classification accuracy (73.9% and 80.4% top-1 accuracy, respectively) on ImageNet dataset, and fine-tuned these models using the PlantCLEF 2015 training data (91759 plant pictures belonging each to one of the 7 types of view (Branch, Fruit, Leaf, Stem, Leaf, Leaf-Scan, Flower)). Beyond the fine-tuning strategy, we investigated different data-preprocessing methods for improving the performance and then we compared their results.

However, although deep CNN models have demonstrated impressive performance on image classification tasks, there is no clear understanding of how they make predictions and why they perform well. For fine-grained image classification it is particularly important to ensure that the CNN models indeed learn the most discriminative features for each category and ignore the irrelevant background correctly, otherwise the models are not generalized enough and will lead to a large test error due to overfitting. To tackle this issue, some visualization approaches have been proposed in the past few years: Class Activation Map(CAM) [ZKL+16] and Gradient Class Activation Map(Grad-CAM) [SCD+16] aim to localize the important region for making prediction of the input image by finding the strongly activated neurons on the last convolutional layer and upsampling this response until the input layer; Deconvnet [ZKTF10] and Guided Backpropagation [SDBR14] visualize the high resolution features captured by the CNN models by inverting the data flow of a CNN, going from neuron activations in higher layer which we are interested in down to the input layer. In our work, we employed these methods on our models and combined them to interpret the classification results generated by our fine-tuned CNN models.

This bachelor thesis is divided into six parts: In the first chapter, the introduction, we introduce the fine-grained plant image classification task and how we apply the deep learning approach on it. Besides, we illustrate the importance of understanding how the network models making prediction and which algorithms we have adopted for explaining the network model behavior. In the next section, we present general information about convolutional neural networks and their components (Sections 2.1 and 2.2). Besides, we explain the network architectures and the design intuition of the Inception models, in particular the Inception-V2 and Inception-Resnet-V2 models (Section 2.3). In the third part, we elaborate the general algorithm used for training deep convolutional neural network (Chapter 3.1 and 3.2), and some widely used techniques for improving the performance and accelerating the training procedure (Chapter 3.3 and 3.4). In addition, we introduce our data preprocessing methods (Chapter 3.5) in this part. In Chapter 4 we illustrate some visualization techniques and how we adopt them for interpreting the classification results given by our network model. The results of our networks, i.e. their performance on the PlantClEF 2015 dataset and the training curves (Section 5.2) as well as the visualization of our models (Section 5.3) are discussed in the fifth part. At the end, Chapter 6 contains our discussion and conclusion based on our experiment results.

Chapter 2

Artificial Neural Network

In this chapter, we list the basic concepts of artificial neural network 2.1, specifically the convolutional neural network 2.2. Besides, we elaborate the network topology and the intuition behind the design for the architecture of Inception models 2.3.

2.1 Artificial Neuron

The basic computational unit of each artificial neural network is a single neuron, which is a simplified model of nerve cells in human brains.[Kar] In the biological model, each neuron receives input signals from its dendrites and sum up these signals in the cell body. Once the total input signal strength is greater than a certain threshold, the neuron fires and generate output signals which will be sent to other neurons along its axon. The signal strength is controlled by the synaptic strengths which are learnable and thus influence the human learning.

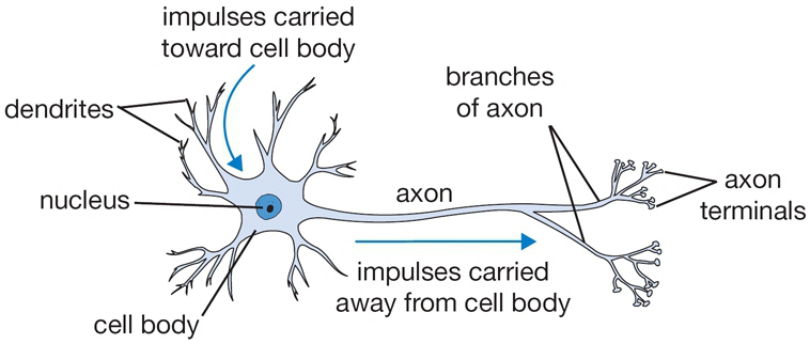


Figure 2.1: biological neuron model in human brains [Kar]

In the mathematic model, the time delay caused by the spikes in the biological

model are not considered, and thus the computation is simplified. Besides, the synaptic strengths are defined as **weights**, which along with **bias** serve as the main parameters need to be learned in a model. The **net input** is then defined as the weighted sum of all the inputs, which simulates the final signal captured by the cell body of a biological neuron. This weighted sum then passes through an **activation function** and the final output of the neuron is generated according to the different activation function. In short, each neuron performs a dot product with the input and its weights, adds the bias and applies activation function to produce an output. The formal definitions are as follows:

$$net_j = \sum_i x_i w_{ij} \tag{2.1}$$

$$o_j = \phi(net_j - \theta_j) \tag{2.2}$$

with:

- x_i - the i-th input value
- w_{ij} - the weight of connection between neuron i and j
- ϕ - the activation function
- θ_j - the threshold(bias) of neuron j
- o_j - the output of neuron j

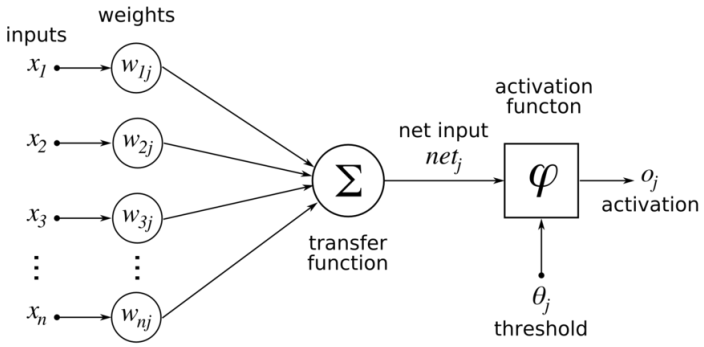


Figure 2.2: mathematic model of neuron [Wik]

Each neuron contains an activation function unit, which should be nonlinear and differentiable. In our work, we choose the *Rectified Linear Unit(ReLU)* [HSM+00][NH10] as the activation function. The definitions of ReLU and its derivative are given in Equation 2.3 and 2.4. Figure 2.3 shows the function graph.

$$f(x) = max(0, x) \tag{2.3}$$

$$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases} \tag{2.4}$$

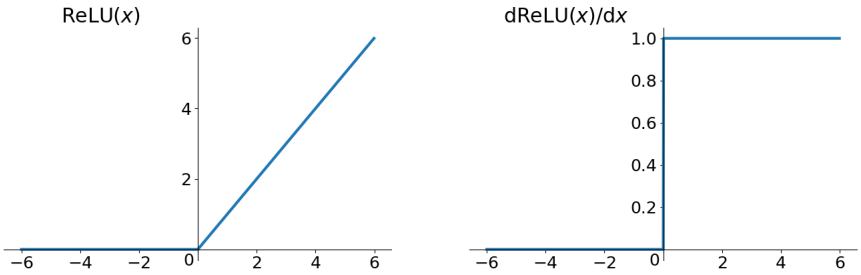


Figure 2.3: Graph for ReLU activation function and its derivative

2.2 Convolutional Neural Network

In this section, we elaborate the three main components of a typical Convolutional Neural Network (CNN), namely Convolutional layer (2.2.1), Pooling layer (2.2.2) and Fully-connected layer (2.2.3). For the image classification task, a softmax layer (2.2.4) is inserted at the end of the network to generate the final class scores.

2.2.1 Convolutional Layer

The Convolutional (Conv) layer is a combination of neural network and the convolution operation in image processing, and it is actually the core building block of a CNN model. Conv layers are based on the assumption of replicated features, i.e. if a feature detector is useful in one place of an image, it is likely that the same feature detector would be useful somewhere else. This replication can greatly reduce the number of free parameters to be learned, since we no longer need to connect neurons to all neurons in the previous layer, but can use **filters (kernels)** which only look at only a small region in the input image (the filter size is called the **receptive field** from a biological viewpoint.) and slide the filter windows across the whole image to generate feature maps. Different feature maps learn to detect different features, which allows each patch of image to be represented by features in many different types. Hence, the usage of Conv layers can still extract information carried by the image while achieving considerable reduction in computational complexity.

Conv Layers contain three hyperparameters, which control the size of the output volume: the **filter size**, **stride** and **zero-padding**. Equation 2.5 shows the relation between the input volume size and the output volume size. Apparently, if the result given by this equation is not an integer, then the hyperparameter combination is not feasible because the size (width,height) of a image can never be a float number. Figure 2.4 displays the convolution process with hyperparameters stride=2 (in both directions), filter size=3

(height and width) and zero-padding=1 (in both directions).[DV16]

$$O = \frac{(I - F + 2P)}{S} + 1 \tag{2.5}$$

with:

P	-	zero-padding
F	-	filter size
S	-	stride
O	-	output size
I	-	input size

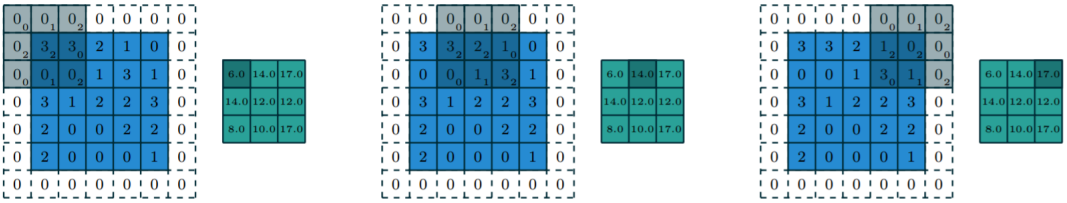


Figure 2.4: A convolution operation for an input of size 5×5. The hyperparameters are: stride=2 (in both directions), filter size=3×3 and zero-padding=1 (in both directions). This will generate an output feature map of size 3×3. [DV16]

The example above in Figure 2.4 is only two-dimensional. In practice, the data always have a depth value greater than one. (e.g. an RGB image has three channels and thus has depth=3. For a neuron in a CNN model with many feature maps, the depth value may be even higher.) For an input with depth we need to operate over all 3-dimensional volumes, and that the filters always extend through the full depth of the input volume.

2.2.2 Pooling Layer

In a typical CNN architecture, pooling layers are normally inserted between successive convolutional layers. Pooling is a downsampling operation which reduces the number of inputs to the next layer of feature extraction, thus decreasing the amount of parameters and allowing more different feature maps in the network. This way, pooling also prevent overfitting in some sense. In practice, there are two common types of pooling operation, namely **max pooling** and **average pooling**.

Similar to Conv layers, the output volume size is controlled by the hyperparameters stride and filter size. The difference is that zero-padding is not

allowed in a pooling layer. relation between the the input volume size and the output volume size can be described by Equation 2.6. [Kar]

$$O = \frac{(I - F)}{S} + 1 \tag{2.6}$$

with:

F	-	filter size
S	-	stride
O	-	output size
I	-	input size

The most common form of a pooling layer is to apply filters of size 2×2 with a stride of 2, which actually downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. By max pooling, the maximum in each small 2×2 region is taken , while the average value is taken by average pooling. Figure 2.5 demonstrates how max pooling and average pooling with filter size=2 and stride=2 work.[TG17]

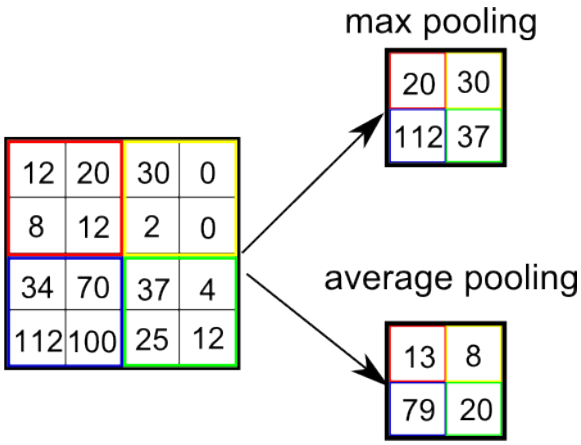


Figure 2.5: Max pooling and average pooling operation for an input of size 4×4. The hyperparameters are: stride=2 (in both directions), filter size=2×2 and zero-padding=2 (in both directions). This will generate an output feature map of size 2×2. The input is divided into four 2×2 parts, which corresponds to the filter size. By max pooling, the maximum of each small region is taken, while average pooling takes the mean of each part. [TG17]

Global Average Pool

In most modern deep neural network models, a global average pooling layer [LCY13] is used after the last convolutional layer and before the final softmax operator to replace the traditional fully connected layers. The basic idea is to

take the average of each feature map and feed the resulting vector directly into the softmax layer. This way, the amount of parameters is greatly reduced since pooling layers do not contain trainable parameter but fully connected layers do. In our work, we use the Inception-V2 and Inception-Resnet-V2 models, both of which use a global average pooling before the final softmax layer in the structure. Figure 2.6 compares the traditional fully connected layer with the global average pooling layer.

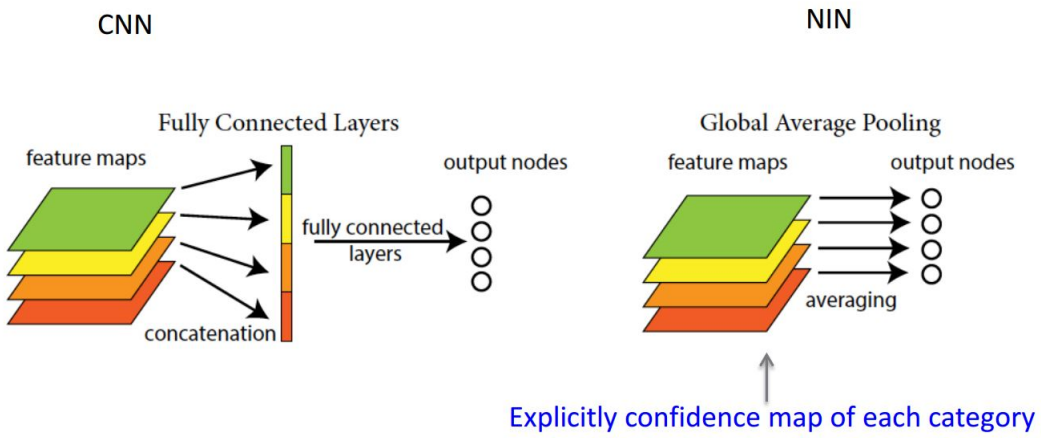


Figure 2.6: A comparison between fully connected layers and the global average pooling layer. Left: the fully-connected layers are used in traditional CNN models such as VGG-net[SZ15] and Alex-net [KSH12]. Right: Network-In-Network(NIN) [LCY13] uses a global average pooling at the end of the model. The mean value of each feature map of the last convolutional layer is taken and directly passed to the final softmax layer. [DLW+]

2.2.3 Fully Connected Layer

In a regular feed-forward neural network model, the fully connected layer is the most common layer type. Figure 2.7 shows the topology of a neural network in which each neuron of the hidden layer is pairwise connected, i.e. fully connected, to every neuron in the adjacent layers[Kar]. For CNN models, the fully connected layers are normally used at the end of the network to compute the final class scores.

2.2.4 Softmax Classifier

For multiclass classification task, a **softmax layer** is used at the end of a network to generate the final class scores. The softmax function generalizes the **logistic (sigmoid) function** (See Figure 2.8), which can only be used

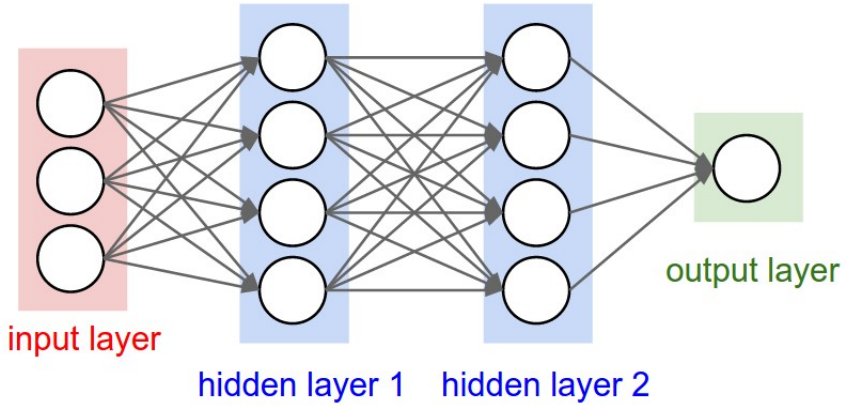


Figure 2.7: A feed-forward neural network with two fully connected layers, i.e. the hidden layers in this case. Each neuron (depicted by circles) of the fully connected layer is pairwise connected (depicted by arrow) to every neuron in the adjacent layers.[Kar]

for binary classification.

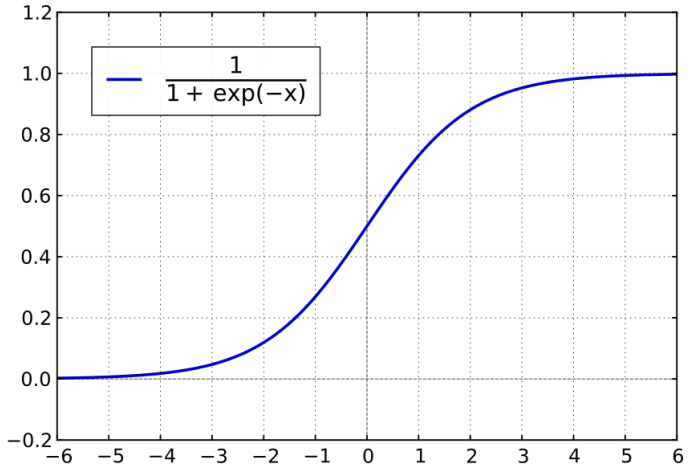


Figure 2.8: Graph for logistic(sigmoid) function $f(x) = \frac{1}{1+e^{-x}}$

The softmax function takes as input a n-dimensional vector of real numbers and squashes these values to a n-dimensional vector of values between zero and one that sum to one. This function is a normalized exponential and the formal definition is as follows:

$$o_j = f(net_j) = \frac{e^{net_j}}{\sum_k^C e^{net_k}} \tag{2.7}$$

with:

o_j - the output of neuron j.
f - the softmax function.
net - the input vector. net_j represents the j-th element of the input vector.
C - the total number of classes. ($C = 1000$ for PlantCLEF 2015 dataset)

The derivative of the softmax function has a quite nice form and thus can be calculated easily in the backpropagation algorithm (Chapter 3.1):

$$\text{if} \quad i = j: \quad \frac{\partial o_i}{\partial net_j} = o_i(1 - o_i) \tag{2.8}$$

$$\text{if} \quad i \neq j: \quad \frac{\partial o_i}{\partial net_j} = -o_i \cdot o_j \tag{2.9}$$

If we set net_i as the unnormalized log probability of class i, then the softmax score can be interpreted as the normalized categorical probability assigned to the correct label given the input x_i parameterized by W.

$$P(y_i|x_i; W) = \frac{e^{net_{y_i}}}{\sum_j e^{net_j}} \tag{2.10}$$

with:
x_i - the input vector.
W - the weight matrix.
y_i - the correct label.
net - the function mapping: $net = Wx_i$.

Cross-entropy Loss Function

The loss function corresponds to softmax classifier is the **cross-entropy loss function** [NBJ02]. From the viewpoint of information theory, it can be written in terms of entropy and the Kullback-Leibler divergence, and thus get its name as cross-entropy.

$$E = -\sum_j t_j \log o_j \tag{2.11}$$

with:
E - the cross-entropy error.
o_j - the softmax value output of neuron j.
t_j - the target value. It is 1 if j is the correct label, and 0 otherwise.

2.3 Inception Models

Inception models refer to a CNN architecture family developed by Google Incorporation which achieve the state-of-the-art results in ImageNet [JWR+09] Large-Scale Visual Recognition Challenge. In our experiment, we investigated the Inception-V2 [IS15][SLJ+14] and the Inception-Resnet-V2 [SIVA16] models. In this section, we elaborate both of these methods.

Inception-V2

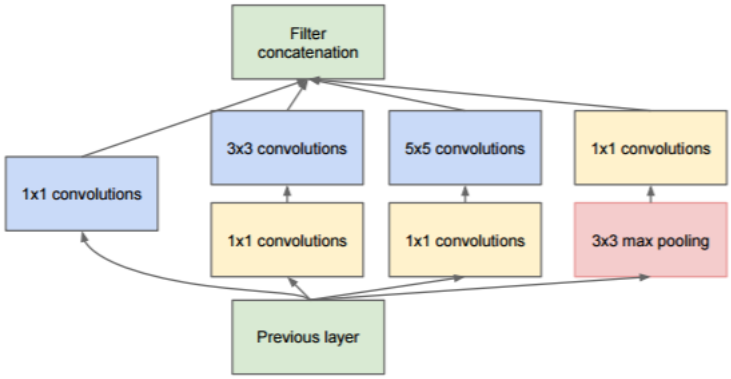
Inception-V2 is a further development of the original Inception model (GoogleNet)[SLJ+14]. The basic components, i.e. the **Inception module** (See Figure 2.9), are highly similar in these network architectures, since both architectural decisions are base on the intuition of multi-scale processing and sparse connection.

type	patch size/stride	output size	input size
input	-	$224 \times 224 \times 3$	-
convolution	$7 \times 7/2$	$112 \times 112 \times 64$	$224 \times 224 \times 3$
max pool	$3 \times 3/2$	$56 \times 56 \times 64$	$112 \times 112 \times 64$
convolution	$1 \times 1/1$	$56 \times 56 \times 64$	$56 \times 56 \times 64$
convolution	$3 \times 3/1$	$56 \times 56 \times 192$	$56 \times 56 \times 64$
max pool	$3 \times 3/2$	$28 \times 28 \times 192$	$56 \times 56 \times 192$
10 Inception	See Figure 2.9	$7 \times 7 \times 1024$	$28 \times 28 \times 192$
global average pool	7×7	$1 \times 1 \times 1024$	$7 \times 7 \times 1024$
linear	-	$1 \times 1 \times 1024$	$1 \times 1 \times 1024$
convolution	$1 \times 1/1$	$1 \times 1 \times 1000$	$1 \times 1 \times 1024$
softmax	-	$1 \times 1 \times 1000$	$1 \times 1 \times 1000$

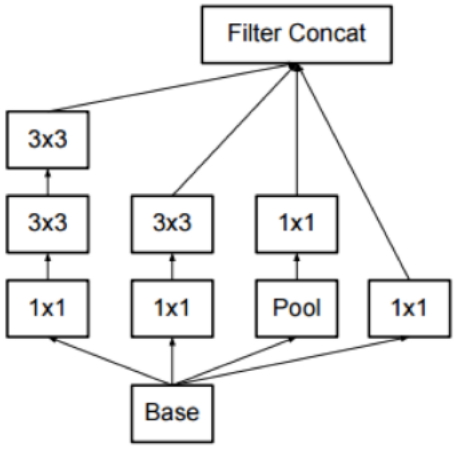
Table 2.1: Outline of Inception-V2 architecture

In the original Googlenet [SLJ+14], the local structure is a clustering of $1 \times 1, 3 \times 3$ and 5×5 filters and the feature maps are concatenated after the convolutional layer (Figure 2.9(a)). As this module consists of filters with different receptive fields, patterns of different scales in the input image can be captured and merged after the filter concatenation. One more important point of this structure is the employment of 1×1 convolutional layers. In the original papers [SLJ+14] and[LCY13], it is illustrated that 1×1 convolution not only increase the representational power of neural network, but also reduce the dimension of input data, thereby removing computational bottlenecks and enable the increasing in depth and width of the networks.

There exists two main difference between Inception-V2 and the original



(a) Inception module in GoogleNet(the first version of Inception)



(b) Inception module in Inception-V2

Figure 2.9: 2.9(a): filters of different sizes($1\times1, 3\times3, 5\times5$) are used to capture features of different scales. All these feature maps are then concatenated and passes to higher layers of the network. 2.9(b): the 5×5 filter is replaced by a stack of two 3×3 filters in the Inception-V2 model. [SLJ+14]

Googlenet (Inception-V1). The most remarkable change is the Batch-Normalization (See Chapter 3.4 for detailed explanation.) which accelerates the training process by reducing *Internal Covariate Shift*. This algorithm now becomes a standard component of most the modern network architecture.

The other main difference is that Inception-V2 replace all the 5×5 filter by a stack of two 3×3 filter 2.9(b). This follows the idea that a stack of small filters can cover a large receptive filed but has fewer parameters [SZ15].

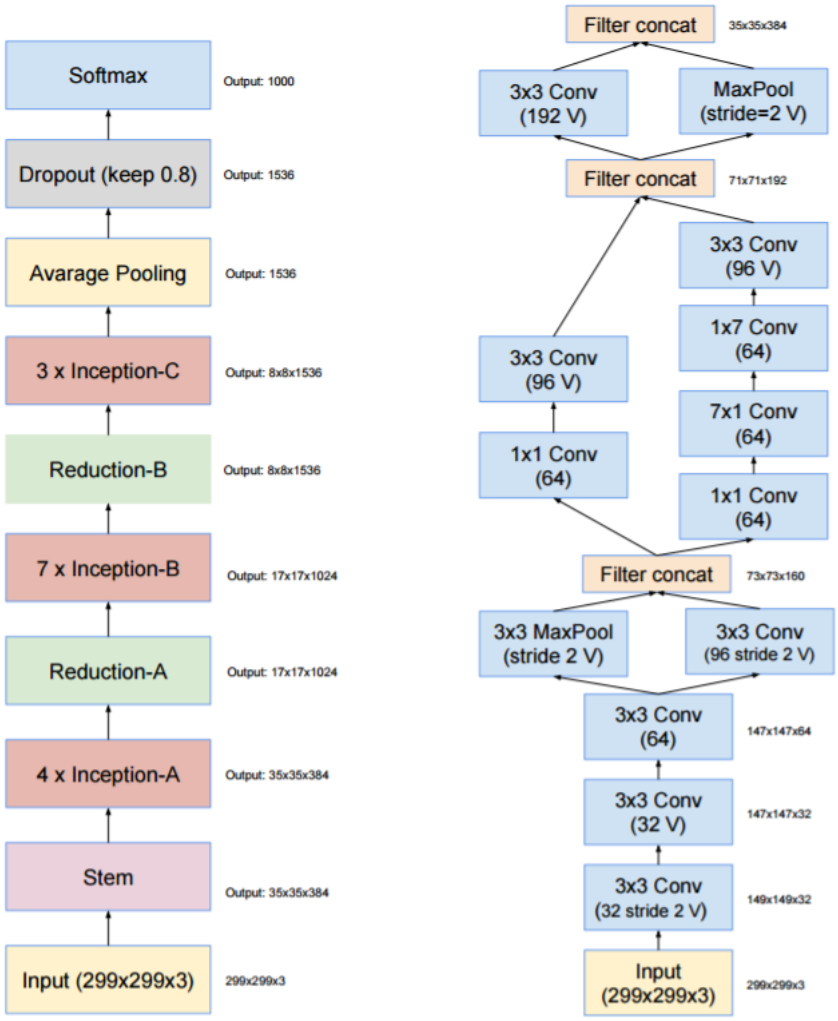
Table 2.1 outlines the structure of Inception-V2, which includes:

- A 7×7 convolution with 64 filters and stride=2
- A 3×3 max pooling layer
- A 1×1 convolution wit 64 filters and stride=1
- A 3×3 convolution with 192 filters and stride=1
- A 3×3 max pooling layer
- 10 Inception modules with 4 branches:
  branch 0: a 1×1 convolution
  branch 1: a 1×1 convolution followed by a 3×3 convolution
  branch 2: a 1×1 convolution followed by two 3×3 convolutions
  branch 3: a average pooling followed by a 1×1 convolution
- A global average pooling layer
- A linear mapping layer with batch-normalization and dropout
- A 1×1 convolution with 1000 filters and stride=1
- A softmax layer which outputs the final class scores

Inception-Resnet-V2

Inception-Resnet-V2 achieves the state-of-the-art performance in ImageNet [JWR+09] Large-Scale Visual Recognition Challenge. It is a combination of the **Deep Residual Net(ResNet)**[HZRS15], which was the winner of the ILSVRC 2015 classification task [RDS+15], and the Inception models.

The basic idea of ResNet [HZRS15] is to insert "shortcut connections" that performs identity mapping into the network, so the model learn to approximate the residual function $\mathcal{H}(x) - x$ instead of the original function $\mathcal{H}(x)$. It is



(a) The overall structure of Inception-Resnet-V2 model

(b) The structure of the 'Stem' part in a Inception-Resnet-V2 model

Figure 2.10: Schema for the overall Inception-Resnet-V2 model and the 'Stem' part. The output size of each level is shown on the right hand side of each diagram. [SIVA16]

asserted that this change considerably simplifies the training and prevents degradation problem occurs in deep neural networks. Following this idea, Szegedy *et al.* insert a shortcut connections into a basic Inception module so as to build a Inception-ResNet module (See Figure 2.11).

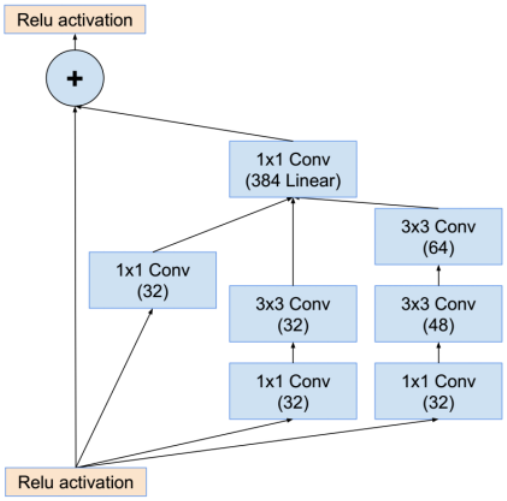


Figure 2.11: Schema for Inception-ResNet-A module(35×35 grid) of the Inception-Resnet-V2 model. [SIVA16]

Chapter 3

Training of Neural Networks

In the previous chapter, we elaborate all the basic components of an artificial neural network and explain how we assemble them to obtain a complex CNN architecture. To solve the plant image classification problem, we now need to train the networks and let them make predictions according to the input images.
In this chapter, we first illustrate the general training algorithm used in our experiments (Chapter 3.1 and 3.2). Then we present some widely used techniques which improve the performance and accelerate the training procedure (Chapter 3.3 and 3.4) . Finally, we introduce the data preprocessing methods we applied in practice (Chapter 3.5).

3.1 Backpropagation

Training the neural network is exactly an optimization problem of finding the parameter set (weight and bias) which minimizes the cost function. To solve such an optimization problem, the most common way is to use the **gradient descent algorithm**, which means the parameters are updated in the direction to decrease the cost function in each step. However, although the gradient descent algorithm is straight forward and easy to implement, it is normally computationally hard to compute the gradients, i.e. the partial derivatives, of the cost with respect to each parameter. Backpropagation algorithm [RHW+88] solve this problem by using the chain rule to propagating the gradients from the output end to the input end of a network.

In backpropagation algorithm, an additional variable $\delta_i = \frac{\partial E}{\partial net_i}$ is introduced, and the gradients of the cost w.r.t. the weight and bias are:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} = \delta_j \frac{\partial net_j}{\partial w_{ij}} = \delta_j o_i \tag{3.1}$$

$$\frac{\partial E}{\partial b_i} = \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial b_i} = \delta_j \frac{\partial net_j}{\partial b_i} = \delta_j \tag{3.2}$$

with:

- E - the cross-entropy error.
- net_j - the input of neuron j.
- w_{ij} - the weight of the connection from neuron i to j.
- b_i - the bias of the neuron i.
- o_i - the output of the neuron i.

Apparently, in order to compute the gradients, we should calculate both δ and $\frac{\partial net_j}{\partial w_{ij}}\left(\frac{\partial net_j}{\partial b_i}\right)$. The key point of backpropagation algorithm is that δ can be computed effectively, if we start computing δ from the output layer and then propagate this value backwards by using chain rule.

As illustrated in chapter 2.2.4, we use the cross-entropy error for multi-class classification task, and the gradient is:

$$\begin{aligned}
\frac{\partial E}{\partial net_i} &= -\sum_j^C \frac{\partial t_j \log(o_j)}{\partial net_i} \\
&= -\frac{t_i}{o_i}\frac{\partial o_i}{\partial net_i} - \sum_{j\neq i}^C \frac{t_j}{o_j}\frac{\partial o_j}{\partial net_i} \\
&= -\frac{t_i}{o_i} o_i(1-o_i) - \sum_{j\neq i}^C \frac{t_j}{o_j}(-o_j o_j) \\
&= -t_i + t_i o_i + \sum_{j\neq i}^C t_j o_i \\
&= o_i - t_i
\end{aligned} \tag{3.3}$$

For a neuron i on the output layer, we get $\delta_i = \frac{\partial E}{\partial net_i} = o_i - t_i$. Now, we need to compute this value for neurons in the hidden layers. The idea is that each hidden activity can affect many output units and can therefore have many separate effects on the final cost, so all these effects must be combined. Following this idea, we derive the rule:

$$\delta_j = \frac{\partial o_j}{\partial net_j}\sum_k \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial o_j} = f'(net_j)\sum_k \delta_k \frac{\partial net_k}{\partial o_j} \tag{3.4}$$

with:

- f - the activation function.

k - subsequent neuron of neuron j.
net_j - the input of neuron j.
o_i - the output of the neuron i.

In our work, ReLU function is used as the activation function. The derivative is :

$$f'(net_j) = \begin{cases} 0 & \text{for } net_j < 0 \\ 1 & \text{for } net_j \geq 0 \end{cases} \tag{3.5}$$

Now we only need to compute $\frac{\partial net_k}{\partial o_j}$. For convolutional layers, this is quite straight forward to obtain: $\frac{\partial net_k}{\partial o_j} = w_{jk}$. As for the pooling layers, we must distinguish between average pooling and max pooling:

$$\text{average pooling:} \quad \frac{\partial net_k}{\partial o_j} = \frac{1}{m} \quad \text{(m denotes the filter size)} \tag{3.6}$$

$$\text{max pooling:} \quad \frac{\partial net_k}{\partial o_j} = \begin{cases} 1 & \text{if } o_j = max(o) \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

So far, all the components needed can be calculated by the backpropagation algorithm effectively, and thus the gradients can be computed with a reasonable computational burden.

3.2 Optimizer

In the previous section, we explain how to compute the gradients of the cross-entropy error with respect to the parameters. The next step is to update the parameters accordingly. Several different methods have been proposed to cope with this task which are all based on gradient descent. In our experiment, we apply the **Stochastic Gradient Descent** and **Adaptive Moment Estimation**. We elaborate both approaches in section 3.2.1 and 3.2.2.

3.2.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is the simplest form of update to change the parameters along the negative gradient direction. As an on-line approach, SGD-Optimizer performs an update for each training sample, thereby reducing the computational cost at every iteration, which is especially effective for large-scale optimization problems.

$$\theta_t := \theta_{t-1} - \alpha \nabla_\theta E \tag{3.8}$$

with:

θ_t	-	the value of parameter θ at time step t.
α	-	the learning rate.
$\nabla_\theta E$	-	the gradient of the cost w.r.t. θ

One drawback of the SGD-optimizer is that it is tricky to set an appropriate learning rate α. Setting this value too high can cause the algorithm to diverge, because it overshoots the optimum at each each step, while setting learning rate too low makes it slow to converge. In the following section, we will discuss a method that adapts learning rate scale for different layers instead of hand picking manually as in SGD.

3.2.2 Adaptive Moment Estimation

Adaptive Moment Estimation (Adam) [KB14] computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. It is proved mathematically that this algorithm naturally performs a form of step size (learning rate) annealing and guarantees convergence. In practice, Adam-optimizer consistently outperforms other methods and achieves the state-of-the-are performance.

The detailed algorithm is as follows:
The first step is to compute the gradient at timestep t:

$$g_t = \nabla_\theta f_t(\theta_{t-1})$$

with:

g_t	-	the gradient at timestep t.
$\nabla_\theta f_t$	-	the derivative of the cost function w.r.t. θ at timestep t
θ_{t-1}	-	the value of parameter θ at time step $t-1$.

Then, Adam employs two variables m and v to denote the 1^{st} and 2^{nd} moment vector, respectively. After computing the gradient at timestep t, the biased first and second raw moment estimate is updated accordingly.

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$
$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

with:

m_t	-	the biased first moment estimate at timestep t.

m is initialized as 0. $m_0 = 0$

v_t - the biased second raw moment estimate at timestep t.
v is initialized as 0. $v_0 = 0$

g_t^2 - the elementwise square of g_t.

β_1 - the exponential decay rates for the 1^{st} moment.

β_2 - the exponential decay rates for the 2^{nd} moment.

And the bias-corrected moment estimates are computed based on the biased moment estimates:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

with:

$\hat{m}_t$ - the bias-corrected first moment estimate at timestep t.

$\hat{v}_t$ - the biased-corrected second raw moment estimate at timestep t.

β^t - β to the power t.

Finally, the update rule for the parameter is:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{(\sqrt{\hat{v}_t} + \epsilon)}$$

with:

α - the step size.

ϵ - the smoothing term which avoids division by zero.

This method has four hyperparameters, namely α, β_1, β_2 and ϵ. $\beta_1, \beta_2 \in [0, 1)$ and the recommended values are $\beta_1 = 0.9, \beta_2 = 0.999$.

3.3 Dropout

So far, we explained how to compute the gradient and how to update the parameters accordingly. Although we can start training already, there remains the potential to improve the performance. **Dropout** [SHK+14] is a simple and effective regularization technique, which helps to prevent overfitting. The key idea is to randomly drop some units during training, but keep all the neurons during testing. This prevents neurons from co-adapting too much, and thus performs well to avoid overfitting.

3.4 Batch Normalization

Batch Normalization (BN) is another recently introduced technique and is widely used in most modern network models. Ioffe *et al.* [IS15] noticed that the distribution of each layer's inputs changes during training, which makes deep neural network extremely hard to train. They called this phenomenon *internal covariate shift*, that the distribution of network activations changes due to the change in network parameters during training. The key idea of BN is to do normalization each scalar feature independently and make it have the mean of zero and the variance of 1. The formal definition of this normalization operation is as follows:

$$\widehat{x}^k = \frac{x^k - \mathbb{E}[x^k]}{\sqrt{Var[x^k]}} \tag{3.9}$$

with:

x - the input vector $x = (x^1, ..., x^d)$. This normalization will be done for each dimension separately

However, simply normalizing each input of a layer might change what the layer can represent. To address this problem, two additional variables γ^k and β^k are introduced, which scale and shift the normalized value:

$$y^k = \gamma^k \widehat{x}^k + \beta^k \tag{3.10}$$

These parameters are also learned along with the original model parameters, and help to recover the original activation learned by the network.

The mathematic details will not be presented here. In short, BN mainly solve the problem of gradient vanishing(explosion) and dramatically accelerate the training in practice.

3.5 Data Preprocessing and Augmentation

In the previous sections, we introduce the mechanisms for training the network. Now the last step before feeding data into the network and starting training is to preprocess the data. There are three main reasons why we should perform data preprocessing and augmentation. The first one is from the practical aspect: the images in a dataset might have various sizes, but the networks only accept input of a fixed size, hence the images should be resized before being fed to the network. Another reason is that theoretically the input data should be zero-centered and each dimension should have the standard deviation. Intuitively, by doing this kind of normalization, we prevent the

gradients from becoming too large and avoid data of each dimension have markedly different scale. This way, we ensure that the objective function is relatively easy to optimize, which means each update step is closed to the right direction points to the optimum instead of oscillating between the saddle points. Lastly, data augmentation decreases the chance of overfitting. The commonly used operations are horizontally flipping, randomly scaling, color distortion etc.

In our experiment, we performs cropping, rescaling, horizontally flipping, color distortion and normalization. The details are listed as follows:

Crop and Resize

As our dataset PlantCLEF [GBJ16] contains plant images of various size, all these input images should be rescaled to a certain size ($224 \times 224 \times 3$ for Inception-V2 model, $299 \times 299 \times 3$ for Inception-Resnet-V2 model). This is the most important part of the data augmentation, since this step might lead to information loss. In order to obtain better prediction results, we test two different methods in our experiment.

The first method we test is to randomly select a small region of the input image and then rescale this small pad to a fixed size. As a result, the shape of the original object in the image might change, since the rescaling can change the aspect ratio.

The other method is to define a size interval whose minimum is larger than the input size of a network, at first. Then we rescale images to a randomly selected size from that interval while preserving the aspect ratio. Afterwards, we randomly crop a path of size $224 \times 224 \times 3$ ($229 \times 229 \times 3$).

Color distortion

For color distortion, we distort the brightness, saturation and hue of the original image by a small factor. Since the color of a plant is also a feature that provide information, we also trained some models without doing color distortion.

Normalization

All the images in our dataset are color image of type *uint8*, i.e. the value interval is $[0, 255]$. We first change all the input images to type *float* and

subtract 0.5 from the values and multiply them by 2. Therefore, the final input values are in $[-1, 1]$.

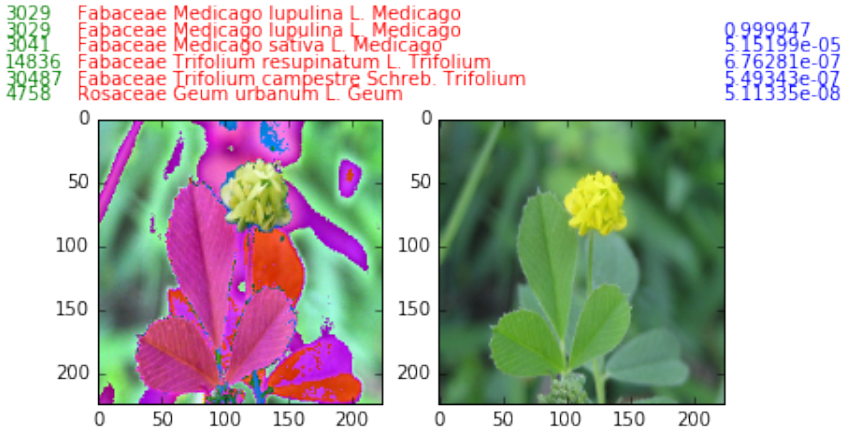
Chapter 4

Visualization

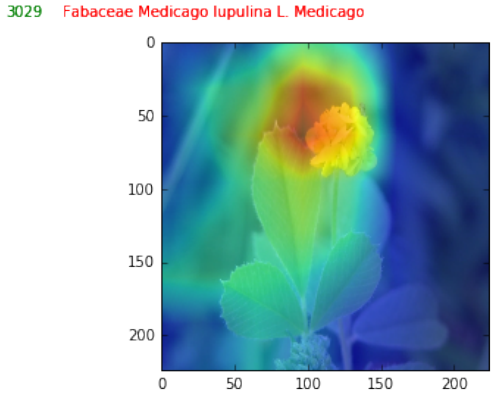
For fine-grained image classification, it is of significance to understand and interpret the classification results generated by the CNN models. To address this issue, we first applied *Class Activation Map* [ZKL+16](Chapter 4.1) and *Gradient-weighted Class Activation Map* [SCD+16](Chapter 4.2) approach to our finetuned Inception-V2 and Inception-Resnet-V2 models to localize the important region of an input image for making prediction, and the corresponding heatmaps are plotted. Subsequently, we use *Guided Backpropagation* [SDBR14] (Chapter 4.4), which is an extension of *Deconvolutional Network* [ZKTF10][ZF14] (Chapter 4.3), to capture high-resolution details learned by the CNN model in higher layers. Finally, we combine these methods to find the fine-grained details learned by the CNN models which greatly contribute to the classification result.

4.1 Class Activation Map

A simple visualization technique called *Class Activation Map* proposed in [ZKL+16] localizes the discriminative regions for a particular class by using the localization ability of a global average pooling layer. The basic idea follows from the fact that most of the CNN networks can retain their remarkable localization ability until the final layer. By definition, global average pooling outputs the spatial average of the feature map of each unit at the last convolutional layer and subsequently, a weighted sum of these values is calculated to generate the final output. Similarly, the *Class Activation Map (CAM)* is defined as the weighted sum of the feature maps of the last convolutional layer. It can be proved mathematically that the CAM directly indicates the importance of the image regions and can identify the discriminative image regions used by the CNN to make prediction.



(a) The classification output



(b) The CAM of a correct classified image

Figure 4.1: The classification output given by Inception-V2 model and the corresponding CAM of a corrected classified image of species *Fabaceae Medicago Iupulina L.Medicago*. (a) The first line show the classId (label) and the species name of the input image. The top-5 predictions and the output scores are shown in the following lines. The left subfigure is the normalized input image and the right one is the original image. (b) The important image region for discrimination is highlighted.

Given a neural network model with a global average pooling as the penultimate layer, the score output F_k of this layer can be computed as:

$$F_k = \frac{1}{N} \sum_{x,y} f_k(x, y) \tag{4.1}$$

with:

N	-	the number of pixels in the feature map
$f_k(x, y)$	-	the activation of k-th feature maps in the last convolutional layer at spatial location (x, y)

Then, the input s_c to softmax layer for a given class c is:

$$\begin{aligned} s_c &= \sum_k w_k^c F_k \\ &= \sum_k w_k^c \cdot \frac{1}{N} \sum_{x,y} f_k(x, y) \\ &= \frac{1}{N} \sum_k w_k^c \sum_{x,y} f_k(x, y) \\ &= \frac{1}{N} \sum_{x,y} \sum_k w_k^c \cdot f_k(x, y) \\ &= \frac{1}{N} \sum_{x,y} M_c(x, y) \end{aligned} \tag{4.2}$$

with:

w_k^c	-	weight corresponding to class c for the k-th feature map
M_c	-	the class activation map for class c. And the spatial element at location (x, y) is $M_c(x, y) = \sum_k w_k^c f_k(x, y)$.

Since the input to softmax s_c is proportional to the sum of the class activation map over all pixels, $M_c(x, y)$ can be interpreted as the importance of the activation at location (x, y) to classify the input image into the category c. However, the computation is based on the usage of a global average pooling layer. Thus, this method is restricted to the CNN network with a global average pooling layer. In this thesis, Inception-V2 and Inception-Resnet-V2 models are used, both of which use the global average pooling before the final softmax layer and thus CAM can be applied here.

4.2 Gradient-weighted Class Activation Map

Gradient-weighted Class Activation Map (Grad-CAM) proposed by Selvaraju *et al.* [SCD+16] is a generalization of the CAM approach, and is applicable to

a wide variety of CNN model-families. The basic idea is to use the gradient information the last convolutional layer of the network to compute the class discriminative localization map (Grad-CAM), which indicates the importance of each neuron to make a prediction.

The procedure to calculate the Grad-CAM of width u and height v for class c $L^c_{Grad-CAM} \in \mathbb{R}^{u \times v}$ works as follows: First, the gradient of class score y^c before the final softmax layer with respect to feature maps A^k of a convolutional layer $\frac{\partial y^c}{\partial A^k}$ are computed. Afterwards, these gradients flowing back are global average-pooled to get the neuron importance weights α^c_k.

$$\alpha^c_k = \frac{1}{Z} \sum_{i,j} \frac{\partial y^c}{\partial A^k_{ij}} \tag{4.3}$$

with:

y^c	-	the score of class c before the final softmax layer
Z	-	the size of the pooling region
$\frac{\partial y^c}{\partial A^k_{ij}}$	-	the gradient of the class score w.r.t. the k-th feature map A^k at location (i, j)

Then the Grad-CAM is defined as the weighted combination of forward activation maps followed by a ReLU unit. A ReLU nonlinearity is used here because only the features that have a positive influence (i.e. pixels whose intensity should be increased in order to increase y^c) on class c matter.

$$L^c_{Grad-CAM} = ReLU(\sum_k \alpha^c_k A^k) \tag{4.4}$$

4.3 Deconvolutional Networks

Deconvolutional Networks (deconvnet) [ZKTF10] proposed by Zeiler *et al.* in 2010 are currently widely used in Semantic Segmentation, Covolutional Sparse Coding, CNN Visualization [ZF14], etc. Given a high-level feature map, the deconvnet approach inverts the data flow of a CNN, going from neuron activations in the layer which we are interested in down to the input layer. If a single neuron is non-zero in the high level feature map, then the reconstructed image shows the part of the input image that is most strongly activating this neuron, and hence represents the region that is most discriminative to this neuron.

Typically, a convolutional layer consists of three main portions (See Chapter 2), namely the convolution, max-pooling and a nonlinear activation function

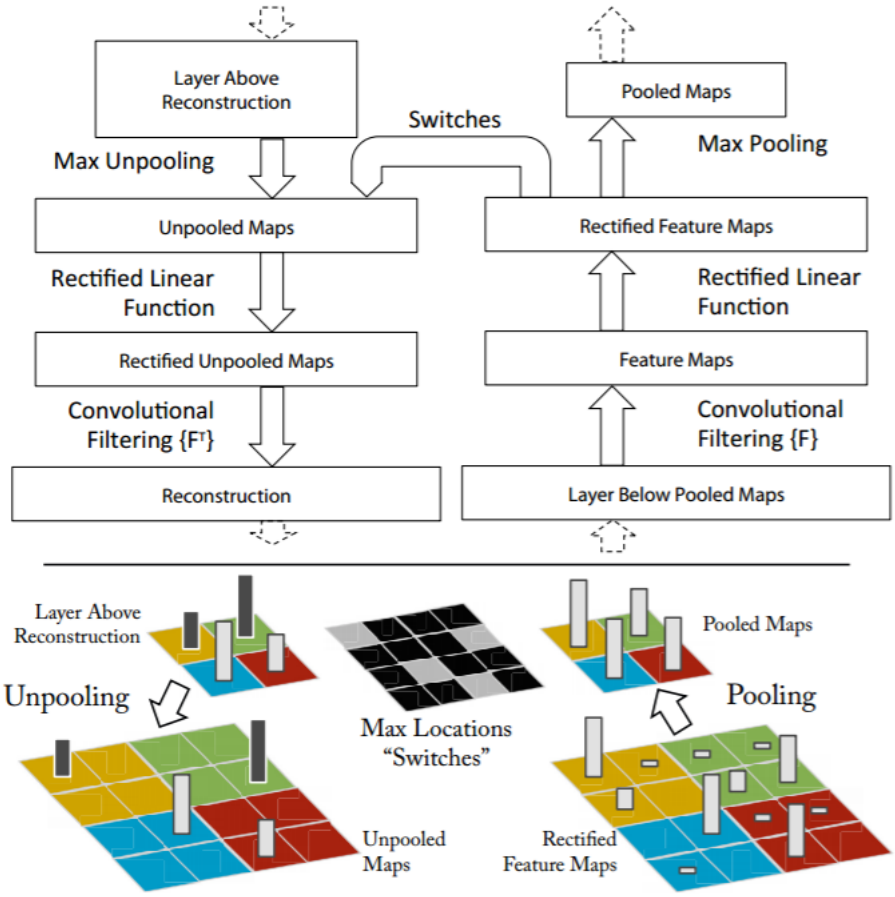


Figure 4.2: The top left part shows a deconvnet layer corresponding to a convnet layer on the top right region. The bottom part illustrates the unpooling operation in a deconvnet: 'switches' record and store the location of the maximum in each pooling region during a forward pass. In the backward pass, the values will be passed to the entries recorded in 'switches'. (The black and white bars represent negative and positive activations respectively. [ZKTF10]

(ReLU in this thesis). In order to obtain a reconstruction from a high layer to the input layer, the reverse of these three operators should be defined explicitly.

In the deconvnet approach, the reversed process of convolution is simply a regular convolutional layer with its filters transposed. By applying these transposed filters to the output of a convolutional layer, the input can be retrieved. However, the reverse of a max-pooling operation is non-trivial, as data about the non-maximum features can be lost. The paper describes a method named *'switches'* in which the positions of each maximum is recorded and saved during forward pass, and when gradients are passed backwards, they are placed where the maximums had originated from. Finally, to reverse the ReLU unit is the easiest. All we need to do is to pass the data through a ReLU again when propagating backwards.

4.4 Guided Backpropagation

An alternative method called *Guided Backpropagation* [SDBR14] visualizes the part of an image that most activates a given neuron by computing the gradient of the class score with respect to the input image activation with respect to the input image. As a gradient-based method, it is indeed a generalization of deconvolution network (deconvnet) and these two methods differ mainly in the way they handle backpropagation through the rectified linear (ReLU) nonlinearity. The relation between gradient-based visualization approach and deconvolution network is discussed in [SVZ13].

Both guided backpropagation and deconvnet perform a simple backward pass of the activation of a single neuron after a forward pass through the network. They are equivalent to a normal backward pass, except that when propagating through a ReLU nonlinear function. As illustrated in Figure 4.3, deconvnet computes the gradient merely based on the gradient signal coming from the higher layer and pass a 0 to the lower layer for the negative entries, whereas the normal backpropagation only propagates the gradient if the corresponding entries in the lower layer is non-negative. (See section 3.1 for detailed explanation)

Guided Backpropagation is a combination of deconvnet and backpropagation: Once at least one of the values from the lower layer and from the higher layer is negative, this value will be masked out and stop flowing back. This process gets its name since it adds an additional guidance signal from the higher layers to usual backpropagation, which prevents the backward flow of negative gradients. Therefore, the gradient signals only pass through the neurons which increase the activation of the higher layer unit we aim to visualize, and

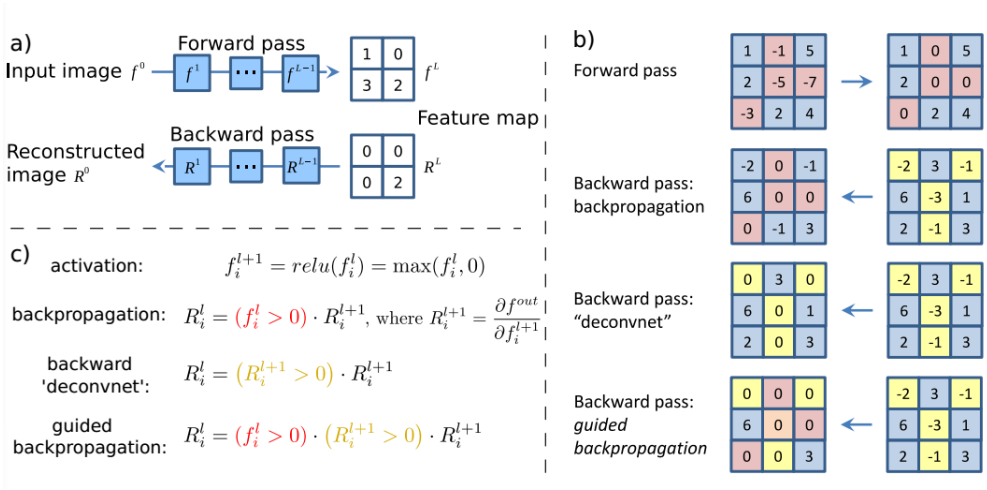


Figure 4.3: a) In order to obtain a reconstruction of a given input image, a forward pass should be performed at first. Subsequently, only the value of one entry which we are interested in will be kept and all other values will be set to zero. After propagating back to the input layer, the result is the reconstruction of the given input. b) A comparison of different methods for propagating back through a ReLU nonlinear function. c) Formal definition of different methods for propagating an activation back through a ReLU function at layer l; [SDBR14]

thus the image region which contributes to the discrimination can be shown in the reconstruction.

In conclusion, both the deconvnet approach and guided backpropagation compute an imputed version of the normal gradient. In practice, guided backpropagation generated a better visualization result since it is in general more robust than the deconvnet approach.

Chapter 5

Evaluation

In this chapter, we present the results of our CNN models on plant image classification task. First, we provide a brief introduction of the PlantClEF dataset in section 5.1. Next, we evaluate our models on the test dataset using two measures: the top-1 accuracy and top-5 accuracy, and outline the results generated by different data preprocessing methods in section 5.2. Finally, we interpret our CNN models by using visualization approaches described in chapter 4.

5.1 Dataset

PlantCLEF 2015 [GBJ16] dataset contains 113205 RGB pictures belonging each to one of the 7 types of view (Branch, Fruit, Stem, Leaf, Leaf-Scan, Flower, Entire) reported into the meta-data, in a xml file with explicit tags. The whole dataset is divided into a training set and a test set.

The training data finally results in 27907 plant observations illustrated by 91759 images with complete xml files associated to them.

The test dataset contains 21446 images with purged xml files, i.e some information is missing.

5.2 Classification Results

In the testing stage, we evaluated our fine-tuned Inception-V2 and Inception-Resnet-V2 models using the top-1-accuracy and top-5-accuracy metric. Top-1 accuracy refers to the proportion of correctly classified images, and the top-5 accuracy is computed as the proportion of images such that the ground-truth

category is one of the top-5 predicted categories.

We used the training set of PlantCLEF 2015 for fine-tuning the pre-trained Inception-V2 and Inception-Resnet-V2 models. The training parameters are set as following: learning rate=0.001, batch size=32, the dropout keep probability=0.8, the input size is $224 \times 224 \times 3$ for Inception-V2 and $299 \times 299 \times 3$ for Inception-Resnet-V2 model (The third dimension corresponds to the RGB channels). Besides, we took 20% of the training data as the validation set and tested our model on the validation set to see how the validation error changed as the learning processed.

network	data preprocessing	optimizer	top-1 (%)	top-5 (%)
Inception-V2	method A	Adam	49.32	72.08
Inception-V2	method B	Adam	52.05	74.03
Inception-V2	method C	Adam	50.05	72.53
Inception-V2	method B	SGD	58.54	80.07
Inception-Res-V2	method A	Adam	56.03	76.99
Inception-Res-V2	method B	Adam	55.84	77.50
Inception-Res-V2	method C	Adam	55.54	76.72
Inception-Res-V2	method B	SGD	65.05	84.57

Table 5.1: classification results on the PlantCLEF 2015 test dataset.

In order to improve the performance, we tried three different combinations of data augmentation methods during training, which are described in chapter 3.5. We named them method A, B, C and the precise details are listed here:

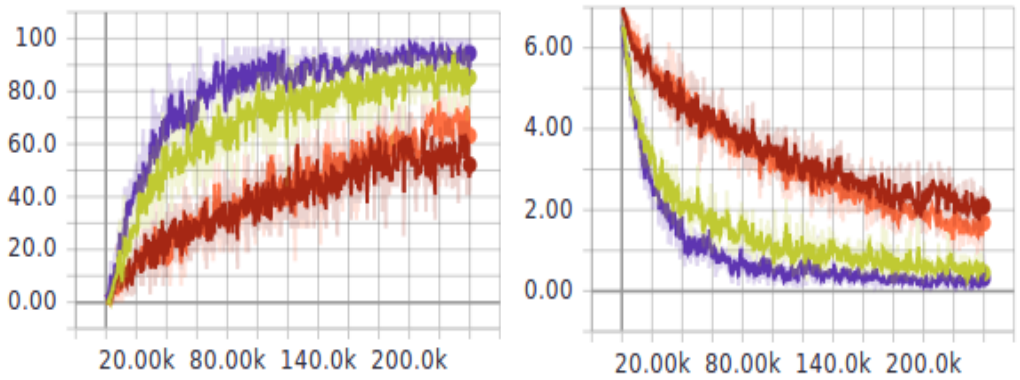
- **method A**: Randomly select a small region of the input image and then rescale this small pad to a fixed size.(This might change the aspect ratio of the original object.) Then we normalized the input image, so the input data values are in the interval $[-1, 1]$. (See chapter 3.5 for more details.)

- **method B**: The cropping and rescaling are identical to method A. But now, the input image can randomly be horizontally flipped and subtle color distortions, e.g. change of the brightness,contrast,hue and saturation, are applied. Finally, the input data is normalized as described in method A.

- **method C**: We first defined the size interval, which is $[256, 512]$ for Inception-V2 and $[320, 512]$ for Inception-Resnet-V2. Then we rescaled the input image so that the smallest side of the image is equal to a randomly selected number from that interval while preserving the aspect

ratio. Afterwards, we randomly cropped a small part from the resized image. The final normalization is identical to method A and B.

During testing, we used a preprocessing approach similar to method C. First, we rescaled the image to a certain size while preserving the original aspect ratio. Then, we performed central cropping and fed the center part into the network.

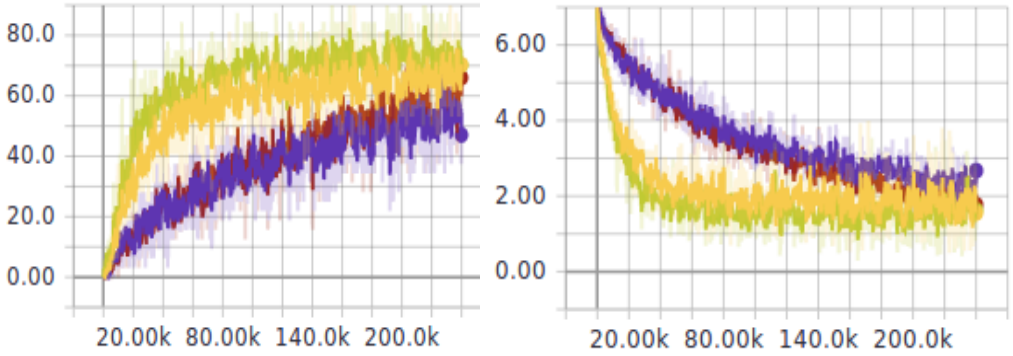


(a) training accuracy (b) training loss

Figure 5.1: (a): The accuracy (in %) of the input training batch during the training is present. Form top to bottom: the purple line - Inception-Resnet-V2 model using Adam optimizer; the greenyellow line - Inception-V2 modle using Adam optimizer; the orange line-Inception-Resnet-V2 model using SGD optimizer; the dark red line- Inception-V2 model using SGD optimizer. (SGD-optimizer took much longer to converge, we only plot the first 240,000 iterations here.) (b): the cross entropy loss of the training batch.

As shown in Table 5.1, the data preprocessing methods exerted an influence on the final prediction accuracy. In general, the model achieved a higher accuracy when color distortion was performed, but the effect of preserving the original aspect ratio is hard to assess. Besides, we noticed that the Inception-Resnet-V2 model always achieved better results when compared with Inception-V2. It is quite straightforward since the network architecture of the Inception-Resnet-V2 model is much more complicated and can extract more information intuitively. However, one iteration took a longer time for Inception-Resnet-V2 and the training process required more time in total.

In our experiments, we also tested both the SGD and Adam optimizer. We noticed that both network models required much more iterations to converge when using SGD optimizer, but the SGD outperformed the Adam optimizer in terms of the prediction accuracy on the test dataset, i.e. the models trained by using SGD optimizer generalize much better. This is an



(a) validation accuracy (b) validation loss

Figure 5.2: (a): The accuracy (in %) of the validation set during the training is present. Form top to bottom: the green line - Inception-Resnet-V2 model using Adam optimizer; the yellow line - Inception-V2 modle using Adam optimizer; the dark red line-Inception-Resnet-V2 model using SGD optimizer; the purple line- Inception-V2 model using SGD optimizer. (SGD-optimizer took much longer to converge, we only plot the first 240,000 iterations here.) (b): The loss on the input batch of the validation set.

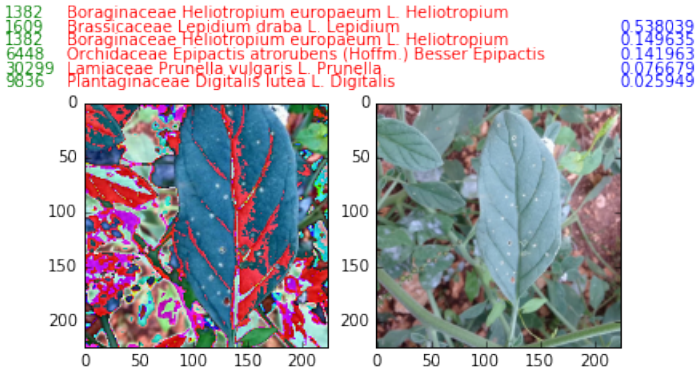
interesting phenomenon because intuitively Adam should be better than SGD optimizer. Figure 5.1 shows the training curve of Inception-V2 and Inception-Resnet-V2 networks. Figure 5.2 presents the validation accuracy and loss during training. And the classification outputs, i.e. the ground-truth category, the top-5 predicted categories and the corresponding prediction scores, are visualized in Figure 5.3(a).

As for the technical details, our implementation is based on the publicly available tensorflow-slim toolbox [GS]. The training is performed on a single NVIDIA TITAN X GPU with 12 GB memory.

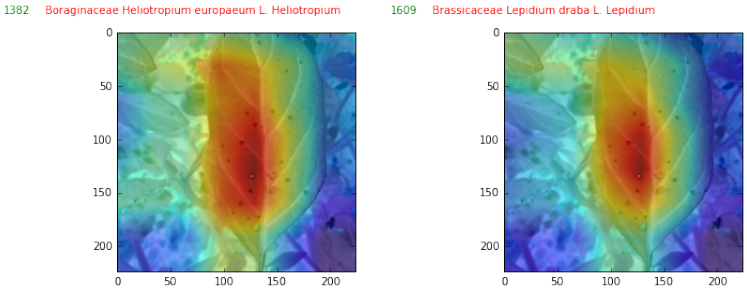
5.3 Visualizing models

In this section, we applied the network visualization approaches introduced in chapter 4 on our fine-trained models.

In our experiment, we first applied the class activation map algorithm on our fine-tuned models to generated the CAM (Chapter 4.1) for the ground-truth category and the top-1 prediction, respectively. Figure 5.3 demonstrates an example generated by Inception-V2 model. In this example, the model did not make a correct prediction, but the ground-truth category is inside the



(a) The classification output



(b) CAM for the ground-truth (c) CAM for the first prediction category

Figure 5.3: (a): The classification results given by a fine-tuned Inception-V2 model. The top-5 predictions and the ground-truth category are shown on the top. The first line refers to the true classId and the species name of the input image. The top-5 predictions are sorted according to the prediction scores and are shown in the following lines. The left image is the normalized input which is fed to the network. The right image is the original input image directly rescaled to 224×224. **(b):** CAM for the ground-truth category. The discriminative region are highlighted in the image. **(c):** CAM for the top-1 prediction, whose classId and the species name of the top-1 prediction are shown above the image. The discriminative region are highlighted. Notice that the CAM differs for different class.

top-5 predicted categories. Figure 5.3(b) and 5.3(c) show the regions which are important for the model to predict the input image as of a certain class. When comparing the CAM in 5.3(b) and 5.3(c), we notice that the strongly activated regions differ when making different decision, which means the important regions are not the same for different categories. But the strongest response is still inside the leaf object in the input image, which makes sense intuitively.

Figure 5.4 shows some examples of CAMs and the original plant images of the PlantCLEF 2015 test dataset. All these images are correctly classified by the Inception-Resnet-V2 model. As shown in the figure, the highlighted regions, i.e. the discriminative image regions used for categorization, in general approximate the extent of important plant organ objects in images, e.g. the fruit,leaf and flower, and the backgrounds are not or only slightly activated.

As for fine-grained classification, it is necessary not only to localize the important regions but also identify the fine-grained details in the images which contribute to categorizing. To address this problem, we apply the guided backpropagation algorithm [SDBR14] (chapter 4.3) to our fine-tuned models to find high-resolution fine-grained details learned by our networks. Then we apply the Gradient CAM approach [SCD$^+$16] (chapter 4.2), which is also a localization technique similar to CAM, to our network models. At the end we combined Grad-CAM and guided backpropagation approaches to visualize the high resolution details used by the CNN models to make predictions. (See Figure 5.5)

As illustrated by Figure 5.5, our model successfully localized the region where there exist a plant organ object, although no supervision on the location of the object was provided. In our examples, our fine-tuned Inception-Resnet-V2 model recognized the flowers and leaves in the input images and ignored the background. Besides, our model captured the fine-grained features which are intuitively important for classifying the plant images correctly, in this case, the contour of the flower petals and the leaf veins.

We also investigated how the features evolve when the network go deeper. To do this, we first fed an input image into our fine-tuned network model and let it perform a forward pass. Subsequently, we used guided backpropagation to pass the gradient information down to pixel space.(See chapter 4.4 for more details.) This way, we obtained the strongest activation for a given feature map of each layer. When projecting down to the input layer, this strongest activation get displayed and the visualization corresponds to the detailed features captured by different network layers. As shown in Figure 5.6, the first convolutional

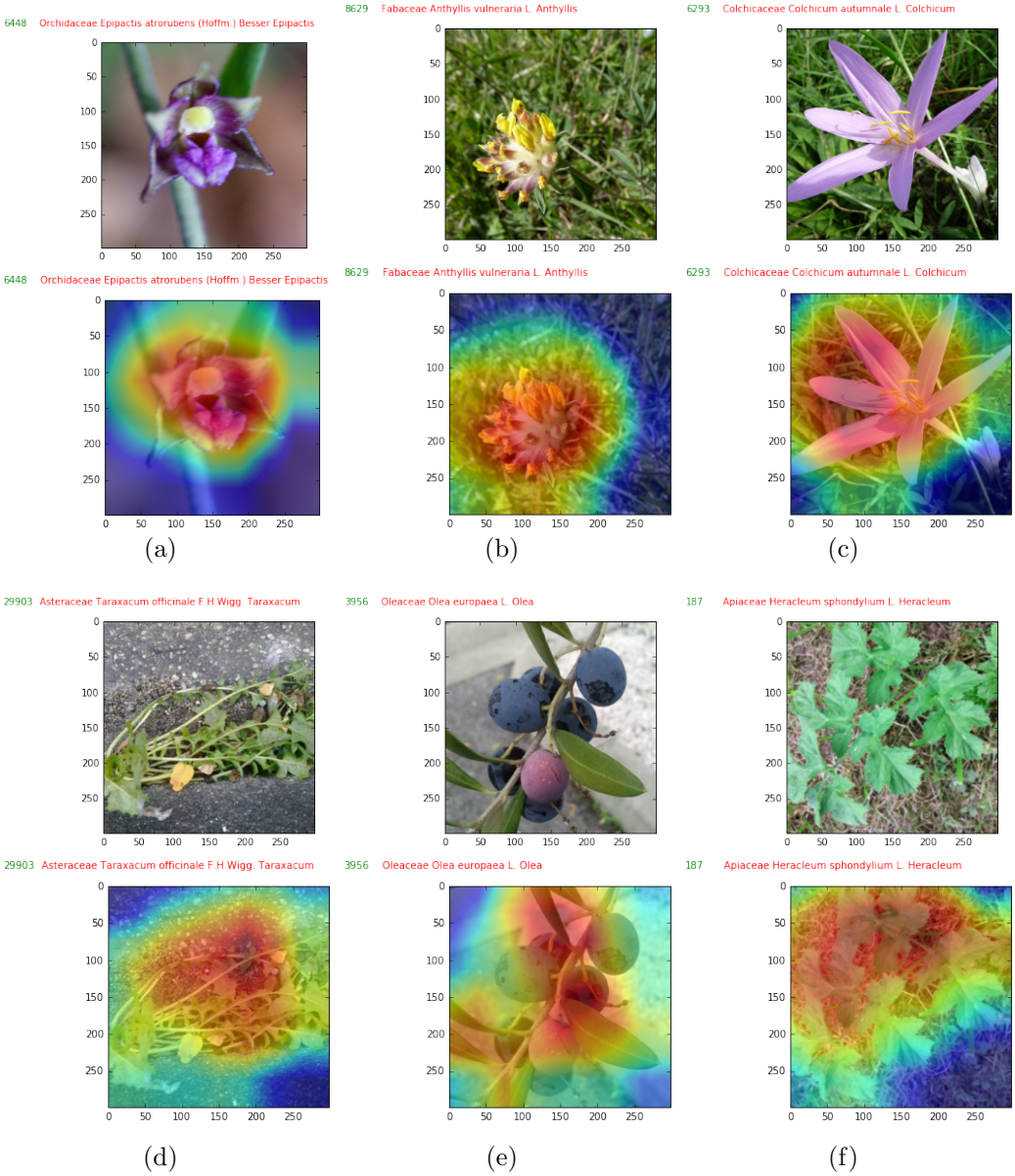
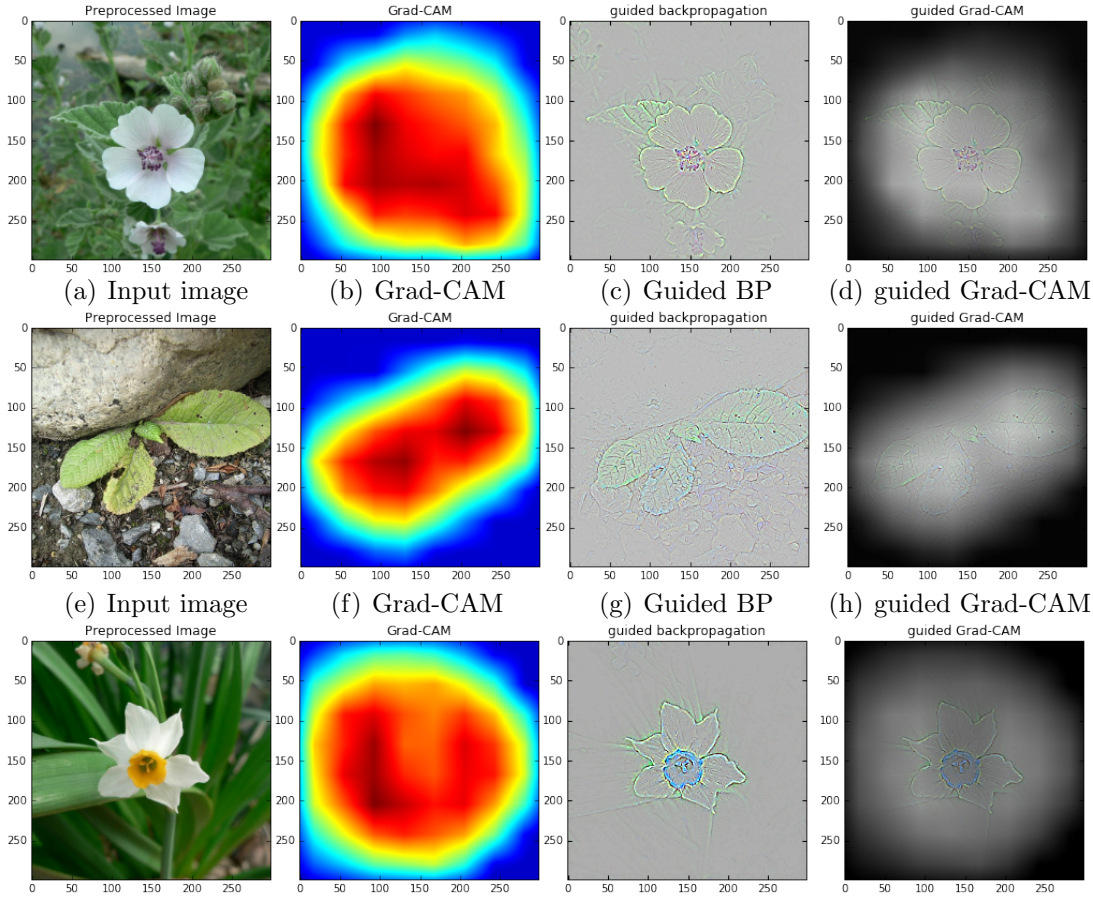


Figure 5.4: The input images and the CAMs for correctly predicted plant images. The red regions are the class discriminative regions. **(a):** a flower of species *Orchidaceae Epipactis atrorubens (Hoffm.) Besser Epipactis* with classId 6448. **(b):** a flower of species *Fabaceae Anthyllis vulneraria L. Anthyllis* with classId 8629. **(c):** a flower of species *Colchicaceae Colchicum autumnale L. Colchicum* with classId 6293. **(d):** the stems of species *Asteraceae Taraxacum officinale F.H.Wigg. Taraxacum* with classId 29903. **(e):** the fruit of species *Oleaceae Olea europaea L. Olea* with classId 3956. **(f):** the leaves of species *Apiaceae Heracleum sphondylium L. Heracleum* with classId 187.

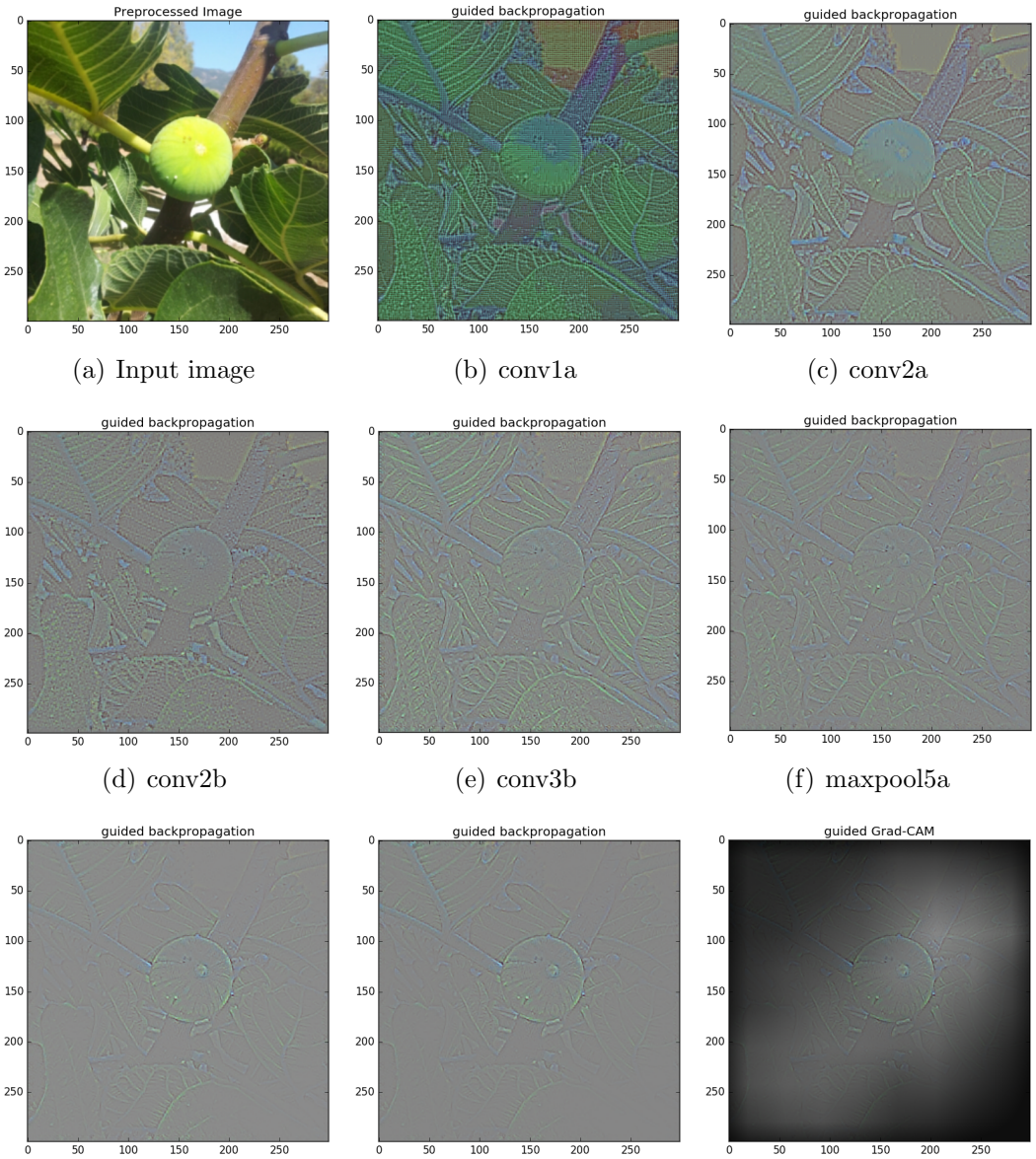


(a) Input image (b) Grad-CAM (c) Guided BP (d) guided Grad-CAM

(e) Input image (f) Grad-CAM (g) Guided BP (h) guided Grad-CAM

(i) Input image (j) Grad-CAM (k) Guided BP (l) guided Grad-CAM

Figure 5.5: (a)(e)(i): the unnormalized resized input image; **(b)(f)(j)**: The Gradient Class Activation Maps are shown by Heat maps. The redder the area, the more important the region for the prediction is; **(c)(g)(k)**: Guided backpropagation visualizes the detailed features captured by the last convolutional layer which contribute to the classification (The contrast is artificially enhanced); **(d)(h)(l)**: Combining Grad-CAM and Guided BP gives Guided Grad-CAM, which displays high resolution features that are important for classifying the plant species in the input image; 1st row: *Species name: Malvaceae Althaea officinalis L. Althaea*(ClassId: 3806); 2nd row: *Species name: Primulaceae Primula vulgaris Huds. Primula*(ClassId: 4353); 3rd row: *Species name: Amaryllidaceae Narcissus tazetta L. Narcissus*(ClassId: 5824)



(a) Input image (b) conv1a (c) conv2a

(d) conv2b (e) conv3b (f) maxpool5a

(g) mix7a (h) conv7b (i) guided Grad-CAM

Figure 5.6: Visualization of features captured by different layers of fine-tuned Inception-Resnet-V2 model using guided backpropagation. From (a) to (h), as the layer becomes higher, the feature is more discriminative. In this example, the last convolutional layer, i.e. conv7b in (h), capture the contour and the texture of the fruit. (i): The bright region shows the detailed feature which is the most important for making prediction. This image show a fruit of speices Moraceae Ficus carica L. Ficus(ClassId: 30126)

layer 5.6(b) roughly recognizes all the edges in the original input image. As the layer becomes higher, some background information is ignored and the last convolutional layer extracts the contour and the texture of the fruit object. The fine-grained features which are most discriminative, i.e. most important for distinguishing between different species, are displayed in the bright region in Figure 5.6(i).

Chapter 6

Discussion and Conclusion

6.1 Discussion

Our main goal is to classify similar images containing plant organs into different categories of plant species with deep convolutional neural network. The PlantCLEF task is comparable with the Imagenet challenge since both datasets contain 1000 classes of objects. But the fine-grained plant image classification task is even harder because of the large within-class variance and the small between-class variance, e.g. leaves of different species can be extremely similar and only have subtle differences but images of the same category might contain different plant organs which are clearly different. So far, the deep CNN have achieved the best performance on this task regarding the classification accuracy. Therefore, we adopted deep CNN models for this fine-grained classification challenge.

Instead of training a network model from scratch, we use the pre-trained CNN models and fine-tune them on our dataset. There are two reasons why we followed this simple fine-tune strategy: First, the computational cost is huge and it is almost infeasible for us to train from scratch since we only possess one GPU with limited memory (12 GB) and it will take too long to converge. The second reason is that the lower layer features are always simple and similar for different dataset and thus can be shared. But it is more likely that the model is trapped into local minimum if trained from scratch due to bad initialization.

Considering the aforementioned reasons, we fine-tuned two different CNN models pre-trained on ImageNet, namely Inception-V2 [IS15] and Inception-Resnet-V2 [SIVA16], which are modified versions of the original Google-net architecture [SLJ+14]. The basic component of both these network models are called Inception-module which is design based on the idea of multiscaled features mentioned in [LCY13]. Each Inception-module contains small filters

of different size to capture features of different scaled and all these features are then combined by performing a filter concatenation at the end of each module. Small filters are used in order to reduce the number of model parameters, and in particular, a 1×1 filter is used to reduce the dimension of input data while increase nonlinearity at the same time. Both of them performs well on the ImageNet dataset, i.e. Inception-V2 achieves a 73.9% top-1 accuracy while Inception-Resnet-V2 obtains a 80.4%. For our PlantCLEF 2015 dataset, both of these models approximately achieve 100% accuracy during training and the best Inception-Resnet-V2 model achieved 65.05% top-1 accuracy, 84.57% top-5 accuracy and the best Inception-V2 model generated 58.54% top-1 and 80.07% top-5 accuracy. These results are worse than that on the ImageNet dataset mainly because of the higher intra-class variance of our dataset. Since each category contains images of different plant organs, which means clearly different input images might be of the same class, it is hard for the last fully connected layer in our models to learn appropriate weights, i.e. the models can hardly find a good feature combination which best represents each category. Thus, the classification performance might be improved, if we modify the existing network architecture to allow different combinations of high level features when making prediction. Since there is no promising solution to this problem so far, there remains a lot to do for further research.

We also noticed that different data-preprocessing methods affected the test results to some degree but are not the decisive factor. And it is hard to determine which method works best because of the variance during training. In general, using color distortion during training improved the network performance. The reason might be that color distortion has the effect of preventing overfitting. In addtion, we found that resizing the image while preserving the aspect ratio always worked better when doing evaluation on the test dataset, and it increased the accuracy by around 2%.

Besides, we investigated two different approaches while performing parameter update, namely the SGD and the Adam [KB14]. The SGD is the simplest gradient descent algorithm but performed surprisingly well on our test dataset. Although it took much longer to converge (Inception-Resnet-V2 needed 240000 iterations, while Inception-V2 required approximately 450000 iterations.) and the training accuracy was relatively low when using SGD-optimizer, the model generalized much better than Adam-optimizer. However, the Adam-optimizer, as shown in our learning curve (Figure 5.1), greatly accelerated the training process by computing individual adaptive learning rates for different parameters. It might be an interesting further topic to understand the behaviors of different optimizers and improve them.

As mentioned in Chapter 1, it is of significance to understand how deep CNN models distinguish different plant species. Specifically, for fine-grained image classification it is necessary to ensure that our models indeed learn the most discriminant features for each category and ignore the irrelevant background correctly. Otherwise they are not reliable and can result in a large test error when generalized to different test dataset. To tackle this issue, we employed different visualization approaches to interpret how CNN model making predictions. As shown in Figure 5.4 and 5.5(b)(f)(j), CAM and Grad-CAM localize the class-discriminative regions of the input images. And our fine-tuned model recognized the plant objects even though there are no location information provided. From this fact we deduce that our model learned to localize the important target objects, e.g. flowers and leaves, at first, and learned the fine-grained details next during the training.

In our next step, we utilized Guided Backpropagation [SDBR14] to visualize the high resolution features captured by the last convolutional layer of our CNN models. And the results are plotted in Figure 5.5(c)(g)(k), which shows the contour of the flowers and the leaves vein in our examples. After combining the Guided Backpropagation with the Grad-CAM methods, we obtain our results shown in Figure 5.5(d)(h)(l). Actually, we obtained the results we expected, i.e. the model looked at the most discriminative fine-grained details (e.g. the flower petal shape, the leaves shape etc.) when making prediction. This provided us inspiration that it might be helpful if we can extract these details and can train the network to select different features when receiving input images contain different types of plant organs.

In Figure 5.6, we investigated the evolution of features when the layers go higher. In this example, the contour of the fruit become clearer from the lowest layer to the highest layer. And at the last convolutional layer, only the center part which included the fruit object are activated and the background are ignored. When we applied this procedure for other images which contains the same plant species but different plant organs, we found that in lower layers, the pixels which contain edge information are strongly activated. As the layer becomes higher, the intuitively most discriminative part of the input image, e.g. the leaves, flowers and fruits, have intenser response. This indicates that networks are able to learn the discriminative features of a plant species no matter which kind of plant organ is provided, which means the network models inherently have the generalization capability to some extent. It might be an interesting topic for further research to understand how much generalization capability each model has, where this ability comes from and how to improve the model architecture design in order to reduce the generalization error.

6.2 Conclusion

In this thesis, we tackle the fine-grained plant image classification task by fine-tuning pre-trained Inception-V2 [IS15] and Inception-Resnet-V2 [SIVA16] models on the PlantCLEF 2015 dataset [GBJ16]. Furthermore, we investigate different data-processing methods and evaluate the model performance on the PlantCLEF test dataset and show our best model yield an acceptable result (65.05% top-1 and 84.57% top-5 accuracy). Finally, we employ the visualization techniques including CAM [ZKL+16], Grad-CAM [SCD+16], Guided BP [SDBR14] and combine them to illustrate that our fine-tuned models learned the discriminative features of each species and used them for categorizing.

Bibliography

[DLW+] Jian DONG, Min LIN, Yunchao WEI, Qiang CHEN, Wei, XIA, Hanjiang LAI, and Shuicheng YAN. Nin, good. http://image-net.org/challenges/LSVRC/2014/slides/ILSVRC2014_NUS_release.pdf.

[DV16] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. https://arxiv.org/abs/1603.07285, 2016.

[GBJ16] Hervé Goëau, Pierre Bonnet, and Alexis Joly. Plant identification in an open-world (lifeclef 2016). In *Working Notes of CLEF 2016-Conference and Labs of the Evaluation forum, Évora, Portugal, 5-8 September, 2016.*, pages 428–439, 2016.

[GS] Sergio Guadarrama and Nathan Silberman. Tensorflow-slim. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/slim.

[HSM+00] Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, (405):947–951, 2000.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. https://arxiv.org/abs/1512.03385, 2015.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. https://arxiv.org/abs/1502.03167, 2015.

[JWR+09] Deng Jia, Dong Wei, Socher Richard, jia Li, Li Kai, and Fei fei Li. Imagenet: A large-scale hierarchical image database. *CVPR*, 2009.

[Kar] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition. http://cs231n.github.io/neural-networks-1/. Accessed 10th May 2016.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980, 2014.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LCY13] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. https://arxiv.org/abs/1312.4400, 2013.

[NBJ02] George E Nasr, EA Badr, and C Joun. Cross entropy error function in neural networks: Forecasting gasoline demand. In *FLAIRS Conference*, pages 381–384, 2002.

[NH10] Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines. *ICML*, 2010.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[RHW+88] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[SCD+16] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. https://arxiv.org/abs/1610.02391, 2016.

[SDBR14] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. https://arxiv.org/abs/1412.6806, 2014.

[SHK+14] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[SIVA16] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. `https://arxiv.org/pdf/1602.07261.pdf`, 2016.

[SLJ+14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. `https://arxiv.org/abs/1409.4842`, 2014.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. `https://arxiv.org/abs/1312.6034`, 2013.

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.

[TG17] Shashank Tyagi and Ishan Gupta. Spatial transformer networks. `https://cseweb.ucsd.edu/classes/sp17/cse252C-a/CSE252C_20170522.pdf`, 2017.

[Wik] Wikipedia. Artificial neuron — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Artificial_neuron`.

[ZF14] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *ECCV*, 2014.

[ZKL+16] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. *CVPR*, 2016.

[ZKTF10] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. Deconvolutional networks. *CVPR*, 2010.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum	Unterschrift
Tübingen, 24.08.2017	Dingfan Chen