university of groningen

faculty of science and engineering

Master's Thesis

Deep Attribute Learning for Zero-shot Recognition and Explainability

Paul Ozkohen
s2575973

Primary Supervisor: **Dr. M.A. Wiering** (Artificial Intelligence, University of Groningen)
Secondary Supervisor: **prof. Dr. L.R.B. (Lambert) Schomaker** (Artificial Intelligence, University of Groningen)

Contents

List of symbols

Vectors and matrices will be displayed in **bold**, matrices are capital letters.

$\mathbf{X}$	Input matrix
$\mathbf{Y}$	Class label matrix
$\mathbf{Y^a}$	Attribute label matrix
n	number of input vectors
$\mathbf{x}^i$	i-th example
$\mathbf{y}^i$	i-th label
D	Dataset containing triplets of examples, class labels and attribute labels
N	dimensionality of the input data
C	number of classes
A	number of attributes
$\mathbf{W}^h$	Hidden layer weight matrix
$\mathbf{W}^o$	Output layer weight matrix
$\sigma(\mathbf{x})$	Activation function
$\mathbf{S}$	Feature map
$\mathbf{K}$	Convolution kernel
$\mathbf{k}$	Number of convolution kernels
$\mathbf{I}$	3-dimensional input image
$\mathbf{p}(\mathbf{a}\vert\mathbf{x_i})$	The attribute predictions for the i-th example
$\mathbf{L^D}$	Class-attribute lexicon for dataset D

List of Figures

Chapter 1

Introduction

1.1 Introduction

Deep learning has received a lot of attention in recent years, due to its success in various fields such as computer vision, signal processing and natural language processing. In 2012, the so-called 'Deep Learning Revolution' was started when a deep, Convolutional Neural Network (CNN) was used to win the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [10]. Since then, deep learning models have been applied to a variety of different tasks instead of more traditional approaches and have achieved state of the art results [11].

While deep learning achieves superior performance in most domains, a recent concern has arisen with respect to the explainability of such deep learning models. These models excel at recognition tasks, but it is often not clear how predictions are made by the model. This is a result of complex, non-linear interactions of many layers of artificial neurons within such deep Neural Networks that leads to their decisions not being interpretable for humans. The application of such 'black box' systems leads to difficulties in certain fields such as medicine, where transparent models are preferred over more accurate but opaque ones. If an automatic system detects a severe disease in a patient, it might not be ethically sound to blindly trust the system without understanding why this classification was made [12, 13]. A recent trend of Explainable Artificial Intelligence (XAI) has surfaced, whose aim it is to make AI systems more transparent, explainable and applicable [14].

Another downside of standard deep learning models is that they require lots of data to train the millions of parameters that these models contain. If the system has to learn to distinguish between different classes, enough samples from all categories should be present. If this is not the case, problems arise as models can become imbalanced and biased towards the more frequently available categories. Moreover, even if the samples for categories are well balanced, a lack of sufficient samples for each can still prevent the network from correctly distinguishing between them. The problem is that not every category might have enough samples, as certain categories may be quite rare. Another problem of standard deep models is that if the system needs to recognize a new class after deployment, the entire model has to be completely retrained keeping this new category in mind, which is costly. Standard deep models do not have the capacity to transfer the learned knowledge from the categories that have been seen to new categories. An exception to this is that a model such as the CNN can be pretrained on some categories, and part of the extracted features can be re-used for a different recognition task. Still, for new classes, the model has to be retrained.

Certain approaches for dealing with this issue of transfer learning have been developed, often called zero-shot or few-shot recognition, whose name is derived from the fact that zero or only very few data points from certain classes that need to be recognized are given [15]. A good zero-shot model is able to train on a set of classes and recognize new, unseen classes by using knowledge gained from the seen classes, without the need for any additional training after the training phase. Such models can be trained on categories that have plenty of samples, then employed to recognize rare ones. Another upside of such a model is that it is not limited in the amount of classes it can recognize, like standard models. Theoretically, new classes can be created on the fly and recognized without having to do any re-training, as long as the new categories are well-defined and share components with the training classes. Recognition of unseen instances by using learned components is also related to how humans learn new categories [16].

1.2 Visual attributes

A possible solution for dealing with the aforementioned issues is introducing an intermediate layer of attributes that lies in-between input data and output classes. By doing this, we can go from recognition of classes to recognition of attributes that a class consists of, where the attributes are easy to interpret for humans [17]. Examples of such attributes include various visual characteristics such as color and shape of an object. More specific visual attributes can be defined, such as the type of fur or teeth of an animal. Other attributes can include visible features of people such as the type of clothes and hairstyle being worn or whether a person is male or female. A model trained on these attributes offers more interpretable predictions. Instead of only predicting an animal to be a cat or a dog, the model can also describe what the animal looks like, what its diet consists of or the environment it lives in, which allows a model to gain a greater understanding of the content in images.

Besides offering more explainability, these attributes also allow for transfer of knowledge from one class to other, possibly unknown ones [8,18]. By specifying for different concepts what attributes they should typically have, a class-attribute lexicon can be built. By learning to recognize attributes within images, new categories can be recognized by first predicting the attributes in the image. The predicted attributes can then be compared and matched with different entries in the lexicon. The most likely or similar class entry in the lexicon is selected as the predicted class. New classes can be created on the fly by adding a new entry to the lexicon with the attributes of the new category. However, such a new category has to be defined in the same attribute space. Also, each category should be sufficiently and uniquely described by the set of attributes.

1.3 Motivation and research questions

The aim of this thesis is to provide an exploration of attribute learning with a deep Neural Network and its effects on learning in both a standard and zero-shot setting. A powerful CNN architecture will be employed for attribute recognition. To recognize the attributes, two classic methods will be used, where one method tries to directly learn the attributes while the second one learns them indirectly via the classes. Additionally, a third method to learn the attributes is experimented with, which uses a triplet loss [3] to learn attribute embeddings.

Standard classification with only output labels will be compared with classification with attributes. Moreover, the effectiveness of attributes for zero-shot recognition will be analyzed. These effects will be tested on three attribute datasets, which vary in size. Additionally, the interpretability of the deep models granted by visual attributes will be explored in the interest of explainable AI. This is done by extracting understandable rules that map from attributes to classes, using decision trees, which can offer insight into the most discriminative and informative attributes of a dataset. Additionally, the explainability of a deep model's predictions will be explored by investigating how a network's attention can be visualized, to understand which areas in an image contribute to certain predictions.

In this project, the primary goal is to investigate the effects and usefulness of deep attribute learning. The main research question is formulated as follows:

How does deep attribute learning improve the applicability of deep learning?

This main research question is answered by answering the following three sub-questions:

1. How does deep attribute learning affect standard classification?
2. How well does deep attribute learning allow zero-shot recognition?
3. How does deep attribute learning improve the explainability and interpretability of deep learning models?

1.4 Outline

The rest of this thesis is sub-divided into the following sections:

- **Theoretical background**. This section explains the theory behind all the methods used in the various different fields, such as Neural Networks, XAI and zero-shot recognition.
- **Methods**. This section describes the experimental setup of this thesis. It details the datasets and frameworks used and the various experiments that have been performed.
- **Results**. This section contains all the results that were obtained from each of the experiments.
- **Discussion**. This final section attempts to answer the research questions asked in the previous section and a conclusion is made.

Chapter 2
Theoretical background

2.1 Artificial neural networks

2.1.1 Supervised learning

Supervised learning is one of the major and most studied fields in the Artificial Intelligence and machine learning literature. Supervised learning algorithms embrace the idea of learning with examples, or learning with a teacher. The algorithms developed in this field require that each input sample is complemented by a label, which indicates the category of the sample. It is up to the algorithm to learn a mapping from input to label, such that new inputs are assigned the correct label.

Generally, supervised classifiers need a distinct training and testing phase. In the training phase, the classifier will observe a feature vector that numerically describes an instance. Such an instance could be a potential customer for a moneylender, or a photo of a dog. The features of an instance depend on the type of sample: in the first example, a binary feature could describe the gender of a person, while a different, continuous feature describes the person's credit history. In the second example, the features describing the dog are the pixels in the image and their color intensities.

Besides the feature vector, a label describing what category the instance belongs to also has to be given. A supervised algorithm then has to learn to predict such a label, given the input feature vector. Formally, a supervised classifier is a function that maps a N-dimensional input vector $\mathbf{x} \in \mathbb{R}^{1 \times N}$ to a label $\mathbf{y} \in \{0, 1, ..., C\}$, where C is the number of categories. $\mathbf{y}$ is also often a binary indicator vector called a one-hot vector, which consists of zeros except for the entry at the index of the correct class, which is set to 1: $\mathbf{y} \in \{0, 1\}^{1 \times C}$.

Each of the N features describes some aspect or characteristic of the example, as mentioned before. For a given supervised algorithm, its task is to find robust correlations and patterns between features within the feature vector and the corresponding label. In this setting, the supervised algorithm performs classification, since $\mathbf{x}$ has to be placed into one of C possible categories. The label can also be any real number: $\mathbf{y} \in \mathbb{R}^{1 \times \mathrm{m}}$. In this case, the algorithm performs regression, which is used for predicting quantitative values for a feature vector, where m is the dimensionality of the values being predicted. A dataset $\mathbf{D}$ is defined as a set of pairs of feature vectors and labels: $\mathbf{D} = \{(\mathbf{x}^1, \mathbf{y}^1), (\mathbf{x}^2, \mathbf{y}^2), ..., (\mathbf{x}^n, \mathbf{y}^n)\}_1^n$, where n is the total amount of examples within the dataset.

In the testing phase, the algorithm will be deployed on unseen examples, whose feature vectors and labels have not been incorporated into the training process. Here, the generalization power of the algorithm is tested: the knowledge gained from the training data is applied to new data. If the supervised algorithm has not memorized the training data too much, it may generalize well to the testing data. However, it is possible that instead of extracting the general patterns within the data, it simply learned the specifics of the training examples too much, resulting in good performance on the training data but bad performance on new data. If not enough training examples are provided or the training examples are of poor quality, performance will also suffer. In this case generalization will be poor, as the new data can be quite different compared to the training data even if they exhibit similar patterns, so the predictions will be heavily biased towards training samples. This is called overfitting. Various methods have been devised to combat the overfitting problem, both on the general level and algorithm-specific level. In general, for classification, balancing the dataset to take into account the underlying distribution of classes is one possible method which can help with algorithmic biases. Different approaches for reducing overfitting have been studied, especially for Deep Learning, which will be discussed in section 3.1.6.

2.1.2 Perceptron

When it comes to creating artificial intelligence, people have classicly been interested in studying the human brain, where real intelligence manifests. Therefore, Artificial Neural Networks (ANNs) have been some of the most studied algorithms in the field of AI. Such networks are loosely based on the human brain, with processing units that resemble neurons which have various inhibitory and excitatory connections with neighboring units. These connections, often called 'weights', are based upon the biological synapses that occur in the brain, which is the process in which electrical or chemical signals are sent from one neuron to another which alter the neuron's state. Many types of ANNs have been proposed over the years in the AI community, with some of them being successful for a variety of applications. The focus of this thesis is on a specific branch of neural networks, which are referred to as 'feed-forward networks'. Such networks process information starting from their inputs, propagating from the inputs to the next layer and so on up until the output layer. Such a network structure can be seen as a directed acyclic graph.

The Perceptron [19] is a simple feed-forward ANN, which has served as the basis for more complicated neural networks that have been built. A single perceptron unit can be seen as a neuron with a collection of weights that are connected to incoming input signals. The connections determine in what way the perceptron is affected by the input, such that certain inputs increase (excite) or decrease (inhibit) the activation of the neuron to various degrees. The neuron's activation can then be used for pattern recognition: the perceptron can learn to become very active when a certain pattern is present in the input, while being dormant when such a pattern is absent. Formally, given an input vector $\mathbf{x}$, the perceptron is a function $f(\mathbf{x})$ that uses a weight vector $\mathbf{w} \in \mathbb{R}^{1 \times N}$ to map the input to some output value, where the output represents the activation of the perceptron. The perceptron has exactly one weight for each input, so the dimensionalities of $\mathbf{x}$ and $\mathbf{w}$ are equal. Given a dataset $\mathbf{D} = \{(\mathbf{x}^1, \mathbf{y}^1), (\mathbf{x}^2, \mathbf{y}^2), ..., (\mathbf{x}^n, \mathbf{y}^n)\}_1^n$, the perceptron classifier's task is to learn a mapping from $\mathbf{x}$ to $\mathbf{y}$, such that $f(\mathbf{x}^i) = \mathbf{y}^i$, for all n input samples. A perceptron uses its weights to linearly combine the given inputs to produce its activation:

$$f(\mathbf{x}) = \sum_{j}^{N} \mathbf{x}_j \mathbf{w}_j = \mathbf{x}^T \mathbf{w}. \tag{2.1}$$

A good set of weights should be defined in such a way that input features that are not important for distinguishing between categories are assigned a small weight, such that their importance in the linear combination is low, while important features, either inhibitory or excitatory, have a weight with a large magnitude. The perceptron can perform both classification and regression. When looking at binary classification, we want the perceptron to map the input to one of two categories, which are often encoded as either $\{0, 1\}$ or $\{-1, 1\}$. However, the output of the perceptron is not restricted to this range because its weights can be arbitrarily large. Therefore, the activation of the network is squashed to be in a certain range, by using a saturating activation function. Examples of such activation functions will be discussed in section 2.1.6. While early algorithms such as the perceptron training rule [19] were used to train the network to find a good weight vector $\mathbf{w}$, most ANNs these days are trained using backpropagation and gradient descent. These methods are discussed in section 2.1.5.

The perceptron's weight vector defines a hyperplane in the input space which separates the positive input examples from the negative ones (in the binary classification setting). This is visualized in Figure 2.1, where positive examples have a label of +1 and negative examples have a label of -1. The decision line is made up of the points in the input space whose dot product with the weight vector equals 0, so they lie exactly on the boundary between the -1 and the +1 regions in the input space. The points and the orientation of the decision plane are thus determined by the values of the weights. By changing the weights, the orientation is also changed and having a

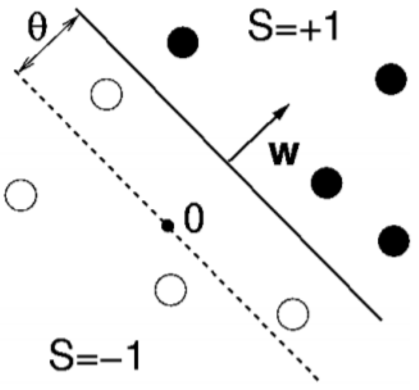


Figure 2.1: The weight vector w and bias weight θ determine the orientation and position of the hyperplane in the input space, respectively. S indicates the positive and negative regions. 0 indicates the origin. Perfect separation is not possible without θ in this example, which is taken from [1].

good set of weights means having a hyperplane that separates the positives from the negatives quite well. However, sometimes it's not possible to separate the positive from the negative examples by only changing the orientation of the hyperplane. By shifting its position, a wider variety of hyperplane placements in the input space are possible, which will allow the perceptron to separate more datasets and thus learn a wider variety of functions that map from $\mathbf{x}$ to $\mathbf{y}$. To add the ability for the hyperplane to shift from the origin, a bias weight is introduced, often denoted as b or θ, which is not connected to an input, but is simply added to the perceptron activation before applying the squashing activation function. When all the features of $\mathbf{x}$ are zero, b will determine the starting position of the decision plane, since the weights are not contributing anything in this case. This bias variable b can be trained, just like $\mathbf{w}$. Incorporating this bias value, the full perceptron equation can be described as follows:

$$f(\mathbf{x}) = \sigma(\sum_{j}^{N}(\mathbf{x}_j \mathbf{w}_j) - b) = \sigma(\mathbf{x}^T \mathbf{w} - b), \tag{2.2}$$

where $\sigma(\mathbf{x})$ is an activation function. The bias b can also be interpreted as a threshold. The dot product $\mathbf{x}^T\mathbf{w}$ must then be greater than or equal to b in order to be classified as the positive class. Perceptrons that are not trained with gradient de-

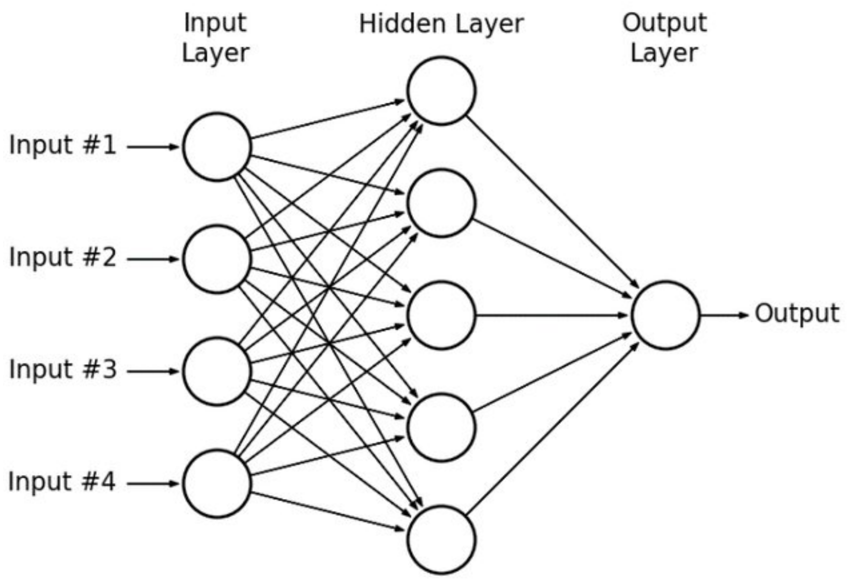


Figure 2.2: An example structure of a multi-layer perceptron, containing one hidden layer with 5 units and an output layer with one unit. The figure is taken from [2]. Here, the 4-dimensional input is non-linearly embedded into a 5-dimensional space, where separation of data points might be better.

scent can use a threshold function $\sigma(x, w, b)$ which can give one of two outputs:

$$\sigma(\mathbf{x}, \mathbf{w}, b) = \begin{cases} +1, & \text{if } \mathbf{x}^T\mathbf{w} \geqslant b \\ -1, & \text{otherwise} \end{cases} \tag{2.3}$$

Neural networks trained with gradient descent need differentiable activation functions, as will be described later.

2.1.3 Non-linearity and the multi-layer perceptron

While the Perceptron is one of the basic units of modern feed-forward neural networks, it is often not used on its own on real problems. While this classifier is able to separate data points with a linear decision boundary, many problems are not linearly separable, especially complex ones found in the real world. This greatly reduces the effectiveness of the perceptron. In order to properly classify such a dataset, a non-linear decision boundary is needed which is not possible with a simple perceptron.

One solution to deal with this problem is to introduce non-linearity in the classifier. This can be done by introducing an intermediate layer of neurons that non-linearly transforms the input, often denoted as the hidden layer [20]. In such a Multi-Layer Perceptron (MLP), inputs are first fed to this hidden layer, whose activations are squashed with a non-linear activation function. This non-linear transform maps the input samples from their original input space, where separation is not possible, to a non-linearly transformed hidden space where linear separation may be possible. The structure of a simple multi-layer perceptron is shown in Figure 2.2. The hidden layer is 'hidden', since its values are never observed from the outside, unlike the input values that are fed to the network and the output layer that computes the predictions for the task at hand.

Since each unit in the hidden layer is just a perceptron with a differentiable transfer function, the activation for one unit is equal to equation (2.2). The full activation vector of the hidden layer, $\mathbf{a}(\mathbf{x})$, can be computed as follows using a vector-matrix multiplication and a vector summation:

$$\mathbf{a}(\mathbf{x}) = \sigma(\mathbf{x}^T \mathbf{W}^H + \mathbf{b}^H), \tag{2.4}$$

with input sample $\mathbf{x}$, hidden weight matrix $\mathbf{W}^H \in \mathbb{R}^{\mathrm{Nxh}}$ and a bias vector $\mathbf{b}^H \in \mathbb{R}^{\mathrm{1xh}}$ containing one bias value for each hidden unit. h is the amount of units in the hidden layer and $\sigma(\mathbf{x})$ is an activation function. After computing the hidden activation vector, it serves as the input to the next layer. If the MLP has one hidden layer, the output layer is computed using the hidden activation as input, which is a non-linearly transformed version of the original input, which may be linearly separable:

$$f(\mathbf{x}) = \sigma(\mathbf{a}(\mathbf{x})^T \mathbf{W}^O + \mathbf{b}^O), \tag{2.5}$$

with hidden activation $\mathbf{a}(\mathbf{x}) \in \mathbb{R}^{\mathrm{1xh}}$, output weight matrix $\mathbf{W}^O \in \mathbb{R}^{\mathrm{hxo}}$ and an output bias vector $\mathbf{b}^O \in \mathbb{R}^{\mathrm{1xo}}$. o is the amount of units in the output layer.

When training an MLP, the hidden layer learns to non-linearly embed the input into a different space in such a way that the output layer is better able to separate the instances, since the output units are simple linear perceptrons. This is why the hidden layer and similar mechanisms are often also referred to as embeddings. In this way, the layers in an MLP can be viewed as cooperating together: the hidden layer(s) learn(s) to improve its embedding of the original input for improved separation, while the output layer learns to better perform classification given the non-linear embedding as input.

2.1.4 Loss functions

The aim of training a neural network is to find good values for all of the weights, such that the network is able to perform its task well, whether that be classification or regression. In modern neural networks, this training is done with mathematical optimization procedures. We can define a function that measures the degree to which a network makes errors, often called an error or loss function. Using optimization techniques, we then aim to minimize this loss function with respect to its variables, the network's weights, which should as a result improve the network's behavior on a given task. Formally, the loss over a set of data points is computed as follows:

$$L(\mathbf{D};\theta) = \frac{1}{n}\sum_{i}^{n} l(\mathbf{D}(i),\theta), \tag{2.6}$$

where $L(\mathbf{D};\theta)$ is the average loss over the training dataset $\mathbf{D}$ given network parameters θ. $l(\mathbf{D}(i),\theta)$ is an arbitrary loss function, taking as input weights θ and an input-output pair $\mathbf{D}(i)$: $\mathbf{D}(i) = (\mathbf{x}^i, \mathbf{y}^i)$. The loss function $l(\mathbf{D}(i),\theta)$ should be defined such that it takes on high values when the network prediction for input sample $\mathbf{x}(i)$ is far away from the ground-truth label $\mathbf{y}(i)$ and low values when the prediction is close to the label.

A popular loss function that is commonly used in classification problems is the cross-entropy loss. For a network with o output units, this loss is defined as follows for multi-class classification:

$$l((\mathbf{x}^i,\mathbf{y}^i),\theta) = -\sum_{j}^{o} \mathbf{y}^i_j log(f(\mathbf{x}^i;\theta)_j), \tag{2.7}$$

where $\mathbf{y}^i_j$ indicates for example $\mathbf{x}^i$ and output node j whether output node j is the correct class (1) for x^i or not (0). $f(\mathbf{x}_i;\theta)_j$ is the output of the j-th output unit of network $f(x_i;\theta)$. Since classes are mutually exclusive in a multi-class classification problem, only one output unit contributes to the loss per data point, as only one $\mathbf{y}^i_j$ will be non-zero. For multi-label classification, more than one output unit can contribute to the loss, since the label for every output unit can be either 0 or 1. Multi-label classification can therefore be seen as multiple binary classification problems, so the multi-label cross-entropy loss is defined as the average binary cross-entropy over all output units:

$$l((\mathbf{x}^i,\mathbf{y}^i),\theta) = \frac{1}{o}\sum_{j}^{o} -(\mathbf{y}^i_j log(f(\mathbf{x}^i,\theta)_j) + (1-\mathbf{y}^i_j) log(1-f(\mathbf{x}^i,\theta)_j)) \tag{2.8}$$

Since the labels are either 0 or 1, one of the two terms in the summation cancels out because of either $\mathbf{y}^i_j$ or $(1-\mathbf{y}^i_j)$ being equal to zero.

2.1.5 Backpropagation

Minimizing the cost function with respect to the weights of the neural network can be done by a first-order optimization method called gradient descent [21,22]. It is a first-order method since it uses first-order derivatives. The gradient of a function, $\nabla\mathbf{f}$, is a vector consisting of the partial derivatives of that function with respect to its variables. Formally, the gradient can be described as follows:

$$\nabla\mathbf{f} = \{\frac{\partial f}{\partial \mathbf{w}_1}, \frac{\partial f}{\partial \mathbf{w}_2}, ..., \frac{\partial f}{\partial \mathbf{w}_{N_v}}\}, \tag{2.9}$$

where N_v is the number of variables of function f. Given a point P in the N_v-dimensional weight space defined by the values of all N_v weights, the gradient of f points in the direction of biggest increase of the value of f from point P. Since we want to minimize f, going against the gradient will turn the weights away from the direction of greatest increase at point P, instead going the opposite direction: the direction of decrease. Given a neural network with randomly initialized weights, the point P starts off in a random location in the gradient space of the cost function. The gradient can then be computed and the weights can be moved in the opposite direction in order to move towards a better minimum. This is repeated many times in order to find weights that minimize f. This method can be seen as a local method, as we look only at the current local point P at a time, defined by the values of the weights, and decide from there where to move. Because of this locality, gradient descent has problems with ending up at a local minimum, instead of the global minimum of the cost function, since we always take the greedy approach by taking the best step each time instead of considering more global information.

In gradient descent, a step-size is defined that controls the magnitude of the steps of $\mathbf{w}$ into the negative direction of the gradient, also called the learning rate and often denoted as α or η. By carefully setting η between 0 and 1, the step-size at each training step is kept small in order to avoid overshooting and ending up in a worse position than the current point P. Setting a proper learning rate is quite important for training, as a high learning rate can lead to overshooting while a low learning rate can lead to very slow convergence. For each training step, the update equation for each weight $\mathbf{w}_i$ is then defined as follows:

$$\mathbf{w}_i = \mathbf{w}_i - \eta\frac{\partial f}{\partial \mathbf{w}_i}, \tag{2.10}$$

where $\frac{\partial f}{\partial \mathbf{w}_i}$, is the partial derivative of loss f with respect to variable $\mathbf{w}_i$.

In order to compute the partial derivatives, a technique called backpropagation can be applied [20]. This method applies the chain-rule of differentiation to calculate the partial derivative of the loss function with respect to some variable through every layer of the network, all the way back to the layer in which the variable resides. The method of backpropagation is related to the credit assignment problem: if a neural network makes a bad prediction on a certain data point, to what extent is each weight to blame? The partial derivative $\frac{\partial f}{\partial w_i}$ estimates how f changes when only looking at weight w_i. Therefore, it measures the contribution of w_i to the error function f, whether that's in the positive or negative direction. Using backpropagation, we can trace the path of contributions back to w_i. An illustration can be given with regard to the MLP structure given in Figure 2.2, which contains 5 hidden units and 1 output unit. First, the contribution of the output unit's output, denoted by $\mathbf{o}^O$, to the value of the loss f is computed as $\frac{\partial f}{\partial \mathbf{o}^O}$, which is dependent on the specific loss function being used. Since the output node's output is dependent on the activation function, we also have to compute the partial derivative of the output to the activation of the output node according to the chain rule of differentiation. Let $\mathbf{o}^O$ and $\mathbf{o}^H$ denote the output of the output layer and hidden layer respectively and let $\mathbf{a}^O$ and $\mathbf{a}^H$ denote the activation of the output layer and hidden layer, respectively. The contribution of hidden unit j, whose weights are denoted as $\mathbf{W}_j^H$, can be computed by computing the chain of derivatives from loss f all the way to $\mathbf{W}_j^H$ as follows:

$$\frac{\partial f}{\partial \mathbf{W}_j^H} = \frac{\partial f}{\partial \mathbf{o}^O}\frac{\partial \mathbf{o}^O}{\partial \mathbf{a}^O}\frac{\partial \mathbf{a}^O}{\partial \mathbf{o}^H}\frac{\partial \mathbf{o}^H}{\partial \mathbf{a}^H}\frac{\partial \mathbf{a}^H}{\partial \mathbf{W}_j^H}. \tag{2.11}$$

$\frac{\partial f}{\partial \mathbf{W}_j^H}$ is thus a vector containing the gradients for each weight of hidden unit j and are used to update unit j's weights as defined in equation 2.10.

Typically, three different kinds of gradient descent are used: online, batch and mini-batch gradient descent. Equation 2.10 denotes an online learning rule, as it updates the parameters according to the gradients separately for each data point. This results in an update for each data point separately instead of considering all the training data points, which leads to a noisier path to a minimum, which can aid the network in escaping from local minima. Online learning is also called stochastic gradient descent (SGD), due to its noisy nature.

In batch gradient descent, the gradients for each variable for each training data point are accumulated. After every data point has been fed to the ANN, the gradients are averaged over the data points and the weights are adapted. Batch gradient descent modifies equation 2.10 as follows:

$$\mathbf{w}_i = \mathbf{w}_i - \eta\frac{1}{n}\sum_j^n \frac{\partial f_j}{\partial \mathbf{w}_i}, \tag{2.12}$$

where f_j is the loss on example $\mathbf{x}^j$. Batch gradient descent is not as noisy as stochastic gradient descent, since it takes into account more data points.

Another variant called mini-batch stochastic gradient descent is used, especially in deep learning, which combines aspects from online and batch gradient descent. As deep learning usually deals with large amounts of data, feeding an entire training set to a deep model might not be feasible. However, only feeding a single data point at a time can also be costly, as the forward pass and backwards gradient computations have to be computed one by one and the amount of computations grows with deeper models. On modern GPU architectures, these types of operations can be sped up for many data points at once. Therefore, mini-batch gradient descent trains on n_b data points at once, benefitting from the stochastic nature of online learning while feeding more data at once in the training procedure. The mini-batch SGD update rule can be written as follows:

$$\mathbf{w}_i = \mathbf{w}_i - \eta \frac{1}{n_b} \sum_{j}^{n_b} \frac{\partial f_j}{\partial \mathbf{w}_i}, \tag{2.13}$$

2.1.6 Activation functions

In the history of Neural Networks, many activation functions have been devised for output squashing and adding non-linear behavior to artificial neurons. As described before, this non-linear transform of neuron activation is what allows an MLP to classify non-linear datasets by first transforming the inputs from the original space into a different space where a linear classifier can be applied. Different differentiable transfer functions exist for performing this task. The most notable ones will be described here.

The logistic Sigmoid function and the Tanh function were traditionally the most popular transfer functions used for Neural Networks. The Sigmoid and Tanh are given by the following equations, respectively:

$$sigmoid(\mathbf{x}) = \frac{1}{1 + \exp^{-\mathbf{x}}} \tag{2.14}$$

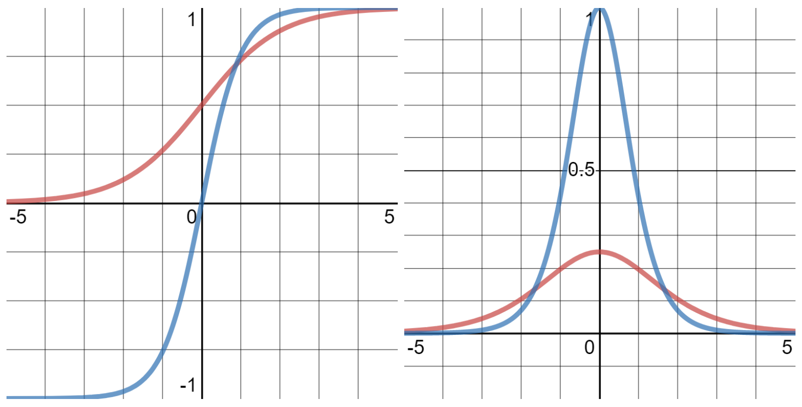
$$tanh(\mathbf{x}) = \frac{\exp^{\mathbf{x}} - \exp^{-\mathbf{x}}}{\exp^{\mathbf{x}} + \exp^{-\mathbf{x}}}. \tag{2.15}$$

The derivatives of these functions are given as follows:

$$\frac{dsigmoid}{dx} = sigmoid(\mathbf{x})(1 - sigmoid(\mathbf{x})) \tag{2.16}$$

$$\frac{dtanh}{dx} = 1 - (tanh(\mathbf{x}))^2. \tag{2.17}$$

The fact that the derivatives of both the sigmoid and tanh can be computed by re-using their values from the forward pass makes them attractive to use from a



(a) Sigmoid and tanh. (b) Derivatives of sigmoid and tanh.

Figure 2.3: Plots of the sigmoid (red) and tanh functions (blue). Both functions have a visible limit, both on the positive and negative end of the x-axis.

computational point of view. The two functions and their derivatives are shown in Figure 2.3.

As can be seen from the graph, both activation functions have an S-like shape. They can be interpreted as a smooth, differentiable version of the threshold function, where the biggest increase happens close to $x = 0$. Both functions saturate when x is very small or large, which can be seen as a maximal inhibition point ($f(x)$ = 0 for sigmoid, $f(x) = -1$ for tanh) and a maximum excitation point ($f(x) = 1$ for both sigmoid and tanh). The use of these activation functions have fallen off in the era of deep learning. With deep learning, many layers are typically stacked on top of each other. The deeper the network becomes, the longer the chain of partial derivatives becomes, such as the one in equation 2.11. In each layer i of some ANN that uses an activation function, an expression of the type $\frac{\partial \mathbf{o^i}}{\partial \mathbf{a^i}}$ occurs, which involves computing the derivative of the activation function used, which is subsequently multiplied in the computation of the gradient. As shown in Figure 2.3, the derivatives of both the tanh and sigmoid function are below 1 in most cases. With many layers, many multiplications with values below 1 have to be done in order to compute the partial derivative of weights in the early layers. These partial derivatives, as a result, will grow smaller and smaller and closer to zero, which will hamper and eventually even prevent learning of earlier layers in a deep model. This is known as the 'vanishing gradient' problem. Similarly, the 'exploding gradient' problem can occur if activation functions are used where the derivative can be higher than 1, which can cause the gradients and subsequently the weights to become very large.

One proposed solution to this problem is a different activation function called the 'Rectified Linear Unit (ReLU), which has become one of the standard activation functions in deep neural networks [11,23]. The ReLU function is defined as follows:

$$ReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} = max(0, x) \tag{2.18}$$

The ReLU is partially the linear identity function $f(x) = x$ when x is positive, and zero otherwise. Clipping all negative inputs to zero is what gives the ReLU its non-linearity. The derivative of the ReLU is given as:

$$\frac{dReLU}{dx} = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.19}$$

Therefore, if the derivative of the ReLU is used in backpropagation terms like $\frac{\partial \mathbf{o}}{\partial \mathbf{a}}$, then this term will reduce to either 1 or 0. While negative activation will be set to 0, the positive gradients will not be reduced in magnitude as they will be multiplied by 1, unlike the sigmoid and tanh functions. This allows the gradients to reach further back in a deep network and improves the training of early layers. For this reason, ReLU has become the dominant activation function in deep learning.

2.1.7 Deep learning

CNNs are deep Neural Networks that have been very successful for various recognition applications. For computer vision, features had to traditionally be manually computed from images, using methods such as the well-known Scale-Invariant Feature Transform (SIFT) [24] or Histograms of Oriented Gradients (hoG) [25], which extract general patterns in the image which could subsequently be used for classification. Therefore, the feature extraction and classification were separate stages in the processing pipeline.

The basic operator that distinguishes CNNs from different Neural Networks is the use of the convolution operator, often denoted by the asterisk ($*$) symbol. The discrete convolution operation in one dimension can be described as follows:

$$s(t) = (\mathbf{x} * \mathbf{w})(t) = \sum_{-\infty}^{\infty} \mathbf{x}(\mathbf{a})\mathbf{w}(\mathbf{t} - \mathbf{a}), \tag{2.20}$$

where $\mathbf{x}$ is an input signal, $\mathbf{w}$ is a parameter vector containing weights for every time step in the input signal and a is the off-set from time t. $s(t)$ therefore calculates the convolution of $\mathbf{x}$ with $\mathbf{w}$ at time step t. In practice, $\mathbf{x}$ and $\mathbf{w}$ are zero everywhere except for a finite number of points. Therefore, this infinite sum can be implemented as a finite sum of multiplications between each signal value and its corresponding weight. With this convolution operator, local patterns in the signal can be found at time t, by weighting the signal value at time t and a finite amount of past values. In other words, a convolution can be seen as a window of fixed size that slides over the input to find patterns within this small interval at different locations.

CNNs are mostly used for grid-like data with a fixed height and width, such as images. Therefore, CNNs employ 2-dimensional convolution. In this scenario, convolution is done along both the height and width dimension of the input image $\mathbf{I}$. The 2D parameter vector $\mathbf{w}$ is usually referred to as a kernel $\mathbf{K}$. In many Neural Network libraries the related cross-correlation function is implemented instead of convolution:

$$\mathbf{S}(i,j) = (\mathbf{K} * \mathbf{I})(i,j) = \sum_{h}^{H}\sum_{w}^{W}\mathbf{I}(i+h, j+w)\mathbf{K}(h,w), \tag{2.21}$$

where $S(i,j)$ is the result at pixel location (i,j) of convolving the kernel $\mathbf{K}$ with input image $\mathbf{I}$. H and W are the height and width of the kernel, respectively. This convolution generates a feature map $\mathbf{S}$ and each element of $\mathbf{S}$ contains the resulting vector dot product between some local area of $\mathbf{I}$ and the kernel $\mathbf{K}$. Similar to the 1-dimensional case, patterns are discovered by weighting a specific location's value and the neighboring values, this time in 2 dimensions. Since the dot product between two vectors is higher the more similar the two vectors are, the convolution of $\mathbf{K}$ and image pixels within the kernel K centered on location $\mathbf{I}(i,j)$ will result in high values in the resulting feature map the more these pixels resembles $\mathbf{K}$. The kernels can therefore be seen as pattern detectors, which fire more when the location in the input that the kernel is currently overlaying represents the pattern it is looking for. The feature map $\mathbf{S}$ can be interpreted as the firing intensities or activations of $\mathbf{K}$ along different positions of $\mathbf{I}$, indicating the presence of the feature. A kernel designed to detect edges will have a large activation in regions of the image that contain edge-like patterns. Traditionally, convolution of images with kernels was already used for image processing, such as filtering, smoothing or edge detection [26]. However, in these applications, the kernels were handcrafted. In a CNN, the kernels are weight parameters that are randomly initialized and trained during supervised learning. Unlike more traditional computer vision methods, this means that the feature extraction is automatically learned and not hand-crafted, which saves a lot of human effort.

Kernel convolution in a CNN brings a host of advantages over more traditional feed-forward neural networks. Unlike a perceptron or MLP, the weights in the CNN are shared among the different locations in the input. This leads to a lot fewer parameters that have to be used. An image usually contains a lot of pixels and every hidden unit in a fully-connected MLP would need one weight for every pixel, leading to a huge amount of parameters. Parameter sharing leads to both a lower computational and model complexity [11].

Another advantageous property of convolution is its equivariance to translation [11]. If the content of an input image is shifted, the kernels of the CNN may still find the patterns that they are looking for, except at a different location. This will also be reflected in the resulting feature map.

One of the standard building blocks of modern CNNs is the convolutional layer. Such a layer contains many kernels, all looking for various patterns. Each kernel is convoluted over the input $\mathbf{I}$, generating one feature map for every kernel. Convolution is a linear operation, so non-linearity is introduced into the network for better separation, which is done by applying an activation function to the resulting feature maps. These feature maps are then stacked to produce a 3-dimensional output volume layer.

A more general version of the kernel is called a filter. Since the input to a collection of kernels is often a 3D volume, such as an an image with red, blue and green color channels, kernels for each of these channels are needed. A filter is therefore a collection of kernels, one for each input channel. The kernels of the filter look for patterns in each of the input's channels and the resulting feature maps for one filter are summed. A convolutional layer therefore takes a 3D input volume and produces another 3D volume of stacked feature maps, which serve as the input to the next layer. The input to a convolutional layer is therefore either the input image for the first convolutional layer, or otherwise a stack of feature maps from the previous layer. The filters in the next layer will look for patterns in the filter activations of the previous layers, which can lead to complex and hard-to-interpret but effective templates. However, the input feature maps to a convolution layer are often first modified by a different CNN building block, called the pooling layer.

Even though the direct connectivity between units in a CNN is lower due to only having local direct interactions, units in higher layers get an increasingly higher receptive field due to indirect connections to units in lower layers. For example, while a 3x3 kernel in a higher CNN layer only has a direct receptive field of 3x3, the input consists of activations of units in the previous layer, which themselves have a receptive field. The higher level kernel therefore indirectly has a bigger receptive field, as it has indirect connections to the inputs within the receptive field of the lower units.

Pooling layers are often applied after convolutional layers to downsample the input. The pooling operation replaces the input feature maps with local summary statistics [11]. Like convolution, pooling is also done using a sliding window, but pooling windows are often spaced apart further than convolution windows. If pooling windows are spaced k pixels apart, the resulting feature maps will be reduced to be roughly k times smaller. The spacing between windows, or alternatively the distance the window moves is often referred to as the stride. The most used pooling operations are max pooling and average pooling, which takes the max or mean of a region of input activations, respectively. Pooling grants the CNN more invariance towards small translations [11]. Translations in the input lead to translations in the output feature maps, but if these translations are small then the features may still fall within the same pooling window, therefore making the output of the pooling operation roughly the same compared as the non-translated input.

After the final convolutional or pooling layer, the extracted features are usually flattened into a one-dimensional feature vector and fed to fully connected layers, which represent a typical MLP, with one or more hidden layers, which performs non-linear transformations of the input features for classification. Both the kernel weights and weights from the fully connected layers are trained during the same backward pass of the network during supervised learning. Both the feature extraction and classification are thus trained in the same connected process. This means that the kernels are learned to recognize robust features, such that the fully connected layer can accurately perform classification. The entire CNN can be trained end-to-end using backpropagation [20].

While CNN kernels are trained for a specific domain, they can sometimes be general enough to be used for other domains. This is known as transfer learning, where a CNN trained on one domain is only partially re-trained for application on a different domain [27]. The kernels learned in a CNN often have a hierarchical nature: in the first layers, the kernels learn to detect simple general features such as edges. In the next layer, filters look for patterns that are combinations of edges by looking at the filter activations of the previous layer. This continues in subsequent layers and at the highest layers more complex, domain-dependent features will be detected, such as eyes or noses in face recognition. While these highest-level features are usually specific for the dataset that is trained upon, the lower level features such as edges can be general enough for most other datasets as well. Higher level features can also be used for transfer learning, if the domains of the source and target datasets are similar enough.

2.1.8 Deep architectures

Many deep learning architectures have been constructed that build upon the CNN concepts described in the previous section. One popular network is GoogleNet, also known as Inception-V1, which was able to win the 2014 ILSVRC [28]. One

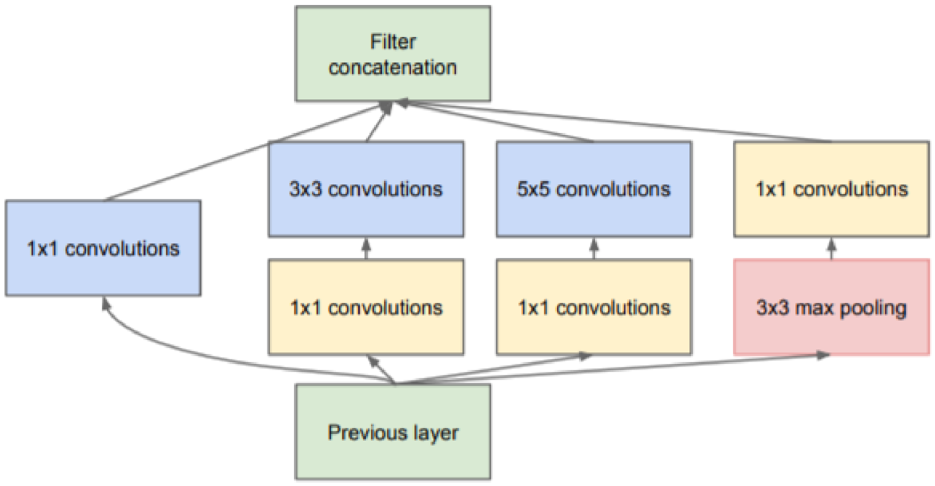


Figure 2.4: Diagram of the Inception modules found in the Inception-V1 architecture, using multiple kernel sizes and dimensionality reduction with 1x1 convolutions.

of the main components of this architecture is the use of Inception modules. One issue in the design of a CNN is the receptive field or kernel size used within the convolutional layers. Small kernels can pick up more fine-grained features, while bigger kernels can detect patterns that span a wider part of the image. The inception modules attempt to address this design issue by including different sets of kernels containing different kernel sizes, which are applied at the same layer. The resulting feature maps from these different filters are all stacked. In this way, features at different scales are extracted and fed to the next layer, which can learn filters to detect patterns in this multi-scale feature volume.

Including many kernels with different receptive fields within one layer could lead to a higher computational complexity because of the higher number of convolutions. To address this, a lot of dimensionality reduction is used within such modules. Before convolving the input with kernel sizes such as 3x3 and 5x5, the amount of input channels is first reduced with 1x1 convolutions with an amount of filters lower than the amount of input channels. A 1x1 convolution is equivalent to first multiplying each input channel by a different scalar weight, then applying non-linearity and finally summing the scalar-transformed non-linear maps. 1x1 convolutions are therefore a relatively cheap method of embedding the 3D input volume into a thinner one. Convolutions with bigger kernel sizes can then be performed on this thinner volume, which is faster than convolution with the original, thicker input volume, since the filters require fewer kernels. The concept of the inception modules is illustrated in Figure 2.4.

For Inception-V2, a very popular optimization technique known as batch normalization is introduced [29]. With batch normalization, a concept known as internal covariate shift is addressed. During training, the parameters at each layer are adjusted, which changes the activation patterns of the layer. Since the layer's activation distribution changes, the next layer will have to re-adjust to this new distribution, which slows down the training process. The inputs to a Neural Network are often normalized such that the features are within a similar, small range, which reduces quick saturation of neurons due to high feature values and stabilizes the input distribution, which helps the network train. Similarly, batch normalization normalizes the outputs of a layer over a mini-batch, to normalize the inputs of the next layer. The normalization is done by first computing the mean μ_b and variance σ_b^2 over mini-batch b for each feature. The inputs x to a layer are then standardized by first subtracting the mean, then dividing by the variance. To prevent the layer from losing representational power, some additional parameters are learned for each batch normalization layer, which can map the normalized activations back to the unnormalized versions in case the batch normalization is detrimental. These parameter vectors γ^l and β^l are learned along-side the other parameters of the network using SGD, for each batch-normalization layer l. The batch-normalization for layer l, $BN(x^l)$, can then be computed as follows:

$$BN(x^l) = \gamma^l \frac{x^l - \mu_b}{\sigma_b^2} + \beta^l, \tag{2.22}$$

where γ^l scales each activation of input x^l and β^l shifts them. Deciding which layers in a network should receive batch-normalization is part of the network design process.

Finally, Inception-v3 introduces various enhancements to the Inception modules from previous versions [6]. Three different Inception modules are introduced, which are placed at different locations within the network. The first module attempts to reduce the computational complexity and amount of parameters further by replacing the 5×5 convolutions in the inception module by 2 subsequent 3×3 convolutions. While 5×5 kernels have a wider receptive field compared to 3×3 kernels, applying a 3×3 kernel to the result of a different 3×3 kernel will still result in a wide receptive field due to the indirect connectivity. These inception modules are placed in the early layers. The second type of Inception modules uses asymmetrical convolutions by replacing $n \times n$ kernels by subsequent $1 \times n$ and $n \times 1$ kernels, which have been found to work well on medium-sized input volumes and are therefore placed in the middle layers. The third Inception module type is placed in the final layers. In these layers, the input volumes will have a small height and width due to the many pooling operations that have been performed. To extract as many different features from these small grids, different asymmetrical filters are employed on the same input and then concatenated.

One final important aspect of the Inception network is its use of Global Average Pooling (GAP) at the end of the network before the prediction layer, instead of fully-connected layers, which was first introduced in [30]. Global Average Pooling takes the final feature maps of the network and averages each channel into a single scalar. This final feature vector is then fed into the prediction layer. One advantage is that a GAP layer contains no parameters, unlike fully-connected layers that are prone to overfitting. Furthermore, it is easier to interpret the connections between output units and the feature maps in the final convolutional layer, as each output unit will have one weight for each feature map.

2.1.9 Deep metric learning

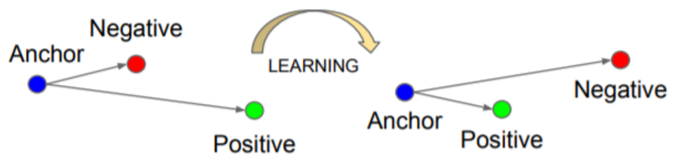


Figure 2.5: An example of embedding learning. In this example, the anchor is closer to the negative sample than to the positive sample in the embedding space. After learning, the anchor is situated closer to the positive [3].

While cost functions such as the cross-entropy have been very successful in Deep Learning, recent research has led to alternative loss functions that have been shown to work well for certain applications. One such an alternative function is called the triplet loss, which has been applied for face recognition and verification [3]. Unlike more traditional CNNs, a network trained with the triplet loss does not contain output units that represent categories. Instead, the final layer of the network contains a fully-connected layer of a certain size. The triplet loss works directly on this embedding layer using a special constraint. During the training of a triplet network, three images are presented to the network at once: the anchor, the positive and the negative. The anchor image represents some class, the positive is a different image of the same class and the negative is an image of a different class. The objective of the network is then to learn a d-dimensional embedding of an image, $f(x) \in \mathbb{R}^d$, such that images of the same class are embedded close in Euclidean space, while images of different classes are far away. This concept is illustrated in Figure 2.5. Classification can then be done by finding an appropriate maximum distance threshold, such that, given an image, all other images with a distance within this threshold will be given the same label as the initial query image. The network therefore tries to optimize the embedding $f(x)$ under the following constraint for each triplet i:

$$d_i^{ap} = ||f(\mathbf{x}_i^a) - f(\mathbf{x}_i^p)||_2^2 \tag{2.23}$$
$$d_i^{an} = ||f(\mathbf{x}_i^a) - f(\mathbf{x}_i^n)||_2^2 \tag{2.24}$$
$$constraint = d_i^{ap} + \alpha < d_i^{an}, \tag{2.25}$$

where d_i^{ap} represents the squared Euclidean distance between the anchor embedding $f(x_i^a)$ and positive embedding $f(x_i^p)$ and d_i^{an} represents the squared Euclidean distance between $f(x_i^a)$ and the negative embedding $f(x_i^n)$. α represents a margin that is enforced between the positive and negative anchor distances. This constraint forces the anchor-positive distance to be small and the anchor-negative distance to be big. The corresponding loss function that incorporates this constraint is the following, for each triplet i:

$$l((x_i^a, x_i^p, x_i^n), \theta) = d_i^{ap} - d_i^{an} + \alpha. \tag{2.26}$$

This loss function is minimized by minimizing d_i^{ap} and maximizing d_i^{an}, given anchor x_i^a, positive x_i^p and negative x_i^n. Increasing the margin α may lead to a larger distance between positive samples for a certain person and negative samples in the embedding space. However, this will also make it harder for the network to learn, as certain high margins may not be feasible to maintain for each class. Theoretically, in order to get a minimum loss of zero, d_i^{ap} needs to be zero. α needs to be counteracted, so d_i^{an} needs to take on the value of α, leading to the loss being $0-\alpha+\alpha = 0$. In this scenario, the margin is perfectly maintained. In practice, this is hard to achieve, as it requires all positive samples from the same class to be in the exact same place in the embedding space in order to have a distance of zero. If α is increased, the network needs to compensate by lowering d_i^{ap} or increasing d_i^{an}.

In order to calculate the distances between the embeddings, a forward-pass through the deep network is required in order to calculate the embeddings themselves. This can be a costly but necessary step in the training process. In order to improve the convergence of a triplet network, quality, informative triplets should be sought. Initially, certain triplets might already have a low loss, due to the random initialization of weights. Therefore, care should be taken to select hard triplets: those triplets with

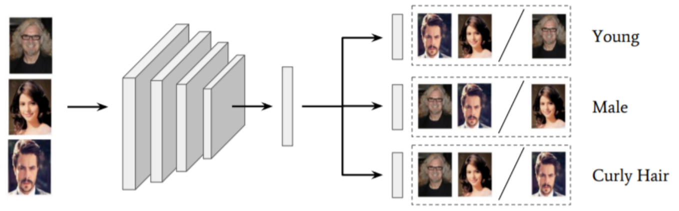


Figure 2.6: The multi-label triplet loss allows a deep network to learn an embedding space separately for each attribute, at the same time. Two images that are positives for one attribute may be negatives for another [4].

a large anchor-positive distance d_i^{ap} or a small anchor-negative distance d_i^{an}, referred to as hard positives and hard negatives, respectively. These triplets will have a large loss and will allow the network to learn the most. There are different selection strategies for triplet selection, also called triplet mining. An often used strategy of triplet mining is online mining: selecting good triplets from images within a mini-batch, instead of gathering them from the entire training dataset, which significantly reduces computation time [3].

Instead of learning only one embedding, the triplet loss can be extended to learn multiple embeddings at once. In a multi-label setting, such as with attribute learning, an embedding can be learned for every class at the same time. For each attribute, an embedding is learned such that both positive and negative instances of the attribute are closer to instances of the same class and further away from the opposite class [4]. An example is illustrated in Figure 2.6. In this case, the triplet loss can be referred to as a multi-label triplet loss function. Since the three images in a triplet may have different attributes, the anchor-positive-negative roles are dependent on the specific attribute. Therefore, for each triplet, the triplet loss is calculated for each attribute separately, then averaged. The multi-label triplet loss for each triplet is given as follows:

$$l((\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k), \theta) = \frac{1}{n_a} \sum_{a}^{n_a} w_{ijk}^a [d_{ia}^{ij} - d_{ia}^{ik} + \alpha], \tag{2.27}$$

for some triplet with images x_i, x_j and x_k. n_a is the number of attributes in the dataset. Applying online mining for multi-label triplet loss would require even more computation time than learning a single embedding, as the quality of a triplet varies for the same triplet of images varies for each attribute. Instead, a triplet weighting scheme can be used to significantly speed up the training [4]. This weighting scheme uses a heuristic that removes the need for computing the embeddings for triplet selection. A triplet weight for each attribute a under triplet (i, j, k), w_{ijk}^a, is computed by measuring the similarity of the attribute labels of the images within the triplet: y^i, y^j and y^k. This is done using the Jaccard index:

$$J(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}, \tag{2.28}$$

which measures how many elements of set A overlap with B, normalized by the total amount of elements. w^a_{ijk} is then computed as follows:

$$w^a_{ijk} = J(\mathbf{y}^i_a, \mathbf{y}^j_a)(1 - J(\mathbf{y}^i_a, \mathbf{y}^k_a))(1 - J(\mathbf{y}^i \backslash \{\mathbf{y}^i_a\}, \mathbf{y}^j \backslash \{\mathbf{y}^j_a\}))(J(\mathbf{y}^i \backslash \{\mathbf{y}^i_a\}, J(\mathbf{y}^k \backslash \{\mathbf{y}^k_a\})) \tag{2.29}$$

which contains four terms. The first two terms are binary and make sure that the triplet is valid. If the anchor and positive do not have the same label for attribute a, or the anchor and negative do not have a different label, the triplet weight for attribute a will be zero. The other two terms measure the overlap in the entire attribute labels between anchor-positive and anchor-negative without the current attribute a. A low similarity between anchor and positive will lead to a higher triplet weight and a higher similarity between anchor and negative also leads to a higher triplet weight. The intuition is that if two images have the same attribute label for one attribute but they have barely any other attributes in common, then the content of the two images is quite dissimilar and the embeddings may be quite different as well. The same heuristic is applied to the negatives, which effectively identifies those triplets that contain hard positives and hard negatives, which will allow the network to train on informative triplets for better convergence. This way of searching for good triplets without actually computing the embeddings beforehand can be viewed as a way of soft triplet mining [4].

After the network is done learning, classification can be performed by first computing the average embeddings of the training set for each attribute for both the positive and negative label. This leads to both a positive and negative centroid in embedding space for each attribute. For a new image, attributes can be predicted by first mapping the image into the attribute sub-spaces. Then, for each attribute, the distance in the sub-space is computed between the new image and both of the centroids. The attribute is then classified as the closest centroid, which is either 0 or 1.

2.2 Explainable artificial intelligence

2.2.1 Introduction

A famous urban legend in the Machine Learning community is the story of a group of researchers using a Neural Network to learn whether a tank is visible in a photo, by presenting photos of tanks and photos without tanks. While the network was able to perform well on images that the researchers had collected and trained the network on, the network failed spectacularly when presented with novel images. It turned out that the tank photos were taken under different conditions, such as time of day, compared to the non-tank images. The network did not learn to detect a tank, but instead learned to detect the time of day. This story shows potential problems of recklessly applying black-box algorithms such as Neural Networks, without understanding what the underlying model has learned. In the medical field, black-box machine learning algorithms may be quite powerful for detecting certain diseases, but without understanding why the algorithm has made a specific diagnosis or detection makes it tricky to use. XAI is a relatively new field that tries to make black-box models more interpretable. Various different types of algorithms are and have been developed in order to facilitate the use of deep Neural Networks. Some of these methods attempt to visualize the information and structure that a CNN has learned, for example by generating images that maximally activate a given CNN filter, which shows the types of patterns that the filter reacts to [31,32]. These types of methods are useful to check whether the network has learned features that are relevant to the task at hand, but they do not provide insight into decisions made by the model at inference time. Instead, this thesis focuses on methods such as rule extraction that extract simple to understand rules from network attribute predictions and methods that can trace network decisions back to the initial input images, in an attempt to visualize the attention of the network towards certain parts of the input. Such an attention heatmap can indicate which pixels contribute the most to a prediction, allowing insight into why a network makes a decision. In the case of unknown dataset bias, attention heatmaps may be a way of discovering this bias. For example, in the tank story, such heatmaps might have focused on the background instead of the tanks, indicating that the network is focusing on the wrong features for the task at hand.

2.2.2 Rule extraction

Unlike Neural Networks, other machine learning algorithms have been developed that offer a more human-interpretable explanation of its decisions. One example of such a "white box" algorithm is the Classification and Regression Tree (CART) [33]. Such a decision tree is trained and constructed on a set of examples, resulting in a tree with multiple nodes where each node represents an if-then-else rule that tests one of the features from the input example. At the bottom of the tree the leaf nodes reside, which represent the outcome classifications. When given a new example, the first rule on the top of the tree is checked. One of the child nodes of the top node is then visited, based on the answer to the rule. This process repeats until a leaf node is reached, which results in the classification that the leaf node is associated with. Such a decision tree therefore allows for extracting classification rules, which determine precisely how an example should be classified based on a set of simple if-then-else tests concerning the features of the example. Decision trees can be constructed for classification and regression and can handle both continuous and categorical input features. The focus in this thesis is on classification decision trees operating on binary input features. Therefore, the nodes in such a tree will have a positive branch (feature is equal to 1) and a negative branch (feature is equal to 0).

To construct a decision tree, a greedy algorithm is often employed. Given a dataset $\mathbf{D}$ with input samples $\mathbf{x}^i$, each with N features, at each iteration of the algorithm a split is proposed using one of the N features. A measure of impurity is defined, which measures the uncertainty of classification at that point in the tree. A commonly used impurity measure is Gini impurity, which is defined as follows in a multi-class setting:

$$Gini(\mathbf{p}) = 1 - \sum_{j}^{C} \mathbf{p}_j^2, \tag{2.30}$$

where $\mathbf{p}_j$ represents the fraction of examples $\mathbf{x}^i$ belonging to class j. If all examples within a node belong to the same class, the Gini impurity reaches its minimum of zero in that node.

Given n training samples, the initial Gini impurity is computed. Afterwards, each of the N features is selected for a split one by one. After every potential split, two child nodes are created and the training examples are distributed to these nodes according to the condition in the parent node. The impurity can then be computed for the child nodes separately and weighted by the proportion of training examples falling into these nodes:

$$Gini_{split}(\mathbf{p}) = \frac{n^+}{n} Gini(\mathbf{p}^+) + \frac{n^-}{n} Gini(\mathbf{p}^-), \tag{2.31}$$

where n^+ and n^- represent the amount of examples falling into the left and right branch, respectively. $\mathbf{p}^+$ and $\mathbf{p}^-$ are the class proportions of examples falling into the left and right branch, respectively. The impurity reduction can then be computed due to splitting on each of the N features as the difference between impurity before and after the split. The feature that results in the biggest impurity decrease is chosen as the feature to be used for the rule at the current node. This process is repeated recursively for every child node, until every training example ends up in a leaf node and every node, except the leaf nodes, tests the input for a specific feature. The algorithm can be considered greedy, as the best step forward in reducing the

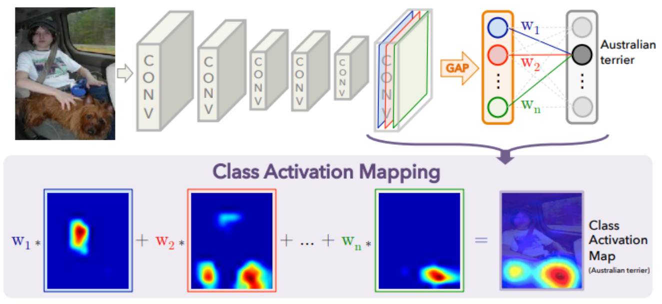
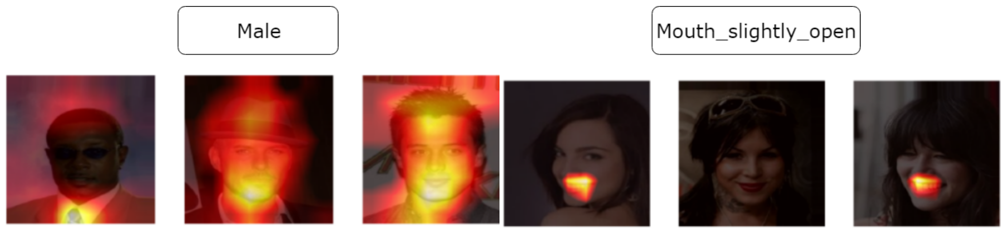


Figure 2.7: Generating class activation maps. For a specific class, this method linearly combines the feature maps that are most relevant for that class in order to generate an attention heatmap, by looking at the weights from that class to the feature maps [5].

impurity is selected by considering only the local information at a given node. Fully growing a decision tree until every training example falls into a leaf node is prone to overfitting, as many rules and branches can be created just to accomodate specific examples, which will not generalize well to unseen examples. Common methods of reducing overfitting in decision tree learning are early stopping and setting a maximum tree depth. Early stopping can be achieved by not doing any further splitting in a node once the impurity within the node reaches a certain minimum, instead of always splitting until impurity reaches 0. The maximum depth of the tree stops the splitting process if this split will result in a tree depth that exceeds the maximum depth. Both the optimal minimum impurity threshold and optimal maximum depth can be determined by cross-validation.

2.2.3 Class activation maps

Class Activation Maps (CAM) are attention heat maps that attempt to localize the presence and extent of a certain object in the input image. By using learned weights or gradient information, the activations of the network can be used in combination with learned filters to localize objects. These types of methods are related to weakly-supervised object localization [34], where only class labels are present and no expensive bounding box annotations have to be used. One method [5] to compute CAM utilizes CNNs with a global average pooling layer. As described in section 2.1.8, the resulting feature vector from the global average pooling results in a mean value for each feature map of the last convolutional layer, which are subsequently mapped to output classes by means of a fully-connected layer. This means that each output



(a) CAM for the attribute 'male'. As can be seen, the heatmaps are concentrated mostly around the chin and a bit around the hair or headwear. For the first image, the CAM is concentrated mostly around the tie, indicating a possible bias in the dataset that mostly men wear ties.

(b) CAM for the attribute 'mouth_slightly_open'. The attention maps are concentrated around the mouth region. The second person does not have her mouth open and the network has a low activation for this attribute. The CAM for this attribute for this image is all zeros, because all pixels contributed negatively to the activation of the corresponding unit.

Figure 2.8: Examples of CAM for two different attributes. The heatmap shows the importance of each pixel to the attribute prediction. Visualisations were generated using the predictions of an Inception V3 [6] model trained on the CelebA dataset [7].

unit of the CNN contains exactly one weight for each of the final channels. As described in section 2.1.7, the deeper layers of a CNN capture more complex, higher level domain-dependent features, such that the final convolution layer contains the highest level of concepts. By combining the activation of filters that detect these concepts, an attention heat map can be created. For example, to localize a dog, we can combine the activation of a filter that detects dog faces, another filter that detects dog tails and another filter that detects a dog's paws. The importance of a feature map for the activation of an output unit is measured by the weights connecting it to the GAP layer. The activation map is computed by taking a linear combination of the final feature maps and these weights:

$$CAM^c = \sum_{i}^{n_d} \mathbf{w}_i^c \mathbf{S}_i, \tag{2.32}$$

where CAM^c is the class activation map for some class c, $\mathbf{S}_i$ is feature map channel i from the output of the final convolution layer, n_d is the total amount of feature maps in the final convolution layer (2048 in Inception V3) and $\mathbf{w}^c \in \mathbb{R}^{1\mathrm{x}n_d}$ contains the weights from the GAP features to class c. This method is visualized in Figure 2.7.

Class activation mapping can also be extended to CNN architectures that do not utilize a global average pooling layer, by using a method called Gradient-Weighted Class Activation Mapping (Grad-CAM) [35]. Without a global average pooling layer, the weights from output units to final feature maps needs to be calculated in a different manner. This can be done by calculating the gradients of an output unit with respect to the final convolution layer feature maps. The positive gradients indicate a positive contribution towards the activation of the output unit of interest. Therefore, for each channel, the gradients of the output unit of interest to the feature maps are computed and averaged, resulting in weights w^c, where a higher weight indicates a higher importance of a feature map:

$$\mathbf{w}_d^c = \frac{1}{HW}\sum_i^H\sum_j^W \frac{\partial \mathbf{o}_c^O}{\partial \mathbf{S}_{ijd}}, \tag{2.33}$$

where w_d^c is the gradient weight of class c for channel d. H is the height of S and W is the width of S. $\mathbf{o}_c^O$ is the output unit corresponding to class c. After calculating the weights, the CAM is computed as in equation 2.32. After generating CAM^c, all negative values are set to zero, as we are only interested in the input pixels that positively contribute to a prediction. With both CAM and Grad-CAM, the computed map is a combination of final feature maps, which usually have smaller height and width compared to the original input image. In order to create an attention map for overlaying the original image, the CAM is upscaled to the original input resolution by using bi-linear interpolation. While CAM and Grad-CAM have been applied in a multi-class setting, not much research has been done on applying these methods in a multi-label setting. This setting can be more interesting, as it can highlight the attention of a network to multiple areas of the same class. Two examples of the CAM method are shown using the CelebA dataset [7] in Figure 2.8. All heatmaps in this thesis use the 'heat' colormap, which ranges from low values to high values in the following order: black-red-yellow-white.

2.3 Zero-shot recognition

2.3.1 Introduction

Zero-shot methods aim to recognize unseen categories by transferring knowledge gained from seen categories. While normal classification models are usually stuck to a set of classes, zero-shot classification attempts to bridge the gap between artificial classification and human-like classification. Theories have been developed on how humans are able to recognize unfamiliar objects by combining familiar components that are present in these objects [16]. Similarly, instead of recognizing objects directly, it might be more useful for machine learning models to recognize the components that visual categories are made up of. When an unfamiliar image is then presented to the algorithm, the components detected within can be combined in order to predict the new category. This can also be used when data of certain categories is sparse. In this scenario, component classifiers can be trained on images that are plentiful, which are then used to classify new classes that can not be properly classified by training on them directly.

In order to realize this, a shared component space has to be defined for each category. Usually, these components are visual and/or semantic attributes [17], which this thesis is focused on. By annotating data with not just the class but also the attributes they contain, we shift the problem from a multi-class to a multi-label classification task, where the attributes are not mutually exclusive. Alternative zero-shot methods have also been studied, where semantic information is not learned per image but are instead treated as auxiliary information, such as a deep visual-semantic embedding model that uses both visual content of images and semantic information from text [36].

2.3.2 Lexicon-based classification

Given a dataset D with classes $\mathbf{c} \in \mathbb{R}^{1\text{x}\mathrm{C}}$ and a set of attributes $\mathbf{a} \in \mathbb{R}^{1\text{x}\mathrm{A}}$, each of the C classes needs to be labeled with an attribute representation $\mathbf{y}_{\mathbf{c}}^{\mathbf{a}} \in \mathbb{R}^{1\text{x}\mathrm{A}}$ and a class label $\mathbf{y}_{\mathbf{c}}^{\mathbf{c}} \in \mathbb{R}^{1\text{x}\mathrm{C}}$, where A is the fixed number of attributes in D. A lexicon $\mathbf{L}^{\mathbf{D}} \in \mathbb{R}^{\mathrm{C}\text{x}\mathrm{A}}$ that maps classes to attributes can then be defined as the collection of class-attribute labels. The attributes should be defined such that each class can be uniquely described by their attribute vector $\mathbf{y}_{\mathbf{a}}^{\mathbf{c}}$. With binary attributes, A attributes can theoretically uniquely describe 2^A different classes. In practice however, many attributes can be heavily correlated, which reduces the uniqueness of many of the labels. Attributes can be a mixture of visual and semantic attributes. While visual attributes are recognizable from an image, semantic attributes can also be detected because of their correlations with visual properties [8].

Two standard ways of classifying attributes and performing lexicon-based zero-shot recognition are Direct Attribute Prediction (DAP) [8] and Indirect Attribute Prediction (IAP) [8], which are illustrated in Figure 2.9. The DAP method first directly learns for each image the attributes that it is annotated with. Classically, this is done by considering the learning of each attribute as a separate binary classification task. Therefore, a separate classifier is used for learning each attribute. For a new image $\mathbf{x}$, the probability of each attribute $\mathbf{a}_i$ occurring in $\mathbf{x}$ is computed with its respective classifier: $p(\mathbf{a}_i|x)$. Doing this for each classifier results in a set of predicted

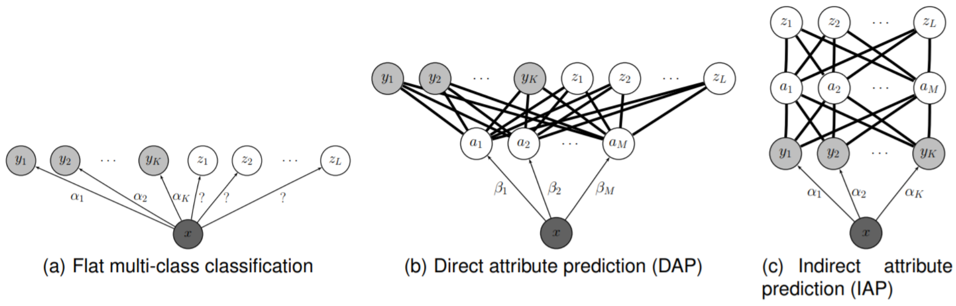

(a) Flat multi-class classification
(b) Direct attribute prediction (DAP)
(c) Indirect attribute prediction (IAP)

Figure 2.9: The DAP and IAP methods [8] visualized. From an image $\mathbf{x}$, DAP predicts a set of attributes $\mathbf{p}(\mathbf{a}|\mathbf{x})$, which maps deterministically to classes via the class-attribute lexicon. The IAP method first predicts classes, then computes the attribute set $\mathbf{p}(\mathbf{a}|\mathbf{x})$ with a weighted sum of class probabilities and class attribute representations from the lexicon. Figure taken from [8].

attribute scores, $\mathbf{p}(\mathbf{a}|\mathbf{x})$:

$$\mathbf{p}(\mathbf{a}|\mathbf{x}) = \{p(\mathbf{a}_1|x), p(\mathbf{a}_2|x), ..., p(\mathbf{a}_A|\mathbf{x})\}_{i=1}^{A} \tag{2.34}$$

The IAP method takes a different approach and first directly predicts the class c_i instead of the attributes: $p(c_i|x)$. This is also done with separate classifiers, where each class is learned in a multi-class setting. This leads to a predicted class vector $p(c|x)$:

$$\mathbf{p}(\mathbf{c}|\mathbf{x}) = \{p(\mathbf{c}_1|x), p(\mathbf{c}_2|\mathbf{x}), ..., p(\mathbf{c}_C|\mathbf{x})\}_{i=1}^{C} \tag{2.35}$$

After computing $\mathbf{p}(\mathbf{c}|\mathbf{x})$, $\mathbf{p}(\mathbf{a}|\mathbf{x})$ is computed with a linear combination of the predicted class scores $p(\mathbf{c}_i|x)$ and their respective lexicon attributes $\mathbf{L^D}_i$:

$$\mathbf{p}(\mathbf{a}|\mathbf{x}) = \sum_{i}^{C} p(\mathbf{c}_i|x)\mathbf{L^D}_i \tag{2.36}$$

The IAP method may have an advantage in that it considers only valid class-attribute combinations, which are then summed [8]. With the DAP method, attributes can be detected that do not occur at all in the correct class. However, a disadvantage of IAP is the fact that classes have to be learned directly, which is not effective when certain classes are rare. Both the DAP and IAP methods are illustrated in Figure 2.9.

When $\mathbf{p}(\mathbf{a}|\mathbf{x})$ is computed using either the DAP or IAP method, classification can be performed using either a distance metric or by using a probabilistic approach, which both use the prior knowledge stored in the lexicon. Distance functions like the Hamming distance can be used to compare $\mathbf{p}(\mathbf{a}|\mathbf{x})$ with all attribute vectors in $\mathbf{L}^{\mathbf{D}}$ [37]. The closest class is then assigned to $\mathbf{x}$. Since the elements of $\mathbf{p}(\mathbf{a}|\mathbf{x})$ are probabilistic, an appropriate threshold has to be applied to each, since the Hamming distance compares two binary vectors. Alternatively, a Bayesian approach can be adopted for finding the most likely class [8]. The probability of each unseen class z given image $\mathbf{x}$ is computed as follows:

$$\mathbf{p}(\mathbf{z}|\mathbf{x}) = \prod_{i=1}^{|\mathbf{a}^z|} \frac{p(\mathbf{a}_i^z|\mathbf{x})}{p(\mathbf{a}_i^z)}, \tag{2.37}$$

where $\mathbf{p}(\mathbf{a}^{\mathbf{z}}|\mathbf{x})$ is the predicted attribute vector, except only the attributes belonging to class z are taken into account, which is specified by $\mathbf{L}^{\mathbf{D}}{}_z$. $|\mathbf{a}^z|$ is the length of the attribute vector and is thus the number of attributes for class z. $p(\mathbf{a}^z)$ is the vector of attribute priors, where again only attributes of class z are used. Therefore, the posterior of class z is computed as the product of the probabilities of the attributes of class z, divided by the priors of those attributes. This reduces the importance of attributes with a higher prior, as they occur often in many classes and thus are less discriminative for a specific class. The priors $\mathbf{p}(\mathbf{a}^{\mathbf{z}})$ can be computed from the distribution of attributes over classes, found in $\mathbf{L}^{\mathbf{D}}$. The image is then assigned to the class with the highest posterior. Alternatively, a different Bayesian approach can be taken that takes every attribute into account, not just the occurring ones for a specific class:

$$\mathbf{p}(\mathbf{z}|\mathbf{x}) = \prod_{i=1}^{A} \frac{p(\mathbf{a}_i|\mathbf{x})}{p(\mathbf{a}_i)}, \tag{2.38}$$

where $p(\mathbf{a}_i|\mathbf{x}) = 1 - p(\mathbf{a}_i|\mathbf{x})$ when attribute $\mathbf{a}_i$ does not occur for class z. The setting described above is known as standard or closed-set zero-shot recognition, where training and testing classes are completely distinct and at testing time only test classes occur. When training and testing classes both occur at testing time, performance drops drastically [15]. This setting is called generalized zero-shot learning and is thought to be more practical than closed-set zero-shot, as in an actual application of zero-shot recognition it may be difficult to determine whether only test classes occur at test times and which ones. Generalized zero-shot learning is beyond the scope of this thesis.

2.3.3 Attribute labelling

While datasets with attribute labels have been constructed, the way images are annotated can vary. One of the standard ways of attribute labelling is to specify labels for each class, then assign each image containing that class its corresponding attributes. This is done in datasets like the Animals with Attributes 2 (AwA2) dataset [15]. This significantly reduces the cost of labelling, as there are typically a lot fewer classes compared to images. Moreover, a lexicon with deterministic class-to-attribute relationships can easily be built with this kind of labelling. However, a downside to this annotation type is that the attribute labels will not always correspond to the actual content in the image. It can be argued that labelling images in this way will hamper attribute recognition [38]. No visual evidence for specific attributes can be found in images with class-level attribute labels instead of instance-level attribute levels. For example, every image belonging to the class ”cat” will have the ”tail” attribute, even if no tail is present in a specific cat image. This might force deep models to learn correlations between attributes instead of learning specific visual features for attributes, as these visual features are often not present despite the annotation indicating that the attribute is present. These correlations may be weaker or non-existant in unseen classes, possibly leading to a lower zero-shot performance. It has been shown that the DAP and IAP methods perform badly in a generalized zero-shot setting [15], possibly as a result of this problem.

Chapter 3

Methods

3.1 Introduction

In this section the experiments that are performed are described in detail. These experiments attempt to answer the research questions that were proposed in section 1.3. To study the effects of attributes on standard classification, classification experiments with and without attributes are described in section 3.4. Zero-shot classification experiments are described in section 3.5. In section 3.6 the explainability experiments are described, which involve the techniques discussed in section 2.2. First, before describing the experiments, the next sections describe the datasets and CNN architecture used in this thesis.

3.2 Datasets

Three datasets of different sizes are used for the experiments, ranging from small to large. The SUN attributes dataset [39, 40] is a small dataset consisting of 14,340 images of 717 different scene categories with 20 images per category. Each image is annotated with 102 attributes, so the dataset has instance-wise labelling instead of class-wise labelling. Due to the small number of images but large amount of classes and attributes, it is expected that achieving good recognition performance on this dataset is difficult. The Animals With Attributes 2 (AWA2) dataset is a medium-size dataset containing 37,322 images of 50 animals, where each animal is annotated with 85 attributes. Therefore, this dataset contains class-wise labelling. Finally, the Large-Scale Attribute Dataset (LAD) [41] is one of the biggest publicly available attribute datasets for zero-shot learning, containing 78,017 images of 230 classes, which are divided into 5 superclasses: animals, fruits, vehicles, electronics and hairstyles. The dataset is labelled in a class-wise fashion and has a total of 359 different attributes.

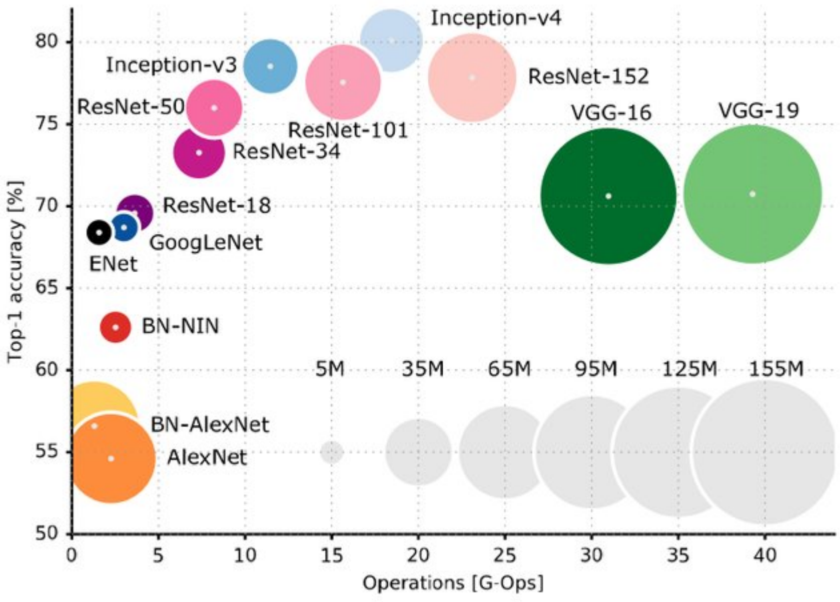


Figure 3.1: Operations versus accuracy on ImageNet of different CNN architectures [9].

3.3 Model and hyper-parameters

The architecture that is used for each experiment is the Inception V3 network, as described in section 2.1.8. This network has been shown to perform well on ImageNet and has a relatively low amount of operations compared to newer and deeper models, as can be seen in figure 3.1. Due to limited computational resources, this network seems like a good choice. Another reason for choosing Inception V3 is the size of the datasets that are available for zero-shot learning. Common datasets used for zeroshot are typically not very large, ranging roughly from 10,000 to 80,000 images, which are all relatively small when it comes to training very deep models. Employing a model with a lot of layers is not used to prevent overfitting. Because of the small dataset sizes, transfer learning is employed using an Inception V3 network pre-trained on ImageNet. The pre-trained network will have already learned a lot of useful features in its many layers, which may alleviate some of the problems that occur when the amount of data is limited. Also, some of the classes of AWA2, SUN and LAD occur in ImageNet, so certain class-relevant features may have already been learned.

Both the Tensorflow [42] and Keras [43] deep learning frameworks are used for training the models. Inception V3 is fine-tuned with a learning rate α of 0.00001. Every time the loss on the validation set does not improve for 3 epochs, the learning rate is halved. If the loss on the validation set does not improve for 10 epochs, the entire training is stopped. This encourages the network to reach a low minimum validation loss. Every time the model reaches a lower minimum compared to before, the network's weight are saved. After training, the model with the lowest validation loss is selected. No regularization techniques besides those intrinsic to Inception V3 are used. No data augmentation such as cropping, shearing or adding noise is added, as these methods did not seem to improve the performance of the network based on preliminary experiments. Cropping can be especially harmful for multi-label classification, as generating crops that are too small might obscure parts of the image that are important for certain attributes. Images are normalized with min-max scaling by dividing each image by 255, assuming that the global minimum and maximum pixel intensities are 0 and 255, respectively. Standardization by subtracting the mean and dividing by the standard deviation of the training images was also tried, but this did not improve the network's learning behavior. Since the pixel means and standard deviations have to be computed and stored for every training set, it is much more convenient to use the alternative min-max scaling method, which seemed to work just as good. The images are resized to 299×299, as these are the standard input dimensions for Inception V3. The Adam [44] optimizer is used for training. The settings described apply to each experiment, unless explicitly stated otherwise.

3.4 Standard classification experiments

In the standard classification experiments, various models are trained to distinguish between categories, both with and without attributes, to see whether classification accuracy can be improved when including attributes in the training process. Since the SUN dataset has a small amount of images and a large amount of classes, learning the classes directly or indirectly with attributes does not lead to satisfactory results. Therefore, no standard classification experiments are performed on this dataset. However, the attribute recognition performance will be computed in all settings for all datasets, including SUN, as the attributes can be recognized. The attribute recognition is measured using the accuracy, precision, recall and f1-score performance measures, which are computed for each attribute separately and then averaged. The accuracy is measured by computing the accuracy separately on the positive and negative class of each attribute and then averaged.

For classification, classes will first be directly learned and attribute annotations will be ignored. The 1000-way softmax layers of the pre-trained Inception V3 model are discarded. The 2048-dimensional global average pooling feature vector is kept as the final layer from the original model and a dense layer of perceptron units is added on top of it. The amount of units is equal to the number of classes. The activations of the final layer are squashed into a probability distribution by using the softmax function:

$$\sigma(\mathbf{o}_i^O) = \frac{\mathrm{e}^{\mathbf{o}_i^O}}{\sum_j^C \mathrm{e}^{\mathbf{o}_j^O}}, \tag{3.1}$$

where $\mathbf{o}_i^O$ is the output of output unit i and C is the number of classes. The categorical cross-entropy loss is the multi-class cost function employed in this setting. Since the occurrences of the classes are not uniform, a weighted categorical cross-entropy loss is employed. First, the count of the most frequent class, N_m, is computed. Then, the weights are computed proportionally to the maximum count:

$$w_i = \frac{N_m}{N_i}, \tag{3.2}$$

where w_i is the weight for class i and N_i is the amount of images of class i.

In the second setting, category labels are ignored and instead the attribute annotations are used to learn to predict the attributes. The architecture is the same as described above, except the amount of units in the final dense layer is equal to the number of attributes instead of classes. The attributes are learned in a multi-label setting, so output categories are not mutually exclusive. While attributes can be learned independently, learning them together by sharing the same layers may help with attribute recognition due to correlations and relationships between them. The activation of the attribute units in the final layer are first squashed into the range [0,1] with the sigmoid function, then the binary cross-entropy loss is used for each unit. Since certain attributes are quite rare, a weighted binary cross-entropy loss is used. First, the proportion of positive samples of each attribute is computed:

$$\mathbf{p}_i = \frac{1}{N}\sum_j^N \mathbf{y}_{ji}^a, \tag{3.3}$$

where $\mathbf{p}_i$ is the prior for attribute i. N can be the number of classes or the total number of images in the dataset. $\mathbf{y}_{ji}^a$ is the label of attribute i for image or class j. Several weighting schemes exist and perform differently depending on the data itself. For the AWA2 and LAD datasets, a weighting scheme is used that increases the weight of rare attributes but does not define weights separately for the positive and negative class of the attribute. The weight for each attribute i is computed as follows:

$$w_i = -2log(\mathbf{p}_i), \tag{3.4}$$

which will assign a larger weight to rare attributes and a relatively lower weight to attributes that occur frequently.

In the third experiment, all available labels will be used by learning both classes and attributes at the same time. The model will contain two output layers and share the same GAP input layer. Learning attributes in addition to learning classes can lead to improved recognition [45]. Both the classes and attributes use their respective weights as described above. The loss function is computed as the sum of both the categorical and binary cross-entropy losses. After training, the class output layer is extracted and used for classification.

The third type of model is a similarity learning model that learns an embedding for each attribute with the multi-label triplet loss as described in section 2.1.11. The anchor α is empirically set to 1.0, but preliminary results have shown that the value of α does not seem to impact the results all that much. Each attribute embedding is normalized by dividing it by its norm, as this has shown to lead to better convergence [3]. The maximum distance between two points in the d-dimensional hypersphere is 2, so a margin α of 1.0 means that the loss tries to enforce a distance between positives and negatives equal to half the embedding space. The attribute embeddings themselves consist of one dense layer of perceptron units per attribute. The embedding size is set to 10. All embeddings take as input the 2048-dimensional feature vector as described before. Triplets are generated by uniformly sampling them together in the current batch. The batch size is set to 189. The 189 images are divided into 63 triplets, which are subsequently fed to the network in a batch of 21 images. Each batch therefore requires 3 forward and backward passes to the network. Since the triplet generation is random, some potentially useful triplets may not be generated at all. To increase the usage of the data in a batch, the procedure described above is repeated a number of times, to try and sample as many possible triplets as possible in the batch. This number of re-sampling is fixed to 20. This increases the number of forward and backward passes to the network in each batch to 60.

After training, the attribute embedding centroids for both the positive and negative class are computed by first feeding all training images to the network. The attribute embeddings are then extracted and averaged for both the positive and negative classes separately, resulting in a positive and negative centroid that are subsequently stored. During testing, test attribute embeddings are extracted from the network from images. The distance of each attribute embedding to its respective positive and negative centroid is computed using the squared Euclidean distance function, which is the same distance function used in the triplet loss. If the test embedding for some attribute is closer to the positive centroid than the negative or vice-versa, it is likely that the attribute falls into that category.

A mapping from the embedding to a binary attribute prediction would disallow the use of Bayesian approaches. Instead, The two distances are normalized using the softmax function to compute attribute prediction $p(\mathbf{a}_i|\mathbf{x}_j)$ given test image $\mathbf{x}_j$ and attribute $\mathbf{a}_i$:

$$p(a_i|\mathbf{x}_j) = 1 - \frac{\exp^{||f(\mathbf{x}_j)_i - \mu_i^+||^2}}{\exp^{||f(\mathbf{x}_j)_i - \mu_i^+||^2} + \exp^{||f(\mathbf{x}_j)_i - \mu_i^-||^2}}, \tag{3.5}$$

where $||f(\mathbf{x}_j)_i - \mu_i^+||$ and $||f(\mathbf{x}_j)_i - \mu_i^-||$ are the distances from the ith embedding to the ith positive centroid μ_i^+ and ith negative centroid μ_i^-, respectively. Lexicon classification then proceeds as usual. It is expected that the triplet models perform better when the amount of images is low, such as for the SUN dataset. For learning using a standard loss function such as cross-entropy, each image only contributes one data point to the classifier in each training iteration. With the triplet loss, the same image is used multiple times in different combinations with other images and in different positions in the triplet, within the same iteration. This increases the usage of the limited data available.

For the standard classification experiments, the data is randomly split into five different folds. The model is then trained and evaluated in a 5-fold cross validation setting, where 4 folds are used for training and 1 fold for testing. Additionally, the testing fold is split in half. One halve serves as the validation set and the other halve as the test set, leading to a 80%-10%-10% train-validation-test split. The performance metrics used depend on the specific model. For class-only models, the mean class accuracy is measured instead of the overall average accuracy on the test set. Since some classes are more rare compared to others, computing the accuracy per class should give a better indication of model performance. For the attribute models, the mean class accuracy metric is also used. Additionally, the accuracy, precision, recall and f1-scores are computed for each attribute.

One method to compute the class accuracy with attributes is to use the predicted attributes together with the lexicon. For lexicon-based classification, both the Hamming distance and the two Bayesian approaches described in section 2.3.2. are used. Since the Hamming distance requires two vectors with binary elements, two different kinds of thresholds are applied. The first threshold is a simple 0.5 threshold, where everything below 0.5 is set to 0 and everything above or equal to 0.5 is set to 1. The second threshold takes into account the positive proportion of each attribute by setting the threshold equal to the prior. Therefore, the thresholds differ among the attributes. For example, if an attribute is quite rare and only occurs in 10% of the images/classes, its threshold is set to 10%, where everything below it is set to 0 and everything equal to or higher is set to 1. The intuition is that predictions for rare attribute may be lower than more common attributes, so a more lenient threshold may be necessary to still detect them. Conversely, a very common attribute receives a high threshold, such that the attribute prediction is only set to 1 if the network is very certain that the attribute is present, as the predictions for a common attribute might be high for many test images.

A second method of computing class accuracy is using the learned attributes as high-level features that serve as input to another learning algorithm. Both a decision tree and an MLP will be trained to map attributes to classes. The amount of hidden nodes and learning rate for the MLP and the maximum depth and early stopping threshold for the decision tree are determined with cross-validation. The raw attribute predictions are used as input for the MLP, while the predictions are binarized for the decision tree.

Since a pre-trained network is used, many of the layers already contain learned features. A decision has to made which layers to finetune and which layers to freeze. Finetuning weights that can not be improved further might disturb the learning of the network. The right amount of layers to train is also determined by cross-validation. For all AWA2 experiments, Only the top 2 Inception blocks and the layers afterwards are trained, while all other layers are frozen. For both the SUN and LAD experiments all layers are trained.

3.5 Zero-shot experiments

In the zero-shot experiments, the same kinds of models are trained as described in the section above: a model that directly learns the classes of images, a model that directly learns the attributes, a model that learns both attributes and classes and a model that learns attribute embeddings. For the model that learns both classes and attributes simultaneously, the attribute layer is extracted instead of the class layer for zero-shot classification. The performance measures and distance functions used are the same as in the standard classification experiments. Attribute recognition performance is also measured, to see if attributes on unseen classes can still be recognized adequately. The same four performance measures as in the standard experiments are used, with one exception. Since classes are distinct between train, validation and test set, it's possible that for some data folds certain attributes do not occur at all. For example, the 'bat' class from the AWA2 dataset is the only category containing the 'flies' attribute. This means that in the fold that contains the 'bat' class images in the test set, the model in this setting only encounters the negative class of ′flies′. During testing, the model will only predicts zeros for this attribute, which will unfairly lower the attribute recognition performance. Conversely, if the ′bat′ class is in the validation set, the attribute is neither trained nor tested upon. In this case, the labels for the ′flies′ attribute will be all zeros and the model will also predict all zeros, which will unfairly increase the prediction performance. To resolve these issues, before computing the performance on the test set, the amount of attributes which have at least 1 positive in the test set is counted. The performance is then summed per attribute and divided by the amount of these ′valid′ attributes, instead of dividing by the total number of attributes. All four performance measures are set to zero for ′invalid′ attributes, effectively ignoring them when computing this adjusted average.

For zero-shot recognition, 5 train/validation/test splits are again created to test the models on different folds of the data. However, in this setting the classes within the train, validation and test set are distinct, as no classes from the test class are allowed to be seen by the model, to evaluate performance on unseen classes. To measure how the models' cross-class generalization is improving during training, the classes in the training set and validation set are also distinct. The zero-shot classification is done in a closed-set manner, such that only testing classes are considered in the testing phase. AWA2 has 50 different classes, of which 35 are put into the training set, 5 into the validation set and 10 into the test set. Only 5 validation classes are used to allow for more training data. 10 test classes are used, which allows comparison to other zero-shot results on AWA2 from the literature. One of the five folds is the original train/test class split that was proposed by the creators of the dataset [8,15,18], to allow for better comparison with previous methods, except that 5 training classes are put into the validation set. The other four folds are created randomly in a such a way that each of the 50 classes occurs exactly once in one of the test sets. The overall performance is computed by averaging the performance over the 5 folds.

The SUN dataset contains 717 classes. A similar approach to previous work is taken, where 580 classes are used for training, 60 for validation and 72 for testing [15]. Again, 5 different folds with different train/validation/test classes are created to improve the robustness of the results. Unlike AWA2, a weighting scheme is used that takes into account both the positive and negative class of each attribute [46]:

$$w_i^+ = \mathrm{e}^{1-\mathbf{p_i}} \tag{3.6}$$

$$w_i^- = \mathrm{e}^{\mathbf{p_i}}, \tag{3.7}$$

where w_i^+ is the weight for the positive class of attribute i and w_i^- the negative one. As the prior of an attribute gets closer to 1, the positive weight approaches 1 as well, such that its weight in the binary cross-entropy is weighted as normal. If an attribute is rare such that its prior gets closer to zero, the negative weight tends to e. This multi-label weighting scheme outperformed the weighting scheme used for the AWA2 dataset. To determine this, it does not help to look at the loss curves of two models trained using different weighting schemes. This is because the weights have a large impact on the error and the local minima that the model is able to reach and is not very indicative of model performance. A model without weighting is able to achieve a very low cross-entropy loss on SUN, but performs poorly on new data and zero-shot recognition as attributes are predicted as the majority class due to the imbalanced attribute distribution. Instead, the weighting schemes were compared by performing zero-shot classification on the validation set. The labels of SUN are labelled per image and not per class and both the class-attribute matrix and the attribute prior probabilities have to be computed. The labels from SUN are not binary but contain four values: 0, 0.333, 0.666 and 1. Training directly on these labels did not lead to improved results, so these labels were instead binarized with a threshold of 0.5. After binarization, the lexicon is built by computing the average occurrence of each attribute for each class. These averages are then binarized again. The priors are computed by averaging the attributes over each class in the lexicon.

3.6 Explainability

The explainability experiments that are performed include rule-extraction and generating visual attention maps. For rule extraction a decision tree is employed, which takes as input the binarized attribute predictions from the attribute model described in the standard classification experiments section. The decision tree uses the attribute predictions to predict the class. On a test image, the attribute network first predicts the attributes, which are then binarized with a 0.5 threshold and provided to the decision tree, which learns to map the predicted attributes to classes. The thresholding is done to allow for simple binary rules that specify whether the attribute is present or not. Additionally, the nodes in the decision tree leading to the classification are extracted, such that the rules are shown which specify which attributes and which rules were used for classification. This leads to a conjunction of binary rules for each class. The same 5 training, validation and testing folds as those in the standard classification experiments are used for the decision tree. The parameters are selected by evaluating the average performance of the decision tree on the 5 validation folds. The decision tree is then trained on both the training and validation data with the best parameters. This is repeated for all 5 five folds and the average accuracy is measured. The mean class accuracy for the decision tree is reported. Rule extraction is applied on both the AWA2 and LAD dataset.

For the attention maps Grad-CAM is used to visualize for certain attributes whether the network can give a pixel-wise explanation for a specific attribute prediction. Both of these methods are related to weakly-supervised localization methods and the datasets do not have ground-truth annotations for attribute locations. Therefore, no objective measure of quality can be assigned to these visualisations, so these will instead be subject to subjective analysis. The attention map experiments are conducted on all three datasets.

For each of the five folds of the SUN dataset, one test class is selected. The attention heatmaps will only be generated for the zero-shot setting, to see whether the weakly-supervised methods work on unseen classes. For the AWA2 and LAD datasets, only two examples will be shown, as they are expected to be less interesting due to their class-wise labelling. For decision tree rule-extraction, 5 classes are selected from both the AWA2 and LAD datasets. This is done only for the standard classification setting, as explicit attribute-to-class mapping is not possible for zero-shot classification.

Chapter 4

Results

In this section, the results of each of the three types of experiments, as described above, are displayed. The results are compared to earlier studies done on zero-shot learning on the datasets used in this research. The explainability results will be discussed and interpreted.

4.1 Standard classification

All standard classification results of AWA2 where attributes are predicted are visualized in Figure 4.1. The three methods are quite similar in their performance, with the DAP method having a slight edge, while the other two methods have lower variance, especially the triplet model results. The triplet model also has a slightly lower performance compared to the other two methods. The highest classification performance is achieved with the DAP model using the standard Bayes method [8, 18], resulting in an average performance of 90%.

The results of models that predict classes without using the lexicon are shown in Figure 4.2. The highest performing model is the one that directly predicts the classes, but this method also has higher variance compared to the other models. The model that learned both attributes and classes at the same time has a slightly lower performance but is more stable. One explanation for this is that its embedding is more generalizable since it has learned to facilitate two different tasks. The MLP that learns to map from predicted attributes to classes was not able to outperform directly learning the classes. It seems that the high-level attribute predictions are worse features compared to the GAP features. The decision tree has the lowest performance of the four methods, but still performs adequately. The 'direct' method results in an average performance of 91%, which is slightly higher than any of the indirect, attribute methods from Figure 4.1.

The attribute classification performances of the three attribute methods are shown in Table 4.1. All three methods achieve a high performance on attribute recognition, which seems to be why their lexicon performances are similar to each other.

While no SUN standard classification is done, the attribute recognition is still

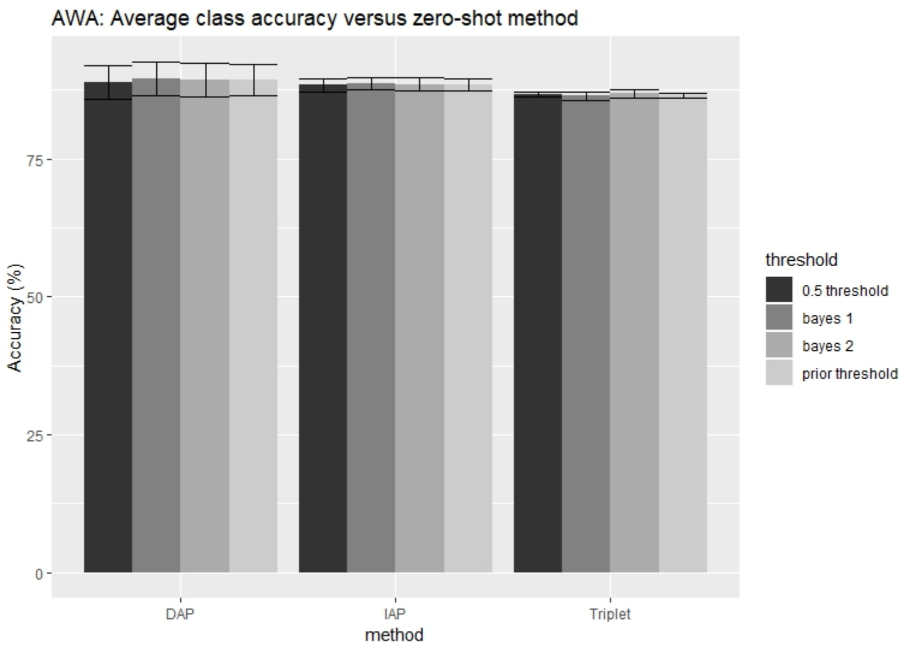


Figure 4.1: Standard class recognition accuracy on the AWA2 dataset for the three different attribute classification methods. The results for the two threshold methods and two Bayesian methods are also shown.

Method	Accuracy	Precision	Recall	F1-score
DAP	**0.97**	**0.98**	0.96	**0.97**
IAP	**0.97**	**0.98**	**0.97**	**0.97**
Triplet	**0.97**	0.97	**0.97**	**0.97**

Table 4.1: Attribute recognition performance on the AWA2 dataset averaged over 5 folds for the DAP, IAP and multi-label triplet methods. Best results are shown in bold.

measured and shown in Table 4.2. The DAP method outperforms the triplet method when it comes to attribute recognition, as the triplet method falters when it comes to precision but has a higher recall than the DAP method. This indicates that the triplet method makes too many positive predictions, leading to a lot of false positives, which reduce the precision of the classifier but are not taken into account in the calculation of the recall. Since the SUN dataset has less images but more at-

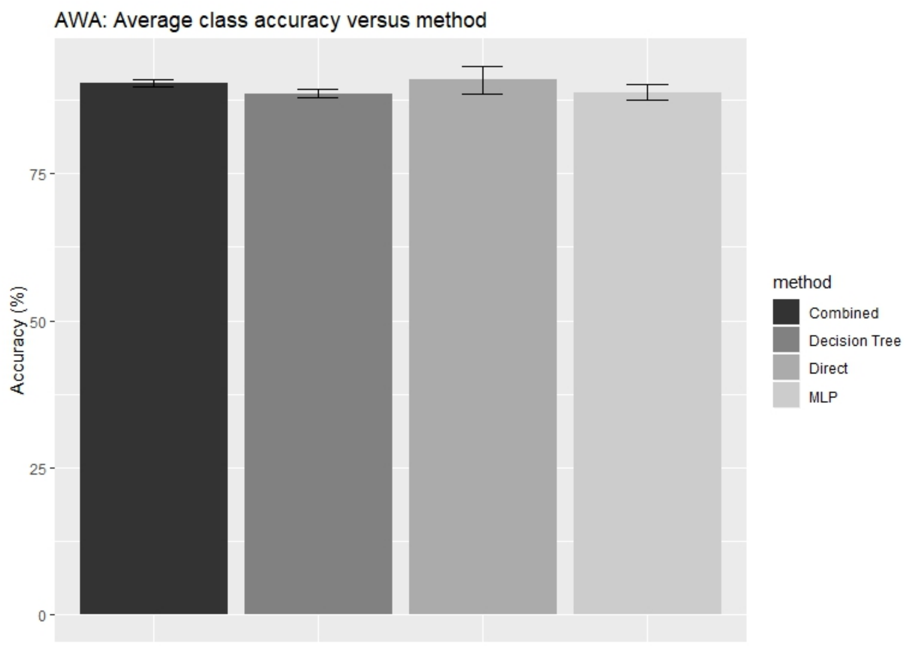


Figure 4.2: Standard class recognition accuracy on the AWA2 dataset for directly learning the classes both with and without attributes, using a decision tree and an MLP.

Method	Accuracy	Precision	Recall	F1-score
DAP	0.77	**0.45**	0.58	**0.50**
Triplet	**0.78**	0.3	**0.64**	0.38

Table 4.2: Attribute recognition performance on the SUN dataset averaged over 5 folds for the DAP, and multi-label triplet methods. Best results are shown in bold.

tributes than the AWA2 dataset, it could be the case that the triplet network was not able to separate the positive and negative centroids for each attribute well enough due to the lack of data.

Similar to the AWA2 results, the lexicon-based results on the LAD dataset are shown in Figure 4.3. Here, the IAP method seems to outperform the other two methods, leading to an average highest performance of 87% with the 0.5 attribute thresholding. This is higher than previous research on standard classification on this dataset, which resulted in a recognition accuracy of 79.52% [41], also using the

Method	Accuracy	Precision	Recall	F1-score
DAP	**0.98**	0.96	0.95	**0.96**
IAP	0.97	**0.97**	**0.97**	**0.96**
Triplet	0.96	0.87	0.93	0.9

Table 4.3: Attribute recognition performance on the LAD dataset averaged over 5 folds for the DAP, IAP and multi-label triplet methods. Best results are shown in bold.

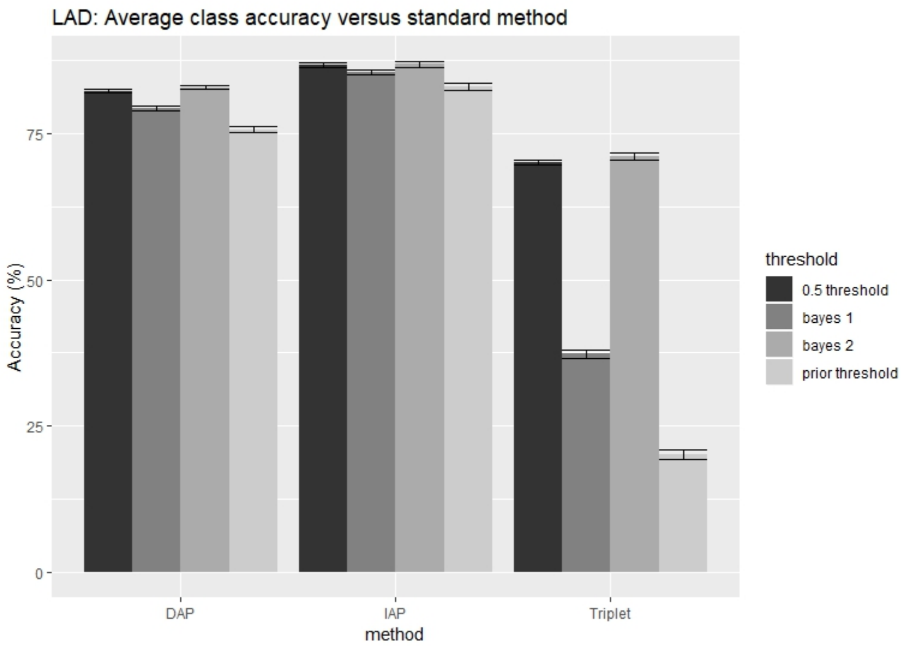


Figure 4.3: Standard class recognition accuracy on the LAD dataset for the three different attribute classification methods. The results for the two threshold methods and two Bayesian methods are also shown.

Inception V3 architecture. The possible improvement in this thesis could be due to the class weighting used. The triplet method seems to perform the worst out of the three. In Figure 4.4 the direct class-prediction model results are shown for the LAD dataset. The MLP model and decision tree model perform worse than the other two. Possibly, the same weakness of the high-level attribute prediction features versus GAP features that was also mentioned before might be the case here. The highest

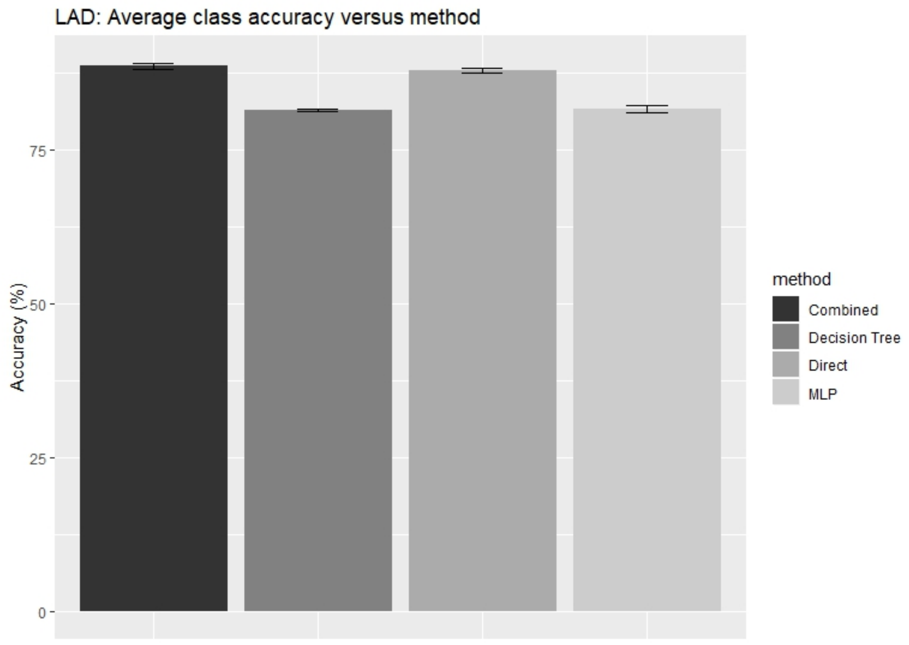


Figure 4.4: Standard class recognition accuracy on the LAD dataset for directly learning the classes both with and without attributes, using a decision tree and an MLP.

recognition performance is by the model that learns both attributes and classes together and achieves an average performance of 89%, which is slightly higher than the highest-performing lexicon-based method. In Table 4.3, the attribute recognition performance is shown. Again, the triplet method seems to perform the worst due to its low precision, which is also observed in the SUN results. One possible explanation for this bad performance is that the LAD dataset contains 359 attributes. It might be the case that not enough data is present in the dataset to properly train 359 10-dimensional embeddings. Since the attributes are not as correctly recognized compared to the other two methods, the lexicon-based classification also suffers.

4.2 Zero-shot recognition

The zero-shot results on the AWA2 dataset are shown in Figure 4.5. Compared to the standard classification results, the classification performance drops from 90% to 58% when dealing with unseen classes. The variance of each method is also significantly higher. This is because the results are averaged over the 5 data splits, but the data splits in the zero-shot setting have entirely different train/validation/test class splits. Each of the five splits therefore has a different distribution of attributes and this can have a big impact on performance. The best performing methods on average are the DAP and triplet method in combination with the 'bayes 2' lexicon matching method. The DAP method has an average classification performance of 59%. Learning both attributes and classes can lead to the highest performance on some of the data splits, as can be seen by the error bars. Previous works have used the traditional DAP method (individual classifier per attribute) with Resnet-101 [47] features, which achieved a classification performance of 58.7% on the standard split from the dataset and 46.1% on a different split [15]. The 'combined' network from this thesis achieves a zero-shot classification accuracy of 67% on the original AWA2 train/test split, which shows that sharing a global representation input with all classifiers and training the classifiers end-to-end together with the rest of the network can significantly outperform the traditional method. It is not known which train/test class split is present in the different split. The 'combined' network achieves an accuracy of 54% on the 'hardest' data split, which is the lowest accuracy of all the five data splits. While this is most likely not the same split as the one from the paper, it is the closest approximation, as the different split is proposed as a harder split compared to the original one. However, the methods from this thesis are outperformed by some of the other zero-shot methods, which are described in that paper.

The zero-shot attribute recognition performance is shown in Table 4.4. The highest average F1-score is 0.61 and is achieved by the triplet method. This is a significant drop from the F1-score of 0.97 on the attribute recognition performance of the standard setting. When the network encounters unseen images, recognition of attributes takes a large hit, most likely because the test images are very different than what the network is used to. The triplet method performs well but still has a slightly lower average precision compared to the DAP and IAP methods.

Next, the zero-shot performance on the SUN dataset is shown in Figure 4.6. The best performance is achieved by the DAP method with an average zero-shot classification accuracy of 30%. This is unfortunately lower compared to the standard DAP method, which was able to achieve a classification performance of 38% using the pre-trained Resnet-101 architecture as feature extractor [15]. It seems that the standard method, which uses individual classifiers that are not trained together with the deep feature extraction module, has an advantage over the end-to-end learning approach used in this thesis when the amount of images is low. The IAP performance on the SUN data is very low, as this method requires directly learning all 717 classes with only 20 instances each, which proves to be too challenging for the model. The attribute recognition performance on the SUN dataset is shown in Table 4.5. The

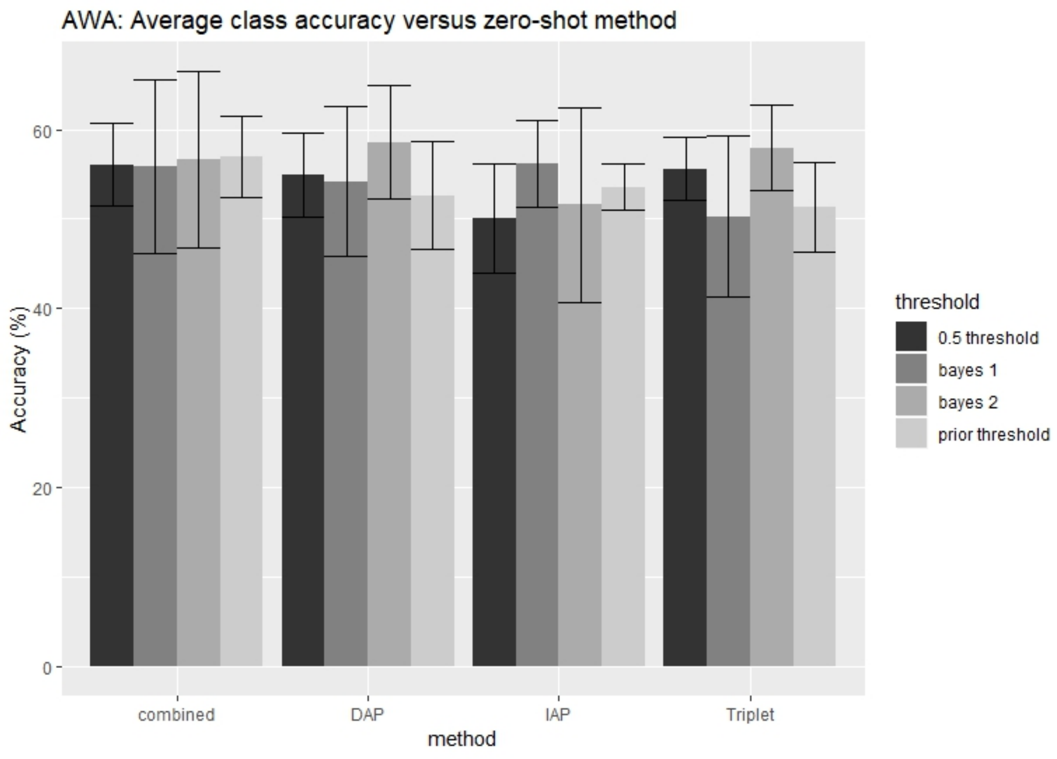


Figure 4.5: Zero-shot class recognition accuracy on the AWA2 dataset for the three different attribute classification methods. The results for the two threshold methods and two Bayesian methods are also shown.

same problems with regards to the triplet method as in the standard classification setting can be seen here as well. Unfortunately, the theory about the triplet method working well when the amount of images is low seems to be false. Interestingly, the F1-score of the DAP method only drops from 0.5 to 0.47, which is not nearly as significant as the drop observed on the AWA2 and LAD dataset.

Lastly, the zero-shot classification results on the LAD dataset are shown in Figure 4.7. The method that learns classes and attributes simultaneously outperforms the other 3 methods, unlike the AWA2 results, resulting in an average zero-shot accuracy of 44%, possibly due to the larger amount of images which may be required to learn a strong, generalizable GAP embedding for both tasks. Again, the triplet method falters due to its low precision, which can be seen in Table 4.6. Similar to the AWA2 results, the F1-score takes a big drop from 0.96 to 0.56. 7 zero-shot

Method	Accuracy	Precision	Recall	F1-score
DAP	**0.7**	**0.71**	0.6	0.6
IAP	0.68	0.7	0.59	0.59
Triplet	**0.7**	0.68	**0.62**	**0.61**

Table 4.4: Zero-shot attribute recognition performance on the AWA2 dataset, averaged over 5 folds for the DAP, IAP and multi-label triplet methods. Best results are shown in bold.

Method	Accuracy	Precision	Recall	F1-score
DAP	**0.76**	**0.43**	0.55	**0.47**
Triplet	**0.76**	0.31	**0.59**	0.38

Table 4.5: Zero-shot attribute recognition performance on the SUN dataset, averaged over 5 folds for the DAP and multi-label triplet methods. Best results are shown in bold.

methods were tested on LAD in previous research [41], which did not include the DAP/IAP methods. The DAP model from this thesis outperforms two of the seven zero-shot algorithms and is on par with 1 on them, which shows that deep attribute learning should not be written off as an ineffective zero-shot method. Furthermore, it seems that the Bayesian approaches are the most effective way of classifying with the lexicon, as they can be used in their raw form and do not need a well-chosen threshold to be found.

Method	Accuracy	Precision	Recall	F1-score
DAP	0.78	0.64	0.57	**0.56**
IAP	0.71	**0.65**	0.44	0.49
Triplet	**0.83**	0.52	**0.7**	**0.56**

Table 4.6: Zero-shot attribute recognition performance on the LAD dataset, averaged over 5 folds for the DAP, IAP and multi-label triplet methods. Best results are shown in bold.

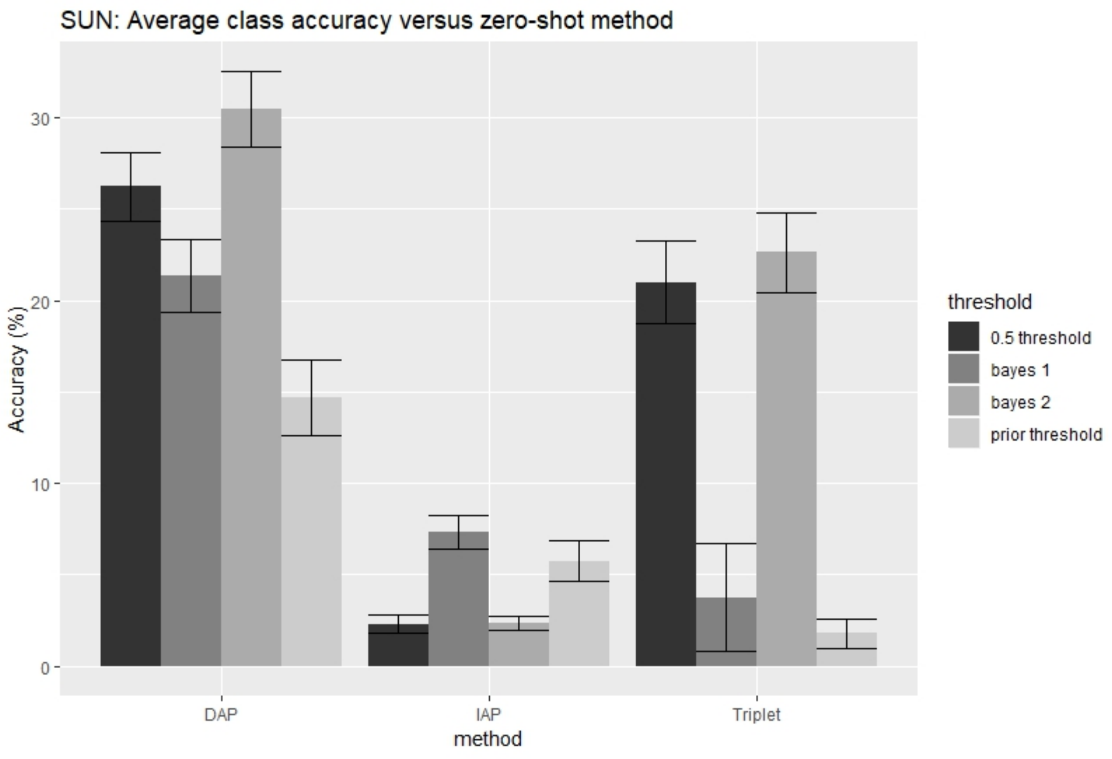


Figure 4.6: Zero-shot class recognition accuracy on the SUN dataset for the three different attribute classification methods. The results for the two threshold methods and two Bayesian methods are also shown.

4.3 Explainability

4.3.1 Attention heatmaps

First, the 2 attention heatmaps are shown for the AWA2 dataset. The original image is shown at the top, while the Grad-CAM heatmaps for three different predicted attributes are shown below it. In Figure 4.8, a sample from the Cat class is shown. As can be observed, the heatmap is distributed over the entire body of the cat, with the highest values near the face region, regardless of the attribute. This would make sense for the 'Meatteeth' attribute, but the attention map for the attribute 'tail' is also focused on the head and not on the actual visible tail of the cat. This seems to imply that the network has learned to detect these attributes mostly based on the

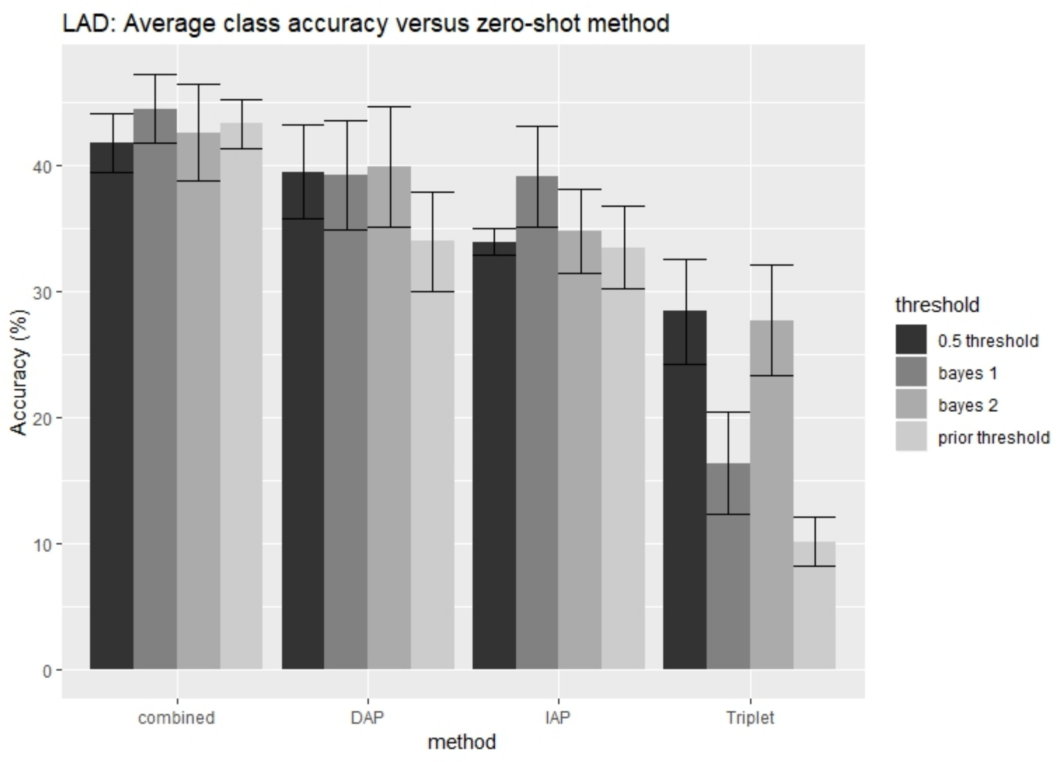


Figure 4.7: Zero-shot class recognition accuracy on the LAD dataset for the three different attribute classification methods. The results for the two threshold methods and two Bayesian methods are also shown.

face of the animal and slightly based on the body and/or fur. A similar pattern can be seen in Figure 4.9. The attention maps are mostly concentrated around the face and body/fur of the animal, with some slight variations between the three maps. Given that the images from the AWA2 dataset are labelled per class and not per image, the unexpected attention of the network could be explained intuitively by speculating about what the network learns in this setting. When images are labelled per class, it can often be the case that an image is annotated with specific attributes that are not visible in that specific instance, as discussed in section 2.3.3. This may weaken the correspondence between visual evidence of an attribute (e.g. a tail) and the annotation (e.g. 'tail present'), as many images without the attribute visible will contain the corresponding attribute label. However, the network will still try to learn to predict the attribute correctly to reduce the loss and has to resort to learning

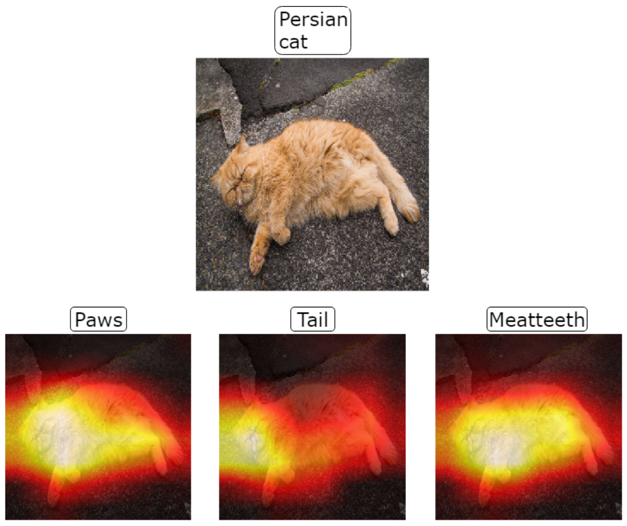


Figure 4.8: Attention heat maps of the ’Persian Cat’ class from the AWA2 dataset. The heatmaps show that the attention of the network is not what one would expect given these three attributes.

different features that are not visibly correlated with the attribute, but still co-occur a lot together with the corresponding label. This is possible, as section 4.1 shows that the attributes on AWA2 and LAD can achieve a very high accuracy and f1-score. This theory explains why the attention of the network visualized by the Grad-CAM method is focused on the face and body/fur of the animal, as these features are often in most images and are probably the most visually discriminative features for the classes.

Secondly, the attention maps for the SUN dataset are discussed. Figure 4.10 contains three attention heatmaps for the ’Cabin’ class. Compared to the AWA2 heatmaps, the ’Cabin’ maps already show a significantly larger variety between the three. The ’Grass’ attribute map shows a concentration of attention to the left of the cabin, around the patch of grass. This map seems to imply that the network actually uses grass-like features for its ’Grass’ attribute prediction. However, it is not clear why the grass in front of the cabin does not receive any attention. One possibility is that the patch to the left is more recognizable as grass due to the lighting and absence of shadow, unlike the other patch. Furthermore, there is some slight attention concentrated on the house. The ’Man-made’ attribute heatmap is mostly concentrated around the cabin itself and mostly around the cabin door. The attention seems to localize ’man-made objects’, which is what one would expect. Most likely, the ’man-

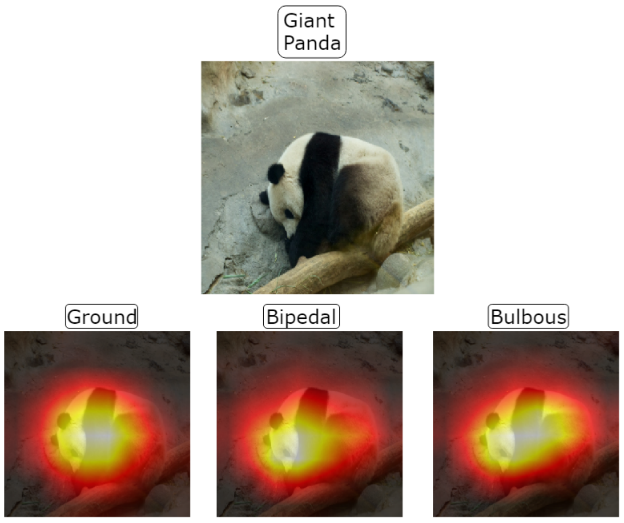


Figure 4.9: 3 attention heatmaps of the 'Giant Panda' class from the AWA2 dataset. Again, the attention of the network does not seem to localize the specific attributes in the image.

made' attribute was annotated for objects such as tents, huts and cabin-like objects in the SUN dataset. The network is thus able to weakly localize such objects with only attribute annotations. The 'Trees' heatmap seems to be mostly concentrated on the large tree to the right of the cabin and the trees in the distance towards the left. However, some attention is also distributed around the top of the house and barely any attention is given to the top of the trees. It seems that the heatmaps are not always exhaustive in that they do not fully cover all the parts of the image that one would expect to be evidence for a specific prediction. A simple explanation for these problems could be that the attribute recognition is just not good enough yet and still has lots of room for improvement, due to the SUN dataset having a relatively low amount of images. This can also be seen in Section 4.1 and 4.2, where the average attribute recognition performance only achieves a maximum f1-score of 0.5. Regardless, even with the current attribute recognition, the attention heatmaps are more discriminative for different attributes than the heatmaps for AWA2 and LAD, but the actual quality of the maps can differ and also depends on how well the specific attributes are recognized. The attention maps shown in section 2.2.3 on the CelebA dataset showed that Grad-CAM maps can be more discriminative. The CelebA dataset contains around 200,000 images, so it's likely that the attribute recognition is better due to almost having twenty times as many images. However,

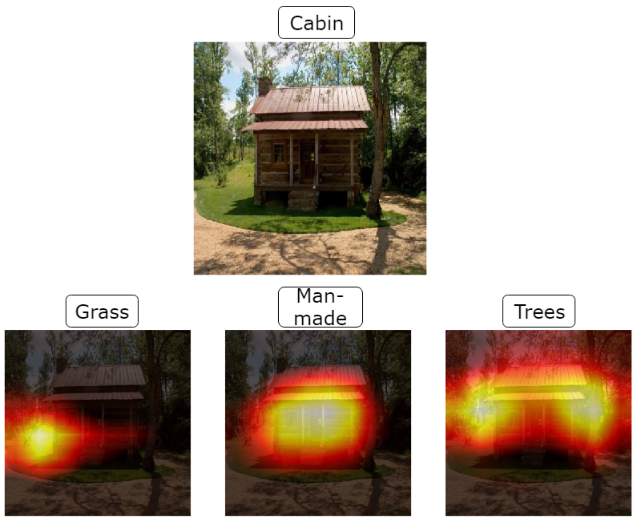


Figure 4.10: Three attribute attention heatmaps on an instance from the 'Cabin' class.

the CelebA dataset arguably contains a smaller visual variety compared to the SUN dataset, mostly showing photos of faces.

From these Grad-CAM maps, it seems that instance-wise labelling results in more intuitive network attention compared to class-wise labelling. Figures 4.11, 4.12, 4.13 and 4.14 show more examples from the other 4 data splits of the SUN dataset. Figures 4.15 and 4.16 show two examples from the LAD dataset.

4.3.2 Rule extraction

In this section, rules are shown for 5 classes of the AWA2 dataset and 5 classes of the LAD dataset. The rules are binary TRUE/FALSE rules and are shown here in formal logic, where $\wedge$ stands for conjunction and $\neg$ indicates negation. The rules for AWA2 are shown in Table 4.7 and the rules for LAD are shown in Table 4.8. These rules can help in understanding which attributes are most discriminative for the classes in the dataset. For example, the 'longneck' attribute is the first attribute that is checked in the AWA2 decision tree, as this attribute can already filter a bunch of classes depending on whether or not it's detected. The same goes for the LAD dataset, where the attribute 'is_for_cleaning' serves to distinguish between classes in the electronics superclass and the classes in the other superclasses.

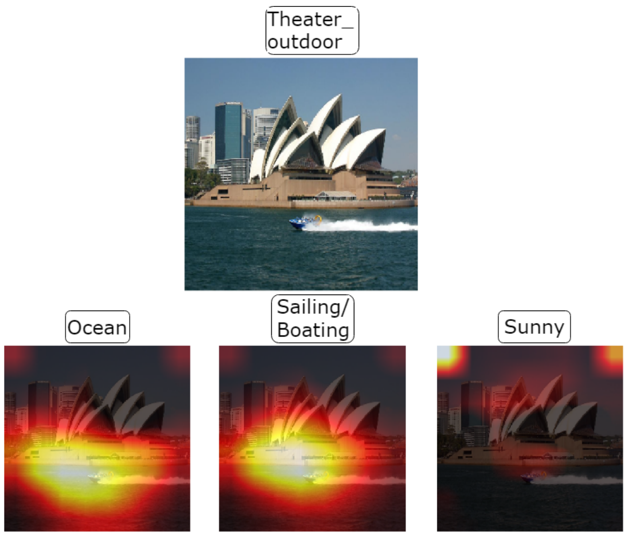


Figure 4.11: Attention heatmaps of an instance of the 'Outdoor Theater' class. The 'Ocean' and 'Sailing/Boating' attribute both focus on the water and the boat and are very similar. The 'Sunny' heatmap is focused mostly on the air, especially the left and right corners, but the reason for this is unknown. Possibly, the attention for some attributes can be attributed to the locations that they often occupy in the images, such that the network learns where to approximately expect these attributes. Images with the attribute 'Sunny' often have the sun or clear blue skies near the top of the image.

class	rules
antelope	$\neg longneck \wedge \neg hops \wedge \neg desert \wedge hooves \wedge big \wedge \neg smelly \wedge \neg spots$
sheep	$\neg longneck \wedge \neg hops \wedge \neg desert \wedge hooves \wedge \neg big \wedge weak \wedge \neg brown$
rhinoceros	$\neg longneck \wedge \neg hops \wedge \neg desert \wedge hooves \wedge big \wedge smelly \wedge \neg active$
otter	$\neg longneck \wedge \neg hops \wedge \neg desert \wedge \neg hooves \wedge \neg tusks \wedge smart \wedge ocean$
horse	$longneck \wedge gray \wedge \neg paws \wedge \neg arctic \wedge \neg spots \wedge buckteeth \wedge \neg horns$

Table 4.7: Example rules extracted from 5 AWA 2 classes. Rules have been shortened to fit on the page.

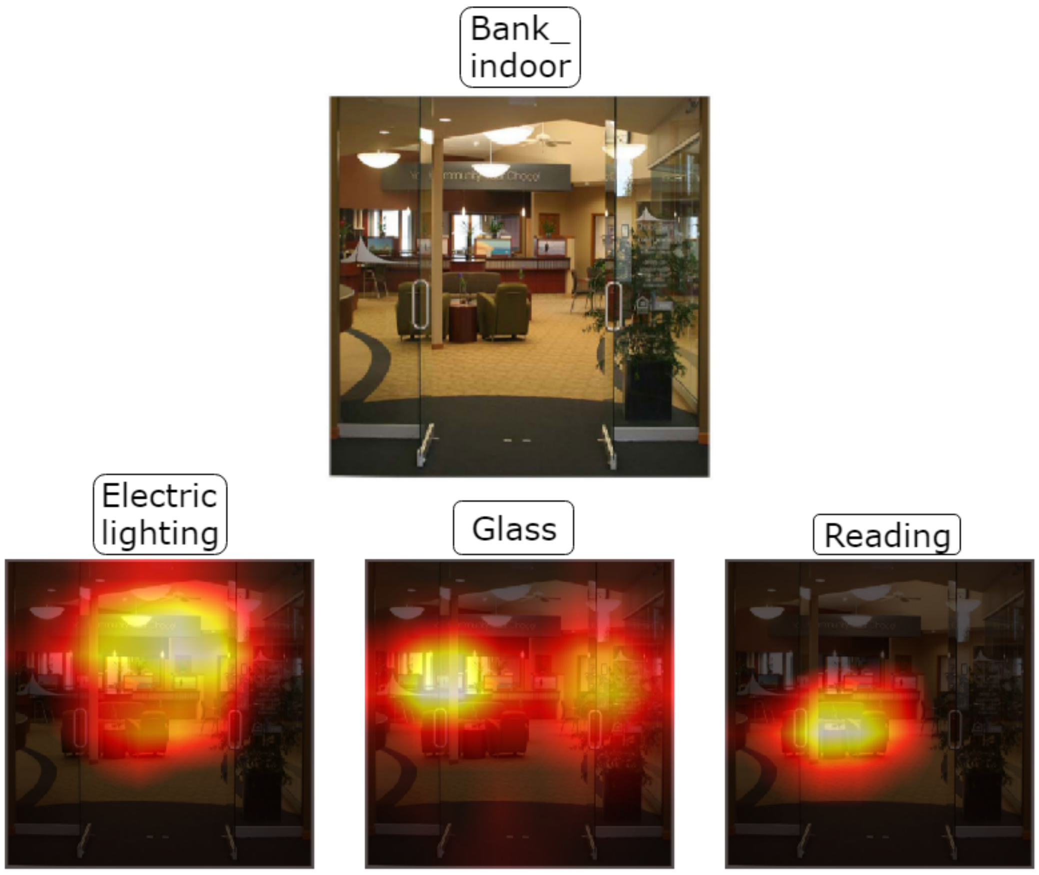


Figure 4.12: Attention heatmaps of an instance of the 'Indoor Bank' class. The attention of the network for the 'Electric lighting' is focused mostly on the light generated by the lamps and the lamps themselves. The 'Glass' attribute attention map is focused mostly on the left and right glass parts of the entrance, but does not fully cover them. The 'Reading' attribute attention map is very concentrated on the small table and chairs in the middle of the image.

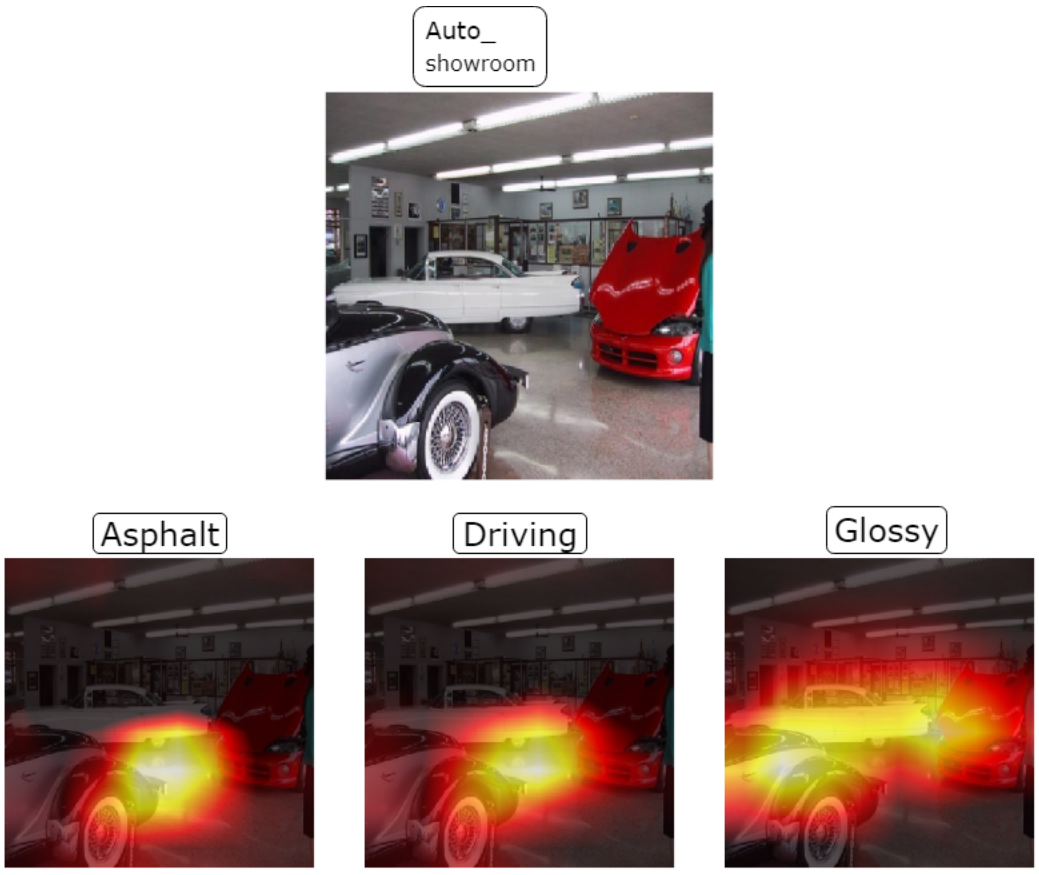


Figure 4.13: Attention heatmaps of an instance of the ’Auto Showroom’ class. The ’Driving’ and ’Asphalt’ attention maps are mostly focused partly on the ground and lower parts of the surrounding cars and indicate that these two attributes are correlated. The ’Glossy’ attention map is mostly concentrated on the glossy surfaces of the cars in the image.

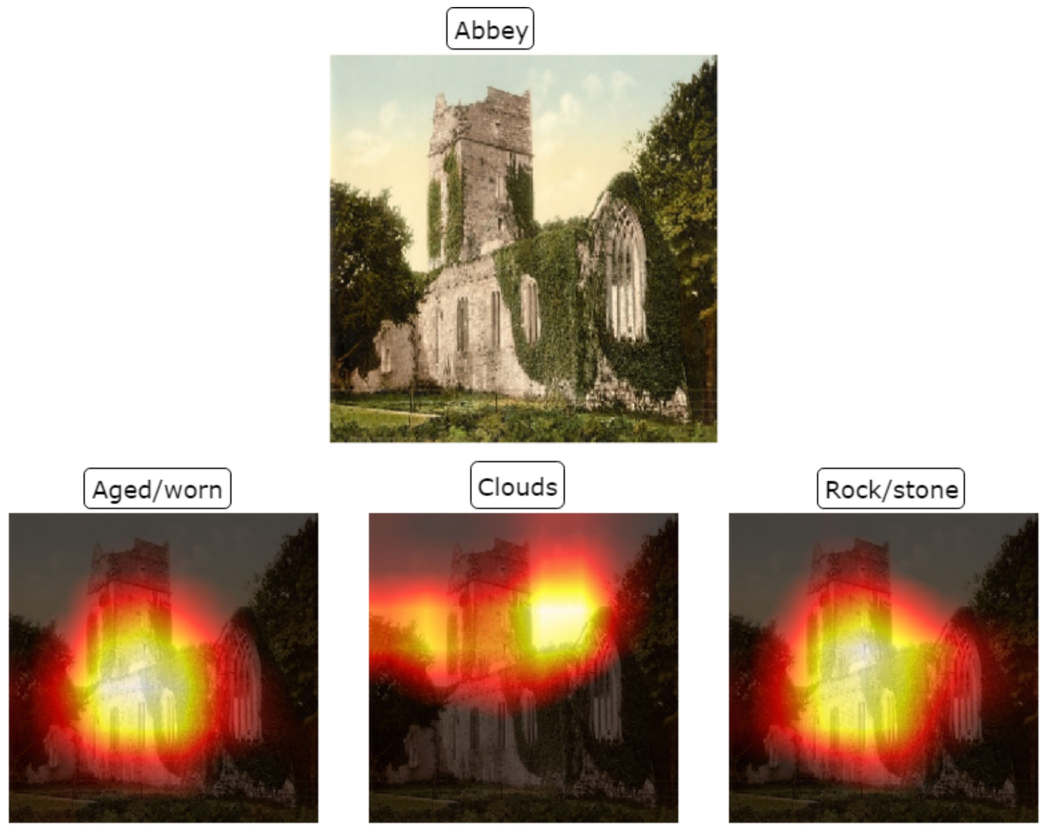


Figure 4.14: Attention heatmaps of an instance of the 'Abbey' class. Both the 'Aged/worn' and 'Rock/stone' are both focused on the abbey itself. The 'Clouds' attention map is concentrated most strongly on the clouds behind the abbey, especially those to the right.

class	rules
jellyfish	$\neg is\_for\_cleaning \land \neg is\_golden \land is\_transparent \land is\_smooth$
washing machine	$is\_for\_cleaning \land \neg is\_for\_office\_use \land \neg can\_refrigerate$
watch	$\neg is\_for\_cleaning \land \neg is\_golden \land \neg is\_transparent \land has\_wristband$
wave hair	$\neg is\_for\_cleaning \land is\_golden \land is\_sexy \land is\_mature \land is\_cute$
table lamp	$\neg is\_for\_cleaning \land \neg is\_golden \land \neg is\_transparent \land \neg has\_wristband$

Table 4.8: Example rules extracted from 5 LAD classes. Rules have been shortened to fit on the page.

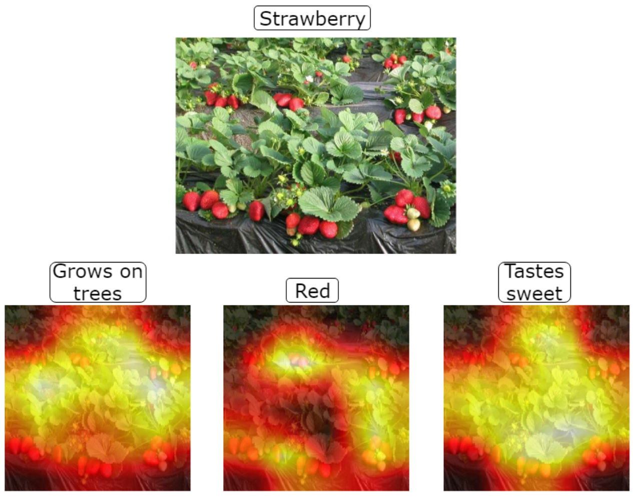


Figure 4.15: Attention heatmaps of an instance of the 'Strawberry' class. The 'Grows on trees' and 'Tastes sweet' attention maps seem active in many areas of the image, but most strongly on the plants. Interestingly, the 'Red' attention map is focused mostly on the strawberries and less on the plants, which actually seems plausible.

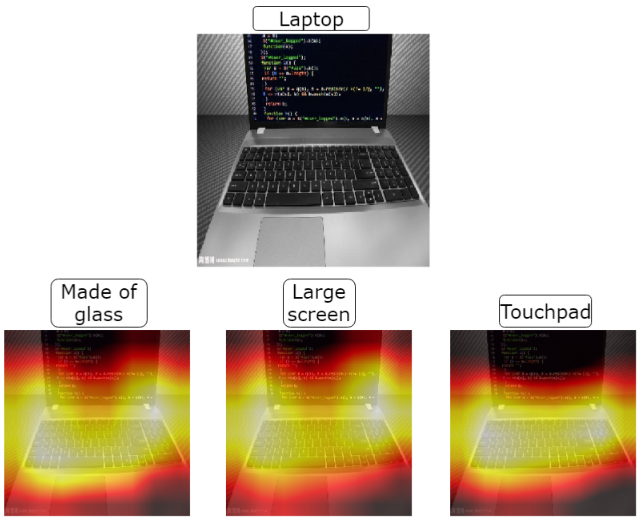


Figure 4.16: Attention heatmaps of an instance of the 'Laptop' class. All three attention maps are focused mostly on the keyboard, instead of on the screen as one would expect for the first two attributes.

Chapter 5

Discussion

5.1 Answers to research questions

How does deep attribute learning affect standard classification?

As seen in the standard classification experiments, directly learning the classes without attributes often outperforms or performs just as well as a deep model that learns attributes. It therefore does not seem worth the effort of labelling a dataset with attributes just to improve the classification performance of a deep learning system. Attributes should be considered when the system needs to generalize beyond the training classes for few-shot recognition, or when one desires more explanability from their deep model.

How well does deep attribute learning allow zero-shot recognition?

The deep attribute learning experiments on AWA2, SUN and LAD resulted in an average zero-shot classification of 58%, 30% and 44%, respectively. Besides the SUN results, the zero-shot capability of attributes seems reasonable compared to other methods, but there is still room for improvement, as the zero-shot recognition performance is a big step down from standard classification. One problem is that the attribute recognition of the deep model takes a nosedive when it comes to unseen classes, which indicates that the attribute classification becomes biased towards the attributes of seen classes, which is an observation that has been made in other research [15]. As a result, the DAP and IAP methods have fallen out of favor in the zero-shot community in favor of other zero-shot methods [15]. Another argument that has been made in the zero-shot literature is that great care must be taken when employing pre-trained deep models, as the dataset that the network has been pre-trained on (Imagnet) might contain classes that also occur in the zero-shot test set, which gives the network information about test classes that it shouldn't have in the zero-shot setting [15]. This seems reasonable, as 6 out of 10 classes (chimpanzee, giant panda, leopard, persian cat, pig, hippopotamus) from the standard test set of

AWA2 occur in Imagenet. While the results of this thesis on this specific data split are among the highest, one other data split has similar results (hamster, ox, gorilla, chihuahua, dalmatian, moose, walrus, weasel, dolphin, German shepherd), where most of them also occur in Imagenet. It therefore seems that the pretraining on unseen classes indeed unfairly increases the results. The remaining 3 data splits used in this thesis are harder to get good results on, compared to the other two. It is possible that the DAP and IAP methods still have potential in a deep learning setting, but the datasets available do not accommodate them due to either their small size or their class-wise labelling. Looking at both the attribute recognition performances of all three datasets in both a standard and zero-shot setting, it seems clear that the attribute recognition on the SUN dataset, which has instance-wise labelling, generalizes better to unseen classes. This can also be seen in the Grad-CAM attention maps, where plausible explanations for predictions can be made for certain attributes, even though the images are unseen by the deep model. The AWA2 and LAD attention maps indicate that these networks focus mostly on the most discriminative features of the class, such as the face and body of an animal, or the keyboard and screen of a laptop. With class-wise labelling, it seems the network learns kernels that detect features related to the classes for attribute recognition. When presented with unseen classes, the network begins to falter and becomes confused because the unseen classes may not resemble the training classes. Previous research on generalized zero-shot recognition (open set) has shown an extremely low zero-shot performance for AWA2, namely 0% on unseen classes when training classes can also occur in the test set [15]. This makes sense, as instances from the training classes align much more with the learned features, which are heavily biased towards the training classes, as discussed before. With an instance-wise labelled dataset like the SUN, it seems that the network learns kernels for the attributes themselves and not necessarily the training classes. When presented with unseen classes, the attribute recognition does not drop because the learned filters depend less on the training classes and can therefore detect the attributes in the unseen instance. It therefore seems like instance-wise labelled class datasets are the way to go to improve both zero-shot recognition and explainability. The downside, however, is that labelling each individual image with attribute annotations is quite expensive and this is probably the reason why no big publicly available instance-wise labelled attribute dataset exists at the time of writing.

How does deep attribute learning improve the explainability and interpretability of deep learning models?

When it comes to scene understanding, learning attributes offers more insight into what the network sees. By modeling the attention of the network, it is possible to extract visual explanations of predictions with methods such as Grad-CAM, even with an average recognition F1-score of 0.47. The clarity and explainability of these maps are, however, strongly dependent on how well the attributes are recognized and if the attributes are labelled per instance instead of per class. The attributes themselves can also have an impact, as certain attributes are harder to visually attribute to the specific parts of the image, such as the 'agile' attribute from the AWA2 dataset. As data volume increases and attribute recognition improves, it seems plausible that the localisation capability of deep network attention also improves. These attention maps can then be used to help users of the network to get a better understanding of why deep networks make certain predictions and even improve datasets after inspecting faults of these predictions. While the attention maps from networks trained on class-wise labelled attribute datasets are not useful for localization, they were still able to intuitively help understand what the network focuses on and how the different labelling methods influence the network, in the end.

Rule extraction can potentially help with attribute selection. By building a decision tree that maps attributes to classes, the most important and least important attributes can be identified. Some of the attributes on the lower parts of the tree could then be pruned, as long as the lexicon entry for each class stays unique. This could also improve a deep model that's trained using the multi-label triplet loss, as fewer attributes lead to fewer attribute embeddings that need to be trained, resulting in fewer parameters and less data needed. Looking at the correlation statistics of the attribute predictions in combination with the decision tree might help in this pruning process.

5.2 Conclusion

Deep attribute learning can offer more explainability of deep models by allowing predictions that are more informative than simple categorical predictions. The explainability is further enhanced by identifying the areas in the image that contribute to specific predictions by visualizing the deep model's attention, to understand why the network makes certain predictions of what it sees in an image. This thesis shows that these attention maps can work but are not perfect, as they do not always exhaustively cover the expected areas of interest and can sometimes cover more areas than they should. However, this is expected to improve as the available datasets also improve. When it comes to standard object classification, adding attributes does not improve the recognition ability of the model. The main reasons for annotating a dataset for attributes is for more explainability and for zero-shot prediction. Better attribute prediction on unseen classes may also help with generalized zero-shot recognition. Zero-shot classification by using deep attribute learning shows promise, but still leaves a lot of room for improvement and is outperformed by other state of the art zero-shot algorithms [15]. With bigger and better datasets, the full potential of deep attribute learning might be realized, leading to better unseen class generalization and better interpretability of network predictions.

Bibliography

[1] Michael Biehl. *Supervised Learning - An Introduction: Lectures given at the 30th Canary Islands Winter School of Astrophysics*, volume 01/2019 of *Machine Learning Reports*. Machine Learning Reports, 4 2019.

[2] Hassan Mohamed, Mohamed Zahran, and Oliver Saavedra. Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: Case study el burullus lake. 2015.

[3] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 815–823, 2015.

[4] I. Nigam, C. Huang, and D. Ramanan. Learning universal embeddings from attributes. December 2018.

[5] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. *CoRR*, abs/1512.04150, 2015.

[6] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[7] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[8] C. H. Lampert, H. Nickisch, and S. Harmeling. Attribute-based classification for zero-shot visual object categorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):453–465, March 2014.

[9] Chris Kawatsu, F. Koss, Andy Gillies, Aaron Zhao, Jacob Crossman, Ben Purman, Dave Stone, and Dawn Dahn. Gesture recognition for robotic control using deep learning. In *Conference: Ground Vehicle Systems Engineering and Technology Symposium*, 2017.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436 EP –, May 2015.

[12] R.B. Altman. Artificial intelligence (AI) systems for interpreting complex medical datasets. *Clinical Pharmacology & Therapeutics*, 101(5):585–586, 2017.

[13] Andreas Holzinger, Chris Biemann, Constantinos S. Pattichis, and Douglas B. Kell. What do we need to build explainable AI systems for the medical domain? *CoRR*, abs/1712.09923, 2017.

[14] H. Hagras. Toward human-understandable, explainable AI. *Computer*, 51(9):28–36, September 2018.

[15] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata. Zero-shot learning - a comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2018.

[16] Irving Biederman. Recognition-by-components: a theory of human image understanding. *Psychological review*, 94 2:115–47, 1987.

[17] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth. Describing objects by their attributes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1778–1785, June 2009.

[18] C. H. Lampert, H. Nickisch, and S. Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 951–958, June 2009.

[19] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, October 1986.

[21] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951.

[22] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, 23(3):462–466, 09 1952.

[23] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks, 2011.

[24] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.

[25] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893, June 2005.

[26] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986.

[27] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[28] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[30] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[31] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. *arXiv e-prints*, page arXiv:1311.2901, Nov 2013.

[32] Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015.

[33] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

[34] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Is object localization for free? - weakly-supervised learning with convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 685–694, June 2015.

[35] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-CAM: Why did you say that? visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[36] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Marc´ Aurelio Ranzato, and Tomas Mikolov. Devise: A deep visual-semantic embedding model. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2121–2129. Curran Associates, Inc., 2013.

[37] Sheng He and Lambert Schomaker. Open set chinese character recognition using multi-typed attributes. *CoRR*, abs/1808.08993, 2018.

[38] Dangwei Li, Zhang Zhang, Xiaotang Chen, Haibin Ling, and Kaiqi Huang. A richly annotated dataset for pedestrian attribute recognition. *CoRR*, abs/1603.07054, 2016.

[39] Genevieve Patterson and James Hays. SUN attribute database: Discovering, annotating, and recognizing scene attributes. In *Proceeding of the 25th Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[40] Genevieve Patterson, Chen Xu, Hang Su, and James Hays. The SUN attribute database: Beyond categories for deeper scene understanding. *International Journal of Computer Vision*, 108(1-2):59–81, 2014.

[41] Bo Zhao, Yanwei Fu, Rui Liang, Jiahong Wu, Yonggang Wang, and Yizhou Wang. A large-scale attribute dataset for zero-shot learning. *CoRR*, abs/1804.04314, 2018.

[42] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas,

Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[43] François Chollet et al. Keras. https://keras.io, 2015.

[44] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[45] Yutian Lin, Liang Zheng, Zhedong Zheng, Yu Wu, Zhilan Hu, Chenggang Yan, and Yi Yang. Improving person re-identification by attribute and identity learning. *Pattern Recognition*, 95:151161, Nov 2019.

[46] D. Li, Z. Zhang, X. Chen, and K. Huang. A richly annotated pedestrian dataset for person retrieval in real surveillance scenarios. *IEEE Transactions on Image Processing*, 28(4):1575–1590, April 2019.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.