Comparing Feature Pooling Methods and Fisher Vector in Convolutional Neural Networks

Mark Post
Leiden University
m.g.s.post@umail.leidenuniv.nl

ABSTRACT

A lot of research has been done in the field of image classification. Image classification id the field of machine learning where the computer determines what the object or objects in the image are. There are a wide variety machine learning methods are designed. They can loosely be divided in 'old'- and 'new'-school methods. There are not many papers comparing the developed methods. The developers of a methods usually compare their method with others, but can we trust that their comparison is unbiased. In this work, we will compare different configurations of the AlexNet convolutional neural network and compare one configuration of the AlexNet with a Fisher Vector neural network. The different configurations of the CNN consist out of three different feature pooling methods, average, maximum and stochastic pooling and four different data augmentation combinations, no, crop, flip and crop and flip augmentation.

We used the Caffe deep learning framework to set up and implement the various network configurations. Both the Fisher Vector and data augmentation are implemented within the Caffe framework. We used twelve different setups for our experiments. We ran them by using the ImageNet LSVRC-2012 dataset consisting out of 1.2 million training and 50,000 validation images of 1,000 different categories. For the experiments of with the Fisher Vector neural network we used a 10,000 and 1,000 (training respectively validation) images large sample of the ILSVRC-2012.

Unfortunately, we could not get the Fisher neural network working and training. Thus, we do not have any results for those experiments. Fortunately, we do have results for the other experiments. There was a clear difference in results with the different pooling methods. Maximum pooling preformed best with an error rate of 21.8% with crop and flip augmentation. There was no clear best augmentation over all the feature pooling methods.

1. INTRODUCTION

In this work, we will compare several configurations of convolutional neural networks as well as comparing a convolutional neural network setup with the Fisher Vector in combination with a neural network.

One of the problems, that systems based on convolutional neural networks and/or Fisher Vector, is image classification. Image classification is a problem in computer vision. It is the task of extracting information from the image and classifying based on that information what the topic of the image is.

The difficult part in image classification is how to classify the image. The computer 'sees' the image just as a matrix of numbers. This matrix has the width and height of the image. The depth of the matrix determines whether the image is grey-value or RGB.

The problem of image classification is a trivial problem in a wide variety of applications. These applications include among others image search in a image database and processing and identifying objects on satellite images. In the first example application, there is a database containing images. The system searches through this database to obtain images with particular visual content. There are satellite images involved in the second example. In this application, the system needs to classify what the objects on the image or subimage are.

Classifying by hand is possible and the human brain is fast, but it is impossible to keep up doing this with humans. The amount of new data is too large for humans. Also, letting a humans search through a database containing millions or maybe even billions of images is too slow. Computers are fast and do not get tired. However, a computer 'sees' the image just as a sequence of bits. It needs some kind of method to classify them.

There is a wide variety machine learning methods to classify the topic of an image. These methods can loosely be divided in 'old'- and 'new'-school methods as well as in supervised and unsupervised learning methods. The old-school methods include Fisher Vector[13] and bag of words (also known as bag of features), whereas the new-school methods include methods based on convolutional neural networks. The convolutional neural networks and support vector machines (used in combination with Fisher vector) are supervised learning methods. Gaussian mixture models (used in earlier stage of Fisher vector) is an unsupervised learning method.

In supervised learning methods, the system knows what the answer should be. The parameters are adjusted based on the error of the estimation compared to the desired result. Unsupervised methods do not know what the desired result is. They 'learn' based on the differences in the input.

In this work, we will compare several configurations of the convolutional neural network as well as comparing the convolutional neural network with a Fisher Vector neural network. In the first part we compare several configurations of feature pooling and data augmentation methods on the AlexNet[6] convolutional neural network. In these experiments the feature pooling methods include maximum, average and stochastic pooling. The data augmentation methods consist of flip augmentation, crop augmentation and a combination of both. In the second part, we compare a configuration of the AlexNet with a Fisher Vector in combination with a neural network. The Caffe[4] deep learning framework is used for the experiments.

Papers in the subject of image classification are usually either from the old-school or new-school. When the writers compare their variant of a method they compare it to other methods. However, those experiments could be biased to the method designed by the writers of the paper. This work does not present a new method, but gives an unbiased comparison between the old-school Fisher vector and new-school feature pooling in combination with convolutional layers. This is done with the well-known dataset ILSVRC 2012.

This work will continue with a section about the related work. Followed by a chapter describing the theoretical side of the used methods. After that a chapter about the experiments and their results. Finally, chapters with the conclusion and discussion.

2. RELATED WORK

In both the 'old'- and 'new'-school areas are a lot of work done and are many papers published. In this section we will highlight a few them, which were used for this work.

The paper written by Krizhevsky et al. [6] introduces the AlexNet convolutional neural network. This is a well-known convolutional neural network. At the time of publishing it was one of the deepest CNNs. The network was designed to work with large datasets such as the ILSVRC-2010 and ILSVRC-2012.

An evaluation of subsampling versus maximum pooling was done in work by Dominik Scherer et al. [14]. They compared both methods in a CNN and concluded that maximum pooling was superior to subsampling. In another paper by Matthew Zeiler et al. [17] is stochastic pooling introduced. They replace the conventional deterministic pooling, such as maximum pooling, with a stochastic procedure. In their experiments, they compared stochastic pooling with maximum and average pooling and showed that their method has the lowest error during test time.

In the paper by Jorge Sánchez et al. [13] is the image classification method Fisher Vector introduced. It was introduced as an alternative for the Bag-of-Visual words approach. Based on the paper by Jorge Sánchez et al. [13], have Florent Perronnin et al. [9] proposed a hybrid architecture combining Fisher Vector with neural network. Their network contains an unsupervised set of layers, where Fisher Vectors are computed, and a supervised set, several fully connected layers. They obtained an accuracy just shy of the one achieved by the AlexNeton the ILSVRC-2012 dataset.

In work by Ken Chatfield et al. [1] is a wide variety of settings for CNNs and Fisher Vector neural networks compared. This similar of what we will do in this work.

A framework for machine learning called Caffe is introduced by Yangqing Jia et al. [4]. It is a C++ library containing bindings for training and deploying general-purpose convolutional neural networks and other deep models.

In our work we will use the AlexNet as base for our experiments as convolutional neural network. In this CNN we will compare different feature pooling methods similar to the work of Matthew Zeiler et al. [17] as well as several data augmentation methods. We will use the Fisher Vector as representative for the 'old'-school methods. Our setup with the Fisher Vector is based on the work of Florent Perronnin et al. [9] although we will use the Caffe framework.

3. METHODS

Before going into details of the compared methods, we will give a brief overview of neural networks and convolutional neural networks. At the end of this chapter we will also describe data augmentation briefly. This method is used in the experiments.

3.1 Neural Networks

A neural network is an artificial neural network inspired by the biological equivalents. It is used to estimate or approximate functions. It is a machine learning method. Thus, it has a learning phase, where all parameters in the model are trained. After it has been trained it can be used on other (similar) input.

The most basic neuron in a neural network is the perceptron. The output is either 1 or 0. Each input x_i has a weight w_i. The output is 1 if $\sum_i w_i x_i$ is greater or equal to some threshold value. If it is smaller, the output is 0. This is shown in Equation 1. The allowed input of a perceptron is either 0 or 1. A visualization of the perceptron is shown in Figure 1.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases} \quad (1)$$

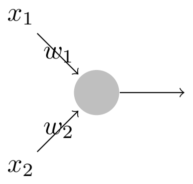


Figure 1: Example perceptron

Another type of neurons is the sigmoid neuron. This type neuron allows real value inputs between 0 and 1. A sigmoid neuron has similar to a perceptron weights on every input and an overall bias. The output value of a sigmoid neuron can be simplified by the equation shown in Equation 2,

$$\sigma(z) \equiv \frac{1}{1+e^{-z}} \equiv \frac{1}{1+\exp(-\sum_j w_j x_j - b)}, \quad (2)$$

where $\sigma(z) = \sigma(w \cdot x + b)$.

An example of a neural network is shown in Figure 2. The left-most layer is called the input layer. The middle layer is called a hidden layer. The hidden middle layer contains the sigmoid (or perceptron) neurons. At the end, the right-most layer, is the output layer. This example has two input, three hidden and one output neuron.

The input layer is a very simple layer. Each neuron in this layer receives an input value and redirects it to its output without manipulating it. The output layer is the output of the neural network. The neurons in this layer either say true or false depending on the criterion they are set on. However, they can also do the final calculation and output the probability that their criterion is true based on the output of the last function. This probability is usually a value between 0 and 1.

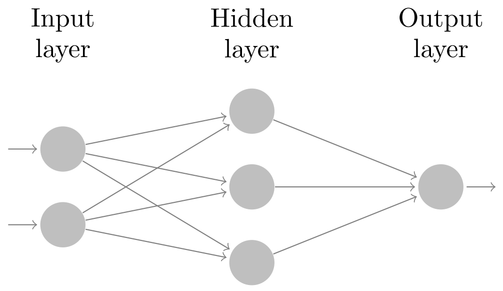


Figure 2: Example of a simple neural network. It has two input, three hidden and one output layer.

The neural network in Figure 2 is called a single-layer network. A neural network can have more than one layer. One type of such networks is called multi-layer perceptrons (MLPs). Both these networks are feed-forward neural networks, which means that there are no loops in the network[8].

3.2 Convolutional Neural Networks

Convolutional neural networks are neural networks with multiple layers. These layers include layers that are not fully connected. One reason for this is too reduce the number of weights the network has. The difficulty with increasing the number of layers is, that the more layers the network has, the more connections and thus parameters the network has. All these parameters need to be trained, which can make training very slow. Convolutional neural networks are often used in image and video recognition.

3.2.1 Layers

The not fully connected layers are called convolutional layers. Some of these layers are followed subsampling or pooling layers. The convolutional layer gets an image or a previous feature map as input. It contains small groups or collections of neurons. These groups of neurons are called (local) perceptive fields. Each perceptive field has a certain size, $K \times K$. Each of the K^2 inputs have a weight. The weights are shared. Which means that the filter, the small group of neurons, has the same weights. So, the filter moves over the input 'image' of the layer and at any point the weights for each input is the same. The output of a convolutional layer are a set or stack of feature maps.

The final layers of a convolutional neural network are usually fully connected layers. These layers are also sometimes called classification layers. The fully connected layers combine the feature maps from the final convolutional or pooling layer into a one-dimensional vector. The final layer of these fully connected layers outputs a vector containing the probability for each of the categories in case of the classification problem[8, 2, 14, 3].

The convolutional layers can make use of Rectified-Linear unit (ReLU). It is an activation function defined as: $f(x) = max(x, 0)$. So, it computes the output as x if $x > 0$. They can also use Local Response Normalization (LRN). It is used to normalize the output of the ReLU. The normalized activity $b^i_{x,y}$ is given by the expression

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}, \quad (3)$$

where $a^i_{x,y}$ is the activity of kernel i at position (x, y) after applying ReLU, N the number of kernels. The sum in the expression runs over n adjacent kernel maps at the same spatial position. The constants k, n, α and β are hyper-parameters and we used $k = 1$, $n = 5$, $\alpha = 10^{-4}$ and $\beta = 0.75$ as described in Krizhevsky et al. [6] (except for the value of k, which is by default 1 in Caffe[1]).

3.2.2 *Feature Pooling*

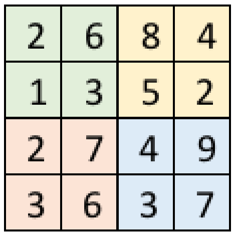
Recent convolutional networks have a pooling layer after some of their convolutional layers. The pooling layers reduce the resolution of feature maps generated by the convolutional layers. Thus, the pooling layers are a aggregation method for a local neighborhood.

There are several pooling methods available for use in convolutional neural networks. The most popular one is maximum pooling. Some of the other options are average (or mean) pooling and stochastic pooling.

The pooling is done by using the $n \times n$ filter of the corresponding pooling method. These filters are applied in combination with a certain stride. The stride is the 'step-size' of the filter. A stride of 0 would mean that the filter stays forever at the same place. Using a stride of 2 causes the filter to do steps of 2 in the y and x direction.

For example, assume we have a feature map of size 4×4, a filter of 2×2 and a stride of 2, then we would have four places where the filter is applied. This is shown in Figure 3. The resulting reduced feature map is reduced by a factor two and has a size of 2×2.

[1]See: http://caffe.berkeleyvision.org/tutorial/layers.html



2	6	8	4
1	3	5	2
2	7	4	9
3	6	3	7

Figure 3: Input feature map for feature pooling

In Equation 4 the output size of a filter is shown. In this equation is W the length of the input image, K the size of the filter, P the padding and S the stride. Where padding is the number of pixels added at each side of the image. If we would apply this to the feature map of Figure 3, then the W is 4, K is 2, P is 0 and S is 2. When we use these number in Equation 4, we end up with $(4 - 2 + 0)/2 + 1$ and a output size of 2.

$$O = \frac{W - K + 2P}{S} + 1 \quad (4)$$

The one of the most popular filters is the maximum pooling filter. The maximum pooling filter chooses the maximum value of the input for each pooling region. An example is shown of this in Figure 4.

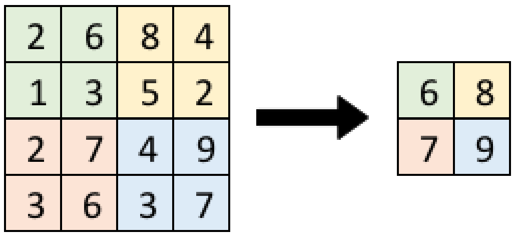


Figure 4: Example of maximum pooling. The maximum value of each region is chosen

In average pooling (also called mean pooling), the average for each pooling region of the input is calculated. Equation 5 shows the formula for this filter and Figure 5 is an example of this.

$$b_{p,q} = \frac{\sum_{i=0}^{K} \sum_{j=0}^{K} a_{i,j}}{K^2} \quad (5)$$

Stochastic pooling is a pooling method that randomly picks the activation within the pooling region in line with a multinomial distribution. This distribution is given by the activities within the current pooling region. There are two variants of stochastic pooling. One of them is used during training, whereas the other during test time.

The first step of the variant used during training time is to compute each probability p_i for each region j in

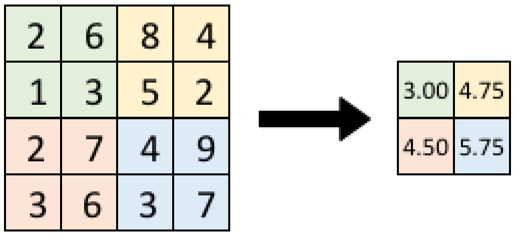


Figure 5: Example of average/mean pooling. The average value for each region of the input is calculated.

the pooling region. This probability is computed using the equation

$$p_j = \frac{a_i}{\sum_{k \in R_j} a_k}. \tag{6}$$

Then a location l from the multinomial distribution based on p is chosen. This pooled activation can be written as

$$s_j = a_l \text{ where } l \sim P(p_1, \ldots, p_{|R_j|}. \tag{7}$$

This version of stochastic pooling introduces noise during test time. So, an alternative is used. This version is a probabilistic form of averaging. The activations are weighted by their probability p_i and summed

$$s_j = \sum_{i \in R_j} p_i a_i. \tag{8}$$

Figure 6 shows an example of the stochastic pooling. It shows in the first image the activations in the pooling region. In the second image, the computed probabilities for each region j within the pooling region (each region is a different color). Finally the last image shows the sampled activation (a) and probabilistic form of averaging (b).

3.3 Fisher Vector

The Fisher vector is an image representation that is designed for high classification accuracy and is sufficiently efficient for large-scale image processing. It is a method based on the Fisher Kernel (FK). The Fisher Kernel is a method combining generative and discriminative methods. A sample's is characterized by its deviation, which is measured by the computed gradient of the sample log-likelihood, from the generative model. The result is a vector representation called the Fisher Vector.

Generating a Fisher Vector happens in several steps. The first step is to generate the samples from the images. We choose to use SIFT feature descriptors for this. Using these feature descriptors, a generative model, a Gaussian Mixture Model (GMM) is generated. In the last step are the Fisher Vectors generated. It uses the SIFT feature vectors and the Gaussian mixture means, covariance's and prior probabilities.

3.3.1 *SIFT and PCA*

As described above, we use SIFT features as samples of the image for the Fisher Vector. Scale-invariant feature transform is an algorithm to detect and describe features in an image. It was first published by David Lowe in 1999[7].

There are four major steps in SIFT. The first step is to detect scale-space extrema. The system searches over all image locations and identifies potential interesting points. The next step is to localize keypoints. A number of locations is selected based on measures of their stability. Each of the selected ones get a detailed model fit. In the third step are one or more orientations assigned to each location. At this step are locations referred as keypoints. The future operations are performed on the image data transformed by these assigned orientations as well as scale and location. The last step is to generate a keypoint descriptor based local image gradients at the selected scale in the region around each keypoint. This keypoint descriptor is usually a 128 dimension vector[7].

The writer of the 2004 paper about SIFT mentions that an image with a typical image size of 500×500 pixels gives about 2,000 features. Since we are using 256×256 pixels images, we can expect to have about 525 features. All these feature points have 128 dimensions. This results in $525 \cdot 128 = 67,200$ values per image. Using this on a large dataset can result in a large memory usage and long computation times.

A solution to this problem is principal component analysis (PCA). By using PCA the dimensionality of SIFT features can be reduced. A method doing this is called PCA-SIFT[15]. We use something similar. We generate first SIFT-features and then reduce their dimensionality by using PCA. PCA-SIFT uses a slightly different approach to generate its feature points. According to the writers of the papers by Chatfield et al.[1], the number of dimensions can be reduced to 80. This reduction results in $525 \times 80 = 42,000$ values per image.

PCA uses an orthogonal transformation to convert the set of SIFT features into a new set of values that has a dimensionality equal or lower than the original set. This new set of variables is called principal components. This set is defined such that the first principal component has the largest possible variance[5].

3.3.2 *Gaussian Mixture Model*

The Gaussian mixture model of a weighted sum K component Gaussian densities. The Gaussian parameters are denoted by $\lambda = \{\mathbf{w}_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, k = 1, \ldots, K\}$, where $\mathbf{w}_k$, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are respectively the mixture weight, the mixture mean and the mixture covariance vector. The Gaussian mixture model is given by the equation

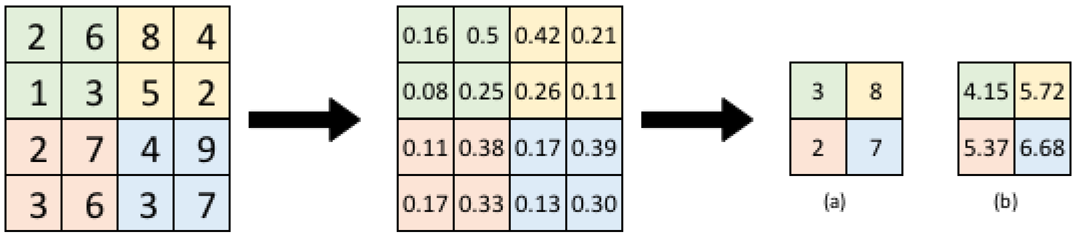


Figure 6: Example of stochastic pooling. The first image is the input feature map. The second image the probabilities of each value within a region. The third part are the resulting 2×2 pooled feature maps, where (a) is the one during training and (b) the one at test time.

$$u_\lambda(\mathbf{x}) = \sum_{k=1}^{K} \mathbf{w}_k u_k(\mathbf{x}), \tag{9}$$

where $\mathbf{x} = \{\mathbf{x}_t, t = 1, \ldots, T\}$ is the D-dimensional sample of feature vectors, $\mathbf{w}_k$, $k = \{1, \ldots, K\}$, are the mixture weights and u_k is the Gaussian k. This Gaussian k is given by the equation

$$u_k(\mathbf{x}) = \frac{1}{(2\pi)^{D/2}|\mathbf{\Sigma}_k|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)'\mathbf{\Sigma}_k^{-1}(x - \boldsymbol{\mu}_k)\right]. \tag{10}$$

This equation has the constrain

$$\forall_k : \mathbf{w}_k \geq 0, \sum_{k=1}^{K} \mathbf{w}_k = 1. \tag{11}$$

The GMM parameters are estimated on a dataset by a Expectation-Maximisation algorithm. This algorithm is used to optimize a Maximum Likelihood (ML) criterion. This means, it wants to find the model parameters that maximize the likelihood of the GMM given the training data. The GMM likelihood can be written as

$$p(X|\lambda) = \prod_{t=1}^{T} p(\mathbf{x}_t|\lambda) \tag{12}$$

where $X = \{\mathbf{x}_t, t = 1, \ldots, T\}$ is the training data.

The EM algorithm is an unsupervised learning method that iterates between the E (expectation) and M (maximization) step. In the E-step the model is updated by using the current estimations of the GMM parameters. This is defined for component k and vector t as

$$q_{tk} = \frac{\mathbf{w}_k u_k(\mathbf{x}_t)}{\sum_{l=1}^{K} \mathbf{w}_l u_l(\mathbf{x}_t)}, \tag{13}$$

where $u_k(\mathbf{x}_t)$ and $u_l(\mathbf{x}_t)$ are the Equation 9. The GMM parameters are in the M-step re-estimated. It uses the model generated in the M-step. The re-estimation is done with the equations for the weights

$$\mathbf{w}_k = \frac{\sum_{t=1}^{T} q_{tk}}{\sum_{t=1}^{T}\sum_{l=1}^{K} q_{tl}}, \tag{14}$$

means

$$\boldsymbol{\mu}_k = \frac{\sum_{t=1}^{T} q_{tk}\mathbf{x}_t}{\sum_{t=1}^{T} q_{tk}} \tag{15}$$

and covariances

$$\mathbf{\Sigma}_k = \frac{\sum_{t=1}^{T} q_{tk}(\mathbf{x}_t - \boldsymbol{\mu}_k)(\mathbf{x}_t - \boldsymbol{\mu}_k)^T}{\sum_{t=1}^{T} q_{tk}} \tag{16}$$

The algorithm can either be initialized with a random initialization of the GMM parameters or it can be initialized by pre-clustering with a k-means algorithm[11].

3.3.3 Fisher Vector

Let $\mathbf{x} = \{\mathbf{x}_t, t = 1 \ldots T\}$ be a sample of T D-dimensional PCA reduced SIFT feature vectors. The Fisher Vector of this sample X is

$$\mathscr{G}_\lambda^{\mathbf{x}} = \sum_{t=1}^{T} L_\lambda \nabla_\lambda \log u_\lambda(\mathbf{x}_t). \tag{17}$$

The Fisher Vector is the sum of normalized gradient statistics for each descriptor. The L_λ is the square-root of the inverse of FIM, Fisher Information Matrix. The FIM is diagonal as showed in Sánchez et al. [13]. The gradients of a single descriptor $\mathbf{x}_k$ with the parameters of the GMM, $\lambda = \{\boldsymbol{\alpha}_k, \boldsymbol{\mu}, \mathbf{\Sigma}_k, k = 1, \ldots, K\}$, are

$$\nabla_{\boldsymbol{\alpha}_k} \log u_\lambda(\mathbf{x}_t) = q_{tk} - \mathbf{w}_k, \tag{18}$$

$$\nabla_{\boldsymbol{\mu}_k} \log u_\lambda(\mathbf{x}_t) = q_{tk}\left(\frac{\mathbf{x}_t - \boldsymbol{\mu}_k}{\boldsymbol{\sigma}_k^2}\right), \tag{19}$$

$$\nabla_{\boldsymbol{\sigma}_k} \log u_\lambda(\mathbf{x}_t) = q_{tk}\left[\frac{(\mathbf{x}_t - \boldsymbol{\mu}_k)^2}{\boldsymbol{\sigma}_k^3} - \frac{1}{\boldsymbol{\sigma}_k}\right], \tag{20}$$

where the weight parameters are re-parametrized to $\boldsymbol{\alpha}_k$ to avoid enforcing the constrain from equation 11 and

q_{tk} is equation 13. Using that the FIM is diagonal, we can coordinate-wise normalize the gradient vectors. This results in the equations

$$\mathscr{G}^{\mathbf{x}}_{\boldsymbol{\alpha}_k} = \frac{1}{\sqrt{\mathbf{w}_k}} \sum_{t=1}^{T} (q_{tk} - \mathbf{w}_k), \tag{21}$$

$$\mathscr{G}^{\mathbf{x}}_{\boldsymbol{\mu}_k} = \frac{1}{\sqrt{\mathbf{w}_k}} \sum_{t=1}^{T} q_{tk} \left(\frac{\mathbf{x}_t - \boldsymbol{\mu}_k}{\boldsymbol{\sigma}_k^2} \right), \tag{22}$$

$$\mathscr{G}^{\mathbf{x}}_{\boldsymbol{\sigma}_k} = \frac{1}{\sqrt{\mathbf{w}_k}} \sum_{t=1}^{T} q_{tk} \left[\frac{(\mathbf{x}_t - \boldsymbol{\mu}_k)^2}{\boldsymbol{\sigma}_k^3} - \frac{1}{\boldsymbol{\sigma}_k} \right]. \tag{23}$$

The equations 21, 22 and 23 are the components of the Fisher Vector. $\mathscr{G}^{\mathbf{x}}_{\boldsymbol{\alpha}_k}$ is a scalar, whereas $\mathscr{G}^{\mathbf{x}}_{\boldsymbol{\mu}_k}$ and $\mathscr{G}^{\mathbf{x}}_{\boldsymbol{\sigma}_k}$ are D-dimensional vectors. The final Fisher Vector is obtained by concatenating $\mathscr{G}^{\mathbf{x}}_{\boldsymbol{\alpha}_k}$, $\mathscr{G}^{\mathbf{x}}_{\boldsymbol{\mu}_k}$ and $\mathscr{G}^{\mathbf{x}}_{\boldsymbol{\sigma}_k}$ for $k = 1, \ldots, K$. It has a dimensionality of $E = (2D + 1)K$[13, 10].

In our experiments, we use an improved Fisher Vector as proposed by Perronnin et al. [10]. We use both the power and ℓ_2 normalization. The power normalization is based on the idea that the Fisher Vector becomes sparser when the number of Gaussians increases. The power normalization "unsparsifies" the Fisher Vector. The ℓ_2 normalization is based on the idea this image contains a background image-independent and an image-dependent part. During the normalization is the image-independent information approximately discarded from the Fisher vector signature.

3.4 Data augmentation

In the Krizhevsky et al.[6] and Chatfield et al.[1] papers are methods described to reduce overfitting. These methods are designed to artificially enlarge the used dataset. The methods described are taking random cropped patches and flipping. The first method works by taking a random cropped patch from the original image. The original image has a size of 256 × 256 pixels. During training time is a random patch of 224 × 224 pixels extracted. Taking a 224 × 224 sized patch means there are $32 \cdot 32$ available patches and thus increasing the dataset size by a factor 1,024. During test time is always the center patch used. The second method is simple. It flips the image on its horizontal axis. So, the top becomes the bottom and vice versa. This increases the dataset by a factor two. Combining the two methods enlarges the dataset by a factor of 2,048.

4. EXPERIMENTS

In the experiments, we compare in the first part different convolutional neural network configurations and in the second part a convolutional neural network configuration with a Fisher Vector based neural network. For the experiments, we use the Caffe framework as well as some other libraries such as OpenCV, VLFeat, Boost and Eigen.

4.1 Caffe Framework

For the experiments, we use the Caffe framework[4]. Caffe is a deep learning framework originally developed by UC Berkeley. The framework is written in C++. A model of a network does not need to be programmed, but is defined in a separate text file.

We use the example definitions for the AlexNet. The most of the code was already in Caffe included. However, we had modified Caffe to some extent for the use data augmentation and Fisher vector.

4.2 Dataset

In the experiments we the ILSVRC-2012 (ImageNet)[12] dataset. The ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) 2012 dataset is rather large. It is a subset of the ImageNet database, which contains 15 million images. The images are of various sizes and are divided in 1,000 categories. The trainings set contains about 1,000 images of each category, which results in about 1.28 million images. The validation set and test set have respectively 50,000 and 150,000 images with the same 1,000 categories.

Due to computation speed in the experiments with the Fisher Vector, we used a sample for those experiments. The sample is from the ILSVRC 2012 dataset and contains 10,000 images for the trainings set and 1,000 for validation set. The sample is obtained by shuffling (using the shuffle function from the C++ Standard Library) the data set and extracting 10,000 and 1,000 respectively from the trainings and validation set. The categories in the dataset are numbered from 0 to 999. The samples only contain images from the first 100 categories.

4.3 Experimental Setup

We will be doing two main experiments. In the first setup, we will compare different configurations of the convolutional neural network with feature pooling and data augmentation. In the second setup, we will compare one CNN configuration with Fisher Vector in combination with a neural network. Each configuration in both experiments is configured in a model. These models are described in section 4.4.

4.3.1 Pooling and Augmentation Experiments

In the first experiment, we compare different configurations of feature pooling and data augmentation. We use the AlexNet by Krizhevsky et al.[6] as a base for our experiments. As of feature pooling methods, we will use the three methods described earlier: average, maximum and stochastic pooling. We will use them in combination with four different data augmentation methods. The first method is using no augmentation. Then we have flip and crop augmentation and finally a combination of flip and crop augmentation. The dataset will be artificially enlarged with these setups by respectively zero, two, 1,024 and 2,048 times.

In these experiments, we will use the full ILSVRC-2012 dataset. However, the images will be resized to 256×256 pixels. In all configuration will the mean image of the dataset be subtracted from the image. Each iterations have a mini batch of 256 images. Table 1 gives an overview of the settings used for training with the Caffe framework.

Setting	Value	Description
Base learning rate	0.01	Learning rate at start
Learning policy	step	Decrease learning rate in steps
Gamma	0.1	Drop learning rate by factor 10
Step size	100,000	Drop learning rate each 100,000 iterations
Max. iterations	450,000	Stop learning after 450,000 iterations
Momentum	0.9	
Weight decay	0.0005	Rate that the weight at nodes decay

Table 1: Overview settings used in Caffe for training in the pooling and augmentation experiments

4.3.2 Fisher Vector Experiments

In the second experiment, we compare the convolutional neural network with the Fisher Vector in combination with a fully connected neural network. For the CNN we use one of the setups from the other experiments: the AlexNet network with maximum pooling and no data augmentation. For the Fisher Vector setup, we implemented the Fisher Vector in the Caffe framework using the VLFeat[16] library. Our code is based on code from Martin Hjelm on Github[2]. The Fisher Vector is computed in the data (input) layer.

During training of the network, we use a pre-trained Gaussian Mixture model. This model is trained on the trainings dataset with an C++ implementation using the VLFeat[16] library. Our code for this was also based on the C++ implementation of Fisher Vector by Martin Hjelm.

For these experiments, we use a sample of the ILSVRC-2012 dataset. The sample of the training set has 10,000 images of 100 categories and the validation set has 1,000 images of the same 100 categories. Each iteration has a mini batch with 16 images. We use this size, because of the computation time of the Fisher Vectors and the smaller dataset. During training we use the same settings as in the other experiments as seen in Table 1.

4.4 Models

Each configuration or network in the experiments has a model. The model is how the network is specified in Caffe. The models can be divided in two main groups. The first group contains models that are convolutional neural networks and based on the AlexNet network. The other group contains one model and is a fully connected neural network in combination with the Fisher Vector.

4.4.1 Pooling and Augmentation Experiments

The models used in the feature pooling and data augmentation experiments are more or less all the same. There are only some small differences in the settings of some of the individual layers.

The models are based on the AlexNet network proposed by AlexNet by Krizhevsky et al. [6] and has 12 layers. These layers consist of one data layer, five convolutional, three pooling and three fully connected layers. Table 2 gives an overview of the network.

Layer	Contains	Size output
Data	Data augment.	3, 224, 224
Convolutional 1	ReLU, LRN	96, 54, 54
Pooling 1		96, 27, 27
Convolutional 2	ReLU, LRN	256, 27, 27
Pooling 2		256, 13, 13
Convolutional 3	ReLU	384, 13, 13
Convolutional 4	ReLU	384, 13, 13
Convolutional 5	ReLU	384, 13, 13
Pooling 5		384, 6, 6
Fully connected 6	ReLU, dropout	4,096
Fully connected 7	ReLU, dropout	4,096
Fully connected 8		1,000

Table 2: Overview of the network used in the pooling and augmentation experiments. First column is the name and type of the layer, second column is other things that layer contains (ReLU is Rectified-Linear unit and LRN is Local Response Normalization) and the third column is the size of the output of the layer (convolutional and pooling layers: number of, height and width of feature maps).

The first layer in the network is a data layer. This is the input of the network. It loads the images into memory and applies the data augmentation depending on the configuration. The necessary data augmentation is done based on a probability for each method. When crop augmentation is applied, then the height and width offset randomly chosen. In our experiments, we crop the images to 224 × 224 pixels, so the crop augmenta-

[2]See: https://github.com/MartinHjelm/fishercaffe

tion chooses one of the 1,024 possible offset combinations. When crop augmentation is not applied, then a centered crop of the image is used. With flip augmentation, the order how the image is copied to the output changes. When it is applied, then we start copying the bottom row of pixels first and then work our way to the top row. When it is not applied, we work from the top to the bottom. Flip augmentation has a probability of 0.5 to be applied.

After the data layer follow two convolutional layer with each a feature pooling layer. These are followed by two convolutional layers without pooling and finally a convolutional layer with feature pooling. Table 3 gives an overview of the configuration of the convolutional and pooling layers. The kernel size is the size of the kernel, the stride is the intervals at which to apply the kernel or filter to the input, padding is number of pixels to add to each side of the input and group is the number of groups the input of output the number of feature maps are separated. The i^{th} output group channels will be only connected to the i^{th} input group. All the convolutional layers and all but one of the fully connected layers use Rectified-Linear unit. The first two convolutional layers have local response normalization.

Layer	**Configuration**
Convolutional 1	Kernel size: 11 - Stride 4
Pooling 1	Kernel size 3 - Stride 2
Convolutional 2	Kernel size 5 - Group 2 - Padding 2
Pooling 2	Kernel size 3 - Stride 2
Convolutional 3	Kernel size 3 - Padding 1
Convolutional 4	Kernel size 3 - Group 2 - Padding 1
Convolutional 5	Kernel size 3 - Group 2 - Padding 1
Pooling 5	Kernel size 3 - Stride 2

Table 3: Configuration of the convolutional and pooling layers

At the end of the network are three fully connected layers. Two of them use dropout. Dropout is a technique to reduce overfitting. We use the strategy proposed in the paper by Krizhevsky et al. [6]. It sets the output of each neuron to zero with a probability of 0.5. By doing this the amount of data is halved and so also the computation time of following layers reduced. During test time no data is dropped, but the values of all neurons is multiplied by 0.5.
The last fully connected layer has 1,000 outputs. This is the number of categories in the dataset and in used to determine the accuracy of the network.

4.4.2 *Fisher Vector Experiments*

As described in the experimental setup, we have two setups for the Fisher Vector experiments. This means that we have two models. The first model for the convolutional neural network is basically the same as the models described above. See Section 4.4.1 for more details. The only difference during these experiments is that the input image size is different. This results in different output sizes of the layers. An overview of the new output sizes is shown in Table 4.

Layer	**Size output**
Data	3, 128, 128
Convolutional 1	96, 30, 30
Pooling 1	96, 15, 15
Convolutional 2	256, 15, 15
Pooling 2	256, 7, 7
Convolutional 3	384, 7, 7
Convolutional 4	384, 7, 7
Convolutional 5	384, 7, 7
Pooling 5	384, 3, 3
Fully connected 6	4,096
Fully connected 7	4,096
Fully connected 8	1000

Table 4: Overview of the CNN network used in the Fisher Vector experiments. First column is the name and type of the layer and the second column is the size of the output of the layer (convolutional and pooling layers: number of, height and width of feature maps).

The second model is the model for the Fisher Vector network. This network has four layers. All but the first layer is fully connected layers. The Fisher Vector is calculated in the data transformer of the data layer. This is also where data augmentation is done in the pooling and augmentation experiments. An overview of the network is shown in Table 5

Layer	**Contains**	**Size output**
Data	Fisher Vector	1, 1, 40,960
Fully connected 1	ReLU, dropout	4,096
Fully connected 2	ReLU, dropout	4,096
Fully connected 3		100

Table 5: Overview of the network used in the pooling and augmentation experiments. First column is the name and type of the layer, second column is other things that layer contains and the third column is the size of the output of the layer.

The first layer in the network is the data layer. In this layer are the images loaded into memory. Also, the Fisher Vector of the image computed is in this layer. This is done as described in Section 3.3. The output is a Fisher Vector for each of the images in the mini batch.

The data layer is followed by three fully connected layers. The first of these layers have ReLU and dropout. The last layer has 100 outputs. Each of these outputs represent one of the possible categories.

4.5 Accuracy

In all of the experiments we work with the image classification problem that we introduced in the introduction. In this problem, we want to classify the topic or category of the image. So, the network gets an image as input and returns a vector or probabilities of each possible category as output. In our experiments this vector has a size of 1,000 and 100 for respectively the pooling and augmentation experiments and Fisher Vector experiments.

Based on these probabilities we can compute the accuracy of the network. A widely-used method are the top-1 and top-5 error rates. We will also use these measures. The top-1 error rate is the number of incorrectly classified images divided by the total number of classified images. The top-5 error rate is the number of incorrectly classified images, where the category of the image was not part of the five categories with the highest probabilities, divided by the total number of classified images.

The accuracy of the networks is measured during on the validation set. When the training is finished, the snapshots of the model weights at different number of training iterations are tested on the validation. Caffe has built-in functionality to do this. The network is run for 1,000 iterations of the validation set. Each iteration has a mini batch of 256 images and at the end returns a top-1 and top-5 accuracy. At the end the final accuracy is computed by taking the mean of respectively all top-1 and top-5 accuracies. The error rates can be computed as 100 minus the accuracy.

5. RESULTS

In this section are the results of the experiments described in the last section. The results are averages over 1,000 iterations an are computed over the validation set of the datasets described in Section 4.2.

5.1 Pooling and Augmentation Experiments

The results of the pooling and augmentation experiments are summarized in Table 6. These results are the error rates on the validation set of the ILSVRC-2012 dataset and are averaged over 1,000 iterations of 256 images. The table shows for each configuration the top-1 and top-5 best (lowest) error rate. Between parenthesis is the iteration shown at which this error rate is achieved. In this table are average, maximum and stochastic pooling shortened as respectively AP, MP and SP. The data augmentation methods are showed as DA_0, DA_C DA_F and DA_CF. This means respectively for no data augmentation, crop augmentation, flip augmentation and the combination of crop and flip augmentation.

Configuration	**Top-1**	**Top-5**
AP DA_0	46.9594 (450k)	23.1551 (450k)
AP DA_C	47.1793 (450k)	23.3125 (450k)
AP DA_F	46.1410 (400k)	23.1031 (300k)
AP DA_CF	46.7117 (450k)	23.1844 (450k)
MP DA_0	46.2633 (450k)	22.5355 (450k)
MP DA_C	46.1891 (400k)	22.5121 (450k)
MP DA_F	46.0070 (400k)	22.1965 (450k)
MP DA_CF	45.7270 (400k)	21.8363 (450k)
SP DA_0	48.2676 (450k)	24.0590 (450k)
SP DA_C	48.1926 (450k)	24.0945 (450k)
SP DA_F	48.5410 (450k)	24.3672 (450k)
SP DA_CF	48.0168 (450k)	24.0461 (400k)

Table 6: Overview of the best error rates (in %) of the different configurations. Each configuration has a top-1 and top-5 error rate. Between parenthesis is the iteration shown at which this error rate is achieved.

In the results, we can see that all top-5 error rates are always lower than the top-1 error rates. We can also see that the most best error rates are at 450 thousand iterations. However, some lowest error rates are at 400 or even 300 thousand iterations. The configurations that have an error rate not at 450 thousand iterations have that mostly for the top-1. Exceptions are AP DA_F and SP DA_CF.

When we look at the data augmentation methods per feature pooling methods, then we can see that the differences between the data augmentation methods are minor. The difference between the largest and smallest error rate are 1.0383%, 0.5363% and 0.5242% for top-1 and 0.2094%, 0.6992% and 0.3211% for respectively average, maximum and stochastic pooling. In average pooling the configuration with flip augmentation performs best in the top-1 as well the top-5 error rates. The worst here is the configuration with crop augmentation. In both maximum and stochastic pooling does CF augmentation perform the best. In maximum pooling no augmentation and in stochastic pooling flip augmentation perform the worst.

When we look at the different feature pooling methods we can see that one pooling methods performs better than another. Stochastic pooling performs the worst in the experiments and maximum pooling the best. The difference between the best performing configuration for stochastic and average pooling 1.3051% for top-1 and 0.9430% for top-5. For average and maximum pooling, it is 0.9847% for top-1 and 1.2981% for top-5.

5.2 Fisher Vector Experiments

Unfortunately, we do not have any results for the Fisher Vector experiments. Despite many attempts and different approaches, we were not able to get the Fisher Vector neural network working and learning. Because of that, we also did not run that CNN configuration in this experiments. More details can be found in Section 7, the discussion.

6. CONCLUSION

Some papers like to show their lowest error rate based on one iteration. They do a number of iterations on the test or validation set and show to lowest error rate achieved. This is however not an accurate measure of the performance of the method or network. Our results as stated earlier are an average over 1000 iterations on the validation set. This is a more accurate measure than taking just one iteration.

As described in the introduction, in Section 1, we compared several configurations of the AlexNet convolutional neural network and we compared one of these configurations with the Fisher Vector in combination with a neural network. In the first part of the experiments we compared feature pooling methods maximum, average and stochastic pooling as well as data augmentation as crop and flip augmentation. So, which of the configurations we used performs best? As in which has the lowest top-1 and top-5 error rate. And how performs the Fisher Vector in combination with a neural network compared to a convolutional neural network setup?

Unfortunately, we cannot answer the question. We do not have any results for the Fisher Vector neural network since we could not get it working. More about this is explained in the discussion in Section 7.

In the results in the last section, Section 5, we can see that all results are quite close together. There are some conclusions we can and cannot make. The data augmentation methods are not decisive in which configuration works best. The configurations without augmentation were never the best, but also never the worst. Both the crop and flip augmentation were once the worst performing augmentation, but in the flip augmentation performed best with the average augmentation. We can conclude that the combination crop and flip augmentation performs better than crop augmentation alone. We can also conclude that maximum pooling performs best from all three feature pooling methods. The maximum pooling with crop and flip augmentation has an error rate of over 2% lower than all stochastic pooling methods configurations. All average pooling methods have a lower error rate stochastic pooling. So, from these pooling methods stochastic pooling performs the worst under these conditions.

To summarise this, we do not have any conclusions on the Fisher Vector neural network compared to a convolutional neural network. In the other experiments maximum pooling performs better then all configurations of average and stochastic pooling. The combination of crop and flip augmentation performs in all experiments better than just crop augmentation.

As of future work, more types of data augmentation could be compared. For example changing the brightnesses of the different channels of the image or using more transformations on the image such as stretching. Also, the effect of even longer learning could be researched. How would the comparison be if we had done one million or more iterations instead of the 450 thousand now? In this work, we only used the AlexNet CNN. It would be interesting to compare different convolutional neural networks and compare who performs best in which situation.

7. DISCUSSION

As stated earlier, we do not have any results for the Fisher Vector experiments. In the project for this work where many setbacks and challenges. This section will give an overview of some of them, some possible reasons why the Fisher Vector experiments did not work and ends with some future work how it possibly could work.

At first the pooling and data augmentation experiments. For the data augmentation in these experiments, we first tried to implement them in a custom layer in Caffe. However, for some reason the network did not learn and was stuck on its initial accuracy. We solved this by using an implementation where the data augmentation was done in the data transformer of Caffe. This data transformer is used in the data layer of the network.

Our implementation of the Fisher Vector ended up not working at all. We tried several ways of implementing it as well as several experimental setups. The Fisher Vector generation consists of two parts: the generation of the GMM and the calculation of the actual FV based on that GMM. At first we created a custom layer in Caffe that calculates the FV with the GMM read from an external file. The GMM was generated in an external piece of program, which did work. This approach did not work and was very slow. So, in the second attempt we build the generation of the GMM into the FV layer. However, this is not what we want. In this approach a GMM is created for each mini batch. Also during test time, which means that we train the GMM for a mini batch. We do not want to train at test time.So, we choose to use generate the GMM separate from the Caffe framework.

We tried a few more things with the Fisher Vector layer, but could not get it to work. Our final attempt was to incorporate the Fisher Vector calculation into the data transformer. We choose to try this, because the data augmentation in the data transformer did work.

For the Fisher Vector we also tried several experimental setups. The generation of the GMM as well as the Fisher Vector calculation was slow. At first we used the full ILSVRC-2012 dataset. Due to memory issues and calculation time, we could not generate a GMM of the full dataset. We solved this by generating the GMM of a small sample of the dataset, but the experiments did not work. Also, the sample was with 1,280 images too small to represent the 1.28 million image dataset. Next we resized the full ILSVRC-2012 dataset to 128×128 pixels images.

We could now generate a GMM of a larger sample, but not the full dataset. In our last setup, we generated a sample as described in Section 4.2 of 10,000 and 1,000 images for respectively the training and validation set. The size of this sample is small enough that we could generate a GMM for the full training set. Also the network would require less training, because of the smaller dataset and fewer number of categories. However, also this last attempt did not work.

There are several reasons possible why the first attempt of the data augmentation and Fisher Vector implementations did not work. First of all, in the custom Caffe layers we only implemented the forward step and not the backward step. Having not implemented the backward step could be the reason why the layers did not work. The backward steps are used to calculate the gradient and eventually the loss of the network. In our case the gradient and loss might not have been correct and causing the network not to learn. Another possible reason is that there might be an error in the code we wrote. Although we used code from another source and a library for the GMM and FV, we might made an error. A possibility is also that something in Caffe does not work together well with our code. However, this is hard to determine.

In the final Fisher Vector experiments, could also the dataset be the cause of it not learning. We used 100 categories with each 100 images. The sample might be too small to make the network learning with this number of categories.

The papers by Perronnin et al.[9] and Chatfield et al.[1] had a similar setup with the Fisher Vector in combination with a neural network as ours. However, both works did not use the Caffe framework. The source code of Chatfield et al. is online available, but it is written in Matlab a so not usable for us.

As of future work our code might work by changing the dataset. We used a sample that might be too small with this number of categories. So, maybe by either reducing the number of categories or increasing the number of images per category, the network combining the Fisher Vector and neural network starts learning.

8. REFERENCES

[1] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*, 2014.

[2] D. C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237, 2011.

[3] A. Giusti, D. C. Cireşan, J. Masci, L. M. Gambardella, and J. Schmidhuber. Fast image scanning with deep max-pooling convolutional neural networks. *arXiv preprint arXiv:1302.1700*, 2013.

[4] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[5] Y. Ke and R. Sukthankar. Pca-sift: A more distinctive representation for local image descriptors. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–506. IEEE, 2004.

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

[7] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[8] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[9] F. Perronnin and D. Larlus. Fisher vectors meet neural networks: A hybrid classification architecture. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3743–3752, 2015.

[10] F. Perronnin, J. Sánchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *European conference on computer vision*, pages 143–156. Springer, 2010.

[11] D. Reynolds. Gaussian mixture models. *Encyclopedia of biometrics*, pages 827–832, 2015.

[12] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[13] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3):222–245, 2013.

[14] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks*, pages 92–101. Springer, 2010.

[15] R. E. G. Valenzuela, W. R. Schwartz, and H. Pedrini. Dimensionality reduction through pca over sift and surf descriptors. In *Cybernetic Intelligent Systems (CIS), 2012 IEEE 11th International Conference on*, pages 58–63. IEEE, 2012.

[16] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[17] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.