UNIVERSITY OF TWENTE

MASTER THESIS

Image Similarity Ranking Applied to the Forensic Domain

Author:
Dylan de Bie

Supervisors:
Dr. Ing. Erik Tews
Dr. Nicola Strisciuglio
Dr. Andreas Peter
Simone Ariëns (NFI)
Mattijs Ugen (NFI)

Faculty of Electrical Engineering, Mathematics and Computer Science

November 24, 2020

Abstract

This research focuses on investigating the possibilities of an efficient scalable image similarity ranking algorithm useful within the forensic domain. It should be able to handle large datasets of images and that is not bound to object categories. This algorithm will be used to rank images within a dataset based on similarity to an object of interest in an ongoing investigation.

Normally classification models are used to classify the images in a dataset to make it easier to go through the large datasets. One could for example be interested in finding evidence of guns on a confiscated device. A downside of using normal classification techniques is that you often need to train your model on a specific category, in order to perform well on the type of data you are looking for. In the case of guns, those models could work well, but when new categories need to be classified unknown to a classification model like shipping containers, the model needs to be extended and retrained to include that category for classification.

This research investigates the possibility of combining Convolutional Neural Networks together with distance metrics for image similarity ranking. Similarity performance analysis and different time analysis have been performed to look for a suitable algorithm.

The results show that it is indeed possible to combine Convolutional Neural Networks with distance metrics to create a well performing image similarity ranking algorithm that is also able to perform well on data unknown to the network. In general, keeping similarity performance and time constraints into account, it would be best to use an InceptionResNetV2 neural network together with the cosine distance metric.

Acknowledgements

I would like to thank my university supervisors Erik and Nicola as well as my NFI supervisors Simone and Mattijs for their support during this project. I would also like to thank the rest of the FBDA department at the NFI for their suggestions, kindness and most importantly for making me feel part of the team. It is a great group of people and it highlights the importance of having nice colleagues. Finally, I would like to thank all my friends and family for their support and love. Without them I would not have been able to accomplish this.

Contents

List of Figures

List of Tables

1 Introduction

Huge amounts of seized data within the forensic domain consist of images. In this field the term image is often used to describe a copy of a storage medium, but in this research we use the term images when we talk about pictures. Big steps are being made in using classification techniques to describe and analyse images within a dataset. This makes it easier to go through the images in a dataset, as you could easily search for instances of the category you are interested in. When you are for example looking for evidence of weapons in a dataset, you could analyse the images that are classified as weapons by the classification model. One of the downsides of this technique is that you have to retrain your classification model every time a new category arises as the model does not know how to cope with this data. An example could be that you have a model that is able to classify objects in your dataset into either being drugs, or being a gun. If it turns out that you have to find evidence of a knife in this dataset, the model does not know how to deal with your data, as there is no knife category yet. Recreating and retraining a model to deal with new categories could take a lot of time, making it interesting to look into alternative ways to support investigators looking for evidence in large datasets.

Two examples of existing reverse image search algorithms are the algorithms used by Google [7] and TinEye [28]. In the case of Google you are able to insert an image of which you would like to see where to find this image on the internet, or to show any similar looking images to your input image. TinEye also supports those features, as well as an tracking system that is able to show you a daily report of your image occurrence on the internet.

As an alternative approach to using classification models that are bound to specific categories, one could try to use image similarity ranking. One approach could be to use the generated image output features of a convolutional neural network (CNN). After removing the classification layers, one is able to generate a list of distances between the images of the dataset, and the reference image you use to find similar looking images using distance metrics. This way, forensic investigators will be able to find relevant information to their case faster compared to manually searching through all images. We only have to look at the top of the returned list for relevant images. By removing the classification layers, we hope to make the models more useful for previously unseen and untrained categories as well.

CNNs are an increasingly used method within the image analysis sector. Additionally, distance metrics are commonly used methods to calculate the distances between vectors. This research will mainly focus on using existing CNNs together with different distance calculation methods. It would benefit investigators that have to deal with large datasets of images, especially within the forensic domain. By finding an algorithm that is able to efficiently rank a dataset consisting of images based on similarity within a considerable amount of time, investigators would be able to solve cases that otherwise would be considered as too time consuming. The outcomes of this study would be useful for investigators within the forensic domain, but could perhaps also be used as reference material within other domains.

1.1 Motivation

Due to the increasing amount of digital data, it will be harder for forensic investigators to analyse every bit of information when working on an investigation. There are many cases in which the amount of data gathered is too large to deal with. One could for example think about analysing a large dataset of images that could help solving some crime. When someone has to go through all gathered images manually to inspect their content, it would be very time consuming. Another important aspect is to keep the data confidential, so uploading sensitive images to Google servers using their search algorithms is not appropriate.

It would be interesting to find a way to do the analysis of finding relevant images automatically, efficient and locally. One could for example make use of an image similarity ranking algorithm that automatically tries to rank the images based on similarity to objects of interest within an investigation. A possible addition could be to remove duplicates to reduce the size of the dataset. This limits the number of images the investigator needs to go through which saves time.

There are some challenges when using an image classification algorithm. For example they are often not really suitable to deal with previously unseen data. Additionally, training such an algorithm could take a lot of time, which is often limited in an investigation. When an algorithm takes for example a year to execute, it would in many cases not be feasible to use in an ongoing investigation.

Our image similarity ranking algorithm should be time efficient and able to rank images based on similarity accurately on previously unseen images as well. Memory complexity plays an important role since datasets are very large in a forensic setting. Therefore, our algorithm should work on an decent everyday working machine.

1.2 Current Limitations

In forensic cases you have a large dataset with images you would like to sort based on similarity to the object of interest. Unfortunately, reverse image search algorithms like Google and TinEye do not provide such an option. You are able to insert an image to see similar looking results, but you are not able to rank your own dataset using for example Google's reverse image search function. Forensic cases also mainly consist of very sensitive information, which should be remained within the forensic team. By using an algorithm like Google's reverse image search, you have to upload an image of a case to the Google servers, which is not what you want as an investigator. Ideally, you would like to keep every bit of information within your own team of investigators.

Another limitation of current existing systems is that they are mostly trained to recognise specific categories. You could for example use an algorithm that is trained to be very good at predicting images of guns, but does not perform well on previously unseen categories that are unknown to the network. When the algorithm does not know the category the results will be very poor. Ideally you would have an algorithm that performs well on previously unseen categories as well. Otherwise you need to retrain an existing algorithm every time when encountering a new category, which takes a lot of time.

Time could also be a major limitation. When you work with very large datasets, you want to reduce the time it takes to generate useful features of a single image from a CNN used for similarity ranking as much as possible. The faster these features per image are generated, the more time efficient the network is when the dataset becomes larger. Therefore, it is important that the used CNN is able to extract the features of an image very quickly and that also the distance calculation method is able to compute the distances between images quickly.

Of course, there will always be a hardware limitation. In general, it holds that the faster the CPU or GPU, the faster the algorithm will be able to execute. Furthermore the more memory we have available, the more data we are able to temporarily store in RAM. Ideally, we would like to have hardware limitations to be the bottleneck of the algorithm and have the other limitations mitigated as much as possible to have the algorithm optimised as much as possible on the available hardware.

1.3 Objectives

The main focus will be on finding out whether it would be possible to come up with a scalable image similarity ranking algorithm. This algorithm should able to have an high similarity performance on objects the algorithm is trained on, and preferably also on untrained objects. Furthermore the algorithm should be capable of executing within a considerable time span.

The objectives are to examine existing CNNs to be used. These CNNs will be combined with distance metrics to see if we are able to create an image ranking algorithm that is useful within the forensic domain. Several analysis methods will be used to determine how well a combination performs. For similarity ranking performance we are for example interested in the precision at k and the inverse recall at q. We are also interested in different time measurements. These measurements include model build time, feature extraction time, and distance calculation time. These and other methods are described in section 3.4. The used CNN distance metric combinations will be analysed to gather results on which combinations perform well.

Ideally, we come up with a conclusion that one combination will perform best in most scenarios giving the investigators a well working setup for image analysis based on the obtained results. It could turn out that there is not one solution suitable for every situation. In that case we at least provided insight in the performance of the different models, and perhaps an indication of which model distance metric combination to use in which circumstances.

1.4 Research Questions

The main research question of this research will the following:

How could image similarity ranking be applied within the forensic domain, to support investigators in ongoing investigations?

Several questions will be investigated to come up with an answer for the research question.

1. What CNNs could be interesting to use for an image ranking algorithm?
2. What types of distance metrics could be interesting to use for an image ranking algorithm?
3. How will different parameters affect the performance of the CNN distance metric combinations?
   - What is the impact of the type of CNN used on the similarity ranking performance, the model build time and the feature extraction time?
   - What is the impact of the type of distance metric used on the similarity ranking performance and the distance calculation time?
   - What is the impact on the similarity ranking performance, when using classes the CNNs are not trained on?
   - What is the influence of using multiple reference images as input on the similarity ranking performance?

By answering these questions, we should get a general idea of how this image similarity ranking algorithm could be implemented. Question 1 and 2 will be evaluated by looking into existing literature. The other questions will be answered by creating a script that is able to analyse the performances of different CNNs together with different distance metrics.

2 Background

2.1 Convolutional Neural Networks

In this section, some general information about the structure of CNNs is given. Existing well known CNNs will be shortly discussed as well as their contributions to the evolution of CNNs. Furthermore, the networks used throughout this research will be described.

2.1.1 General Information

There are many different CNNs nowadays that are useful when working with image recognition and image classification. The structures of the networks are all different, but they all make use of the same principles. A simplistic view of a CNN is given in figure 2.1. This figure shows some of the basic elements of a CNN. These elements will be discussed briefly.

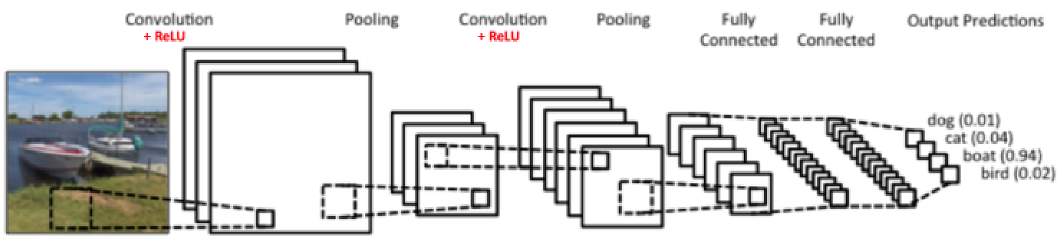

FIGURE 2.1: Simplistic view of a convolutional neural network [13].

Convolutional Layer

The convolutional layer is used to extract features from the image you feed into the network. It makes use of filters to achieve this. These filters slide over the input matrix. The dot products between the filter and the input matrix are computed. An activation map will be created yielding all the dot products between input matrix and filter. Different filters could be applied to extract different types of features. The output of the convolutional layer is the stacked result of the activation maps along the depth dimension. The depth dimension is in this case the number of colour channels. In a coloured image this number is 3 indicating the RGB values, which is short for red, green and blue values. These are integer values between 0 and 255, indicating the intensity of the colour.

Activation Functions

Activation functions are used to add nonlinearity into neural networks. These functions decide whether neurons in the network are activated or not. Different types of activation functions could be used when working with neural networks. Some of the popular ones are briefly described here. The most commonly used activation function in CNNs is the ReLU function. The ReLU function stands for rectified linear activation unit. This function returns the value directly if greater than 0 and 0 otherwise. The main reasoning behind using the ReLU function is that training the network will be less time consuming and it often yields better results.

A second used activation function is the sigmoid function. This function outputs values between 0 and 1. It normalises the output of each neuron. One of the downsides of this activation function is the vanishing gradient problem. When training a network the gradient could become really small using this activation function. This prevents weights from updating their values which affects the training of the network. Another downside is that it is computationally expensive.

A third often used activation function is the tanh function. This function returns a value from -1 to 1 which is different than the output values between 0 and 1 generated using the sigmoid function. Another difference with the sigmoid function is that the gradient of the tanh function is steeper. The described activation functions are depicted in figure 2.2.

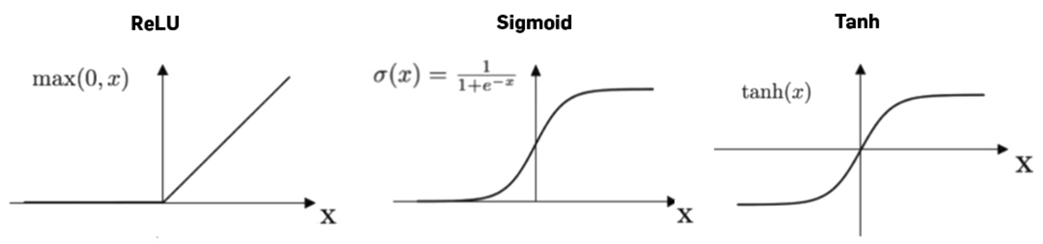

FIGURE 2.2: Some of the commonly used activation functions with CNNs [15].

Pooling Layer

There are different types of pooling layers. These layers are used to reduce the amount of parameters and the number of computations in the network. It applies a filter to the matrix to reduce the size while trying to maintain information. The most commonly used types of pooling are max pooling and average pooling. In the case of max pooling, the filter takes the maximum value within the area of the filter. In the case of average pooling, the average value of the area is calculated. This process is depicted in figure 2.3. In this case, the filter is a 2x2 matrix and it uses a stride of 2 indicating the filter step size.

Fully Connected Layer

Before entering the fully connected layers, the input is flattened. By flattening, the multidimensional output from the convolutional or pooling layers is converted into a 1 dimensional vector consisting of all values. In the fully connected layers, every neuron of a layer has a connection to all neurons in the next layer. Those connections are left out in the simplistic view figure. In these layers features that show characteristics of a specific class should have an high probability of belonging to that class. The fully connected layers are most of the time followed by a softmax activation function that converts a vector of values into a vector of probabilities yielding specific classes. In this research, layers such as the classification layer will be removed to remove the classification aspect. The feature vectors of the images will be extracted from intermediate layers.

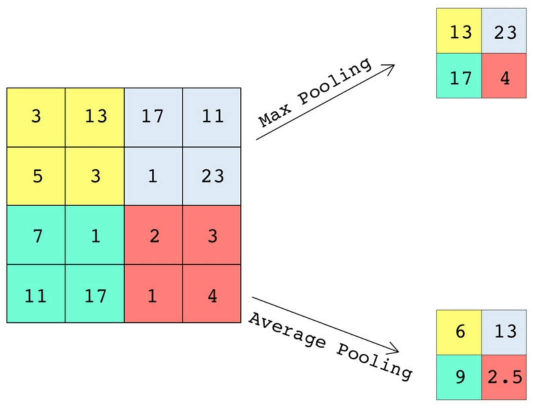

FIGURE 2.3: Visual representation of max and average pooling [1].

2.1.2 LeNet-5

Lots of research is done on using CNNs for content based image retrieval. One of the first described CNNs is the LeNet-5 network by LeNet [17]. This network was used to distinguish handwritten text. This network consists of seven layers as well as an input layer that takes an input image of 32x32 pixels. It makes use of two convolutional layers, one fully connected convolutional layer, two average pooling layers, a fully connected layer and an output layer which is a fully connected softmax layer indicating the probabilities of ten possible outcomes indicating the digits from 0 to 9.

2.1.3 AlexNet

Since around 2010, the developments of CNNs started to grow significantly. ImageNet, which is a large image database, decided to create challenges regarding visual recognition called ILSVRC [10] which stands for ImageNet Large Scale Visual Recognition Challenge. Some nowadays well known networks were winners of those ILSVRC challenges. One of those networks is AlexNet [16], which is the winner of the ILSVRC challenge of 2012. The top-5 accuracy error score of this network and some other to be discussed networks are depicted in figure 2.4. The top-5 accuracy error score indicates the percentage of images of which the correct class is depicted in the top 5 estimated classes by the model. Some interesting things to mention about AlexNet are that it has an highly optimised implementation for the usage with GPUs and that they have trained AlexNet with two GTX 580 3GB GPUs in parallel. Both GPUs use approximately half of the neurons and GPU communication occurs only in certain layers. The network consists of eight layers of which five layers are convolutional layers and three layers are fully connected layers. Another interesting design choice is the use of the ReLU nonlinearity function (discussed earlier) instead of the traditional tanh function. To reduce overfitting, AlexNet makes use of two data augmentation techniques. The first technique involves generating new images out of existing images, by taking a subset of the image. The second technique involves altering intensities of the RGB channels in the training images. One more method used to reduce overfitting is making use of dropouts. By using this method, neurons could be ignored at training stage with a probability of 0.5.

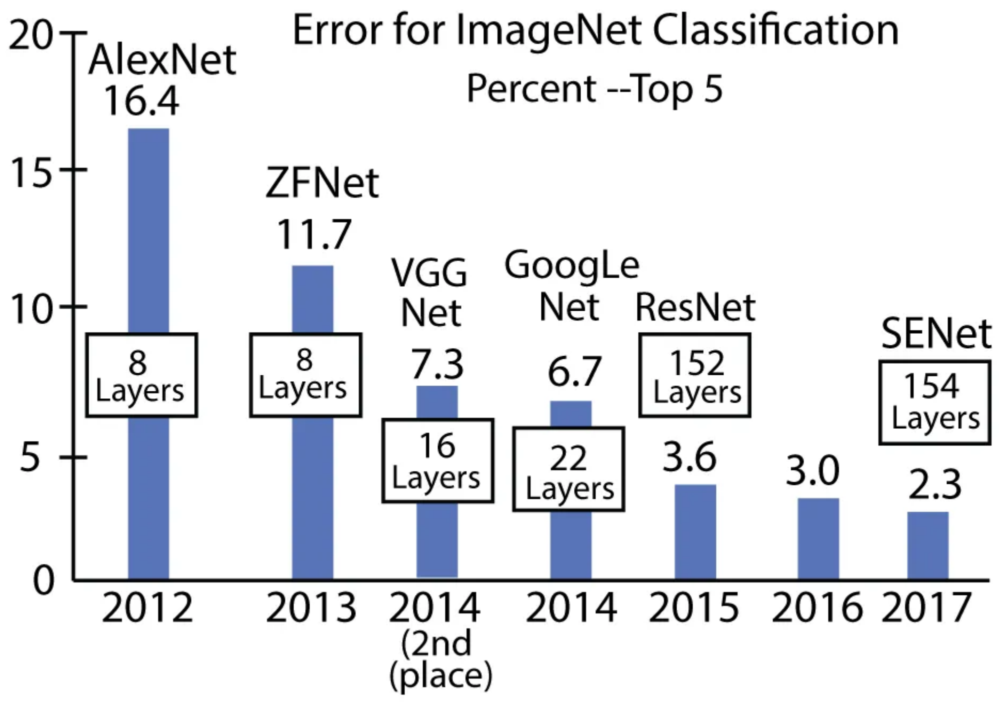

FIGURE 2.4: Winners of the ILSVRC challenge per year from 2012 to 2017 [19].

2.1.4 ZFNet

Zeiler and Fergus [31] developed a way to visualise activity within a model. This could be used to identify how different features are interpreted by the model instead of guessing what happens. They call it Deconvnet, which does the opposite of a convnet. It maps features to pixels instead of mapping pixels to features. They analysed different existing CNNs to check what could be improved on those existing models to increase the performance. They have for example analysed AlexNet and adapted it based on the results they obtained when visualising the activity within the model, which increased the overall performance. One interesting key point is that they demonstrated that a model trained for classification, is highly sensitive to local structure in the image, instead of using broad scene context. For experimenting, they used a model pretrained on ImageNet and trained a new softmax classifier on top. They also created a model from scratch. It turns out that the model created from scratch is not able to handle small datasets very well, while the pretrained model does very well, outperforming previous state of the art networks. The pretrained model can generalise well to other datasets. This principle is also called transfer learning.

2.1.5 GoogLeNet

Szegedy et al [24] developed a new CNN called GoogLeNet also known as Inception v1, which won both the ILSVRC 2014 detection challenge as well as the Classification challenge. GoogLeNet is a 22 layers deep and makes use of the Inception module architecture. One of the main differences with this neural network in comparison to other networks at that time, is that it made use of 1x1 convolution layers in the middle of the network, and global average pooling nearly at the end of the network, instead of using fully connected layers. The 1x1 convolution layers are used to greatly reduce the computation time of the network. GoogLeNet uses 12 times fewer parameters than the previously discussed AlexNet. One of the main contributions of this paper is that GoogLeNet did not follow the traditional structure of just stacking layers sequentially. Instead it made use of creative structuring of the different layers that turned out to be also very efficient in improving the accuracy of the model.

Inception v1 was the foundation for many more networks including Inception v2, Inception v3, Inception v4, InceptionResNet and InceptionResNet v2 [26], [25]. The latter ones are created with inspiration of the ResNet networks described in section 2.1.7. The computational cost of Inception v3 is similar to the cost of InceptionResNet and the computational cost of Inception v4 is similar to the InceptionResNet v2 network. The InceptionResNet v2 network will be used in this research and is 164 layers deep. Another network that will be used in this research is the Xception network [5]. This network is also created by Google. The name Xception stands for extreme version of Inception. It outperforms the Inception v3 network of Google by using a modified version of Depthwise Seperable Convolution used in the Inception v3 network. The Xception network is 71 layers deep.

2.1.6 VGGNet

VGGNet [22] made improvements to the previously discussed AlexNet. They figured out that it would be more efficient to use same sized smaller filters on top of each other than using for example a single large filter, as it would lower the computational cost and making it able to learn more complex features that are better representative, as normally features are getting more complex when getting deeper in a network. The VGG network uses 3 fully connected layers instead of using for example an average pooling layer as used in GoogleNet. VGGNet was the runner up in the ILSVRC 2014 competition. Two popular networks derived from this paper are VGG16 and VGG19. VGG16 has 13 convolutional layers and 3 fully connected layers of which the last layer consists of a softmax layer. VGG19 has 16 convolutional layers and also 3 fully connected layers with the last layer being a softmax layer. These networks will be used in this research.

2.1.7 ResNet

He et al. [9] came up with an residual learning framework that made it easier to train deeper neural networks by tackling the degradation problem meaning a saturated accuracy when increasing the depth of the network. They competed in the ILSVRC 2015 classification and won the first place. With ResNet, they are able to train much deeper networks that still have less complexity than for example VGGNet. They introduced a new neural network layer named The Residual Block. It uses Skip Connection identity mapping, which is used to add the output of a previous layer to the next layer. Traditionally, the previous layer only feeds into the current layer. By skipping some layers, you should at least be able to match the accuracy of a more shallower network mitigating the degradation problem. Consequently, you should be able to come up with much larger networks than the traditional networks that do not skip layers. One of the networks described in this paper is the ResNet50 network. This network consists of 48 convolutional layers, 1 maxpool layer and 1 average pool layer. This network will be used in the research.

2.1.8 NasNet

Zoph et al [32] designed the NasNet search space. They experimented with building network architectures using small datasets and transferring these architectures to work with larger datasets as well. Normally building model architectures requires significant architecture engineering. By using smaller datasets to build the network architecture and transferring the architecture, this progress will become less computationally expensive. Their composed neural networks all have convolutional layers with identical structure. Those layers do however have different weights. They found that their NasNet network architecture build with the CIFAR-10 dataset show state-of-the-art accuracy when used with ImageNet. In fact all of their models exceed the accuracy of other human designed models. The NasNetLarge network will be used in this research.

2.2 Reverse Image Search using GPUs

The research conducted by Johnson et al. [12] focuses on presenting an algorithmic structure of similarity search methods to be used with GPUs. The most important reason for this is that GPUs are able to reach higher speeds when for example computing k-NN graphs in comparison to a CPU. This makes it less time consuming when performing a reverse image search task. The paper discusses the architecture of the GPU and compares k-selection on a CPU with a GPU and some challenges that arise from using a GPU instead of a CPU for k-selection. Their aim is to perform k-selection as fast as the time required for scanning the input once. In other words, it could not be more efficient due to hardware limited factors. The results are promising. It turns out that using GPUs can significantly improve the speed of executing the reverse image search algorithm in comparison to using it with a CPU. These findings make it interesting to consider the usage of a GPU when performing the research.

2.3 Existing Reverse Image Search Implementations

In this section, some existing reverse image search implementations will be briefly discussed.

2.3.1 pyCBIR

Araujo et al. [2] came up with a program called pyCBIR. This interactive system allows users to search through specified databases of images and returns the set of suggested images based on approximate ranking. The program includes 10 different similarity metrics to measure the distance using different sets of features of both labeled and unlabeled images. This paper uses this newly developed software to evaluate on three types of CNNs. The paper uses four different types of datasets that are publicly available. One of the datasets used is the flicker material database [21]. This database consists of ten common materials and is designed for material recognition. This dataset might also be interesting to use within the forensic domain depending on the types of images one is looking for. When for example weapons need to be analysed on images, it might be interesting to use this dataset to train to detect metal objects on sets of images.

The paper also discusses some methods to improve the speed of the used algorithms. One suggestion is to use Principal Component Analysis [29] that is used to extract the most useful information as possible by using as little feature information as possible. In other words, only the features that are most important in decision making will be used to distinguish. This method will improve the computational cost as it reduces the amount of data being used when computing.

The paper mentions some important points that could perhaps be useful when performing this research. Feature reduction is an important factor used to reduce the computation time. Furthermore one should take into account the image sizes being used when trying to use a reverse image search algorithm. Ideally, you would like to resize the images to a point that they are not too big to use for analysis. Large images cause a very long computation time. Furthermore you would keep the size of the images large enough to maintain valuable information. When images are resized too a point that they become too small, valuable information within the images could potentially disappear. This paper also uses Sensitive Hashing Forest [3] to save computation time when finding the nearest neighbours. The program has not been tested as the GitHub repository is deleted and it was not possible to find the code anywhere else.

2.3.2 Reverse Image Search with Elasticsearch

Schulz [20] uses Elasticsearch to find similar looking images. First they extract the relevant features from a CNN and then they implement those feature vectors in Elasticsearch [6], which is a distributed, open source search and analytics engine for all types of data. Together with the use of the euclidean distance metric, they are able to represent the images based on the obtained distances sorted from low distance to high distance. A benefit of using Elasticsearch is that it is very fast in searching. When you are only interested in a specific set of images, Elasticsearch will be very fast in looking for the results. Elasticsearch could be interesting when you would like to come up with an algorithm that is able to search for the relevant images very fast.

2.4 Distance Metrics

In this section different distance metrics will be discussed that could be useful when calculating the distances of the extracted image features from the CNNs. Each of these distance metrics will be tested with the networks to check which combination works best. It will also be tested whether using a different distance metric for the feature distance calculation has any influence at all on the resulting ranked image list. We should consider the time taken to calculate the distances using the different approaches. If one approach yields for example slightly better ranking results but is for example way slower in calculating the distances than using another metric, then the other metric could perhaps be more useful based on the circumstances.

2.4.1 Euclidean Distance

Euclidean distance is known as being the straight line distance between two points. This metric could be used to calculate the shortest distance in n-dimensions. The formula used to calculate the distance between two points in two dimensions is:

$$d(A,B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$$

in which A and B are the two points of which you would like to measure the distance, a1 and b1 are the coordinates of the points in the first dimension and a_2 and b_2 are the coordinates of the points in the second dimension.

In the case of an n dimensional space, this formula will become:

$$d(A,B) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

This metric could be useful to compare the distances of the features extracted from the reference image and the features extracted from the remaining images in the dataset. Here we assume that the lower the total distance between the features of two images is, the more similar those images should be. In this way, we should be able to rank all images in the dataset based on similarity towards the image of interest. The euclidean distance is depicted in figure 2.5a.

2.4.2 Manhattan Distance

The Manhattan distance metric has a different approach than the euclidean distance metric. Instead of taking the straight line distance between two points, it is calculated by the sum of absolute differences between points across all dimensions. The formula to calculate the Manhattan distance in a two dimensional space is stated as:

$$d(A,B) = |a_1 - b_1| + |a_2 - b_2|$$

in which A and B are the two points of which you would like to measure the distance, a1 and b1 are the coordinates of the points in the first dimension and a_2 and b_2 are the coordinates of the points in the second dimension.

In the case of an n dimensional space, this formula will become:

$$d(A,B) = \sum_{i=1}^{n} |a_i - b_i|$$

The Manhattan distance is depicted in figure 2.5b.

2.4.3 Cosine Similarity

The cosine similarity metric could be used to measure the cosine angle between different vector pairs. The difference with, for example, euclidean distance is that it measures the correlation between features instead of the features them selves. Making it possible to find vectors with similar patterns.

The formula to calculate the cosine similarity is stated as follows:

$$cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

in which A and B are vectors and A$_i$ and B$_i$ are components of the vectors. The cosine distance is derived from the cosine similarity as follows:

$$distance = 1 - cos(\theta)$$

The cosine similarity is depicted in figure 2.5c.

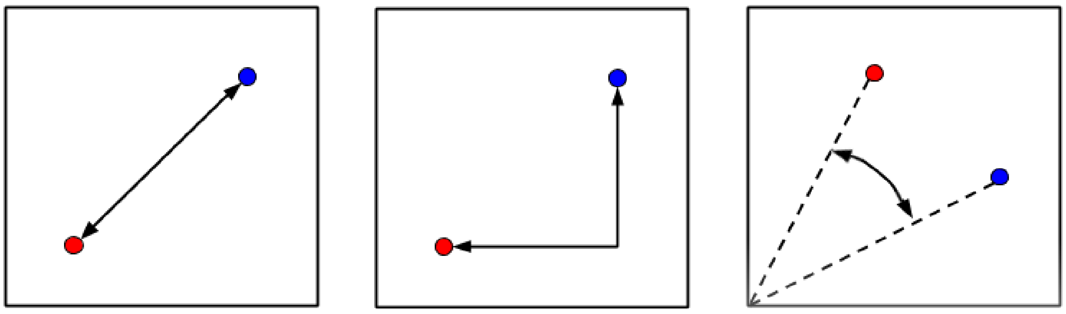
(A) Euclidean Distance (B) Manhattan Distance (C) Cosine Similarity

FIGURE 2.5: Visual representation of different distance metrics [18].

2.4.4 Modified Distance

Ivanov et al. [11] described a new metric to improve the recognition of faces in an image. They have tested different existing distance metrics on a group image to check how many faces they are able to identify and compare the results to their own created distance metric called modified distance. They have compared their method against some well known distance metrics like euclidean distance and angular cosine distance. Their results look quite promising. When tested on recognition, the accuracy of their metric outperformed every other metric used with an increased accuracy of at least 15%. When tested on clustering, their method outperformed every other method with an increased accuracy of least 11%.

$$f\left[(x_1 \quad y_1 \quad z_1),(x_2 \quad y_2 \quad z_2)\right]=$$

$$\left(\frac{|x_1-x_2|^2+|y_1-y_2|^2+|z_1-z_2|^2}{(x_1+y_1+z_1)(x_2+y_2+z_2)}\right)^2+\left(\frac{x_1x_2+y_1y_2+z_1z_2}{\sqrt{(x_1^2+y_1^2+z_1^2)(x_2^2+y_2^2+z_2^2)}}-1\right)^2+$$

$$\left(|x_1-x_2-2y_1+2y_2+z_1-z_2|^2+|2x_1-2x_2-y_1+y_2-z_1+z_2|^2+|x_1-x_2+y_1-y_2-2z_1+2z_2|^2\right)^2 /$$

$$4\left(|2x_1-y_1-z_1|^2+|x_1-2y_1+z_1|^2+|x_1+y_1-2z_1|^2+|2x_2-y_2-z_2|^2+|x_2-2y_2+z_2|^2+|x_2+y_2-2z_2|^2\right)^2$$

This paper only focuses on face recognition, but perhaps it would be interesting to look into the possibilities of using their modified distance metric with an image similarity ranking algorithm as well. The formula is quite large, thus perhaps will be less time efficient than using for example the cosine similarity algorithm.

3 Methodology

In this section the methodology is described to carry out the research to give answer to question 3 as indicated in section 1.4:

How will different parameters affect the performance of the CNN distance metric combinations?

The results obtained from this question together with the results from question 1 and question 2 that are discussed in the background literature will be combined to formulate a conclusion to the main research question. A short description is given of the approach to give answer to the sub-questions used to answer question 3. The applied methods and some limitations regarding the applied methods will be discussed as well as the reasoning for using the applied methods.

3.1 Approach

We have developed a script that is able to combine existing CNNs with different types of distance metrics and analyse their performance in terms of similarity ranking performance and calculation times. With this script we are able to give answer to the following stated sub-questions:

SQ 1: What is the impact of the type of CNN used on the similarity ranking performance, the model build time and the feature extraction time?

By analysing the results of the different CNNs being used, while keeping the distance metric consistent, we are able to see the impact on both similarity ranking performance, as well as on model build time and feature extraction time.

SQ 2: What is the impact of the type of distance metric used on the similarity ranking performance and the distance calculation time?

By analysing the results of the different distance metrics being used, while keeping the CNN consistent, we are able to see the impact on both similarity ranking performance, as well as on the distance calculation time.

SQ 3: What is the impact on the similarity ranking performance, when using classes the CNNs are not trained on?

By analysing the results of the script on classes the models are not trained on, we will be able to see how well the models are able to cope with unknown classes.

SQ 4: What is the influence of using multiple reference images as input on the similarity ranking performance?

By analysing the similarity ranking performance of the resulting ranked list when using multiple reference images as input and compare it with that of the resulting ranked list when using a single reference image, we will be able to see the effects of using multiple reference images on the similarity ranking performance.

By combining the results obtained from these sub-questions, we are able to gain more insight into the effects of the usage of different parameters on the performance of the CNN distance metric combinations. By combining the answers of these questions we will be able to yield a CNN distance metric combination which is either: 1) the fastest in terms of distance calculation time 2) the best in terms of similarity performance, or 3) a combination of the previous two. Combining these findings with the answers derived from the literature study, we are able to give answer to the main research question.

3.2 Design

3.2.1 Code

The code for the script has been written in Python, as this language is used a lot when working with data analysis and it has a lot of useful modules. The CNNs will be created by using the TensorFlow and Keras modules [27], [14]. One is able to load pretrained networks. Those networks are pretrained on ImageNet. You are able to cut the network at a certain point and obtain the output feature vector from that layer. This feature vector will then be used with distance metrics to create the ranked list of images. This research is mostly focused on excluding the last layers of the network. Some more shallow layers have been tried, but this decreased the overall performance. Therefore, similarity ranking on shallow layers has not been investigated further. The code is publicly available for everyone to download and use in order to be able to reproduce the results obtained within this research, as well as to test the code on your own datasets. A README file will also be made available with instructions on how to use the code and the modules needed for the code to work [4]. Additional obtained results will also be available here.

3.2.2 Used Convolutional Neural Networks and Distance Metrics

The CNNs as well as the distance metrics used for the analysis done in this research are all depicted in table 3.1. The distance metrics are also described in Section 2.4. The CNNs are chosen based on the background literature, popularity, support for pretraining, as well as performance on image classification tasks [14].

Convolutional Neural Networks
Xception
VGG16
VGG19
ResNet50
InceptionResNetV2
NasNetLarge

Distance Metrics
Euclidean
Manhattan
Cosine
Modified

TABLE 3.1: CNNs and distance metrics used in the experiments.

3.2.3 Database

A database is created using SQLite3 [23]. This database is used to store all the results that are used for the analysis. Whenever the script is used on a dataset, information is stored about the dataset, the reference image used, as well as all the information calculated when analysing a CNN distance metric combination.

3.2.4 Datasets

Caltech-256
One of the datasets used in this research is the Caltech-256 dataset [8]. The images within this dataset are publicly available for everyone to see and use. This dataset has images of 257 classes that are all placed in separate class folders. The dataset has 30607 images in total. The object categories within this dataset are extremely diverse. It is the successor of the Caltech-101 dataset with more classes and more images per class. The obtained results are generated using the entire dataset. This means that one is able to reproduce the same obtained results when using the dataset. Some of the reference images from the Caltech-256 dataset used in this research are shown in figure 3.1. The names of all the reference images used are added to the code repository. This makes it possible to recreate the same exact results in terms of similarity performance.

FIGURE 3.1: Some of the 100 reference images used when working with the Caltech-256 dataset.

FIRE
This dataset used by the Netherlands Forensic Institute has images of 16 classes with a total of 117.920 images. FIRE stands for Forensic Image Recognition Engine. The dataset uses images from cases as well as crawler data from the internet. The main difference between this forensic dataset and Caltech-256 is the quality of images. Forensic data often consists of images of low quality, images taken from weird angles, or images that are out of focus. Reference images have been used from 15 out of these 16 classes. The 16th class is the 'other' category that contains images not belonging to any other class. The classes together with the number of images located inside each class could be seen in table 3.2.

Class	# of images	Class	# of images
bank card	509	military vehicles	650
cannabis	425	money	112
cannabis plantation	59	narcotics	1008
chemicals	568	narcotics packaging	792
fire arms	577	pornography	4028
identity documents	787	screens	247
IS flags	132	shipping containers	1072
knife	1798	others	105156

TABLE 3.2: Table containing FIRE dataset information.

3.2.5 Used Machines

This research is performed on two different machines. All results using the Caltech-256 dataset are obtained on a machine running Ubuntu 20.04 LTS with kernel version 5.4.0-48-generic using a GeForce GTX 1060 6GB GPU and 16GB of DDR4 RAM. All results using the FIRE dataset are obtained on a machine running Linux Mint 19 with kernel version 4.15 using a GeForce GTX 1080 8GB GPU with 32GB of DDR4 RAM.

3.3 Design Limitations

Of course, there are also limitations about the design. The used CNNs are for example only pretrained using ImageNet. Different datasets could have been used as pretraining to see their effects on the results. Furthermore, there is a possibility that there are images of Caltech-256 that are also used during pretraining by ImageNet, since ImageNet has over 14 million images. Only the classification layers are removed from the networks, making the derived feature vectors still leaning towards specific classes. It has been tried to apply distance calculations to more shallow layers as well, but since the similarity performance decreased significantly it has not further been investigated.

Furthermore, the script only makes use of six CNNs as well as four different distance metrics. More metrics and networks could be tested that could perhaps show different results. Another limitation in the design is the way the modified distance metric is calculated. The other distance metrics are all calculated by making use of the same module, but the modified distance metric is implemented by myself as there was no support for that metric yet. This could perhaps have influence on the calculation time.

There are also limitations in code optimisation, but it is at least been tried to keep everything consistent. Even though there might be optimisations to improve calculation times, the time comparisons are all done in the same way. This shows time efficiency of one method over another still in a reliable way, except for the calculation of the modified distance metric as already discussed.

3.4 Analysis Methods

In this section the methods that are used to gather information regarding the similarity performance and the time measurements are explained.

3.4.1 Precision at k

The precision when classifying images is indicated with the following formula:

$$precision = \frac{TP}{TP + FP}$$

TP short for True Positives indicate the number of images correctly classified as the target class. FP short for False Positives indicate the number of images classified as being of the target class, but are actually from another class. In this research a ranked list will created and assumed is that images belonging to the target class, which is the class belonging to the reference image, should be on top. When the precision at for example k: 50 is calculated, the first 50 images of the ranked list will be checked. In this research k will always be smaller than the number of images belonging to the reference class. This means that when the algorithm performs well, all images that are checked should belong to the reference class. Using this assumption the formula could be adopted to:

$$precision\ at\ k = \frac{TP}{k}$$

Every image that is labeled as not being of the reference class is considered to be a false positive. Suppose we have a ranked list of images and a reference image of dog. We would like to have as many dogs at the top of the ranked list as possible, as the higher on the list, the more similar the images should look like. If we take a k value of 50, we are interested in the first 50 images of the ranked list. If 25 of those 50 images are actually dogs, this yields a precision of 0.5.

3.4.2 Inverse recall at q

Another measurement method used in this research is the inverse recall at q. With the inverse recall at q insight is given into what fraction of the ranked list one needs to go through to reach a certain fraction q of the class of interest. This will be useful to check the amount of data needed to find the target data. The inverse recall at q is indicated with the following formula:

$$Inverse\ recall\ at\ q = \frac{number\ of\ images\ passed\ to\ reach\ q}{total\ number\ of\ images\ in\ the\ dataset}$$

Suppose a dataset has a total of 100 images and 10 of those images are from the class of interest and one uses a q value of 0.5. This means that one is looking to find half of the target images in the dataset which is in this case 5. A ranked list will be created based on similarity to the reference image. The result will be the number of images needed to reach 5 target images divided by the total number of images in the dataset. When the 5th target image is found when looking at the 20th image in the ranked list, the inverse recall at q: 0.5 will be 20 divided by 100 which is 0.2. This means that 20% of the images in the sorted list need to be seen to find 50% of the target images.

3.4.3 AUIRC

We also make a plot of the inverse recall for every network distance metric combination. With this plot we are able to calculate the AUIRC which is short for Area Under the Inverse Recall Curve. This value gives an overall indication of the performance in terms of inverse recall. The lower the AUIRC the better the model will be in general. We should note that the optimal value will depend on the situation. The x axis denotes the number of images belonging to the reference class and the y axis denotes the total number of passed instances.

3.4.4 Time Measurements

For time measurements, we are interested in the time it takes to build the pretrained CNN. In other words, the time needed to load in the pretrained CNN from Keras.

We are interested in the time it take to extract the features of the images in the dataset from this network, and the time it takes to calculate the distances between reference image and images of the dataset used to generate the sorted list of images.

3.5 Method Limitations

There are several limitations when using the specified methods. We are using image classification scoring techniques with our image ranking algorithm as we need to use some sort of metric to judge the performance of the algorithms. The algorithms do not make a prediction of the class an image belongs to, instead it shows a distance which we use to determine the similarity between the images. We assume that the shorter this distance is, the more similar the images look. What could happen is that an image that belongs to another class than the reference class you are looking for could be more similar to your reference image than an image that actually contains something of the reference class. In our case this image will cause the precision to drop and inverse recall to rise, as it does not contain content of the reference class. In reality the algorithm does a proper job when putting an image in front of the list that is not from the reference class, but in fact more similar to the reference image than an actual image from the reference class that looks totally different.

There are some inconsistencies within the datasets with regards to the amount of images belonging to a specific class. Some of the classes have more images than other classes. This could perhaps influence the obtained results a bit, however this should not necessarily be disadvantageous, as in a real life scenario, there will also never be an equal amount of images of every class you are interested in. Due to the fact that not every class have an equal amount of images within and the way the AUIRC is calculated, the optimal value of calculated AUIRC will fluctuate a little per class, however this will not have a big influence as the amount of images per class do not fluctuate that much, and the larger the datasets are in relation to the amount of images per class, the more this problem is mitigated.

Due to the limitations of the datasets, it could sometimes be that an image is labeled as one class, but also shows content of another class. This could influence the obtained results as sometimes an image will be considered as not similar to the reference image, even though the content of the reference image is also visible on the target image.

4 Results

In this section the results obtained throughout the research will be shown. The Caltech-256 dataset along with the FIRE, used by the Netherlands Forensic Institute will be analysed on similarity ranking performance as well as on calculation times. Furthermore the results of the 5 best performing classes from the FIRE dataset will be depicted as well as the results of 5 classes the pretrained CNNs are not trained on. The networks are all pretrained using ImageNet. The networks are trained to be able to classify objects into 1000 different categories [30]. We used images not belonging to any of those classes to represent images unknown to the networks. With the Caltech-256 dataset we have also compared the usage of multiple reference images to see if it will improve the similarity ranking performance.

4.1 Caltech-256

4.1.1 Similarity Performance

The similarity performances are all measured by using reference images of 100 different classes, and analyse them with every CNN distance metric combination. This means that for every reference images, a ranked dataset is created for every combination.

AUIRC

A summary of the obtained results by looking at the AUIRC is shown with boxplots in figure 4.1. The AUIRC is depicted on the y axis, the CNNs used are shown on the x axis and the different colours indicate the distance metrics used. Interesting to observe in this figure is that the results when using the modified and cosine distance metric are very similar and are more accurate than for example using the euclidean and the Manhattan distance metrics. When looking at the boxplots of the different CNNs, one can see that the interquartile range (IQR) of the InceptionResNetV2 network using the modified distance metric is the smallest. In addition, the maximum of that boxplot is the lowest too. They are closely followed by that of the Xception network, although this network has more larger outliers. The network with the least large outliers using the cosine or modified distance metric is the VGG16 network.

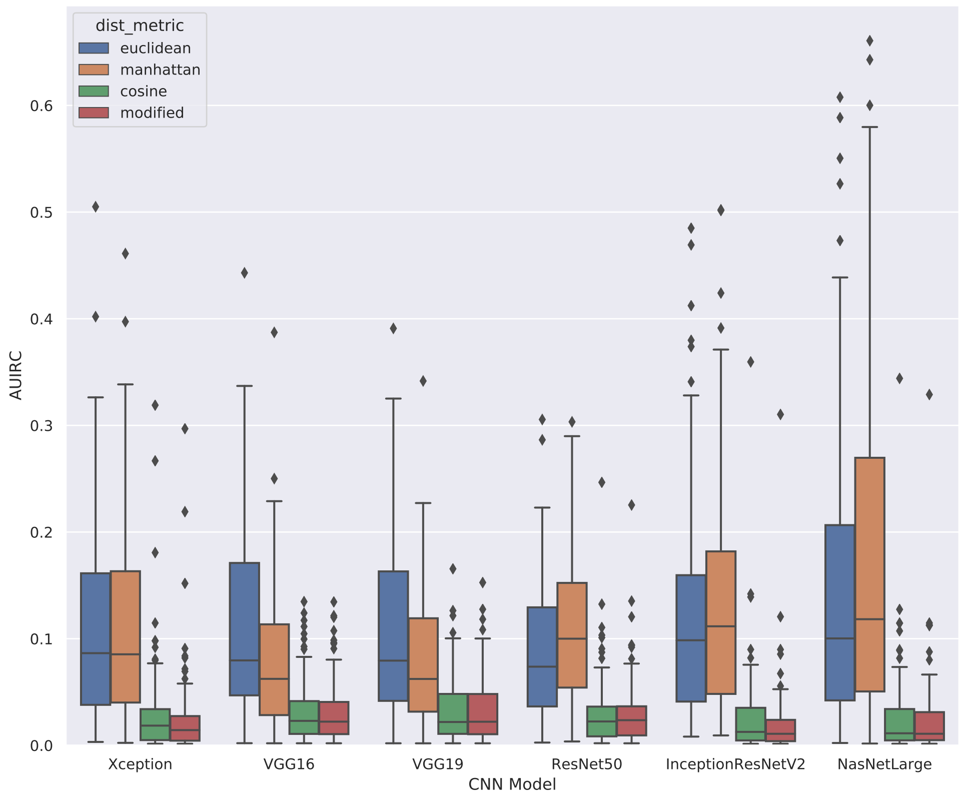

FIGURE 4.1: Results for the AUIRC with the Caltech-256 dataset using 100 classes.

A scatterplot of the average AUIRC is created per model and per metric. This is depicted in figure 4.2. The y axis represents the average AUIRC over the 100 different runs per model metric combination. The x axis represents the distance calculation time in seconds, which indicates the time it took to calculate the distances between all images within the dataset to the reference image used. The colours represent the used model and the different shapes indicate the metric being used. Within this plot we can especially get a bit more insight into the clusters that formed. It can be seen that the cosine as well as the modified distance metric are clustered pretty nicely and show little difference in terms of the calculated AUIRC. The euclidean as well as the Manhattan distance metric are however more spread out and show larger deviations. The biggest difference between the cosine and modified distance metric lies within the distance calculation time.

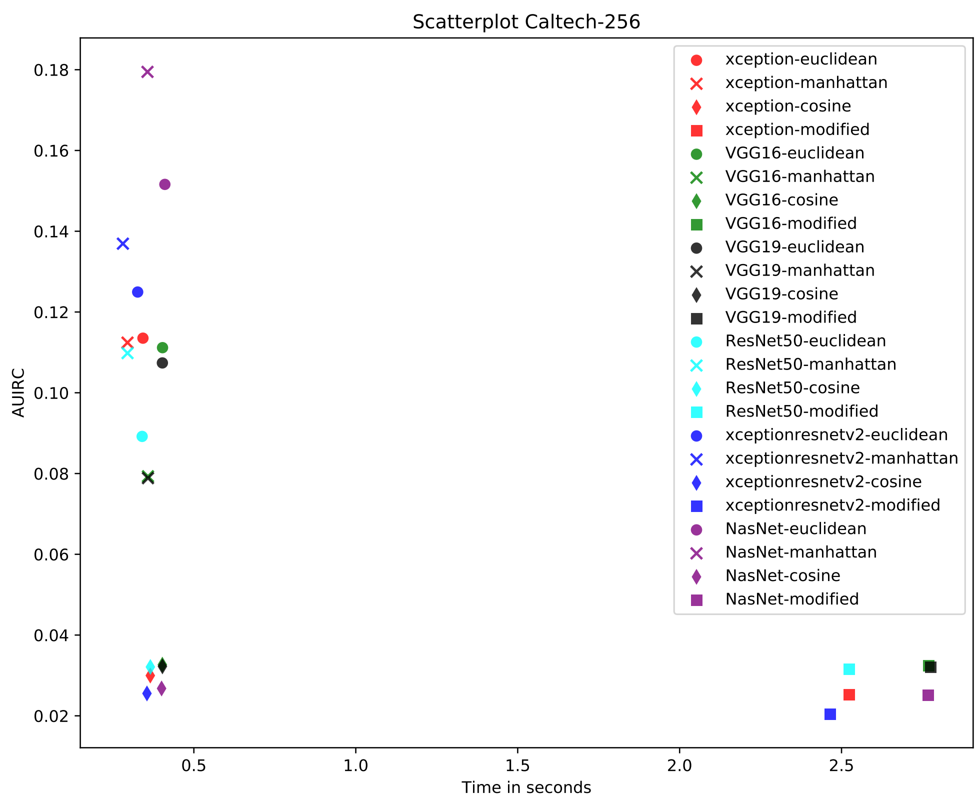

FIGURE 4.2: Scatterplot of the average AUIRC on the Caltech-256 dataset using 100 classes

Inverse recall at q

The boxplots of the inverse recall at q: 0.5 are depicted in figure 4.3. The inverse recall at 0.5 is depicted on the y axis, indicating the fraction of the amount of images one need to go through to have found 50 percent of the images belonging to the reference class. The x axis depicts the CNNs and the colours indicate the distance metrics used. When looking at the IQR of the boxplots, one can see that especially in the case of the cosine and the modified distance metric, the values are all very similar, independent of the network that is used. Also, the maximums of those boxplots are very similar. The outliers of the different networks when using the cosine or modified distance metric are also very similar, apart from a very few larger outliers.

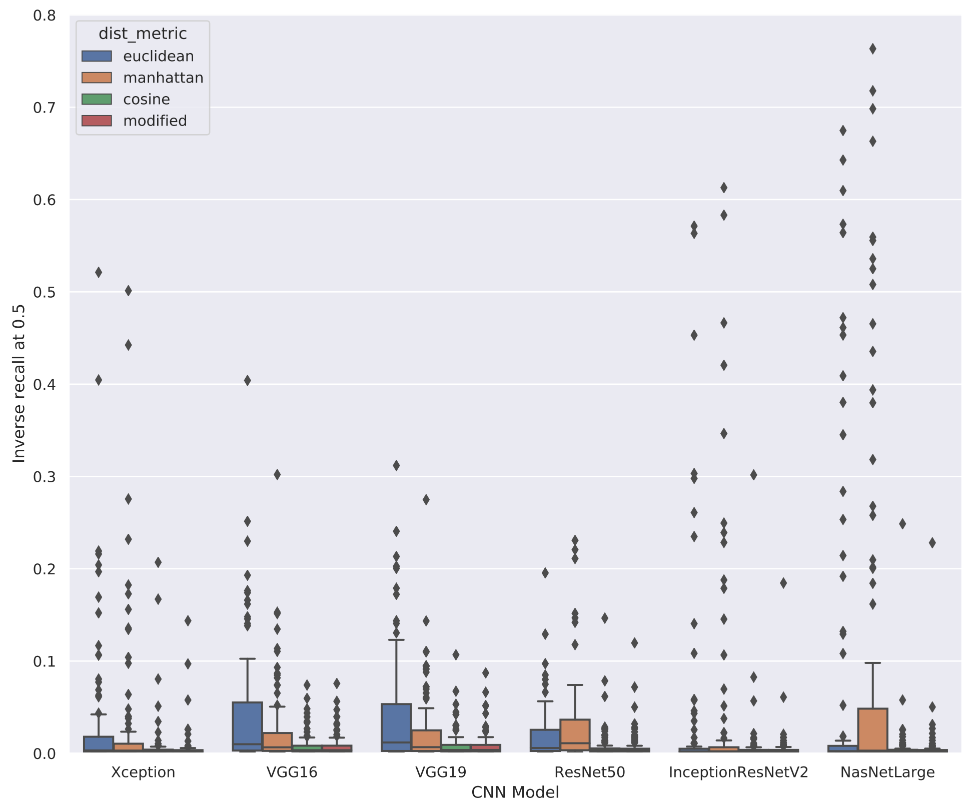

FIGURE 4.3: Results for the inverse recall at 0.5 on the Caltech-256 dataset using 100 classes.

Precision at k

The boxplots of the precision at k: 50 are depicted in figure 4.4. The precision at 50 is depicted on the y axis, indicating the fraction of the amount of samples within those 50 images belonging to the reference class. The x axis depicts the CNNs and the colours indicate the distance metrics used. When looking at the IQR of the boxplots, one can see that they are all very spread out, making it very inaccurate. Also, the minimums and maximums of the boxplots are very spread out in all cases. Based on these boxplots, InceptionResNetV2 as well as NasNetLarge perform best. Also in this case, it seems that the modified distance metric or the cosine distance metric are most suitable to use. The possible reasoning behind these spread out results will be discussed in section 5.1.1.

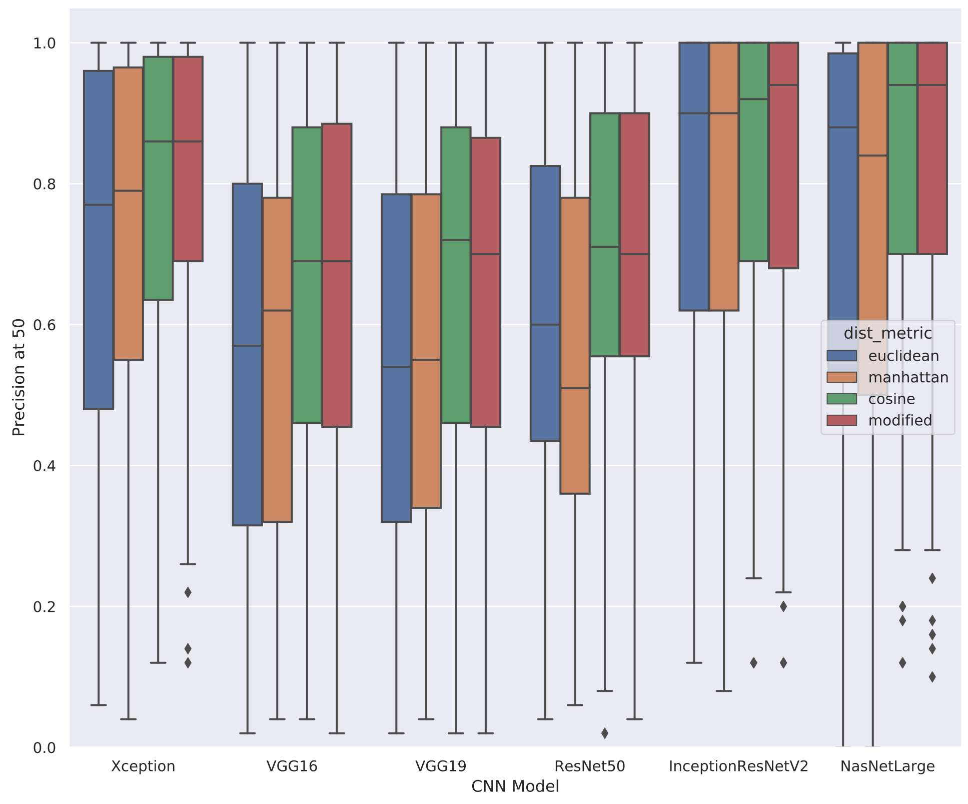

FIGURE 4.4: Results for the precision at 50 on the Caltech-256 dataset using 100 classes.

4.1.2 Time measurements

Model build time

The build time of the different CNNs is depicted in figure 4.5. The y axis yields the time in seconds and the x axis yield the different models. In total 32 runs are performed per model to give a clear indication of the average build time needed per model. The standard deviation is included. It can be seen that the VGG16 and VGG19 models have the highest standard deviation. Furthermore, the build time of the Xception, VGG16, VGG19 and ResNet50 models are all very similar followed by InceptionResNetV2 that takes about 3 times as long as the other three mentioned. NasNetLarge is considered to be slowest in terms of model build time being 5 to 6 times slower than Xception, VGG16 and VGG19.

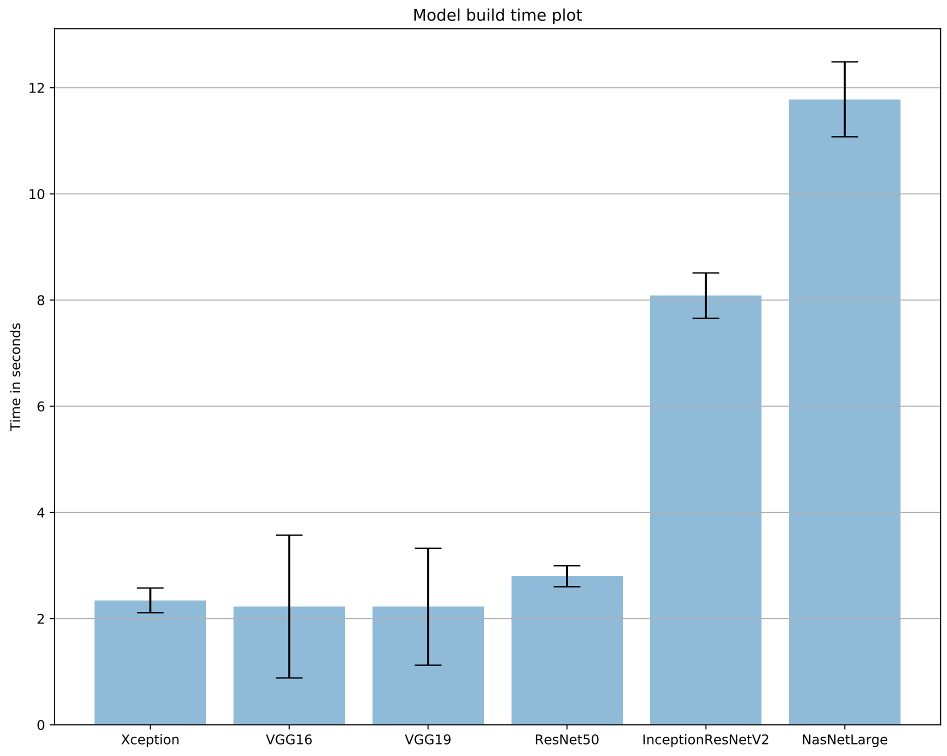

FIGURE 4.5: Build time of the different CNNs when working with the Caltech-256 dataset.

Features extraction time

The feature extraction time per 10.000 images of the different CNNs are depicted in figure 4.6. The y axis yields the time in seconds and the x axis yield the different models. In total 32 runs are performed per model to give a clear indication of time needed to extract the features. The standard deviation is included. It can be seen that the VGG16, VGG19 and ResNet50 are fastest in extracting the features from the Caltech-256 dataset. Xception and InceptionResNetV2 take a little more than twice as long to extract the features. NasNetLarge is considered to be the slowest being almost nine times slower than the VGG16, VGG19 and ResNet50. The standard deviations are very small, making it an accurate time estimation.

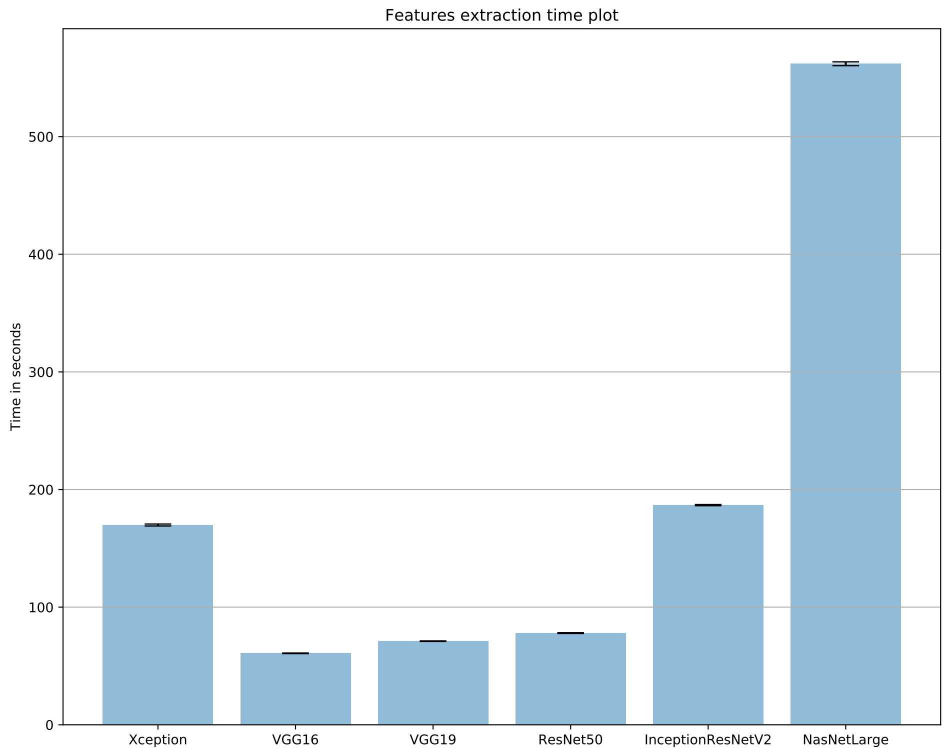

FIGURE 4.6: Feature extraction time per 10.000 images using the Caltech-256 dataset.

Distance calculation time

The distance calculation time of the different CNNs are depicted in figure 4.7. The y axis yields the time in seconds and the x axis yield the different models. In total 48 runs are performed per model to give a clear indication of time needed to calculate the distances between the images in the dataset to the reference image. The standard deviation is included. It can be seen that the time needed to calculate the distances are very similar for the euclidean, Manhattan and the cosine distance metric with a very similar standard deviation. The time needed to calculate the distances using the modified distance metric is more than 8 times slower. Also the standard deviation is considered to be a bit larger, making it a bit less consistent.

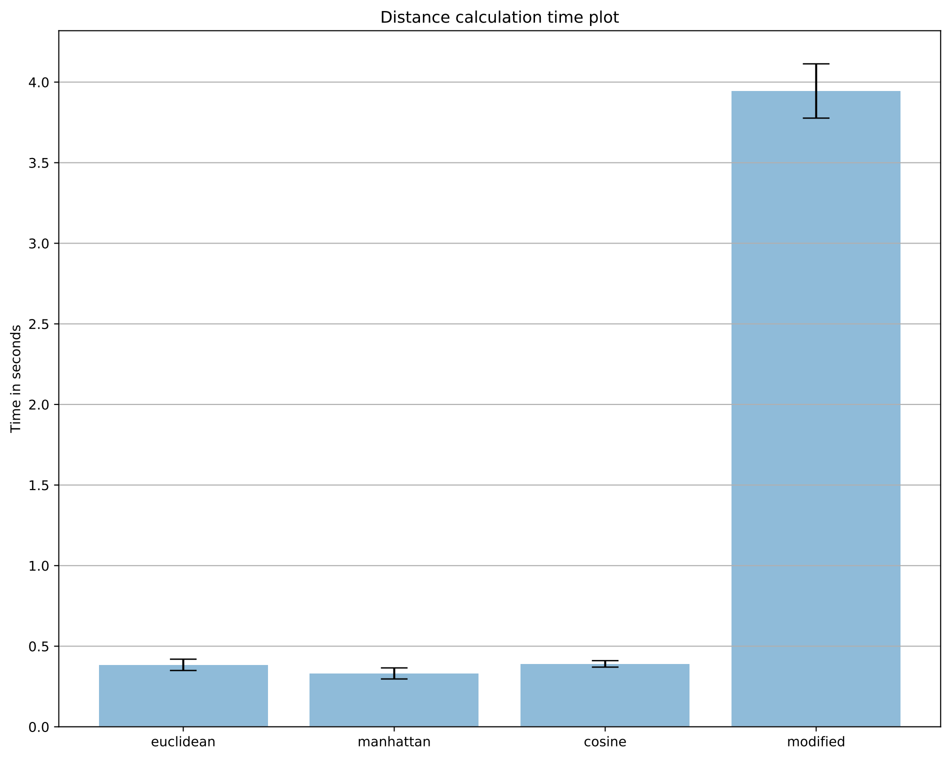

FIGURE 4.7: Distance calculation time of the different distance metrics on the Caltech-256 dataset.

4.1.3 Multiple reference images

Analysis has been done to investigate the influence of using multiple reference images on the similarity ranking performances. For this part of the analysis it is decided to make use of the InceptionResNetV2 network together with the cosine similarity metric. The reason for this is that this combination seems to perform well based on the previous obtained results. Furthermore, investigating into every single combination again will yield a lot of extra results. The approach is the same as when investigating on the influence of a single reference image, but in this case we have created the ranked list per reference image and summed up the overall distances. Again we sorted the results based on the shortest distance to the reference images. We have investigated the influence of using 1 up to 5 reference images. Again 100 classes are used to have more insightful information in general.

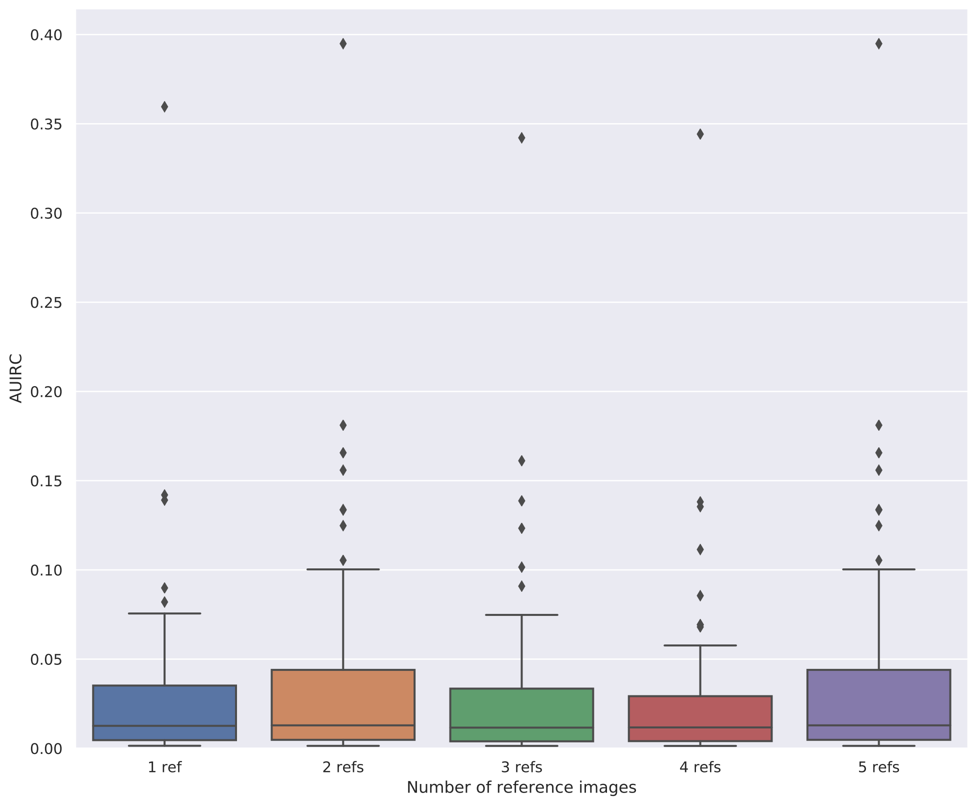

FIGURE 4.8: The AUIRC boxplot comparisons using multiple reference images with the InceptionResNetV2 network and the cosine distance metric.

The obtained results for the AUIRC is shown in figure 4.8. It can clearly be seen that the depicted boxplots are looking very similar when a fluctuating number of reference images is used. The results for the inverse recall at 0.5 are shown in figure 4.9. Also, in this figure the boxplots look very similar when fluctuating the number of reference images used. When looking at the precision at 50 results in figure 4.10, the results seem to be fluctuating a bit more. Still it does not seem that the number of reference images used has a big effect on the similarity performance. Using 1 reference image shows near similar results than using 3 or 4 reference images. Using 2 reference images seems to yield a similar boxplot as using 5 reference images, both seeming to perform less than using for example 1 reference image.

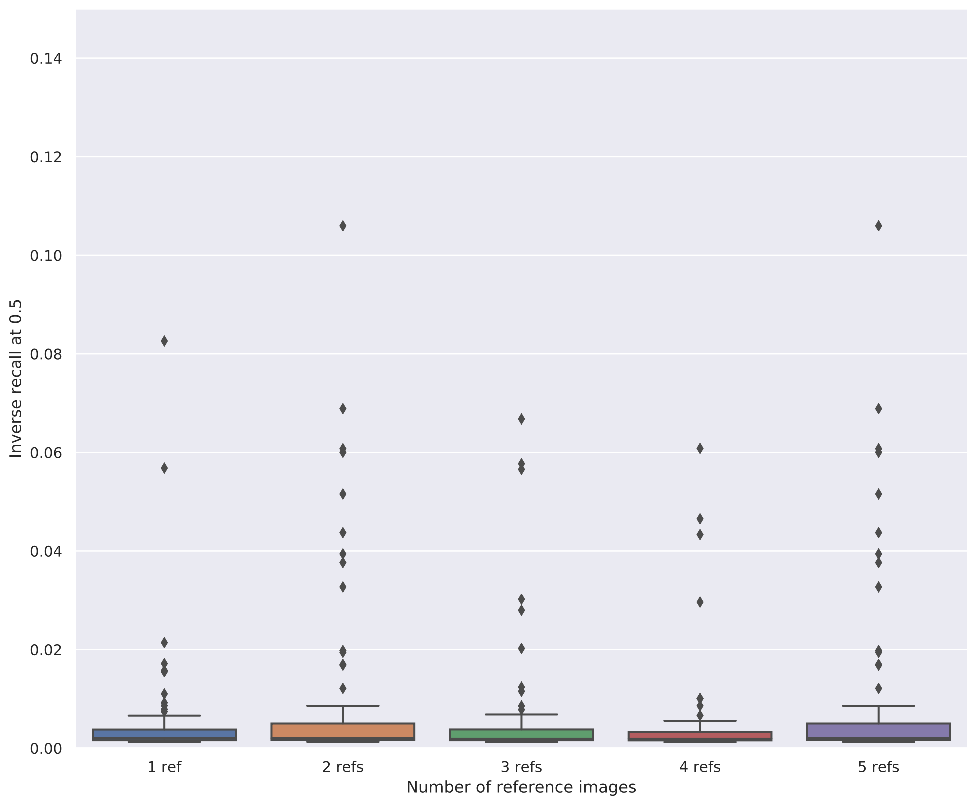

FIGURE 4.9: The inverse recall at 0.5 boxplot comparisons using multiple reference images with the InceptionResNetV2 network and the cosine distance metric.

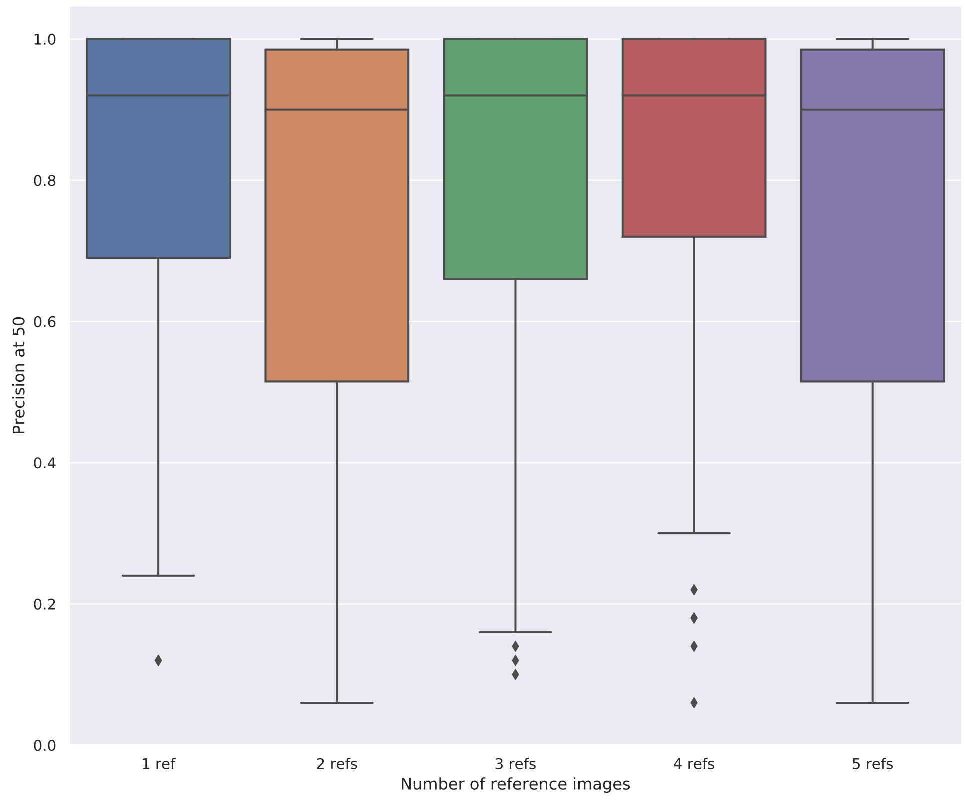

FIGURE 4.10: The precision at 50 boxplot comparisons using multiple reference images with the InceptionResNetV2 network and the cosine distance metric.

4.2 FIRE

4.2.1 Similarity Performance

The similarity performances are all measured by using reference images of 15 different classes, and analysing them with every CNN distance metric combination. This means that for every reference image, a ranked dataset is created for every combination.

AUIRC

A summary of the obtained results by looking at the AUIRC is shown with boxplots in figure 4.11. The AUIRC is depicted on the y axis, the CNNs used are shown on the x axis and the different colours indicate the distance metrics used. Interesting to observe in this figure is that the results when using the modified and cosine distance metric are again very similar and are more accurate than, for example, using the euclidean and the Manhattan distance metrics. This behaviour is similar to the AUIRC plot of the Caltech-256 dataset. When looking at the boxplots of the different CNNs, one can see that the IQR of the Xception Network using the modified distance metric is the smallest. In addition, the maximum of that boxplot is the lowest too, followed by that of InceptionResNetV2. They both have two outliers with fairly similar values. This was also the case with the Caltech-256 dataset, but in that case Xception with modified distance metric scored a little higher than that of InceptionResNetV2.

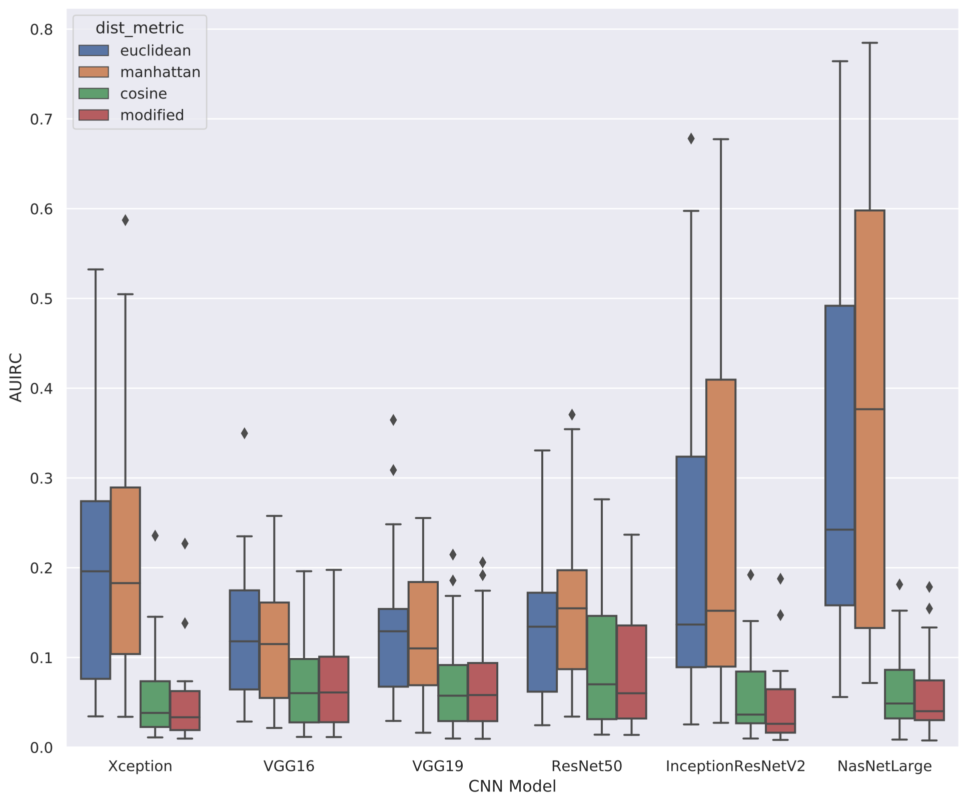

FIGURE 4.11: Results for the AUIRC with the FIRE dataset using 15 classes.

A scatterplot of the average AUIRC is created per model and per metric. This is depicted in figure 4.12. The y axis represents the average AUIRC over the 100 different runs per model metric combination. The x axis represents the distance calculation time in seconds, which indicates the time it took to calculate the distances between all images within the dataset to the reference image used. The colours represent the used model and the different shapes indicate the metric being used. It can be seen that the cosine as well as the modified distance metric are clustered pretty nicely and show little difference in terms of the calculated AUIRC, as was also the case with the Caltech-256 dataset. The euclidean as well as the Manhattan distance metric are however more disturbed and show larger deviations. Also, with the FIRE dataset the biggest difference between the cosine and modified distance metric lies within the distance calculation time.

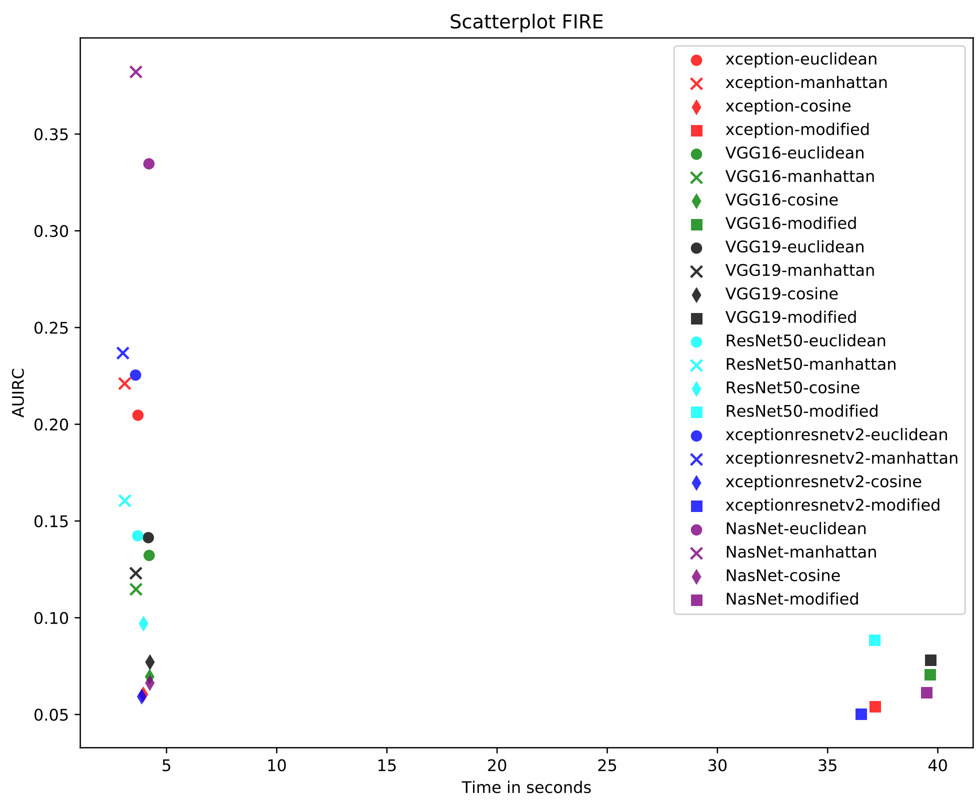

FIGURE 4.12: Scatterplot of the average AUIRC with the FIRE dataset using 15 classes.

Inverse recall at q

The boxplots of the inverse recall at q: 0.8 are depicted in figure 4.13. The inverse recall at 0.8 is depicted on the y axis, indicating the fraction of the amount of images one need to go through to have found 80 percent of the images belonging to the reference class. The x axis depicts the CNNs and the colours indicate the distance metrics used. When looking at the IQR of the boxplots, one can see that especially the cosine and modified distance metrics of Xception and the modified metric of InceptionResNetV2 and NasNetLarge show very similar results. Also, the maximums of those boxplots are very similar. The outliers when using InceptionResNetV2 or NasNetLarge using the modified distance metric are also fairly similar.

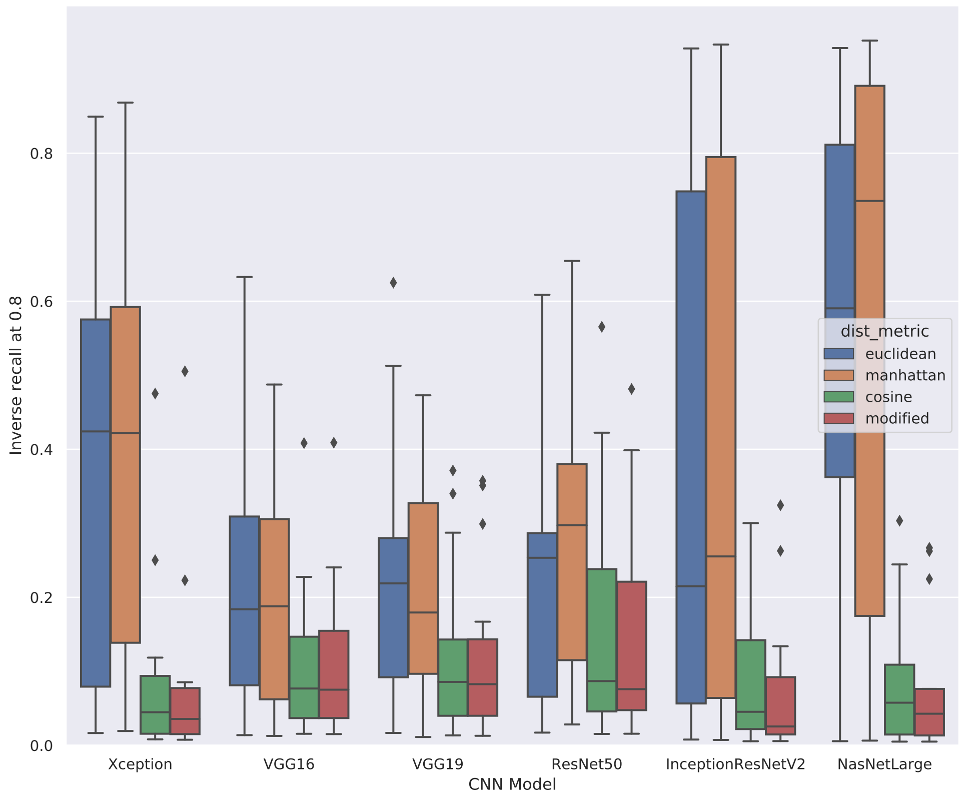

FIGURE 4.13: Results for the inverse recall at 0.8 with the FIRE dataset using 15 classes.

Precision at k

The boxplots of the precision at k: 100 are depicted in figure 4.14. The precision at 100 is depicted on the y axis, indicating the fraction of the amount of samples within those 100 images belonging to the reference class. The x axis depicts the CNNs and the colours indicate the distance metrics used. When looking at the IQR of the boxplots, one can see that they are all also very spread out as is the case with the Caltech-256 dataset, making it very inaccurate. We have to note in this case however that only the results of 15 classes are used, making it a less reliable observation. Also, the minimum and maximum values of the boxplots are very spread out in all cases. Not much can be said about this figure, apart from the observation that InceptionResNetV2 as well as NasNetLarge seem to achieve the highest precision in combination with the cosine and modified distance metric. The possible reasoning behind these spread out results will be discussed in section 5.1.1.

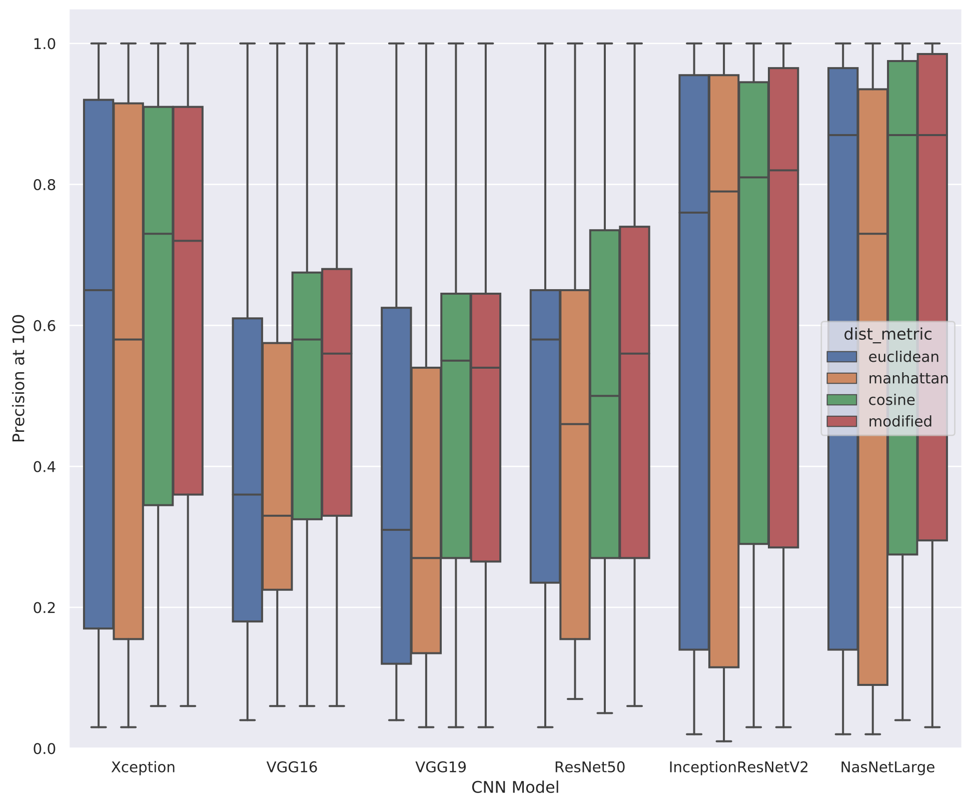

FIGURE 4.14: Results for the precision at 100 with the FIRE dataset using 15 classes.

4.2.2 Time measurements

Model build time

The build time of the different CNNs is depicted in figure 4.15. The y axis yields the time in seconds and the x axis yield the different models. In total 24 runs are done per model to give a clear indication of the average build time needed per model. The standard deviation is included. It can be seen that the standard deviations are all fairly similar in terms of spreading. ResNet50 seems to have the largest standard deviation. Furthermore, the build time of the Xception, VGG16, VGG19 and ResNet50 models are all very similar followed by InceptionResNetV2. NasNetLarge is considered to be slowest in terms of model build time.

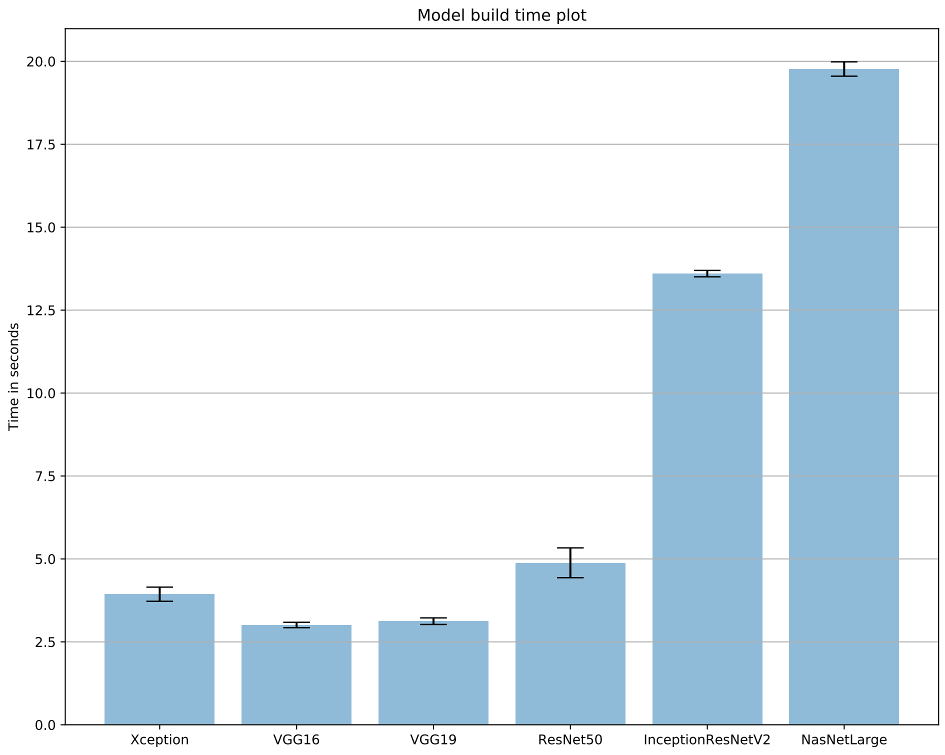

FIGURE 4.15: Build time of the different CNNs when running with the FIRE dataset.

Features extraction time

The feature extraction time per 10.000 images of the different CNNs are depicted in figure 4.16. The y axis yields the time in seconds and the x axis yield the different models. In total 24 runs are performed per model to give a clear indication of time needed to extract the features. The standard deviations are included. The InceptionResNetV2 and NasNetLarge model seems to be the most consistent in terms of feature extraction time based on the standard deviations. Only the NasNetLarge model really stands out in terms of feature extraction time, the other models all show near similar feature extraction times.

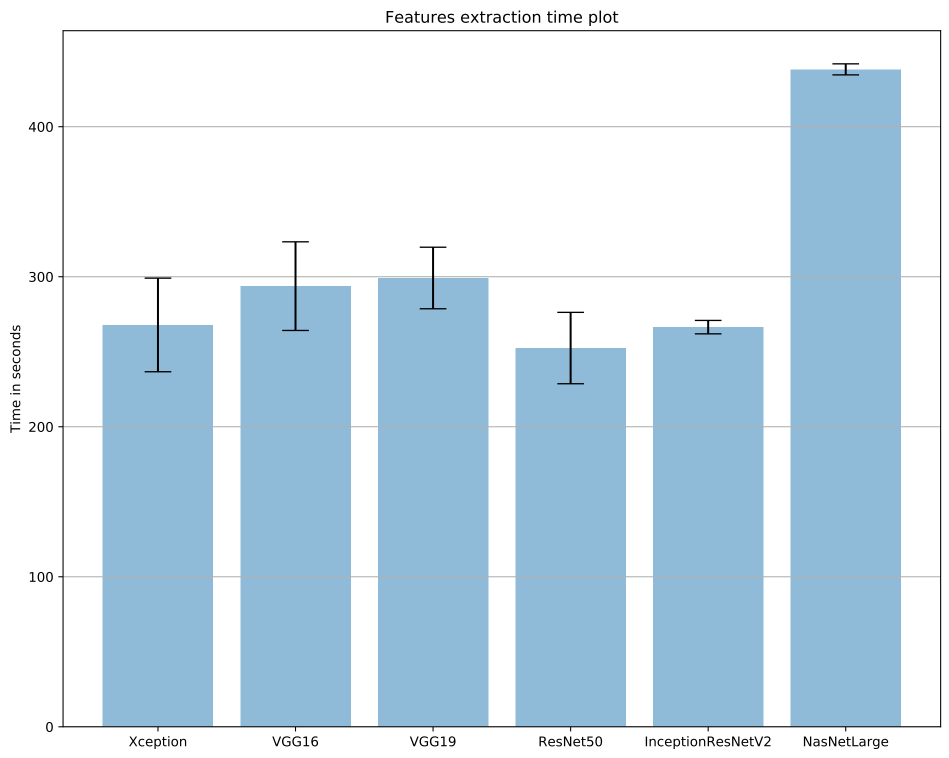

FIGURE 4.16: Feature extraction time per 10.000 images using the FIRE dataset.

Distance calculation time

The distance calculation time of the different CNNs are depicted in figure 4.17. The y axis yields the time in seconds and the x axis yield the different models. In total 28 runs are performed per model to give a clear indication of time needed to calculate the distances between the images in the dataset to the reference image. The standard deviations are included. It can be seen that the time needed to calculate the distances are very similar for the euclidean, Manhattan and the cosine distance metric with a very similar standard deviation. The distance calculation time of the modified distance metric is once again way slower. Also, the standard deviation is considered to be a bit larger, making it a bit less consistent.

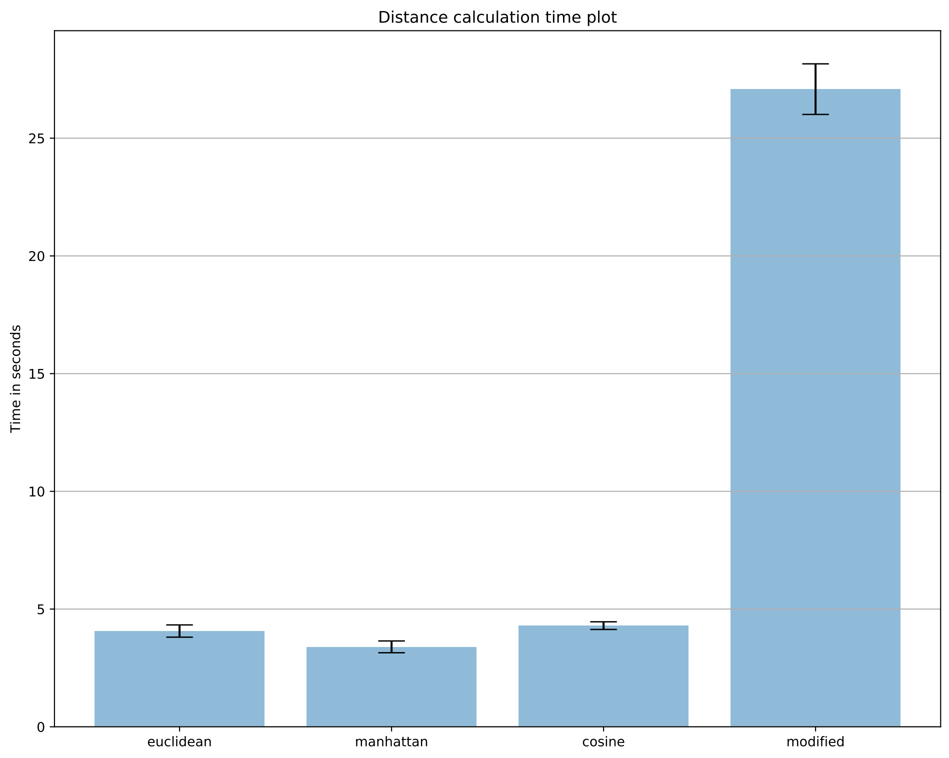

FIGURE 4.17: Distance calculation time of the different distance metrics with the FIRE dataset.

4.2.3 Best classes

Results from 5 of the best classes using the FIRE-Training dataset are depicted in this section. Per best class the prediction and inverse recall values are stated as well as the CNN distance metric combination used to achieve those results. More combinations could be stated when they yield the same results. In the case of the inverse recall at 0.8, a range of 0.003 starting from the best inverse recall value is used to give more insight.

Precision at k

Class	Precision at 100	Model	Metric
firearms	1.00	Xception NasNetLarge	Euclidean, Manhattan, Modified Euclidean, Manhattan, Cosine, Modified
pornography	1.00	Xception VGG16 VGG19 ResNet50 InceptionResNetV2	Euclidean, Cosine Euclidean, Manhattan, Cosine, Modified Euclidean, Manhattan, Cosine, Modified Euclidean, Manhattan, Cosine, Modified Euclidean, Manhattan, Cosine, Modified
shipping containers	1.00	Xception VGG19 Resnet50 InceptionResNetV2 NasNetLarge	Euclidean, Manhattan, Cosine, Modified Cosine, Modified Euclidean, Cosine, Modified Cosine, Modified Euclidean, Manhattan, Cosine, Modified
knife	0.99	NasNetLarge	Euclidean, Manhattan, Cosine, Modified
military vehicles	0.99	NasNetLarge	Euclidean, Manhattan, Cosine, Modified

TABLE 4.1: 5 best classes of the FIRE dataset based on precision at 100.

Inverse Recall at q

Class	Inverse Recall at 0.8	Model	Metric
military vehicles	0.0050-0.0073	InceptionResNetV2 NasNetLarge	Manhattan, Cosine, Modified Euclidean, Manhattan, Cosine, Modified
shipping containers	0.0074-0.0079	NasNetLarge	Euclidean, Manhattan, Cosine, Modified
firearms	0.0076-0.0079	Xception InceptionResNetV2 NasNetLarge	Modified Modified Modified
screens	0.0084-0.0112	VGG19 InceptionResNetV2 NasNetLarge	Manhattan Cosine, Modified Cosine, Modified
bank cards	0.0096-0.0107	Xception InceptionResNetV2	Cosine, Modified Cosine, Modified

TABLE 4.2: 5 best classes of the FIRE dataset based on inverse recall at 0.8.

4.2.4 Untrained classes

Results of 5 untrained classes using the FIRE dataset are depicted in this section. Per untrained class the prediction and inverse recall values are stated as well as the CNN distance metric combination used to achieve those results. More combinations are stated when they yield the same results. In the case of the inverse recall at 0.8, a range of 0.003 starting from the best inverse recall value is used to give more insight.

Precision at k

Class	Precision at 100	Model	Metric
pornography	1.00	Xception	Euclidean, Manhattan, Modified
		NasNetLarge	Euclidean, Manhattan, Cosine, Modified
cannabis	0.96	InceptionResNetV2	Modified
bank cards	0.95	InceptionResNetV2	Cosine
screens	0.88	NasNetLarge	Manhattan
IS flags	0.26	InceptionResNetV2	Cosine

TABLE 4.3: 5 untrained classes of the FIRE dataset based on Precision at 100.

Inverse Recall at q

Class	Inverse Recall at 0.8	Model	Metric
pornography	0.0391-0.0580	Xception	Cosine, Modified
		VGG16	Euclidean, Manhattan, Cosine, Modified
		VGG19	Manhattan, Cosine, Modified
		ResNet50	Cosine, Modified
		InceptionResNetV2	Manhattan, Cosine, Modified
cannabis	0.0255-0.0576	InceptionResNetV2	Cosine, Modified
		NasNetLarge	Cosine, Modified
bank cards	0.0096-0.0282	Xception	Cosine, Modified
		VGG16	Cosine, Modified
		VGG19	Cosine, Modified
		ResNet50	Cosine, Modified
		InceptionResNetV2	Cosine, Modified
screens	0.0083-0.0282	Xception	Euclidean, Manhattan, Cosine, Modified
		VGG16	Euclidean, Manhattan, Cosine, Modified
		VGG19	Euclidean, Manhattan, Cosine, Modified
		ResNet50	Euclidean, Manhattan, Cosine, Modified
		InceptionResNetV2	Euclidean, Manhattan, Cosine, Modified
		NasNetLarge	Cosine, Modified
IS flags	0.0285-0.0580	Xception	Cosine, Modified
		InceptionResNetV2	Cosine, Modified
		NasNetLarge	Cosine, Modified

TABLE 4.4: 5 untrained classes of the FIRE dataset based on Inverse Recall at 0.8.

5 Discussion

In this chapter the results of the research will be discussed and explained, as well as their limitations.

5.1 Similarity Performance

When comparing the similarity performances of the models in most cases the NasNetLarge model and the InceptionResNetV2 model perform best. In the case of image classification, those models also outperformed other models. The InceptionResNetV2 and the NasNetLarge models are models that achieve some of the highest results in terms of accuracy when validated on the ImageNet Validation set [14]. We are not working with image classification, but since only the classification layers are removed, the resulting ranked list could still show some similarity when comparing distances between images to image classification.

When using the cosine and modified distance metric, it seems that the influence of the CNN being used is less significant than when using the euclidean or Manhattan distance metric as also made visual in the AUIRC scatterplot in figure 4.2 and figure 4.12. Perhaps this could be because the cosine and modified distance metric both show less deviations in terms of similarity performance in general for every model, making the overall results more accurate than when using the euclidean or Manhattan distance metric and thus showing less deviation per model.

5.1.1 Similarity Limitations

One of the biggest limitations is the precision calculation in all scenarios. In both datasets used there are classes that are very related making the resulting score of the precision often way lower than it actually should be. As an example there is a class in the Caltech-256 dataset that contains frogs, and a class that contains toads. What happens in this case is that when calculating the precision at 100 on the frogs class, the frogs and toads are so similar, that they are mixed up in the resulting ranked list, making the prediction of the model very poor if all toads are on top of the list. In reality this could still be counted as very similar objects. This described scenario is visualised in figure 5.1 in which the first value indicates the position of the image in the ranked list and the second value of the image name indicates the class number. In this case the reference image is depicted on position 1. This scenario is also the case when making use of the FIRE dataset, as in that case there are for example the cannabis and cannabis plantation classes, as well as the narcotics and narcotics packaging classes that are sometimes hard to distinguish. In this scenario they would both be interesting for the investigators.

Another limitation is that there are a lot of images within the Caltech-256 that have multiple objects on them but only belong to one class. What could happen in this case is the models detect the object of interest on the image, but the image is flagged as being part of another class as that object is also visible on the image. This will also cause the precision estimation to drop and the inverse recall value to rise.

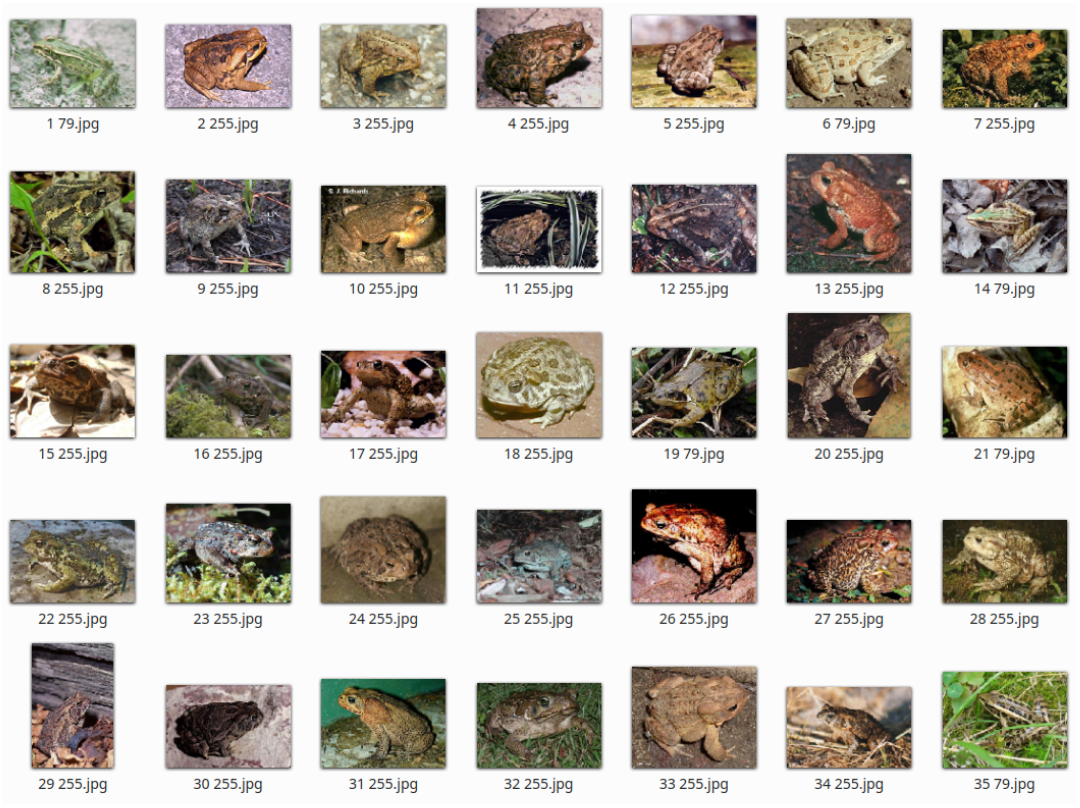

FIGURE 5.1: InceptionResNetV2 output using the cosine distance metric with frogs (79) and toads (255).

5.2 Time Measurements

Based on model build time and feature extraction time, the VGG16, VGG19 and ResNet50 model seem to belong to the fastest models. When results need to be found quickly and the features of the datasets have not been extracted already, those models would be most suitable to be used. This could cost a bit in terms of similarity performance, as there are models that have a better similarity performance that take longer to execute. InceptionResNetV2 and NasNetLarge are examples of such models of which the latter is the slowest model.

An explanation for the differences in time probably lies with the way the models are structured. Especially a combination of many parameters and a lot of layers seem to slow the feature extraction process, which has the highest influence on the total execution time.

The different distance calculation methods also influence the execution time, but do not weigh against the measured feature extraction times. In most use-cases, this is however the most important time, as the features are often already extracted from the model. The modified distance metric is considered to be the slowest, while the other three distance metrics seem to perform almost similar in terms of speed. An explanation for this probably lies with the complexity of the modified distance metric formula, making the computation of the distances more time consuming.

5.2.1 Time Limitations

A computer should not be doing a lot of other work when measuring times to have the most accurate time measurements. It is tried to mitigate other computer activity as much as possible when calculating these measurements, but it could have happened that there was some disturbance from time to time. This scenario is however mitigated as much as possible by running the program multiple times to have a more accurate estimation of the results.

The time measurements of the FIRE dataset have been done on another machine as I did not have the right permissions to have access to the dataset myself. This machine might have been slightly more powerful than my own machine. This however does not influence the distance and model comparisons, as they are compared per dataset, meaning that per dataset the device used for the time measurements has been kept consistent.

6 Conclusion

This section describes the conclusion of the research. First we will give an answer to question 3 by answering the 4 underlying sub-questions. After that we will combine the results of question 1 and 2 derived from the background literature together with the answer of question 3 to give answer to the main research question.

Question 3 is stated as follows:

How will different parameters affect the performance of the CNN distance metric combinations?

To give a clear answer to this question, the 4 underlying sub-questions will be answered.

SQ 1: What is the impact of the type of CNN used on the similarity ranking performance, the model build time and the feature extraction time?

When analysing the obtained results of the similarity performances, it can be clearly seen that the used CNN influences the obtained similarity performance. It seems that this influence is higher depending on the distance metric being used. When looking for example at the different CNNs using the Manhattan distance metric, you can see that this influence per model is way higher as well as the standard deviation than when comparing the models using the cosine distance metric.

When looking at the build time, it can be clearly seen that the CNN influences the time needed to build the model. Xception, VGG16, VGG19 and ResNet50 are all performing relatively similar in terms of build time, but the InceptionResNetV2 and NasNetLarge models have a significantly larger build time. In terms of feature extraction time, especially the NasNetLarge network shows a way longer feature extraction time in comparison to the other networks. VGG16, VGG19 and ResNet50 show very similar feature extraction times, followed by Xception and InceptionResNetV2.

Based on these observations and assuming that the distance metric that seems to perform best will be used with the CNNs, the InceptionResNetV2 model would probably be the best option as in terms of similarity performance it performs in general better or almost as good as the other models in terms of AUIRC, inverse recall and precision. In terms of time measurements it is not the best model to use, but considering the better similarity performance results compared to other models probably most suitable depending on the situation.

SQ 2: What is the impact of the type of distance metric used on the similarity ranking performance and the distance calculation time?

When keeping the used CNN consistent and changing the distance metrics being used, we can clearly see that the distance metric being used has a big impact on the performances of the models. In almost all scenarios and with every CNN using the cosine or modified distance metric yields better results in terms of similarity performance than using the euclidean or Manhattan distance metric. This makes the cosine or modified distance metric most interesting, similarity performance wise, when working when working with image similarity ranking using distance metrics.

In the case of the distance calculation time, especially the euclidean, Manhattan and cosine distance metric show near similar results. Being almost equally as fast. The modified distance metric shows less promising results. Taking these observations into account, one could say that the impact of the distance metric being used is very high and that it is best to use the cosine distance metric when working with reverse image ranking based on a combination of similarity ranking performance and distance calculation time.

SQ 3: What is the impact on the similarity ranking performance, when using classes the CNNs are not trained on?

When looking at the obtained results from the FIRE dataset, we can see that untrained objects are in some cases a bit harder for the models, but in most cases still useful especially in terms of inverse recall. The screens class as well as the bank cards class are also considered to be part of the best performing classes indicating that untrained classes could still perform well on existing models not knowing these classes. The IS flags class which performs the worst, has a precision of 0.26 meaning that 26 out of the first 100 images are considered to be an IS flag. In terms of inverse recall at 0.8, 3000 images need to be looked at to find 80 percent of the IS flags in the best scenario instead of looking at more than 100.000 images which saves an considerable amount of time. Overall it would still seem useful to use an image similarity ranking algorithm when working with untrained classes as it would still save a considerable amount of lookup time based on the obtained results.

SQ 4: What is the influence of using multiple reference images as input on the similarity ranking performance?

It turns out that there is no real difference between using 1 or multiple reference images based on the obtained results from using the InceptionResNetV2 model together with the cosine distance metric. The overall similarity ranking results are nearly similar in all cases when looking at the AUIRC, the inverse recall at 0.5 and the precision at 50. When time is an important factor one could thus better use 1 reference image to save time, as it does not seem to make a lot of difference compared to using multiple reference images.

The main research question of this thesis is:

How could image similarity ranking be applied within the forensic domain, to support investigators in ongoing investigations?

To answer this question a couple of things need to be taken into consideration. When the focus only lies with similarity performance and execution time is no issue or the features already have been extracted from the models, the best option would be to use the InceptionResNetV2 or NasNetLarge model together with the cosine or modified distance metric as these models yield in general the best results. They also seem to perform well in general on object categories unknown to the models.

Whenever there is not much time to gather results and time is very limited, it would be best to ignore the NasNetLarge model as its feature extraction time is significantly slower than that of the other models and this is the most time consuming factor. It would then be a better option to go with either the Xception model as this still shows promising similarity results or with the InceptionResNetV2 model when there is a bit more time available.

When build time and feature extraction time is not taken into account and the only time dependent factor is the distance calculation time, the best metric to use is the cosine distance metric, as it is significantly faster than the modified distance metric and outperforms the euclidean and Manhattan distance metric.

6.1 Future Work

There is still a lot to explore for further investigation.

- More CNNs as well as distance metrics could be tested for improvements.
- Feature vectors from more shallow layers could be tried to see the effects on the similarity performance.
- More datasets could be used to give more insight into the similarity performance. For now only two datasets are being used. Perhaps adding more datasets give more insight into for example the precision of the models.
- Perhaps alternatives to precision at k and inverse recall at q could be used to determine the similarity performance of a model. As already stated in the discussion, the results look sometimes a lot worse than they actually are due to classes being near similar or images having several different objects that could in theory belong to multiple classes but are only labeled as one class.
- More insights should be given on the performance of the models on classes the models are not trained on.
- It would also be interesting to see what the effects are on the models when working with image augmentation. Do the models perform differently when reference images are for example rotated or grey scaled.
- When analysing the influence of using multiple reference images we have taken the distance output lists for single reference images and added the distances together to have a ranked image list based on multiple reference images. An interesting thing for future work is to combine the input features of the reference images and compare that to the images in the dataset instead of combining the output lists. When this turns out to be effective, this could save computation time as only the reference image features need to be combined and the distances between the other images of the dataset will be only calculated once instead of once per reference image.

References

[1] Nura Aljaafari. “Ichthyoplankton Classification Tool using Generative Adversarial Networks and Transfer Learning”. PhD thesis. 2018.

[2] Flavio H.D. Araujo et al. “Reverse image search for scientific data within and beyond the visible spectrum”. In: *Expert Systems With Applications* 109 (2018), pp. 35–48.

[3] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. “LSH forest: self-tuning indexes for similarity search”. In: *Proceedings of the 14th international conference on World Wide Web* 14 (2005), pp. 651–660.

[4] Dylan de Bie. *cnn-distance-research*. https://gitlab.com/d.r.debie/cnn-distance-research.

[5] François Chollet. “Xception: Deep Learning with Depthwise Separable Convolutions”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 1800–1807.

[6] Elasticsearch. *Elasticsearch*. https://www.elastic.co/what-is/elasticsearch. Accessed: 2020-05-28. 2010.

[7] Google. *Google image search*. https://images.google.com/. Accessed: 2020-04-08. 2001.

[8] G Griffin, AD Holub, and P Perona. *The Caltech 256*. http://www.vision.caltech.edu/Image_Datasets/Caltech256/. Accessed: 2020-06-15.

[9] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.

[10] ILSVRC. *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*. http://www.image-net.org/challenges/LSVRC/. Accessed: 2020-04-15. 2010.

[11] Sergei Evgenievich Ivanov, Nataliya Gorlushkina, and Anton Govorov. “The Recognition and Classification of Objects Based on the Modified Distance Metric”. In: *Procedia Computer Science* 136 (2018), pp. 210–217.

[12] Jeff Johnson, Matthijs Douze, and Hervé Jégou. “Billionscale similarity search with GPUs”. In: *IEEE Transactions on Big Data* (2017), pp. 1–12.

[13] Ujjwal Karn. *An Intuitive Explanation of Convolutional Neural Networks*. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. Accessed: 2020-11-05.

[14] *Keras Applications*. https://keras.io/api/applications/. Accessed: 2020-08-20.

[15] Daniel Kobran and David Banys. *Activation Function*. https://docs.paperspace.com/machine-learning/wiki/activation-function. Accessed: 2020-11-05.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems* 25 (2012), pp. 1097–1105.

[17] Yann LeCun et al. “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86 (1998), pp. 2278–2324.

[18] *Measuring Distance or Similarity*. https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781785882104/6/ch06lvl1sec40/measuring-distance-or-similarity. Accessed: 2020-10-30.

[19] *Principles of Deep Learning*. https://principlesofdeeplearning.com/index.php/is-deep-learning-getting-too-deep/. Accessed: 2020-10-28.

[20] Rocco Schulz. *Building a Reverse Image Search with Elasticsearch*. https://blog.mimacom.com/elastic-img-similarity-search/. Accessed: 2020-05-28. 2020.

[21] Lavanya Sharan, Ruth Rosenholtz, and Edward H. Adelson. "Accuracy and speed of material categorization in real-world images". In: *Journal of Vision* 14 (2014).

[22] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks For Large-Scale Image Recognition". In: *ICLR 2015* (2014), pp. 1–14.

[23] *SQLite3*. https://docs.python.org/2/library/sqlite3.html. Accessed: 2020-08-20.

[24] Christian Szegedy et al. "Going Deeper with Convolutions". In: *Computer Vision and Pattern Recognition* (2015), pp. 1–9.

[25] Christian Szegedy et al. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *AAAI*. 2017.

[26] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 2818–2826.

[27] *Tensorflow*. https://www.tensorflow.org/install. Accessed: 2020-08-20.

[28] Tineye. *TinEye Reverse Image Search*. https://tineye.com/. Accessed: 2020-04-08. 2008.

[29] Svante Wold, Kim Esbensen, and Paul Geladi. "Principal component analysis". In: *Chemometrics and Intelligent Laboratory Systems* 2 (1987), pp. 37–52.

[30] yrevar. *imagenet1000*. https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a. Accessed: 2020-08-15.

[31] M.D Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks". In: *Computer Vision ECCV 2014* 8689 (2014), pp. 818–833.

[32] Barret Zoph et al. "Learning Transferable Architectures for Scalable Image Recognition". In: *CoRR* (2017).