Visualizing Important Areas for Facial Verification
A Deep Learning Evaluation with Artificial Noise Injection

Andreas Herman
Abstract

A Deep Learning Evaluation with Artificial Noise Injection

Andreas Herman

Facial verification is a growing area of research, where heavily trained neural network models obtain superhuman performance on recognizing individuals. However, because of network models’ complex nature, it is hard to know what areas of the image are used. This thesis introduces three visualization techniques, to better understand the importance of different facial areas when performing verification. A method for measuring the visualizations’ hierarchical correctness was established, evaluating the model’s ability to preserve and damage the verification performance, when treated as artificial pixel noise. This is done for two types of networks, which are different in their architecture and their input dimensionality (one taking full color and the other taking grayscale images).
Acknowledgments

I would like to thank Krishnan Shibu and Stephane Herman for kindly agreeing to review this thesis, despite not being experts in the field of computer science. They provided constructive feedback with grammatical and structural improvements.
# Contents

Abstract iii
Acknowledgments iv
Contents v
List of Figures vii
List of Tables vii

1 Introduction 2
1.1 Motivation 3
1.2 Aim 4
1.3 Research Questions 4
1.4 Delimitations 4
1.5 Related Work 5

2 Theory 6
2.1 Deep Learning 6
2.2 Regularization in Neural Networks 11
2.3 Convolutional Networks 14
2.4 Face Recognition 16
2.5 Saliency Map 18

3 Method 20
3.1 Environment 20
3.2 Datasets 20
3.3 Evaluation of Verification 23
3.4 DAR Pipeline 23
3.5 General Training Procedure 24
3.6 Face-ResNet 25
3.7 LightenedCNN C 26
3.8 Visualization Techniques 27

4 Results 31
4.1 Model Performances 31
4.2 Artificial Noise 32

5 Discussion 36
5.1 Results 36
5.2 Method 37
5.3 Wider Context 39

6 Conclusion 40
List of Figures

1.1 Google Ngram on Facial Recognition. .................................................. 3
1.2 Deep Convolutional Neural Network. .................................................... 3

2.1 Plot of hyperbolic tangent, sigmoid and linear rectifier. ......................... 8
2.2 Plot of hard tanh and absolute value rectifier. ..................................... 9
2.3 Example of convolution with kernel of size 3x3 applied on input of size 5x5. 15
2.4 Example of image-specific class saliency maps. .................................. 18

3.1 MS-Celeb-1M dataset image distribution over identities. ........................ 21
3.2 Verification accuracy for models with different noise thresholds evaluated on LFW View 1. A curve was fitted to the data using polynomial regression (polynomial of degree 2). ................................................................. 22
3.3 Detection-Alignment-Recognition (DAR) pipeline. .............................. 23
3.4 RConv - Complex layer ................................................................. 26
3.5 Activation maps using the images’ activations as gradients. Created for Face-ResNet. ................................................................. 29
3.6 Space maps by combining 25,600 random encoding maps. Created for Face-ResNet. ................................................................. 30

4.1 ROC curves of the models used in this work compared to human performance. . 32
4.2 Images subjected to 50% insignificant pixel noise using the 2M trained Face-ResNet-model, presented as black. .................................................. 33
4.3 Activation map visualizations on images exposed to different types of insignificant noise, using Face-ResNet. .................................................. 34
4.4 Face-ResNet incrementally exposed to artificial pixel noise. .................... 35
4.5 LightendCNN C incrementally exposed to artificial pixel noise. ............... 35
4.6 Difference between insignificant and significant noise. ........................ 35
List of Tables

3.1 Face-ResNet Architecture ................................................. 25
3.2 Face-ResNet Trainings ......................................................... 26
3.3 LightenedCNN C Architecture ............................................. 27
3.4 LightenedCNN C Trainings ..................................................... 28
4.1 2M Face-ResNet Model (Best LFW View 1 Accuracy) .................... 31
4.2 2M Face-ResNet Model (Best MS-Celeb-1M Loss) ......................... 31
4.3 2M B LightendCNN C Model (Best LFW View 1 Accuracy) ............... 31
4.4 2M B LightendCNN C Model (Best MS-Celeb-1M Loss) .................... 32
4.5 Difference Areas ............................................................... 34
Many problems in the domain of computer science are easy for humans to conceptualize, in
the sense that we can manually program a machine to find a solution, such as finding the
shortest path between two points. However, for some problems it is hard to write algorithms
that generalize well to a variety of possible outcomes, such as web search ranking, movie
recommendation, object identification in images or letting an autonomous helicopter learn
how to fly. These problems can be approached by letting machines learn about the problem,
and how to solve it, by themselves.

Machine learning[16, 21] is widely used today in industry and in research. Its interest is
growing rapidly, since computational power is becoming more available and we are possess-
ing large sets of data, in reaction to the growth of the Internet.

Many machine learning techniques are limited in the way of processing data in its raw
form. These techniques requires careful engineering by manually extracting features, which
are to be used to transform the raw data into an internal representation, which is later used
by the learning system.

Representation learning[14] is a set of techniques that automatically learns features from
raw data. Deep learning[20] networks, which are neural networks[16, 21, 20] with many lay-
ers, is a part of this family of methods. It uses multiple layers to represent the data, obtaining
a slightly more abstract representation for each layer. Each layer has its own type of features
as input, which have been constructed by the previous layers (e.g. pixels, edges, shapes), and
are learned from the data using general learning procedures.

Because deep learning requires very little feature engineering, the technique can easily
be applicable in industry, and not only by research specialists. In response to availability of
using the technique, opportunities for deep learning will most likely increase even more, and
affect our society in many new ways.

A particular neural network architecture which has received a lot of recent attention is
convolutional neural networks (CNN)[13]. This architecture is inspired by the organization
of the brain’s visual cortex, where neurons answers to stimuli for restricted regions of space.
CNNs are widely used in image-, video- and sound recognition, and also in natural language
processing.
1.1 Motivation

Facial recognition/verification, where the main goal is to identify whether a pair of images present the same person, is a growing research area. Heavily trained neural network models obtain superhuman performance on recognizing individuals and company giants like Google, Facebook and Microsoft conduct large scale research in this area. Google’s FaceNet was trained on 200M images (8M unique identities)[24] to obtain almost perfect performance, and neural networks seem to constantly astound with its capabilities of handling large data sets.

Despite that neural networks are powerful models, there are several potential drawbacks attached to it. One being harder to bypass. The drawback of being computationally heavy can be solved by obtaining specialized hardware. The drawback of being an overparameterized model[21] can be solved by introducing sparsity in the network’s architecture or obtaining more data[20]. However, the drawback of being an extremely complex model, viewed by many as a black box, is something that is hard to circumvent. The concept of how the model works in itself is nothing strange, but how the model makes its decisions with respect to the data is arguable. The reason lies in the model’s nature, by sequentially transforming data into new feature spaces, where the concept of what makes sense in connection to the input is lost on the way. This also applies to the domain of facial verification, where convolutional neural networks (CNN) learn how to best distinguish different individuals’ faces. Here an uncertainty lies in what areas of the face are used for prediction.

A possible way to determine this could be to try all possible input combinations and observe when the model performs well, and use this to establish facial area dependencies. However, using the Face-ResNet network[10], it would involve approximately $10^{77680}$ possible attempts. For comparison, the number of atoms in the Universe is estimated to be between $10^{78}$ and $10^{82}$.

Figure 1.1: Google Ngram on Facial Recognition.

Figure 1.2: Deep Convolutional Neural Network[17].
An unraveling technique for CNNs is to construct a saliency map for a given class, which weakly localizes the representation of the class in image space[25]. This technique should also be relevant for the purpose of facial verification. CNNs, as shown in figure 1.2, consists of several layers of units which the data is propagated through. When performing facial verification, the last hidden layer of a CNN (a layer in between the input and the output layer), trained to classify different individuals, is used as an encoding of the face. When comparing two facial images, the Euclidean distance between the encodings determines the outcome of predicting same or different identities. By visualizing encodings, instead of classes, it should be possible to localize image areas that are primarily used when performing face verification (since some areas are more discriminant than others[25]). However, it may be the case that these areas are not independently important, but were highlighted because of their conjunction with other facial areas.

1.2 Aim

This work will investigate ways to visualize facial important areas when performing verification. This will be approached by first training CNN models to obtain good prediction performance on the facial verification task. Later, different techniques for visualizing image-important areas will be created. These techniques will be evaluated by a method, created for this work, which captures the visualizations’ correctness in terms of representing the model’s hierarchical dependencies in the image. Lastly, a method will be created to evaluate how much of the image the model actually uses. This will in turn make it possible to emphasize how robust the model is to this type of artificial noise, of removing targeted pixels.

1.3 Research Questions

This work will first investigate various ways of visualizing facial areas that are important to the model.

Q1: How can important areas for performing facial verification be visualized in the image?

A method will be established to compare the visualization techniques in terms of hierarchical correctness. This method involves removal/preservation of high/low pixels in the visualizations, further denoted as exposure to artificial pixel noise. A pixel’s effect on the verification performance will make it possible to determine its importance to the model.

Q2: How is the facial verification performance affected by exposure to artificial pixel noise?

Two networks with different input dimensionality will be evaluated in terms of robustness, on their ability to handle artificial noise, to investigate how the models make their predictions. This in turn also investigates how much of the image the model actually uses.

Q3: How well does a 1-input channel model handle artificial pixel noise compared to a 3-input channel model?

1.4 Delimitations

This work will not focus on trying to train the best performing models given the data. It will instead create reasonable good models, (comparable with state-of-the-art) and use them for conducting the experiments. Network architectures will be selected from related work, with availability in Caffe framework[11] as a criteria. The networks will not be selected by conducting an empirical study, but simply chosen by the papers’ published results, in conjunction with suitability for this work.
A lot of studies have been done which employ deep neural networks for facial recognition/verification.

Schroff et al\cite{24} proposed a triplet-based loss function for tuning CNNs to perform better in face recognition context. However, these models were trained on a much larger data set than is available for this work (200M images, 8M identities) and obtained a verification accuracy of 99.63% on the LFW data set\cite{2}.

Parkhi et al\cite{22} proposed a procedure to create a reasonably large data set (2.6M images, 2,622 identities) and used triplet loss (different network architectures than Schroff et al) on this data set to obtain a verification accuracy of 98.95% on LFW. The training procedure roughly corresponds to the work of Schroff et al, with exception of pretraining before using triplet loss.

Wen et al\cite{9} proposed the center loss function, which makes use of class centroids to help supervise the training of CNNs. Compared to training with triplet loss, where mini-batches of triplets are carefully selected to increase the discriminative power of the learned features, this approach does not involve careful selection of examples, and is therefore more computationally efficient. They obtained a verification accuracy of 99.28% on LFW using a small dataset (under 500K images and 20k identities). Center loss function will also be used in this work.

Taigman et al\cite{12} made use of twin-like Siamese networks, training on a Facebook collected dataset (4.4M images, 4,030 identities), to obtain 97.0% verification accuracy on LFW. They also combined several models into ensembles, giving the distances as input to a support vector machine (SVM)\cite{16}, improving the performance to 97.35% accuracy.

When visualizing image discriminant areas for CNNs, Simonyan et al\cite{25} proposed a way of localizing relevant areas for certain classes. These areas were further propagated in the image by color continuities to create a saliency map covering the entire object.

Dodge et al\cite{18} evaluated how CNNs are affected by image quality. They showed that their models are susceptible to different kinds of image distortions, especially blur and noise. The noise was applied by targeting each pixel’s color component, by adding a Gaussian random value. The noise of this work will however be more directed and carefully selected, replacing the target pixels with uniform random color values.
The theory section covers both the fundamentals for understanding and working with CNNs for different applications, but also the necessary parts for conducting the experiment. This includes facial verification using CNNs, and visualization of class-discriminant areas in the image.

2.1 Deep Learning

Deep learning networks are neural networks with a deep architecture of layers, and is the quintessential for deep learning models, where the goal is to approximate some function $f^*$[20].

Feed-forward Networks

Feed-forward networks[16] can be described as series of functional transformations. Each basis function is a nonlinear function of a linear combination of some input $x$. The nonlinearity comes from an activation function, which is given the input linearly combined with the coefficients (adaptive parameters, which are to be learned). In neural network context, these coefficients are more commonly referred to as weights.

When applying the transformations, $M$ linear combinations of the input variables $x_1, ..., x_D$ are first constructed. This is known as the activation of unit $j$, and is calculated by

$$a_j = \sum_{i=1}^{D} w_j^{(1)} x_i + w_j^{(1)}$$  \hspace{1cm} (2.1)

where $j = 1, ..., M$ is a unit in that layer. The superscript indicates which layer the unit is in (in this case the first layer). Note that the bias unit with its weight $w_j^{(1)}$ is added separately. A more computationally convenient form would be to introduce a variable $x_0=1$ and simply handle it as an input variable (then the activation can be calculated as a single matrix multiplication)[16].

The activation can now be transformed using a nonlinear activation function, to obtain the output of the unit. The calculation is done by

$$z_j = h(a_j)$$  \hspace{1cm} (2.2)
where \( h \) is some nonlinear activation function (described below). The \( z_j \) values are then used as input to the next layer, which again introduces its own bias unit[16]. The generalized equation for this is

\[
a_j = \sum_{i=0}^{D} w_{ji} z_{ji} = W_j Z_j^T
\]

where \( z_i \) is the input from the previous layer (or the data variables denoted with \( z_i \)).

This process of calculating the output for each layer in the network, and ending up making prediction on the target variable(s), is called forward-propagation. An error estimate can then be done on the prediction, which represents how well the network performs.

Cost Functions

An important choice when designing a neural network is the choice of cost function (an objective function which is to be minimized). A simple choice of cost function would be using the sum-of-squares

\[
E(w) = \sum_{n=1}^{N} (\hat{y}(x_n, w) - y_n)^2
\]

where \( \{x_n\} \) is the input vectors, \( w \) the weights, \( y_n \) the target and \( \hat{y} \) the prediction function. However, by using a probabilistic interpretation, a more general view of the training can be obtained[16].

In most cases the principle of maximum likelihood is used (cross-entropy between the training data and the predictions). Cross-entropy is equally described as the the negative log-likelihood. The form of this cost function depends on the model, where some terms that do not depend on the model may be discarded. An advantage of deriving the cost function from the maximum likelihood is that it automatically gets determined from the model[20]. The cross-entropy for a binary classification problem with likelihood function

\[
p(\hat{y}|w) = \prod_{n=1}^{N} \hat{y}_n^{y_n} (1 - \hat{y}_n)^{1-y_n}
\]

is given by

\[
E(w) = -\frac{1}{N} \sum_{n=1}^{N} [y_n \ln(\hat{y}_n) + (1 - y_n) \ln(1 - \hat{y}_n)]
\]

where \( \hat{y}_n = \hat{y}(x_n, w) \) is the prediction.

An important property for the cost function is that the gradient is large enough to guide the learning algorithm. Functions that saturate (becomes very flat) do not support this property, since the gradient becomes very small. The negative log-likelihood (cross-entropy) helps to prevent this problem, since it cancels out \( \exp \) for output units that has it in its activation function[20].

It is practically more convenient to maximize the logarithm of the likelihood function, since it is monotonically increasing of its argument[16].

Output Units

Output units and their activation functions should be chosen based on the nature of the target variables. The output activation function is only used to transform the activation, at the final step of the neural network, into the wanted format of the output. The functions listed bellow are three common choices for activation functions[16, 20].

Linear unit: \( \hat{y} = a \)
2.1. Deep Learning

Logistic (Sigmoid) unit: \[ \hat{y} = \sigma(a) = \frac{1}{1 + \exp(-a)} \] \hspace{1cm} (2.8)

Softmax unit: \[ \hat{y} = \text{softmax}(a)_i = \frac{\exp(a_i)}{\sum \exp(a_j)} \] \hspace{1cm} (2.9)

The output units using these functions first performs an affine transformation to obtain \( a_k \), as described earlier. For a regression problem, the identity (2.7) is suitable\[16\]. For binary classification, the logistic sigmoid function (2.8) is suitable, since it gives a Bernoulli distribution with range \((0, 1)\) and has nice gradient properties (ensures strong gradient when the model is uncertain)\[20\]. For a multiclass problem, the softmax function (2.9) is suitable, since it gives a Multinoulli distribution which can represent \( i \) classes (outputs a vector of values ranging \((0, 1)\) and sums to 1, to be a valid probability distribution)\[20\].

**Hidden Units**

There are many possible hidden units available, and new types of units is an active research area. However, there are not many theoretical principles stating which units are preferred. Instead, it is more common to try out several ones and decide which one to use based on its performance on a validation dataset\[20\].

![Figure 2.1: Plot of hyperbolic tangent, sigmoid and linear rectifier.](image)

Any unit can be used as a hidden or output unit. A common choice is using the logistic sigmoid (2.8) or the hyperbolic tangent as activation function.

Hyperbolic tangent unit: \[ z = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \] \hspace{1cm} (2.10)

These two functions do have convenient derivatives, which makes them simple to use in back-propagation\[16\]. However, they saturate across most of their domain (both for high and low values of \( a \)), and are strongly sensitive to input near \( a = 0 \). This can make gradient-based learning difficult\[20\]. The hyperbolic tangent is however generally preferred before a sigmoidal function, since it resembles the identity function more (\( \tanh(0) = 0 \) while \( \sigma(0) = \frac{1}{2} \)).

Another choice is the rectified linear unit. It outputs zero across half its domain, and linear activation on the other half.

Rectified linear unit: \[ z = \text{max} \{0, a\} \] \hspace{1cm} (2.11)
2.1. Deep Learning

The left derivative of \( a = 0 \) is \( z' = 0 \) and the right derivative is \( z' = 1 \). It is practically good to initialize the weight of the bias input to be a small positive value to make the unit initially active[20].

Another version of this is the absolute value rectification. It is commonly used for object recognition, where features that are the same under a polarity reversal may occur. Generalizing the function, it may be written as

\[
\text{Absolute value rectification unit: } z = \max \{0, a\} + \min \{0, a\} \tag{2.12}
\]

Yet another alternative is the hard tanh unit, with the looks of a hyperbolic tangent unit in rectifier form.

\[
\text{Hard tanh: } z = \max \{-1, \min \{1, a\}\} \tag{2.13}
\]

Gradient-Based Optimization

In most cases the function being optimized has several optima. This prohibits the solution of simply finding the global minimum/maximum by solving the equation of the function’s derivative being zero. Instead it could be optimized by iteratively moving stepwise to the opposite direction of the function’s derivative (if minimization is the goal)[20]. A local minimum will then be found, which sometimes is good enough. This process could be repeated with different starting points, and result in different solutions.

The gradient of \( f \) is denoted \( \nabla_x f(x) \) and is a vector containing all partial derivatives. Optimization could then be done by moving in the direction of the negative gradient[16, 20]. Gradient descent then gives the point

\[
x' = x - \epsilon \nabla_x f(x) \tag{2.14}
\]

where \( \epsilon \) is the learning rate (often set to a constant). The algorithm converges when every partial derivative of the gradient is zero (close to zero in practise)[20]. If \( \epsilon \) is too big, the algorithm might miss the minimum (by overstepping it), if it is too small, it takes long time for the algorithm to converge.

Since the cost function is defined with respect to the training set, the entire dataset is required to obtain the gradient. Techniques that use the entire dataset are called batch methods[16].
2.1. Deep Learning

Stochastic Gradient Descent

The nonlinearity of neural networks causes most loss functions to become nonconvex. Therefore, neural networks are usually trained using an iterative gradient-based approach where the cost function is driven to low values[20]. Stochastic gradient descent (SGD) is an extension of batch gradient descent.

Large training sets are important for deep learning, but are computationally expensive. This is because the cost functions often use a sum of per-example loss functions. With \( m \) training examples, the cost of the operation is \( O(m) \). By instead guiding the learning algorithm by an approximated estimation of the gradient, using a minibatch (smaller set of samples) instead of all examples, the time needed becomes less severe. The size of the minibatch \( m' \) is typically one to a few hundred, and is usually held fixed. This time-independence of obtaining the estimated gradient is the main reason SGD is used in many different areas, also outside the context of deep learning[20].

Another advantage with SGD is its ability to escape local minima. The reason is, a local minimum for the entire dataset will generally not be the local minimum for each point individually[16]. Therefore, by sampling the dataset each time the gradient is evaluated, the data subset will differ. For some subsets a local minimum will be present, for others it will not.

Backpropagation Algorithm

The iterative algorithm is a process which involves two steps: (1) evaluation of derivatives for the cost function with respect to the weights, (2) adjustment of weights according to the derivatives[16]. The first step is done using backpropagation, and is an efficient way of computing derivatives by propagating errors backwards in the network. The second step uses gradient descent to update the weights, in order to decrease the error.

Many cost functions, as mentioned above, consist of a sum of errors for each data point in the training set. Then the overall error is

\[
E(w) = \sum_{n=1}^{N} E_n(w)
\]  

(2.15)

The cost function \( E_n(w) \) may for instance be defined by maximum likelihood or the sum-of-squares error.

For simplicity, only one point will be considered in this conceptual description (may represent a stochastic gradient descent with a batch size of 1). The network will also only have a single sigmoidal unit, and the output unit will be linear. The network’s performance will be evaluated using the sum-of-squares.

When the forward-propagation process has computed the output for each layer (and the prediction at the final layer) the error can be evaluated. This is done by deriving \( E_w \) with respect to a weight \( w_{ji} \). \( E_n \) depend on the weight \( w_{ji} \) via the summed input \( a_j \) to unit \( j \), and is obtained via the chain rule for partial derivative

\[
\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}
\]  

(2.16)

By introducing the notation

\[
\delta_j = \frac{\partial E_n}{\partial a_j}
\]  

(2.17)

(often referred to as error) and using (2.3) to write

\[
\frac{\partial a_j}{\partial w_{ji}} = z_i
\]  

(2.18)
(2.16) can then be rewritten as

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_iz_i$$

(2.19)

This shows that the derivative can be obtained by multiplying $\delta$ (at the latter unit) with $z$ (at the prior unit)[16]. To evaluate this, $\delta_j$ needs to be calculated for each hidden unit and the output unit, then apply (2.19).

In this description, $\delta_k$ for the output units are computed as $\delta_k = a_k - y_k = \bar{y}_k - y_k$ (because $h$ is the identity). $\delta$’s for the hidden units is calculated by

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

(2.20)

which sums over all units that unit $j$ outputs to. A unit $k$ can also be included by other units. This can be rewritten as

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

(2.21)

using (2.2), (2.3) and (2.17). Here it is assumed that every hidden unit in the network has the same activation function. The derivation can however be generalized to allow different units[16]. It can also further be applied to a cost function which considers several data points by

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}}$$

(2.22)

Starting Values

There are several issues that have to be considered when training a neural network. The issues are due to the model being generally overparameterized, and the optimization problem is nonconvex and unstable, unless certain guidelines are followed[21].

Initializing weights to zero $w_i = 0$ when beginning training of a neural network is strongly discouraged. This is because the derivatives will be zeroes, making the algorithm unable to move (perfect symmetry). Instead the weights should initialized to random values near zeros. Then the model will start as almost linear and become more nonlinear as it is trained. Usually the weights are drawn from a Gaussian or a Uniform distribution[20].

Another approach is to follow the Xavier scheme[19]. It initializes the weights with respect to input and output units. For Caffe, the variance of the distribution using Xavier is $\text{Var}(W) = 1/n_{in}$, where $n_{in}$ is the number of input units. In the paper by Glorot et al[19], it is described as $\text{Var}(W) = 2/(n_{in} + n_{out})$, which also takes the units that the result is fed to into account. The initialization aims to assign the weights so the variance of the input and the output are the same (which is the case assuming the input and weights are independent and identically distributed).

2.2 Regularization in Neural Networks

Neural networks often tend to overfit to the training data because of their massive number of weights[21]. Strategies designed to prevent this is known as regularization techniques[20]. Most strategies are based on regularizing estimators which causes variance to be reduced at the cost of increased bias. This often involves reducing the capacity of the model in order to reduce the generalization error.

For very large datasets, regularization usually confers little reduction in the generalization error, or when the number of classes are small. However, neural networks tend to perform better when having a bigger capacity while being restricted with regularization[20].
Parameter Penalties

This strategy puts restrictions on the weights by inducing a penalty for large values. This is done by adding a penalty norm $\Omega(\theta)$ to the objective function $E$. The regularized objective function $\tilde{E}$ becomes

$$\tilde{E}(\theta; X, y) = E(\theta; X, y) + \lambda \Omega(\theta)$$  \hspace{1cm} (2.23)

where $\theta$ is all the parameters of the model. The intensity of the penalty is controlled by a Lagrange multiplier $\lambda \geq 0$ (when zero it does not penalize at all). Large values of $\lambda$ shrink the weights toward zero (may give poor fit), while small values of allow the network to overfit[20].

The bias parameter is excluded in the regularization process, since it only controls a single variable (while a typical weight specifies the relationship between two variables) and requires less data to fit accurately. Regularizing it could lead to a serious amount of underfitting. The vector $w$ therefore denotes all the weights that should be affected by a norm penalty.

One can use separate multipliers for each layer of the network. This does bring an expense of tuning more hyperparameters but can give a better tuning[20].

There are two common ways of choosing the penalty norm:

$L^2$ Regularization: Is also known as weight decay, and is the most common way of parameter norm penalty[20]. It constrains the weights to lie in a hypersphere by $\Omega(w) = \frac{1}{2} \|w\|^2$.

$L^1$ Regularization: Constrains the weights to lie in region by $\Omega(w) = |w|_1 = \sum_i |w_i|$. It results in a more sparse solution than $L^2$ regularization, since some parameters have an optimal value of zero[20].

When performing regularizing in this way, it is important to normalize the input, since the weights for the first layer are determined by the input. To ensure that the input is treated equally by the regularization process, the mean of the input should be zero and the standard deviation be one[21].

Data Augmentation

Training with more data is usually the best approach to obtain a general model which performs good on new data. One way to do so is by faking new data in appropriate ways. For classification problems, a classifier must support a wide variety of transformations. New data points could then be added by slightly modifying the already existing ones. This is specially suitable for image-related problems, where the input is of high dimensions involving a lot of variation[20].

New image-related data points could for instance be added by translation, rotation or scaling of already existing data. This does often contribute to improved generalization, despite if the model is designed to be invariant to these properties. However, one must be careful not to apply modifications that might change class of the object (for instance by performing a horizontal flip on letters, "b" would appear as "d"). Another modification, with the goal of obtaining a more general model, is to inject noise into the input. Since neural networks are not very robust in its nature, training with random noise could enhance its robustness[20].

Label Smoothing

Datasets often contain mistakes in the target labeling, and maximizing this could harm the model. This can be addressed by modelling noise on the labels, assuming that label $y$ is correct with a probability $1 - \epsilon$[20]. Smooth labeling using softmax with $N$ classes becomes

$$\hat{y} = \text{smooth}_{\text{softmax}}(a) = \frac{(1 - \epsilon) \exp(a_i) + \sum_{j \neq i} \frac{\epsilon}{N-1} \exp(a_j)}{\sum_j \exp(a_j)}$$  \hspace{1cm} (2.24)
2.2. Regularization in Neural Networks

Early Stopping

When training any model, the error on the training set will decreases over time, while the validation set error might begin to rise again (because the model starts to overfit to the training data)[21]. This can be avoided by stopping the training procedure before this happens. The model is trained in search for the best performing weights on the validation set, and stopped when there is no improvement of the best recorded weights, for some specified number of iterations. This could be viewed as tuning a hyperparameter specifying the number of training steps until termination.

The benefits of this technique is its simplicity and effectiveness. In contrast to weight decay, it does not change the learning dynamics by modifying the objective function (where one can get trapped in bad local minima by penalizing too much)[20]. The procedure can also be conducted in parallel, where training and validation is performed on separate machines (or separate GPUs). It can also easily be used together with other regularization techniques.

Because early stopping requires a validation set, the training does not make use of the entire dataset. It is therefore best to perform additional training on the entire dataset when early stopping has completed, to make use of the extra data[20]. This can be done in two ways:

Reinitialize the model: Reinitialize the model and train it on the entire dataset for the same number of steps that was optimally determined by early stopping. However, it is hard to know if one should train for the same number of parameter updates or the same number of dataset passes (epochs). Either way, there is an uncertainty that the minimum occurs after the same number of steps.

Continue training: Perform additional training on the entire dataset, starting with the weights for the model that were optimally determined by early stopping. The validation set (which is now also a part of the training data) is used to monitor when the training falls below the best error rate. However, this might not ever happen, and does not guarantee a termination for the process.

Early stopping could be comparable with weight decay, since weight decay specifies a restriction of the trajectory length of the weight vector, while early stopping stops the trajectory at a good point in space (determined by monitoring the validation set error)[20].

Bootstrap Aggregating

Bootstrap aggregating[20], also known as bagging, is a powerful and reliable regularization technique which combines several models and have them vote on the output. Since different models might not make the same errors, they can compensate each other.

One way of combining models is by averaging their votes. The average error for $k$ models is $\frac{1}{k} \sum_i e_i$, where $e_i$ is the error for model $i$. Suppose the model errors are drawn from multivariate Gaussian distributions with zero mean, variance $E [e_i^2] = v$ and covariance $E [e_i e_j] = v = c$. The expected mean squared error is then

$$E \left[ \left( \frac{1}{k} \sum_i e_i \right)^2 \right] = \frac{1}{k^2} E \left[ \sum_i (e_i^2 + \sum_{j \neq i} e_i e_j) \right] = \frac{1}{k} v + \frac{k - 1}{k} c $$  \tag{2.25}

When the model errors are perfectly correlated ($c = v$) the error becomes $v$, and the averaging has no effect. In the case where the errors are perfectly uncorrelated ($c = 0$) the error becomes $\frac{1}{k} v$, meaning the error decreases linearly with the number of contributing models[20, 16].

Bagging can be done in many ways, for instance: (1) combining different networks, (2) training with different algorithms or objective functions, (3) training on different datasets (e.g. construct $k$ datasets by sampling with replacement from an original dataset).
Since training and evaluating several independent networks requires more memory and runtime, it is common not to use more than five to ten neural networks in an ensemble\cite{20}.

**Dropout**

Dropout\cite{5} is a computationally inexpensive regularization method which can be thought of as a method of bagging a large amount of models, where subnetworks is formed by removing units from a base network (multiplying their input by zero). This makes it possible to represent an exponential number of models, where in reality only one network is used. Units (along with their connections) are dropped by random during training.

Dropout forces a unit to perform well despite which subnetwork it is present in. For instance, if a model learns to detect a face by finding the nose, then dropping a hidden unit corresponds to erasing this information, the unit must learn to detect the face by another feature.

Since dropout is a regularization technique, it aims to reduce the capacity of the model. This typically implies that the base network must be much larger. However, for being simple to implement and it works well with any model, it is the most widely used implicit ensemble method.

**Weight Tying and Sharing**

Introducing weight dependencies is a common way of expressing some prior knowledge into the model. If two problems that are similar are addressed with models $A$ and $B$, it could be reasonable that their weights should be close to each other. One way of doing this is by introducing a penalty norm $\Omega(\theta_A, \theta_B) = \|w_A - w_B\|_2^2$, tying them together in a $L^2$ fashion\cite{20}.

Another, more popular way, is to force the weights to be equal. This can be applied to parts of the same model, and yield the advantage of only managing a subset of the weights\cite{20}. Convolutional neural networks makes use of this by sharing weights for features across different locations (through translation) and incorporates prior knowledge in the network architecture (for instance a photo of a cat still presents a cat even if the pixels are translated one to the right).

### 2.3 Convolutional Networks

Convolutional (neural) networks (CNNs)\cite{13} are feed-forward neural networks specialized for processing data that has a grid-like structure, and is an approach that is neuroscientifically influenced by how the visual cortex in the brain is organized. The name comes from the fact that the network applies a linear mathematical operation called convolution. CNNs use convolution instead of matrix multiplication for at least one layer.

**Convolution Operation**

The convolution operation performs a weighted average over some measurement, based on a probability density function $w(a)$ (kernel), on the input $x(t)$. The output is a feature map, and will be a smoothed estimate of the input.

$$s(t) = (x \ast w)(t) = \int x(a)w(t - a)da \quad (2.26)$$

The convolution operation is commonly denoted with an asterisk. In practise, the values are usually discrete and will then be defined as

$$s(t) = (x \ast w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a) \quad (2.27)$$
When used in machine learning, the kernel is represented by a multidimensional array where the weights are adjusted by the learning algorithm. Convolution is often used over more than one axis. For an image as input a 2D-kernel is used, given by

$$S(i, j) = (K * I)(i, j) = \sum_{m} \sum_{n} I(i - m, j - n)K(m, n)$$  \hspace{1cm} (2.28)

This is illustrated in figure 2.3.

Figure 2.3: Example of convolution with kernel of size 3x3 applied on input of size 5x5.

**Sparse Interaction**

In usual feed-forward networks, matrix multiplication is used to calculate the weights’ effect on the input. In this case every input unit affect every output unit (the network being fully connected). For convolution, the units are sparsely connected and only interact with some units at the next layer. This is done by making the kernel smaller than the input (limiting the number of connections a unit may have)[13]. An effect of this is that the exact location of some features is more relaxed, while also decreasing the computational work dramatically. For a deep CNN, the units will still be able to indirectly interact with large portion of the input, and not lose too much precision. This allows CNNs to efficiently describe complicated interactions between a vast amount of variables.

**Parameter Sharing**

In neural networks, a weight is used exactly once: when computing the input’s effect on the activation for a specific unit. For convolution, the same weight is applied to the same input value (despite the unit). Each member of the kernel is then used at multiple positions of the input. Therefore, the same linear transformation of a small local region is applied across the input[20].

**Equivariance**

Because of the form of parameter sharing, convolution has a property called equivariance to translation. This means the output changes in the same way the input does. A function $f$ is equivariant to a function $g$ if $f(g(x)) = g(f(x))$. For convolution, if $g$ translates the input (mapping image function to another image function, e.g. shifts every pixel some amount), and $f$ is the operation of performing convolution, the result would be the same regardless of the order. This property’s effect on image data would mean, moving an object in the input space would result in moving the object the same amount in the output representation. This is useful for early layers where features (e.g. edges, lines) are similar across the entire image, and the parameters could be shared. Therefore, detecting a feature would be independent of the location, but yet maintain its location in the output representation. However, for later layers working on more complex features, this property is not as useful[20].
Poolings
A convolution layer (obtaining a set of activations) is typically followed by a nonlinear function (comparable with (2.3) (2.2)), which is then passed on to a pooling function. The pooling function reports a summary statistic of a rectangular neighborhood of the output. The output could for instance be the maximum or average of the neighborhood, or a weighted average based on the distance to a center. Pooling makes the representation invariant to small translations of input, focusing more on whether a feature is present than where exactly it is[20]. This can be viewed as adding a prior belief about the input. This can however be inappropriate when it is important to preserve the locations of the features.

The next layer after a pooling layer will have $k$ times fewer inputs to process (where the pooling regions are $k$ pixels apart). This results in improved efficiency and reduced memory consumption. It is also for many tasks essential for handling input of different sizes, where the next layer must have a fixed size input[20].

Convolutional Neural Network Layers
There are different terminologies when talking about components in CNNs. For a simpler terminology, each operational layer is referred to as a layer, such as convolution and pooling. For a complex terminology, a sequence of simple layers are referred to as a layer. This is often done to present huge networks in shorter notation[20].

2.4 Face Recognition
A CNN-based approach on face recognition (and verification) is to learn the face representations purely from the data, rather than using manually engineered features. The output of the last hidden layer is used as an encoding of the face and is put in Euclidean space. Face verification can be done by simply thresholding the distance between two points in this space. Face recognition could be viewed as a k-nearest neighbors classification problem[22].

To make the most of the Euclidean space, which the images are transformed into, the representation should be constructed to maximize the distance between different identity clusters, while minimizing the distance between points of the same identity. Simply forming the encoding based on classifications with softmax would be a shot in the dark, hoping it will give good properties in the Euclidean space. A better alternative is to tune the network according to these properties.

Pretraining CNN
A good idea to make the training easier and faster is to bootstrap the CNN-architecture $\phi$, used for face recognition, by first pre-training the network on a $N$-classification problem (and then tuning for the wanted properties)[22]. When training with softmax, each image $\ell_t$, $t = 1, ..., T$, gets associated with an output vector $x_t = W\phi(\ell) + b \in \mathbb{R}^N$ (by a fully-connected layer of $N$ predictors). These are then compared to the true identity classes $c_l \in \{1, ..., N\}$ (encoded as one-hot vectors) by calculating the empirical softmax log-loss

$$L_S = \sum_t \log \text{softmax}(x_t)_i$$

Training using Triplet Loss
The goal of training with triplet loss is to learn representative encoding vectors that perform well in facial recognition context, by obtaining vectors that are distinctive and compact. Even though this loss function was not used in this work, it is interesting to compare it with center loss[9].
The output of the CNN $\phi(\ell_1) \in \mathbb{R}^D$ is $l^2$-normalized and projected to a $L \ll D$ dimensional space using an affine projection $x_l = W^t \phi(\ell_1)/\|\phi(\ell_1)\|_2$, $W' \in \mathbb{R}^{L \times D}$ [22]. $L$ is the size of the encoding vector and can be set arbitrary (Parkhi et al[22] used $L = 1024$, Schroff et al[24] used $L = 128$).

The triplet loss aims to minimize the distance between an anchor point $x_l^{(a)}$ and a positive point $x_l^{(p)}$ (having the same identity), while maximizing the distance between $x_l^{(a)}$ and a negative point $x_l^{(n)}$ (of different identities). The encoding will also be constrained to be in the $d$-dimensional hypersphere $\|x_l\|_2 = 1$. It tries to fulfill
\[
\left\| x_l^{(a)} - x_l^{(p)} \right\|^2_2 + \alpha < \left\| x_l^{(a)} - x_l^{(n)} \right\|^2_2 \tag{2.30}
\]
where $\alpha$ is the enforced margin between positive and negative pairs. $\forall (x_l^{(a)}, x_l^{(p)}, x_l^{(n)}) \in T$, where $T$ is the set of all possible triplets in the training set. The loss that should be minimized is given by
\[
L_T = \sum_{(a,p,n) \in T} \max \left\{ 0, \left[ \left\| x_l^{(a)} - x_l^{(p)} \right\|^2_2 - \left\| x_l^{(a)} - x_l^{(n)} \right\|^2_2 + \alpha \right] \right\} \tag{2.31}
\]

It is important that only triplets that can contribute to improvement of the model are selected, since many triplets are easily satisfied. Including all triplets in training would result in a slow convergence[24].

Given an anchor points $x_l^{(a)}$, a positive point $x_l^{(p)}$ (hard positive) is selected such that $\arg\max_{x_l^{(p)}} \left\| x_l^{(a)} - x_l^{(p)} \right\|^2_2$ and a negative point $x_l^{(n)}$ (hard negative) such that $\arg\min_{x_l^{(n)}} \left\| x_l^{(a)} - x_l^{(n)} \right\|^2_2$ [24].

It is not computationally feasible to compute these across the whole training set, and they are therefore only selected from within mini-batches (Schroff et al[24] used 1800 as batch size). The mini-batch consists of all positive faces while also selecting some randomly sampled negative faces. However, selecting the hardest negative can lead to local minima early on when training[24]. Instead semi-hard examples are selected as negative ones, which also satisfies $\left\| x_l^{(a)} - x_l^{(p)} \right\|^2_2 < \left\| x_l^{(a)} - x_l^{(n)} \right\|^2_2$ (which are further away from the anchor point than the positive example).

### Training using Center Loss

Another approach to tune the network, similar to triplet loss, is to introduce a center loss signal. This signal jointly supervises the training with softmax loss according $L = L_S + \lambda L_C$ (balanced by a hyperparameter $\lambda$), and yields state-of-the-art accuracy for small training sets (under 500k images and under 20k unique identities)[9].

Compared to triplet loss, where careful selection of triplets has to be taken into account for the approach to be computationally practical and stable, center loss is more efficient and easier to implement. It uses centroids, which are updated during training, to minimize the distance to its belonging points in space. Softmax has the effect of forcing the classes from staying apart, while center loss tries to move points of the same class closer to each other.

The center loss function is given by
\[
L_C = \frac{1}{2} \sum_{i=1}^{m} \left\| x_l - c_{y_l} \right\|^2_2 \tag{2.32}
\]
where $c_{y_l}$ denotes the $y_l$th center point. Updating the points would ideally use the entire training set, taking the average of all points belonging to that class. This would however be inefficient and unpractical. Instead, the centers are updated based on mini-batches.
The learning rate of the center points are controlled by a scalar $a \in [0, 1]$, and are updated by

$$c_j^{t+1} = c_j^t - a \Delta c_j$$  \hspace{1cm} (2.33)

where

$$\Delta c_j = \frac{\sum_{i=1}^{m} \delta(y_i = j)(x_i - c_j)}{1 + \sum_{i=1}^{m} \delta(y_i = j)}$$  \hspace{1cm} (2.34)

based on the gradients of $L_C$

$$\frac{\partial L_C}{\partial x_i} = x_i - c_{y_i}$$  \hspace{1cm} (2.35)

and where $\delta(\text{condition}) = 1$ if the condition is satisfied, 0 otherwise.

### 2.5 Saliency Map

A saliency map captures the object in image-space, as seen in figure 2.4, making it possible to crop out the object areas. Simonyan et al.[25] approached this by highlighting the most discriminant areas in the image, and further propagated this covering further using color continuities, to capture the entire object.

Figure 2.4: Example of image-specific class saliency maps.

When calculating a saliency map for a class of a CNN, the pixels of image $I_0$ are ranked based on their influence on the class score function $S_c(I)$ (for a certain classification). If the influence for class $c$ would be according to a linear score model, $S_c(I) = w^T I + b_c$ could be used, where $I$ is the image (in vectorized form), $w_c$ is the weight vector and $b_c$ is the bias. It is clear that the elements of $w$ emphasize the importance of the pixels in $I$. This is not the case for CNNs, since the class score function is highly non-linear of $I$. However, the score value can be approximated, given an image $I_0$, by the first-order Taylor expansion

$$S_c(I) \approx S_c(I_0) + S'_c(I_0)(I - I_0) = w^T I_0 + b + w^T (I - I_0) = w^T I + b$$  \hspace{1cm} (2.36)

where

$$w = \frac{\partial S_c}{\partial I} |_{I_0}$$  \hspace{1cm} (2.37)

which is the derivative of $S_c$ with respect to image $I$ at point (image) $I_0$. It indicates which pixels affect the class the most.

The derivative of $S_c$ with respect to the image is obtained by a single pass of back-propagation. It is used to obtain the map computed as $M_{ij} = \left| w_{h(i,j)} \right|$, where $h(i,j)$ is the index for element $w$ of pixel $(i,j)$ in the image. This applies for gray-scale images. In the case
of multi-channel images, a color index is needed. If $k$ is the color channel for pixel $(i,j)$, the pixel is indexed as $h(i,j,k)$. The class saliency value is then the maximum value for the color channels over pixel $(i,j)$: $M_{ij} = \max_c \left| w_{h(i,j,c)} \right|$. 
Three major parts were addressed in this thesis. The first step was to train CNN models which verifies faces with good performance. The second step involved visualizing each model’s interest areas in the image. The last step made use of these areas to create artificial pixel noise, to evaluate the performance of noise-induced images over a test set. These steps could be conducted in iterations, which made it possible to perform pre-experiments on poorly trained CNNs.

3.1 Environment

The trainings were conducted on a Nvidia DGX-1, specifically engineered for deep learning training. This machine was provided by the company where the study was conducted. This made it possible to train multiple models on huge datasets within the time scope of this thesis work.

Caffe framework[11] was used in addition to available open source implementations of center loss function[10].

3.2 Datasets

When creating the facial verification system, different datasets were used for different purposes. For training the models, Microsoft’s MS-Celeb-1M dataset for MSR Image Recognition Challenge (IRC)@ACM Multimedia 2016[8] was used. Several independent datasets were used for testing the performance of the trained models.

The MS-Celeb-1M dataset had been constructed by first selecting the top 100K celebrities (according to popularity) and obtain images for each celebrity using public search engines[8]. The dataset had a mean value of 85.65 images per identities, with a total of 99,891 identities and 8,456,240 images. However, in result of being beyond the scale of manual labeling, it contained a lot of noise (mislabeled faces). This noise could damage the deep learning algorithm and result in poorly trained models.

The Labeled Faces in the Wild (LFW) dataset consisting of 13,233 web-collected images, labeled into 5,749 identities)[2] was used for tuning some hyperparameter and testing the models. The dataset was divided into two views. LFW View 1 was used for training and
3.2. Datasets

evaluating the models using the hold-out method (dividing the data into two sets: training and testing). The second view was used for testing the performance of the final models. The LFW protocol has been follow by related work[24, 22, 9].

Data for the views in LFW does partly overlap (training sets of view 1 and view 2) and bias the final test performance[16, 21]. Wolf et al[2] suggests it does not bias the result significantly, because the test sets of View 1 and View 2 do not overlap. This is because the models are not selected by the overlapping data (it is only used for selecting a verification threshold).

The final models were evaluated on two test datasets:

1. **LFW View 2** Evaluations on LFW were made following the standard protocol of unrestricted with labeled outside data[2], which suggest cross-validation[16] on per fold disjoint datasets ($K = 10$ folds) of the provided pairs for LFW View 2. The $k^{th}$ fold was used for testing, and the other $K - 1$ folds to select a verification threshold/train a classification model on the Euclidean distances. Each fold consisted of 300 same and 300 different pairs.

2. **YouTube Faces (YTF)** This dataset consisted of sequences of consecutive frames from YouTube, based on the identities provided by LFW[27]. 5,000 videos from this database had been divided into 10 folds and were evaluated using cross-validation, following the unrestricted protocol. Each fold consisted of 250 same and 250 different pairs. In this work, the entire sequences of frames (which the given pairs suggests) were not used, but only the first images of each sequence. This does affect the outcome of the performance evaluated on this dataset (introducing greater variance), making the test harder.

**Filtering MS-Celeb-1M**

Label smoothing could make label noise less severe[20], but because of lack of available open-source implementations in Caffe, it was not used. Instead the data was filtered in an attempt to remove mislabeled identities.

The MS-Celeb-1M dataset contained a lot of noise, in form of false positives, where other persons were labeled as having the identity of the class. Attempts were made to reduce this
noise by filtering images based on the Euclidean distance to a mean encoding for each class. These distances were obtained by running the dataset over an open source model[10] trained on the CASIA dataset[1] (which is less noisy than MS-Celeb-1M). MS-Celeb-1M was then filtered by removing images that were above a filtering threshold.

The maximum center distance for identities in the LFW dataset was 30.2887, while the distances for MS-Celeb-1M was 38.9908, which implies that the MS-Celeb-1M indeed is noisy. Different thresholds were tested, when training a Face-ResNet model, in the interval of [19, 39] (where 39 implies no filtering). The interval was examined linearly with steps of 1. The trainings were done on a randomly sampled subset of the MS-Celeb-1M dataset consisting of 500k images. This dataset was filtered into smaller subsets (depending on the filtering threshold value).

![Figure 3.2: Verification accuracy for models with different noise thresholds evaluated on LFW View 1. A curve was fitted to the data using polynomial regression (polynomial of degree 2).](image)

The reason for training on smaller datasets when a greater valued threshold is applied is to evaluate the trade-off between noisy data and less data. This is also the case when filtering the entire MS-Celeb-1M dataset. The observed results of the selected models were gathered into a set of 20 samples. Since the observations seemed to contain high variance, polynomial regression was used to fit a curve to the data. This resulted in a filtering threshold between 24 and 25 (keeping around 92% of the images).

A pre-cleaned version of the dataset[6] was also evaluated and compared to this filtering procedure. Filtering threshold 24, 28 and the clean version were used to subsample 1M images from MS-Celeb-1M. These were used to train models for 30 epochs, obtaining 0.94, 0.92 and 0.971 accuracy on LFW View 1, respectively.

In this work the already cleaned version[6] of the MS-Celeb-1M dataset was further used for training models. It consisted of 5,049,824 images (keeping around 60% of the images) with 98,693 identities.
3.3 Evaluation of Verification

The models’ performances were evaluated by performing face verification, where it was determined if pairs of face images \((a, b)\) had the same or different identity. This could be based on the Euclidean distance \(D(a, b)\) between the faces being below a threshold \(\delta\). If \(P_s\) is the set of true same identities, and \(P_d\) the set of true different identities, then the true accepts \(TA(\delta)\) and the false accepts \(FA(\delta)\) are defined as

\[
TA(\delta) = \{(a, b) \in P_s, D(a, b) \leq \delta\} \\
FA(\delta) = \{(a, b) \in P_d, D(a, b) \leq \delta\}
\] (3.1)

This gives the validation rate \(VAL(\delta)\) and the false accept rate \(FAR(\delta)\) as

\[
VAL(\delta) = \frac{|TA(\delta)|}{|P_s|} \\
FAR(\delta) = \frac{|FA(\delta)|}{|P_d|}
\] (3.2)

These two metrics plotted against each other gives a ROC curve[16], which illustrates the performance of a binary classifier for various thresholds.

When classifying by certainty, a probability distribution is favorable compared to a deterministic answer - using a threshold. In this work, logistic regression[16, 21] was instead used to determine if pairs were the same or not (using the intercept value and one coefficient).

Accuracy is probably the most important metric when evaluating the verification performance of the overall model (network model to obtain the distance, logistic regression model to classify into same or different identities). If \(C\) is the number of correctly classified examples and \(T\) is the total number of examples, it is calculated as

\[
Accuracy = \frac{C}{T}
\] (3.3)

Coverage is another interesting metric for the overall model. When classifying based on a probability distribution, a confidence value for the decision is obtained. By discarding decisions below a certain degree of confidence and only classifying on the \(P\) certain examples, the coverage is calculated as

\[
Coverage = \frac{P}{T}
\] (3.4)

3.4 DAR Pipeline

Facial recognition and verification is in many real world applications only one part of a system that automatically detect, align and recognize faces in images. Recognition is part of a Detection-Alignment-Recognition (DAR) pipeline. The earlier steps in the pipeline prepares images for the next step in the pipeline, making the final task successfully easier.

![Detection-Alignment-Recognition (DAR) pipeline.](image)

For this work, the detection step had already been done by the providers of the datasets[8, 2, 27], leaving alignment and recognition untouched.

The performance of the pipeline on the test sets also depends heavily on detection and alignment. Ideally the same aligner should be used for training and testing, since different aligners produce different results. For this work, the cropped version of MS-Celeb-1M[8]...
was aligned using a MTCNN[3] open source implementation[15]. The method for which the alignment implementation was based on was also used by Wen et al[9].

By using this alignment implementation, images that were considered hard by the aligner were left out (2.05% of the images were unsuccessfully handled where no face was detected). Training on the left out images would produce a more robust recognizer. However, because of MS-Celeb-1M’s huge size, it should not affect the end result significantly.

3.5 General Training Procedure

The goal when training was to find weights that maximize the performance of the verification task, by minimizing a loss function: obtaining a good fit while not overfitting to the training set. When training neural networks, different settings works well for different conditions. However, for the experiments of this work, the used settings did overlap to some extent. If nothing else is mentioned, the procedure described below was used.

1. Early stopping was used as regularization method to choose the best performing epoch. The selection of epoch was based on the loss value on a validation set $D_V$, subsampled from MS-Celeb-1M, or the accuracy on LFW View 1. Despite that the loss function is not what actually is being optimized, tuning by a surrogate loss function (acting as a proxy) has the advantages of including conditional probabilities of classes in performance estimation (where for instance a 0-1 loss is intractable)[20]. A surrogate loss function could result in being able to learn more. For instance when the 0-1 expected loss is zero, it is possible to improve the model further by pushing the classes apart from each other. This could lead to improved robustness of the classifier. However, it relies completely on how well the loss function represents the problem.

2. The dataset used for each training was first divided into training set $D_T$ and validation set $D_V$ (using the hold-out method). The training set was randomly sampled from the dataset, making up 80% of the dataset size, leaving 20% to the validation set.

3. The training batch size $B_T$ and validation batch size $B_V$ were set quite high (to make use of the available computational resources and decrease the runtime of each epoch). However, since a minibatch of lower size may has a greater regularizing effect (because of the ability to more easily escape local minima), it might be preferable in some cases[20]. This was however discarded in this work.

4. The number of total iterations were calculated as $E \times |D_T| / B_T$, where $E$ is the number of epochs. The models were tested each epoch on the entire validation set for $|D_V| / B_V$ iterations.

5. The Caffe solver and the network used the following settings:
   a) The weights were initialized randomly using Caffe Xavier.
   b) The trainings were done using stochastic gradient descent with a momentum of 0.9.
   c) The default weight decay value of 0.0005 was often used.
   d) A base learning rate was decreased stepwise by a factor of 0.1 at roughly 33% of the total number of iterations, using a gamma value of 0.1.
   e) Data augmentation was performed by mirroring the images, both for the training and validation set.
3.6 Face-ResNet

The first network was obtained from the Deep Face Recognition with Caffe Implementation GitHub repository[10], and had an architecture as described below. It was trained with center loss from the start, and took three color channels as input of height 112 and width 96. This network achieved state-of-the-art performance for face verification on small datasets (under 500k images and 20k identities)[9].

Table 3.1: Face-ResNet Architecture

<table>
<thead>
<tr>
<th>Layer</th>
<th>#Filters (#Output)</th>
<th>Kernel (size, stride, pad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>conv1a</td>
<td>32</td>
<td>3,1,0</td>
</tr>
<tr>
<td>relu1a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv1b</td>
<td>64</td>
<td>3,1,0</td>
</tr>
<tr>
<td>relu1b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool1</td>
<td></td>
<td>2,2,0</td>
</tr>
<tr>
<td>RConv2_1</td>
<td>64</td>
<td>3,1,1</td>
</tr>
<tr>
<td>conv2</td>
<td>128</td>
<td>3,1,0</td>
</tr>
<tr>
<td>relu2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool2</td>
<td></td>
<td>2,2,0</td>
</tr>
<tr>
<td>RConv3_1</td>
<td>128</td>
<td>3,1,1</td>
</tr>
<tr>
<td>RConv3_2</td>
<td>128</td>
<td>3,1,1</td>
</tr>
<tr>
<td>conv3</td>
<td>256</td>
<td>3,1,0</td>
</tr>
<tr>
<td>relu3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool3</td>
<td></td>
<td>2,2,0</td>
</tr>
<tr>
<td>RConv4_1</td>
<td>256</td>
<td>3,1,1</td>
</tr>
<tr>
<td>RConv4_2</td>
<td>256</td>
<td>3,1,1</td>
</tr>
<tr>
<td>RConv4_3</td>
<td>256</td>
<td>3,1,1</td>
</tr>
<tr>
<td>RConv4_4</td>
<td>256</td>
<td>3,1,1</td>
</tr>
<tr>
<td>RConv4_5</td>
<td>256</td>
<td>3,1,1</td>
</tr>
<tr>
<td>conv4</td>
<td>512</td>
<td>3,1,0</td>
</tr>
<tr>
<td>relu4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>maxpool4</td>
<td></td>
<td>2,2,0</td>
</tr>
<tr>
<td>RConv5_1</td>
<td>512</td>
<td>3,1,1</td>
</tr>
<tr>
<td>RConv5_2</td>
<td>512</td>
<td>3,1,1</td>
</tr>
<tr>
<td>RConv5_3</td>
<td>512</td>
<td>3,1,1</td>
</tr>
<tr>
<td>fc5</td>
<td>512</td>
<td></td>
</tr>
<tr>
<td>fc6</td>
<td>#Classes</td>
<td></td>
</tr>
<tr>
<td>Softmax_Loss+\lambda * Center_Loss</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1 presents the network architecture. Every maxpool, conv, fc and relu layer is described using a simpler terminology. RConv is described using a more complex terminology and consists of several layers. Its filters and kernel represents the convolution layers’ kernels and filters in it. Figure 3.4 shows how the data is passed within this layer. The res layer performs an element-wise summation of two input sources (the input of the RConv and the output of the last relu in it).

Layer fc5 was used as encoding vector to determine the Euclidean distance between a pair of images. It used 512 feature values to encode the face.

Tuning Center Loss $\lambda$-Hyperparameter

Wen et al[9] tuned $\lambda$ for the combined loss of center loss and softmax ($L_S + \lambda L_C$) to be $\lambda = 0.003$ (which should correspond to 0.006 in the open source implementation[10], since $\lambda$ is
divided by 2 in equation 2.32). They did this by testing various $\lambda$ from 0 to 0.1 (grid search[20] by logarithmic scale). In the open source implementation 0.008 was proposed.

In this work, $\lambda$ was fine-tuned on a subset with 500k images of the MS-Celeb-1M dataset in an attempt to find a better performing value. The search spanned over the range (0.004, 0.012) with steps of 0.001 (linear scale) and was evaluated on LFW View 1.

The value 0.008 appeared to be best according to a fitted polynomial regression curve (of degree 2) to the observations. However, the value 0.007 had the highest performing observation. In this work, the value of 0.008 was further used when training this network.

### Training

Trainings were done on sequentially larger datasets in attempts to manually find good hyperparameter settings. This approach was used to successfully improve the performance of the network, and better understand how to tune the hyperparameters.

**Table 3.2: Face-ResNet Trainings**

<table>
<thead>
<tr>
<th></th>
<th>500k</th>
<th>1M</th>
<th>2M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Accuracy (best loss)</td>
<td>0.938</td>
<td>0.971</td>
<td>0.973</td>
</tr>
<tr>
<td>Total Accuracy (best accuracy)</td>
<td>0.951</td>
<td>0.974</td>
<td>0.98</td>
</tr>
<tr>
<td>Epochs</td>
<td>30</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>Weight Initialization</td>
<td>Random</td>
<td>Random</td>
<td>Random</td>
</tr>
<tr>
<td>Base Learning Rate</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Weight Decay</td>
<td>0.0005</td>
<td>0.0005</td>
<td>0.0005</td>
</tr>
</tbody>
</table>

The models mentioned in table 3.2 were trained for respectively 30, 30, 60 epochs. The trainings on 500k and 1M images may not have converged according to the loss value - the best performing epochs were to last ones. Batch sizes of $B_T = 768$ and $B_V = 384$ and a base learning rate 0.1 were used.

### 3.7 LightenedCNN C

The second network was obtained from the *A Light CNN for Deep Face Representation with Noisy Labels* GitHub repository[7], and had an architecture as described below. It took one color channel as input of height 128 and width 128 pixels.
Table 3.3 presents the network’s architecture. Every maxpool, conv, fc and slice layer is described using a simpler terminology. SConv is described using a more complex terminology and consists of several layers. It consists of the layers in the following order: conv, slice and res. SConv’s filters and kernel represents the convolution layer’s filters and kernel in it.

Table 3.3: LightenedCNN C Architecture

<table>
<thead>
<tr>
<th>Layer</th>
<th>#Filters (#Output)</th>
<th>Kernel (size, stride, pad)</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SConv1_1</td>
<td>96</td>
<td>5,1,2</td>
</tr>
<tr>
<td>maxpool1</td>
<td></td>
<td>2,2,0</td>
</tr>
<tr>
<td>SConv2_1</td>
<td>96</td>
<td>1,1,0</td>
</tr>
<tr>
<td>SCond_2</td>
<td>192</td>
<td>3,1,1</td>
</tr>
<tr>
<td>maxpool2</td>
<td></td>
<td>2,2,0</td>
</tr>
<tr>
<td>SConv3_1</td>
<td>192</td>
<td>1,1,0</td>
</tr>
<tr>
<td>SConv3_2</td>
<td>384</td>
<td>3,1,1</td>
</tr>
<tr>
<td>maxpool3</td>
<td></td>
<td>2,2,0</td>
</tr>
<tr>
<td>SConv4_1</td>
<td>384</td>
<td>1,1,0</td>
</tr>
<tr>
<td>SConv4_2</td>
<td>256</td>
<td>3,1,1</td>
</tr>
<tr>
<td>SConv4_3</td>
<td>256</td>
<td>1,1,0</td>
</tr>
<tr>
<td>SConv4_4</td>
<td>256</td>
<td>3,1,1</td>
</tr>
<tr>
<td>maxpool4</td>
<td></td>
<td>2,2,0</td>
</tr>
<tr>
<td>fc5</td>
<td>512</td>
<td></td>
</tr>
<tr>
<td>slice5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>res5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dropout5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fc6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Softmax_Loss+λ*Center_Loss</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Layer fc5 was used as encoding vector to determine the Euclidean distance between a pair of images. It used 512 feature values to encode the face.

**Tuning Center Loss λ-Hyperparameter**

The center loss λ constant was also tuned for this network. It was tuned on an already trained model of 500k images, which had been trained with only softmax (λ = 0). The reason for pre-training the network was that center loss went into undefined values starting from randomly initialized weights with λ = 0.008.

Grid search[20] was performed in exponential scale within the range (0.002, 0.064). λ = 0.008 performed best also for this network.

Parkhi et al[22] did also pretrain in this manner to later tune the model using triplet loss.

**Training**

Trainings were done using the same approach as for Face-ResNet, successfully increasing the size of the dataset. Batch sizes of \( B_T = 256 \) and \( B_V = 256 \) and a base learning rate of 0.001 were used. Training on 2M images using the default weight decay was unsuccessful (the network did not converge). Setting the weight decay to zero solved this issue, which emphasizes a lack of capacity. When a bigger dataset is used less regularization is required[20], and can be damaging instead.

**3.8 Visualization Techniques**

In this work, three types of visualization techniques were created.
3.8. Visualization Techniques

<table>
<thead>
<tr>
<th></th>
<th>500k A</th>
<th>1M</th>
<th>2M A</th>
<th>500k B</th>
<th>2M B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Accuracy (best loss)</td>
<td>0.934</td>
<td>0.951</td>
<td>0.968</td>
<td>0.944</td>
<td>0.972</td>
</tr>
<tr>
<td>Total Accuracy (best accuracy)</td>
<td>0.934</td>
<td>0.952</td>
<td>0.971</td>
<td>0.945</td>
<td>0.974</td>
</tr>
<tr>
<td>Epochs</td>
<td>30</td>
<td>30</td>
<td>60</td>
<td>30</td>
<td>60</td>
</tr>
<tr>
<td>Weight Initialization</td>
<td>Random</td>
<td>Random</td>
<td>Random</td>
<td>500k A</td>
<td>2M A</td>
</tr>
<tr>
<td>Base Learning Rate</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>Weight Decay</td>
<td>0.0005</td>
<td>0.0005</td>
<td>0</td>
<td>0.0005</td>
<td>0</td>
</tr>
<tr>
<td>Center Loss $\lambda$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.008</td>
<td>0.008</td>
</tr>
</tbody>
</table>

### Activation Maps

When inspecting the areas the model looks at in the image, the encoding for facial verification was used as gradients and propagated backwards in the network. This highlights the areas the model considers to be discriminant.

This could be viewed as minimizing an error function which aims to assign everything to zero ($y = 0$), $E_n(\hat{y}) = \frac{1}{2} \hat{y}^2$ (with identity as the output activation function, giving $\delta_i = \frac{\partial E_n(\hat{y})}{\partial \hat{y}_i} = \hat{y}$). The gradient then represents the changes to make in order to improve the results of the error function. More intense gradient areas in the image map are more significant areas for the model.

This could be compared with saliency maps[25], where a class score is maximized. In this work, the absolute value of the class score is of interest, and the score (encoding) variables are highly dependent of each other and cannot be visualized separately, as in the work by Simonyan et al[25].

The color channels were summed together and then normalized as

$$M' = \frac{\|M\| - \min(\|M\|)}{\max(\|M\| - \min(\|M\|))} \quad (3.5)$$

(to be in the range of $(0, 1)$, while also merging the negative and positive scale by taking the absolute value. The direction of the gradients is discarded, since only the trajectory length representing the gradient is important to determine significant areas according to the error function.

### Space Maps

Space maps were created to represent the model’s dependencies for many possible activations. They were created by back-propagating Gaussian random values (mean 0, standard deviation 1) from the network’s encoding layer (fc5). This was done when the image already had been forward propagated as input (which is necessary in order to calculate the gradients). These maps where combined in two different ways

$$M_{ij}^{(\text{max-space})} = \max_d(M_{ij}^{(d)}) \quad M_{ij}^{(\text{sum-space})} = \sum_d(M_{ij}^{(d)}) \quad (3.6)$$

where $M_{ij}^{(d)}$ is the pixel value of row $i$, column $j$ belonging to map $d$. This was done before summing up the color channels and normalization.

The max-space presents the highest points of each map in the combination, while sum-space captures how frequently a pixel is activated and at what intensities.

### Artificial Noise Evaluation

Incremental amount of noise was induced into the images to evaluate the model’s performance given only a subset of true-valued pixels. Four different approaches of creating artificial noise in the images were evaluated: (1) removing pixels at random (by a uniform
3.8. Visualization Techniques

Figure 3.5: Activation maps using the images’ activations as gradients. Created for Face-ResNet.

distribution) (2) removing pixels based on activation maps, (3) removing pixels based on max-space maps, (4) removing pixels based on sum-space maps.

The models were evaluated on LFW View 1 with incremental exposure to noise. Logistic regression models were trained on images exposed to the same noise, in order to only evaluate the neural network models. The discarded parts were assigned to uniformly random values in three channels. The maps were ordered by significance and applied as noise, in both directions. Removal of more significant pixels starting from the least significant, and removal of less significant pixels starting from the most significant. These exposure techniques will be denoted as removal of significant pixels, and removal of insignificant pixels, respectively.

Incremental exposure from both directions were done to capture the pixels’ importance in the ability to preserve and damage the performance. Measuring the pixels’ ability to damage the model’s performance represents how important the pixels are in conjunction with other present pixels. Measuring the pixels’ ability to preserve the model’s performance represents how independently important the pixels are.

The accuracy evaluations were combined into a unified metric by

\[ D(x) = A_{\text{Insignificant}}(x) - A_{\text{Significant}}(x) \]  

where \( A(x) \) is the accuracy obtained with \( x \) percentile amount of noise. \( D(x) \) is therefore the difference between the accuracies obtained with significant and insignificant noise.

The difference metric was further combined to represent the difference over an interval of percentile noise values. This was done by calculating the area under the curve, using the Trapezoidal rule, giving the metric

\[ T(\alpha, \beta) = \sum_{i=\alpha}^{\beta} \frac{D(i/S)_+ + D((i+1)/S)_+}{2} \]  

where the interval of noise \([1, 99]\) (\( \alpha = 1, \beta = 98 \)) is interesting, and was evaluated linearly by steps of 1 (\( S = 100 \)). This could be done on \([0, 100]\) with arbitrary step sizes.

The reason for only taking positive differences (and therefore positive areas) is that measurements at high noise exposure might be disturbed. In order not to affect the rest of the measurements, they were discarded completely (measured as zero).
3.8. Visualization Techniques

Figure 3.6: Space maps by combining 25,600 random encoding maps. Created for Face-ResNet.

The area emphasizes how well the visualization technique represents the hierarchical dependencies in the image, or how robust the model is in general to artificial pixel noise (how much of the image the model actually uses).

- A visualization technique $V_1$ which has greater area than $V_2$, for the same model and test set, does damage the performance more severely when significant pixels are removed. This is because the order of removing significant pixels prioritize a highly damaging pixel before a less so, making the $A_{\text{significant}}$-curve to decrease more quickly. $V_1$ does also preserve the performance better, when insignificant pixels are removed, making the $A_{\text{insignificant}}$-curve decrease more slowly. The area of these curves therefore represents the correctness of the hierarchical ordering of the visualization technique.

- A model $M_1$ which has greater area than $M_2$, using the same visualization technique and test set, is more affected by the order in which the noise is applied. For $M_1$, the performance decreases more quickly when significant pixels are removed, and more slowly when insignificant pixels are removed. This means it makes use of a smaller facial area, and is less dependent on the entire face. This therefore represent the model’s robustness in the context of artificial pixel noise. However, this method relies in how good the visualization technique is.
The result of this work includes both the trained models and their verification performances, but also the performance effects of different types of artificial noise.

4.1 Model Performances

The final models used when conducting the experiments were evaluated on the LFW View 1 validation set and the LFW View 2 and YTF test sets. Models selected by loss on the MS-Celeb-1M validation set were compared to models selected by accuracy on LFW View 1.

Table 4.1: 2M Face-ResNet Model (Best LFW View 1 Accuracy)

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>LFW View 1</th>
<th>LFW View 2</th>
<th>YTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.980</td>
<td>0.980</td>
<td>0.870</td>
</tr>
<tr>
<td>Coverage (95%)</td>
<td>0.765</td>
<td>0.858</td>
<td>0.292</td>
</tr>
<tr>
<td>Coverage (99%)</td>
<td>0.342</td>
<td>0.538</td>
<td>0.053</td>
</tr>
</tbody>
</table>

Table 4.2: 2M Face-ResNet Model (Best MS-Celeb-1M Loss)

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>LFW View 1</th>
<th>LFW View 2</th>
<th>YTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.974</td>
<td>0.973</td>
<td>0.863</td>
</tr>
<tr>
<td>Coverage (95%)</td>
<td>0.722</td>
<td>0.810</td>
<td>0.278</td>
</tr>
<tr>
<td>Coverage (99%)</td>
<td>0.298</td>
<td>0.480</td>
<td>0.067</td>
</tr>
</tbody>
</table>

Table 4.3: 2M B LightendCNN C Model (Best LFW View 1 Accuracy)

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>LFW View 1</th>
<th>LFW View 2</th>
<th>YTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.974</td>
<td>0.977</td>
<td>0.870</td>
</tr>
<tr>
<td>Coverage (95%)</td>
<td>0.674</td>
<td>0.805</td>
<td>0.312</td>
</tr>
<tr>
<td>Coverage (99%)</td>
<td>0.311</td>
<td>0.528</td>
<td>0.075</td>
</tr>
</tbody>
</table>
Table 4.4: 2M B LightendCNN C Model (Best MS-Celeb-1M Loss)

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>LFW View 1</th>
<th>LFW View 2</th>
<th>YTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.972</td>
<td>0.978</td>
<td>0.871</td>
</tr>
<tr>
<td>Coverage (95%)</td>
<td>0.678</td>
<td>0.805</td>
<td>0.306</td>
</tr>
<tr>
<td>Coverage (99%)</td>
<td>0.311</td>
<td>0.526</td>
<td>0.077</td>
</tr>
</tbody>
</table>

Accuracy was measured on all examples ($P = T$) and coverage was measured by the percentage of confidence being above a percentile threshold. The verification was done using logistic regression\([16]\), to obtain a probabilistic outcome.

All models performed much better on LFW than YTF. However, only the first images of the frame sequences were used when evaluating the verification performance on YTF. For some test samples no face was detected. The YTF is more difficult than LFW according to related work\([12, 9, 22]\).

The Face-ResNet network performed better than the LightendCNN C network in general. Three out of four final models performed better than human accuracy (0.9753 on cropped images\([4]\)), on LFW View 2.

The best performing Face-ResNet model was selected by accuracy, and the best performing LightendCNN C model was selected by loss. These were used for further experiments.

Figure 4.1 presents the ROC curves of the models’ performances, using thresholds to perform verification, compared to human performance.

![ROC curves](image)

Figure 4.1: ROC curves of the models used in this work compared to human performance.

## 4.2 Artificial Noise

The techniques for visualizing discriminant areas and possible spaces were used to determine pixels in the image to discard. Randomly distributed pixel noise in the image served as a
4.2. Artificial Noise

comparable base-line. Figure 4.2 presents images which have been subjected to different kinds of noise, discarding 50% of insignificant pixels in the image. The space maps were created with 64 random maps, to keep the runtime relatively low.

![Random Noise](image1)

(a) Random Noise

![Activation Noise](image2)

(b) Activation Noise

![Max-space Noise (64 maps)](image3)

(c) Max-space Noise (64 maps)

![Sum-space Noise (64 maps)](image4)

(d) Sum-space Noise (64 maps)

Figure 4.2: Images subjected to 50% insignificant pixel noise using the 2M trained Face-ResNet-model, presented as black.

Figure 4.3 shows activation maps for the Face-ResNet model when the image has been exposed to different kinds of insignificant artificial pixel noise. Random noise makes the model quickly lose track of key facial areas, while the other types of noise makes the model use smaller areas. This can especially be seen for max-space.

Figure 4.4 (a) shows that a substantial amount of insignificant pixels can be discarded and still achieve well on facial verification. This also applies to figure 4.5 (a), but to a lesser extent. Max-space preserves the verification performance best compared to the other noise types.

Figure 4.4 (b) and 4.5 (b) show that significant activation and sum-space noise damage the performance the most. Max-space is less important in this context, but is still more important than random noise.

Figure 4.6 shows the difference between the significant and insignificant noise. The visualization techniques were evaluated homogeneously and heterogeneously. Table 4.5 shows that out of the homogeneous curves, max-space best represents the hierarchical structure of the image dependencies. However, the heterogeneous curves are equally much better.

The two networks do handle noise very differently. LightendCNN C, in figure 4.5, does seem to react more linearly than Face-ResNet, in figure 4.4. LightendCNN C also has a lower difference area, in table 4.5, for all visualization techniques.
4.2. Artificial Noise

Figure 4.3: Activation map visualizations on images exposed to different types of insignificant noise, using Face-ResNet.

Table 4.5: Difference Areas

<table>
<thead>
<tr>
<th>Network</th>
<th>Activation</th>
<th>Max-Space</th>
<th>Sum-Space</th>
<th>Max-Activation or Max-Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Face-ResNet</td>
<td>14.782</td>
<td>25.250</td>
<td>15.524</td>
<td>26.71</td>
</tr>
<tr>
<td>LightendCNN</td>
<td>9.007</td>
<td>15.086</td>
<td>7.499</td>
<td>17.211</td>
</tr>
</tbody>
</table>
4.2. Artificial Noise

(a) Removal of insignificant pixels

(b) Removal of significant pixels

Figure 4.4: Face-ResNet incrementally exposed to artificial pixel noise.

(a) Removal of insignificant pixels

(b) Removal of significant pixels

Figure 4.5: LightendCNN C incrementally exposed to artificial pixel noise.

(a) Face-ResNet

(b) LightendCNN C

Figure 4.6: Difference between insignificant and significant noise.
The outcome of the experiments and trainings depends on many factors. Neural networks are complex models and it is easy to jump into the wrong conclusions if the assumptions are not carefully thought through. The outcome will be discussed and why it was obtained, and the method used will be viewed from different angles, mentioning other possible approaches.

5.1 Results

The outcome may possibly be viewed in other ways than mentioned here, and serve other research purposes. However, only the relevant topics for this work will be mentioned here, where the visualization techniques and networks are compared with each other.

Facial Verification

The obtained models did achieve similar to human performance (and better). The reason why Face-ResNet performs better than LightendCNN C in general may be because it is a bigger network. LightendCNN C could not be trained using weight decay, emphasizing a lack of capacity. Neural networks tend to perform better when having a bigger capacity while being restricted with regularization[20]. Another reason might be due to the fact that Face-ResNet makes use of all color channels, while LightendCNN C only takes grayscale images.

Comparison of Visualization Techniques

Insignificant max-space noise, presented in figure 4.4 (a) and 4.5 (a), affects the performance the least if discarded. This implies that max-space best preserves the model’s interests. However, max-space does not present the most discriminant areas with highest intensity, as activation and sum-space do. If that were the case, exposure to significant max-space would affect the model in the same way (or worse) as activation and sum-space do.

A reason why max-space more accurately represents the hierarchical importance of areas in the image, according to figure 4.6 and table 4.5, is most likely because the other homogeneous techniques only present the peaks of what is important. A relatively general visualization as max-space does therefore present the model’s needs better. More pixels are ordered
5.2. Method

Correctly in the big picture of the hierarchy. They are not necessarily ordered correctly compared to each other, but are so compared to the background and other less important areas.

Significant activation and sum-space noise, presented in figure 4.4 (b) and 4.5 (b), does seem to damage the model the most. This implies that these maps capture the most critical areas of the face. However, these do not preserve the model’s performance very well when applied as insignificant noise. They do not capture the rest of the model’s interests areas very well.

The reason why activation and sum-space noise do affect the model in similar ways is most likely because the discriminant areas appear more frequently. They are therefore frequently exploited by the model when making predictions for any encoding.

By comparing the difference areas, in table 4.5, for the homogeneous and heterogeneous differences, both heterogeneous curves got greater area. This implies that the visualization techniques can be further improved to more accurately represent the dependence hierarchy.

Observation of Noise Effects

Since activation map areas, which are the most discriminant areas, do not preserve the model’s performance very well when applied as insignificant noise, other parts of the face are important. This indicates that the discriminant areas were not highlighted as independent areas, but in result of their conjunction with other facial areas. This can also be seen in figure 4.3, where different types of insignificant noise is applied. Random noise makes the model lose track of all key facial areas very quickly. Activation and sum-space noise make the model change facial areas as more noise is applied. Max-space does however seem to make use of a smaller portion of the same areas, emphasizing that it best preserves the discriminate areas.

Comparison of Networks

LightendCNN C does seem to be less affected by noise in general, compared to Face-ResNet, where performance decreases almost linearly, as shown in figure 4.5, to the increased amount of distorted pixels. LightendCNN C also has a smaller difference area, than Face-ResNet, in table 4.5 for all types of noise. This implies that LightendCNN C makes use of a greater total area of the face, and is less dependent on a few facial pixels. This idea is strengthened by the fact that Face-ResNet is damaged less by very low amounts of random noise, compared to LightendCNN C. At low amounts of applied noise, there is a lower chance of disrupting highly important facial areas.

Why LightendCNN is affected less by noise in general, and if it considers a greater total area of the face to be of importance, can be for multiple reasons. Face-ResNet taking 3-channel color input and LightendCNN C taking 1-channel color input could be the reason. However, the networks are of different sizes and architectures, where LightendCNN C is of smaller size because less color channels are used (less data). As shown in table 3.2 and 3.4, Face-ResNet could be trained with a weight decay of 0.0005 on 2M images, while LightendCNN C could not. This was most likely because of a lack of capacity. This implies that the networks indeed are not similar in their capacity of handling this quantity of images. It might be the case that a network with lacking capacity looks at a greater total area, and has not pinpointed down the areas that are more discriminant in general.

5.2 Method

Some decisions of this work were lightly evaluated and the approach chosen based on it. These, among other things, are mentioned here.
5.2. Method

Facial Verification

The hyperparameters could have been tuned further to obtain better performance, and more of the MS-Celeb-1M data could have been used. The validation set $D_V$ with 20% of the used data was also not trained on, which is encouraged when using early stopping[20]. This was not done because of time limitations of this work.

When selecting which epoch to use for early stopping, both accuracy on LFW View 1 and loss on the MS-Celeb-1M validation set were evaluated. For Face-ResNet, accuracy appeared to be the better choice, and for LightendCNN C, loss appeared to be. This has to do with how well the loss function represents the true problem[20]. The loss function for LightendCNN C represents the problem better than for Face-ResNet. In this case, the loss function has managed to push the class examples further apart from each other, obtaining a more generalized fit. The outcome of loss being the better choice for LightendCNN C can also have to do with mild overfitting to LFW View 1, where accuracy on LFW View 1 appears to be better than it actually is.

Center loss $\lambda$ was tuned automatically using grid search[20] in this work. The base learning rate and weight decay were tuned manually by intuition of good performing values. There are however other automatic tuning methods, which could have been used, that makes more sophisticated hyperparameter guesses than grid search. Bayesian optimization[26] successfully reduces the search space by the best investigated setting or by the best expectation.

Other algorithms than logistic regression, classifying on the distance outputted from the networks, could have been used. Support vector machines (SVM)[16] were considered and attempted for Face-ResNet (using Scikit learn 0.18.1 SVM default settings), but performed slightly worse than logistic regression. Other regularization methods could have been used when training the logistic regression model than $L^2$-penalty. $L^2$-penalty was compared with $L^1$-penalty and performed slightly better.

Center loss was used in this work to supervise the training of the CNN models. Triplet loss is another well performing loss function which works differently but aims to achieve the same thing. This loss function could have been used instead. According to an open-source comparisons[23], center loss appeared to perform better for their networks, and was used in this work because of it. It would however be interesting to see how the different choice of supervising loss function affects the performance of the networks used in this work.

Artificial Noise Evaluation

When combining multiple color channels into a map, Simonyan et al[25] suggested them to be combined by taking the maximum value over all three channels. In this work, they were summed together to capture the combined importance of each pixel. The maximum approach was however attempted when visualizing the maps, which resulted in less smooth visualizations.

When performing the noise experiments, space maps were generated by combining 64 random encoding maps. More maps could have been used, making the model’s facial interest more explicit. However, this would increase the runtime of the experiment, and was not done because of it.

Even if all possible random encodings were to be generated and combined into a space, it would not capture all of the model’s interest areas. This is because there is not a one-to-one mapping between the image areas and possible encodings. Multiple area combinations can represent the same encoding. The space maps only presents a limited subspace of the true space.

When evaluating the model’s dependencies to certain pixels in the image, noise was applied from two directions. This was done in order to measure the pixel’s ability to damage and preserve the model’s performance. However, when measuring a pixel’s impact for an image that already is exposed to noise, the measuring might be disturbed. When for instance
measuring the impact of a pixel located in the middle of the ordering, 50% of the image will already be discarded. This does most certainly affect the measurement of the model's dependency in the pixel. However, this approach was used to plainly make use of the visualization's natural hierarchical ordering, and keep the runtime low. Evaluating a visualization technique in one direction on LFW View 1 did already take 5 hours. However, assuming that the dependency measurements were done correctly, and were not disturbed by distortion of other pixels, the area does indeed represent the hierarchical correctness of the visualization technique.

It could be questioned why the pixels are removed from significant and insignificant directions. A possibly simpler method would be to remove one pixel at time and measure its impact on the performance, and then compare it with the other pixels to identify its correct location in the dependency ordering. However, the performance impact of removing a single pixel is most likely negligible, where more pixels would be needed to be removed to obtain a measurable difference. This introduces a problem of which pixels to remove together. The method used in this work solves this problem by simply removing pixels according to the visualization technique itself.

The evaluations were done on LFW View 1, and not on the more natural choice LFW View 2, because of code conveniences. View 1 is also smaller than View 2, deceasing the runtime.

**Replicability**

Trying to replicate this work would most likely give similar, but different results. Training a neural network to obtain the same final weights is close to impossible. This is because of the massive quantity of weights, where every one of them is initialized randomly. Another important factor is the deep learning framework used, where small differences such as number of bytes used to represent a floating-point or rounding estimations could result in differences. However, despite that the models might not perform equally, the outcome of the visualization comparisons should be similar. The exposure experiments were done on a training set of 2,200 pairs and a test set of 1,000 pairs. Evaluating the performance effect on a relatively large set of pairs does reduce the chance of the results being an outcome of particular images.

**Source Criticism**

The theory of this work and the decisions made are mostly based on published work. This ensures some degree of validity of the information used. However, online sources regarding Caffe framework and other open-source code were also used. These were only used for implementation purposes.

**5.3 Wider Context**

Neural networks are extremely complex models with a massive number of weights, making it hard to know what the model uses, and how it makes its decisions. Because deep learning is a growing area, where both digital and physical systems might be affected, it raises some ethical and social questions. For instance, face recognition is already used by many intelligence agencies today to identify persons of interest. Relying blindly on these systems' ability to accurately recognize/verify identities is often the only way, because the systems are in themselves too complex to fully understand. However, despite a neural network's ability to make predictions, it can always make an error. This can be seen by simply training a neural network to approximate for instance an interval of a sinus curve. The model will be able to approximate the interval of points it is given, but have much harder (and often wrongly make predictions) on points outside the interval. This emphasizes that the observations that have not been covered can go completely wrong. This could in turn incorrectly lay the ground for juridical decisions, when used for the context of facial verification.
Three visualization techniques were created in attempts to capture what the model considers to be important in the image, when performing facial verification.

The activation maps (similar to sum-space maps) present what is most discriminant and critical in the image. However, they do only capture the peaks of what is important, which in turn is an outcome of the discriminant parts being present in conjunction with other facial areas. These areas are not enough for making good predictions, but are still the most important areas.

Max-space maps present the model’s interest areas in general, but do not capture the peaks as strongly. Max-space does also more correctly represent the model’s hierarchical dependencies in the image. This visualization technique is more suited for answering the question “what the model uses” than “what is most important”.

A method for determining hierarchical dependency correctness was established and was used to compare the visualization techniques. The same method was applied in another context to evaluate how the models were affected by artificial pixel noise. This could in turn emphasize that a model with a lower difference area (according to the method) makes use of a smaller image area when making predictions.

The 1-channel network was in this work less affected by any type of artificial pixel noise, compared to the 3-channel network. This could be because the 1-channel network takes less color channels (less input data). However, because of the networks being of different sizes and capacities, further research has to be conducted to conclude if a 1-channel network is less sensitive to artificial pixel noise, than a 3-channel network, in general.

Future Work

Activation (and sum-space) maps present the most critical areas, while max-space maps present important areas in general. It would be interesting to further investigate new visualization techniques which better represent the model’s hierarchical dependencies in the image, presenting both the critical and the generally important areas.

The space-maps in this work present only a subspace of the true space of important areas. This is because there is not a one-to-one mapping between the image and the encodings. For each layer, the mapping becomes less inverse representable. It would be interesting to visualize the dependencies for different encodings, of several layers. This could be approached
in a tree fashion, where each layer has multiple possible representations in the earlier layer, spanning out to more possible lower-layer encodings. This would result in many possible low-layer encodings to take into consideration, but still dramatically reduces the number of interesting input combinations.

The method of measuring the difference area of insignificant and significant performance when exposed to pixel noise does represent the hierarchical dependencies, if the measurements are assumed to be correct. However, measurements of pixels’ impact on the performance might be disturbed by other noised out pixels. It would be interesting to measure in a way which is sure to be robust to the noised out disturbance, while still being practically possible in terms of computational resources required. For instance, to expose noise based on sub-intervals from the visualization ordering.

It would also be interesting to use the techniques and methods of this work and apply them for usual classification problems. The activation map is similar to the work of Simonyan et al.[25]. However, max-space and sum-space could be modified to show the space of possible outcomes of a class score function. These visualization techniques could be evaluated using the method of this work, incrementally treating pixels as noise. This also applies to evaluating the model in terms of robustness to artificial noise (how much of the image the model uses).
Bibliography


