Metric learning versus Classification for Facial Recognition Model Robustness Against Adversarial Attack.

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Mchechesi Innocent Amos
2497239

Supervised by Dr Richard Klein

June 10, 2023

A research report submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science by coursework and research report in Artificial Intelligence

Abstract

Facial recognition using deep learning models has gained much attention because of its high performance and ability to represent features in the most abstract manner enabling the models to extract the most important features. Researchers found that these deep learning models are susceptible to adversarial attacks, which have the ability to fool them into producing incorrect outputs. Many researchers have looked into methods to make these models robust. Still, they mainly focus on classification models, and adversarial attacks on metric learning models have not received as much attention. In this research, the vulnerability of classification and metric learning models against adversarial attacks was compared.Various adversarial techniques were explored to assess their effects on classification and metric learning approaches in the context of improving model robustness against multiple attacks. The performance comparison revealed that the triplet loss with ResNet-50 backbone outperformed all other models, while the SENet with Cross-Entropy exhibited the lowest performance among the approaches studied.

Declaration

I, Mchechesi Innocent Amos, hereby declare the contents of this research report to be my own work. This report is submitted for the degree of Master of Science in Artificial Intelligence at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.

Signature: 

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the world advances technologically, deep learning is taking centre stage in the artificial intelligence fraternity. Particularly, in facial recognition (FR), deep neural networks (DNNs) lead the race because of their scalability, ability to extract and represent features cost-effectively, and high accuracy scores [Croak 2021].

It is widely understood that adversarial attacks can be used to exploit and fool deep learning algorithms, which mix imperceptible noise into data set samples; this noise is not visible to humans but affects the way that the network extracts features [Huang *et al.* 2015]. When specific noise or perturbations are added to the samples, they cause the deep networks to give incorrect outputs. An adversarial attack is a manipulation of a model to produce wrong outputs. This is done by passing adversarial samples through the model. Adversarial samples are input examples that carry a carefully and intentionally added perturbation, invisible to the human eye but causing the model to give an incorrect output. As shown in fig 1.1, adversarial examples are produced using many different methods from an adversarial creation process. These methods aim to calculate specific noise to make the CNN give high-confidence classification to wrong samples. Given a sample of class y, adversarial noise is calculated so that the model calculates features which leads it to embed the image incorrectly into the features space so that it is misclassified as some other target class, x. Adversarial examples are found in the latent space between decision boundaries and regions of correct classification. This region was found to be significant [Goodfellow *et al.* 2016].

To counter this, models are protected from adversarial examples using methods different methods which include like adversarial training, Bayesian uncertainties, defense distillation, and model ensembles just to mention a few.

There has been a race to make facial recognition models robust since it can be a security risk if models are susceptible to such attacks. As robustness was explored, weaknesses were also explored, which led to different types of adversarial attacks being discovered. There are adversarial attacks that are made for classification models and some made for metric learning models. All these are done through adversarial images but with a slight difference in that adversarial attacks in classification fool models to *misclassify*.

In metric learning, adversarial attacks fool models to embed images into the wrong region of the latent space.

Facial recognition is commonly characterized as a task that involves identifying and characterizing face images, and lately, videos [Raji and Fried 2021]. There are different tasks in facial recognition which include recognition, classification and verification. Recognition is found in both classes, classification is classifying a picture to its rightful identity and verification is verifying that one particular correctly classified identity.

Facial recognition is done through many different methods, which vary from 1) Principal Component Analysis [Tang and Wang 2002] 2) Support Vector Machines [Heisele *et al.* 2007] 3) Genetic Algorithms/Particle swarm optimization [Seal *et al.* 2015] to 4) Neural Networks (Deep learning). Deep learning is, however, found to significantly outperform other machine learning models [Coe and Atay 2021]. The modern deep learning models make use of convolutional neural networks(CNN), for example, the *EfficientNetB0* [Almadan and Rattani 2021], *CoAtNets (Convolution and Self-Attention Network)* [Kvak 2022], *FixEfficientNet-l2* which works by fixing the training protocol on EfficientNET [Touvron *et al.* 2020] and DeepFace's *SFace* [Zhong *et al.* 2021].

The networks learn features of individual samples through multiple convolutional and neural layers. Each subsequent layer performs further feature extraction and transformation. The features learnt are at different abstraction levels, making these models perform well even when pose and facial expressions vary. Architectures in this domain vary dramatically but mainly fall under **classification** and **metric learning**. In this research, the investigation aims to analyze the disparities between the two approaches concerning their vulnerability to adversarial attacks.

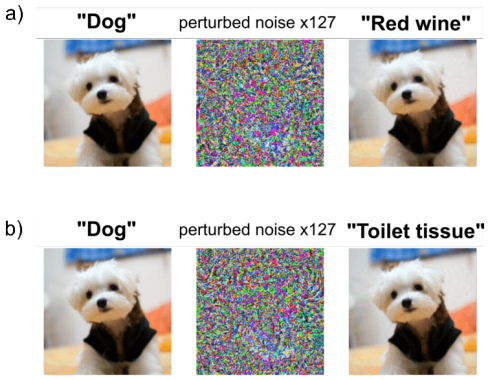


Figure 1.1: Adding perturbation to an original input example, causing the classifying model to give an incorrect output [Machiraju *et al.* 2021].

Much work has been done on adversarial attacks in facial recognition, but there are few publications on adversarial attacks in metric learning. The focus lies in exploring attacks from both classification and metric learning perspectives and assessing their effectiveness within these learning paradigms. The investigation also aims to analyze the vulnerability of the models to these attacks, with the objective of determining which models exhibit better robustness and safety.

Based on the study conducted, it is observed that metric learning models, in general, exhibit lesser vulnerability to adversarial attacks compared to classification models. A comparison was made between the output confidences of two model types, with and without adversarial defenses. It was found that metric learning models tend to display higher confidence in correct classifications and lower confidence in cases of misclassification.

1.1 Contributions

In this section, major contributions to this research were outlined.

- Tested the vulnerability of metric learning-based models and classification-based models against adversarial attacks without adversarial defense methods.
- Tested the vulnerability of metric learning-based models and classification-based models against adversarial attacks with adversarial defense methods.

1.2 Report organisation

The subsequent chapters are structured in the following manner: chapter 2 explores literature and previous work done. Chapter 3 outlines the methodology which used to tackle the research; this is done in multiple phases. Chapter 4 lays out the research plan defined as per the phases outlined in the preceding chapter. Chapter 5 provides a discussion of the results and concludes the report.

Chapter 2

Background Work

2.1 Introduction

This chapter contains a literature review, which covers the technical background concepts and related work in the space. Firstly, concepts and work around facial recognition are considered. This is followed by an exploration of the key concepts under these subjects, which includes state-of-the-art models, training and evaluation, and common attacks and defences of these models. Specifically, Section 2.2 covers deep learning, and Section 2.3 covers facial recognition algorithms. Section 2.3.5 presents adversarial attacks for classification and metric learning. Section 2.4 presents defences against these attacks.

This research focuses on images, however, in this chapter, some information will be obtained from work done on videos.

2.2 Deep Learning

Deep learning has its roots spanning from 1943 when [McCulloch and Pitts 1943] created a neural network model [Foote 2022]. This subject has garnered much attention in practice and research fraternities enabling people to carry out complex machine-learning tasks. Because of the improvements in computing power, researchers have managed to do work and implement deep learning on complex tasks. It is generally defined by [Goodfellow *et al.* 2016] as a powerful type of machine learning type which is capable of learning the real world and representing it in abstract and simpler terms which form a nested hierarchy. This means the capability to learn the complex world, breaking it down into simpler separate representations or concepts, ignoring the unnecessary information for the sake of differentiating uniqueness, in a way only remaining with the most important information which defines a class.

This is achieved by using neural networks with many layers, thus having many parameters. Deep learning is done in various ways henceforth classified according to the technique and architecture used, some of which are the following:

- Supervised learning - also known as supervised-machine-learning. It uses labeled data to accurately train models and algorithms.
- Unsupervised learning - also known as unsupervised-machine-learning, works by analysis on unlabeled data and finds out the concealed patterns in data with without human aid.
- Reinforcement learning - It is concerned with teaching an agent to make a series of choices in response to feedback. In reinforcement learning, the agent interacts with an environment and receives feedback in the form of rewards or penalties based on the actions it performs. Reinforcement learning aims to optimize the cumulative reward over time by learning a policy that links states to actions.

CNNs have exceptional architectural proficiency for image recognition. In face recognition, CNNs have become the powerhouse since they have reached great success in this field.

2.2.1 Data Sets

In this section,a summary of publicly available datasets and those which are accessible with reasonable effort is presented.

The available datasets are shown in 2.1.

Table 2.1: Publicly available data sets for Facial Recognition

Name	Year	IDs	Images	Cropping Needed	Labeled
LFW [Huang *et al.* 2008]	2007	5749	13 233	Yes	Yes
Multi-Pie [Gross *et al.* 2010]	2010	337	750 000	Yes	Yes
CelebFace [Sun *et al.* 2014]	2014	10 177	202 599	Yes	Yes
MS-Celeb-1M [Guo *et al.* 2016]	2016	99 892	8 456 240	Yes	Yes
CASIA-WebFace [Yi *et al.* 2014]	2014	10 575	500 000	Yes	Yes
Wider Face [Yang *et al.* 2016]	2016	32 203	393 703	Yes	Yes
CAS-Peal [Gao *et al.* 2007]	2008	1040	99 594	Yes	Yes
FERET [Phillips *et al.* 2000]	1996	1199	14 126	Yes	Yes
VGGFace2 [Cao *et al.* 2018]	2017	9131	3 310 000	Yes	Yes
MegaFace2 [Nech and Kemelmacher-Shlizerman 2017]	2017	672 057	4 753 320	Yes	Yes

2.3 Facial Recognition Algorithms and Methods

Facial recognition is about detecting a face and adding a correct identity to it. As much as there are many methods and algorithms for doing it, in this research,deep learning models were used. The methods were categorized into classification and metric learning during the study. The first parts of these pipelines are the same: firstly there is face detection in the input images, then input face processing, then training of the recognition model, then feature extraction and the features are fed into the loss function for classification and distance metrics are applied to the features for metric learning.

Face recognition deep learning performance is influenced by pose, illumination, expression and occlusions[Mehdipour Ghazi and Kemal Ekenel 2016]. Consequently, to deal with this particular problem face processing is then put in place. The common face

processing methods are classified as “one-to-many augmentation” and ”many-to-one normalization”. The technique of one-to-many augmentation involves generating multiple images with diverse pose variations from a single image, which aids deep neural networks in understanding representations that are invariant to pose. On the other hand, many-to-one normalization involves deriving a standard pose from images with varying poses (such as non-frontal views), allowing facial recognition to be performed as if the images were captured under controlled conditions. The concept of many-to-one normalization, which utilizes Convolutional Neural Networks (CNNs), will be further elaborated upon. CNNs mostly learn the 2d representation mapping of non-frontal and frontal face images (*frontal images are face images facing to the front*) and use the mappings for image normalization in the pixel space. The displacement field network presented by [Hu *et al.* 2017] learns a shift transformation relationship between pixels in the frontal image and non-frontal face image in the translation layer using the displacement field. All non-frontal images are first transformed into frontal ones, which will be used for facial recognition, making the model pose invariant.

Different architectures are used for facial recognition models like AlexNet [Krizhevsky *et al.*], GoogleNet [Ballester and Araujo 2016], VGGNet [Simonyan and Zisserman 2014], with the state-of-the-art represented by ResNet [He *et al.* 2016] and SENet [Kvak 2022]. The difference then comes from the loss functions. Classification-based learning is based on the cross entropy loss. The networks usually have a softmax layer at the end with one neuron representing each of the possible classes. The cross entropy loss coupled with the softmax layer push the network to try match a one-hot encoded vector where 1 represents the correct identitiy. Metric learning, nonetheless , is based on pairwise sample similarity scores. The network creates an embedding of the input in some latent space where the loss function encourages similar identities to be clustered together in different regions. Functions like the contrastive or triplet loss, aim to reduce the variation within a given class while simultaneously increasing the differences between different classes in the latent space.

Section 2.3.1 now presents various network architectures, followed by classification and contrastive loss functions in sections 2.3.3 and 2.3.4.

2.3.1 Network Architectures

SENet

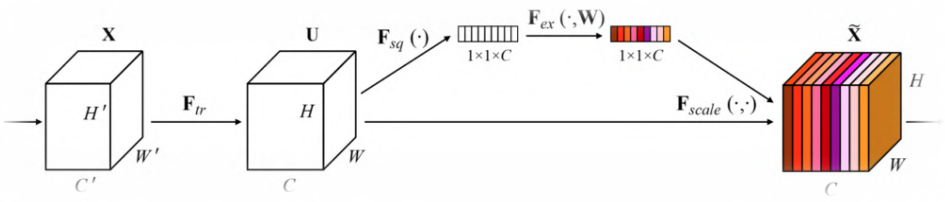


Figure 2.1: Squeeze and excitation representation [Hu *et al.* 2018]

SENet (Squeeze-and-Excitation Network) shown in figure 2.1 is a type of CNN that incorporates a squeeze-and-excitation (SE) module into the architecture to better capture inter-channel dependencies and improve the representation learning capabilities of the network [Hu *et al.* 2018].

The SE module is created to dynamically adjust the channel-specific feature responses through modeling the connections between channels. The SE module has two key parts: a compression step and an activation step.

The squeeze operation reduces the spatial dimensions of the feature maps while maintaining the channel dimension. This is done by using global average pooling, which computes the average of each channel over all the spatial locations. The output of the squeeze operation is a vector with the same number of elements as the number of channels in the feature maps.

The excitation operation then uses this vector to recalibrate the channel-wise feature responses. It does this by first transforming the vector through a fully connected (FC) layer with a ReLU activation, followed by another FC layer that scales the output of the ReLU layer. The output of the excitation operation is also a vector with the same number of elements as the number of channels in the feature maps.

The output of the excitation operation is then element-wise multiplied with the original feature maps to produce the final output of the SE module. This has the effect of adjusting the degree of significance of the different channels based on the interchannel dependencies learned by the SE module.

Mathematically, the SE module can be represented as follows in two parts which are squeeze operation and excitation operations:

- Squeeze operation

$$z = GlobalAveragePooling(x) \tag{2.1}$$

- Excitation Operation

$$r = ReLU(FC(z)) \tag{2.2}$$
$$s = FC(r) \tag{2.3}$$

- Output

$$y_{ij} = s \odot x_{ij} \tag{2.4}$$

where, $x \in \mathbb{R}^{W \times H \times C}$ represents the input feature maps, $y \in \mathbb{R}^{W \times H \times C}$ is the output of the SE module, $z \in \mathbb{R}^{C}$ is the output of the squeeze operation, $r \in \mathbb{R}^{C}$ is the output of the ReLU operation, $s \in \mathbb{R}^{C}$ is the output of the scaling operation and $\odot$ represents the elementwise multiplication at each spatial location. The SE module is typically inserted between the convolutional layers of a CNN, and the network is trained end-to-end to learn the parameters of the SE module and the convolutional layers.

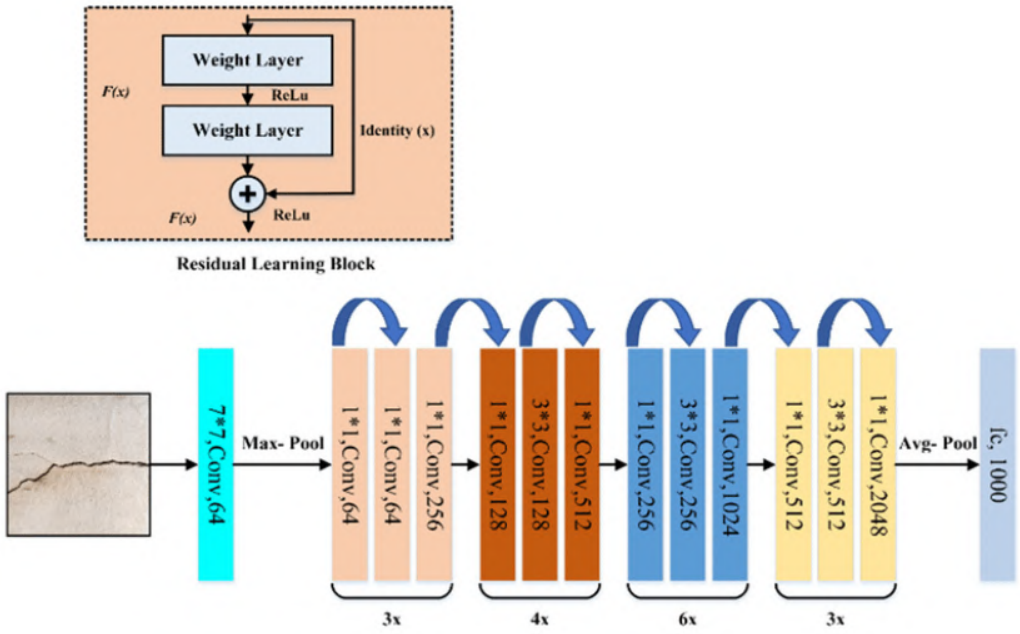


Figure 2.2: Residual Network architecture representation [He *et al.* 2015]

RESNet

Residual Network (ResNet) [He *et al.* 2015] shown in figure 2.2 is a deep learning architecture known for alleviating the issue of vanishings gradient in neural networks, which is a common issue in deep networks that can prevent them from learning effectively.

In ResNet, the architecture is made up of elements called residual blocks, which are designed to allow the network to learn the residual mapping between the input and the output of each block rather than the actual mapping itself. The residual mapping is defined as the difference between the output of the block and the input, or $F(x) - x$ and $F(x)$ represents the output of the block.

The idea behind this approach is the fact that it is much undemanding for the network to learn the residual mapping than the actual mapping, especially for deep networks where the gradients can become very small and the learning process becomes difficult. By learning the residual mapping, the network can effectively "shortcut" the layers and directly learn the desired mapping, which can make the learning process more efficient.

Mathematically, the residual block can be represented as follows:

$$y = F(x, W) + x \tag{2.5}$$

Here, x is the input to the block, W are the weights of the block, $F(x, W)$ is the output of the block without the residual connection, and y is the output of the block with the residual connection.

ResNet typically consists of several residual blocks stacked on top of each other, with the output of one block serving as the input to the next block. The number of residual blocks can vary depending on the depth of the network.

VGG16

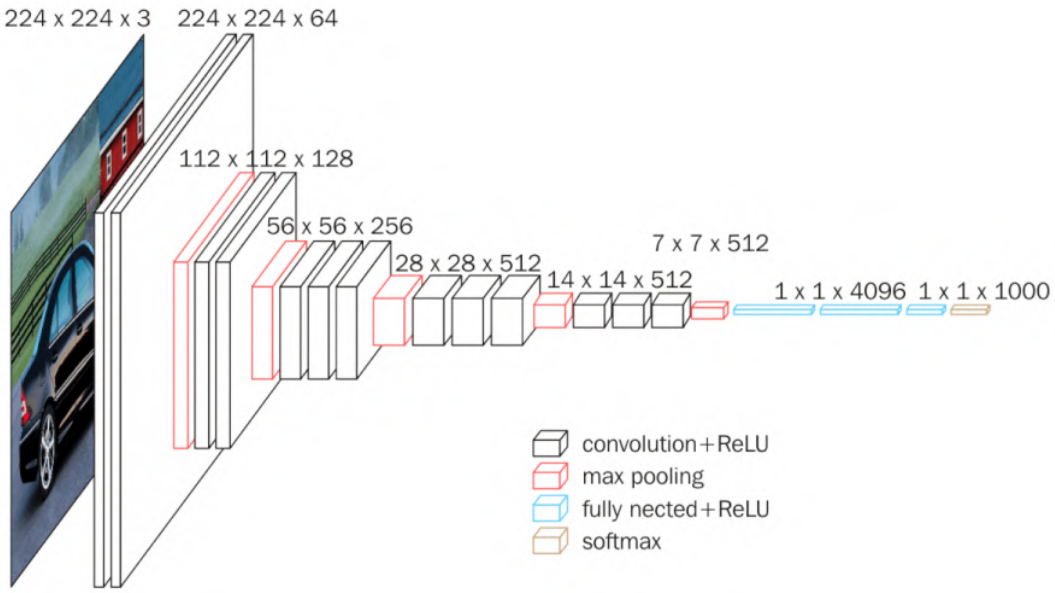


Figure 2.3: VGG16 architecture representation [Simonyan and Zisserman 2014]

VGG16 [Simonyan and Zisserman 2014] shown in figure 2.3 was developed by the Visual Geometry Group (VGG) at the University of Oxford as part of a larger research effort to improve the state-of-the-art in image classification. Specifically, VGG16 was designed for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which is an annual competition held to evaluate the performance of various algorithms on a large dataset of images.

One of the main goals of VGG16 was to develop a deep CNN architecture that could attain high performance on the ILSVRC classification task. To achieve this, the authors of the VGG16 [Simonyan and Zisserman 2014] paper focused on two main design principles: increasing the depth of the network and using small convolutional filters.

Expanding the network depth refers to adding more layers to the CNN, which allows it to learn more complex features from the input examples. This can be especially beneficial for image classification tasks, as images often contain a huge number of features that has to be captured in order to correctly classify them.

Using small convolutional filters, on the other hand, allows the network to capture fine-grained details in the input data. This can be especially important for tasks like image classification, where small details in the input image can be crucial for making accurate predictions.

Overall, VGG16 was designed to be a very deep CNN architecture with a focus on small convolutional filters, in order to achieve high performance on the ILSVRC classification task. The success of VGG16 in this challenge and its widespread adoption in the area of computer vision demonstrate the effectiveness of these design principles

Xception

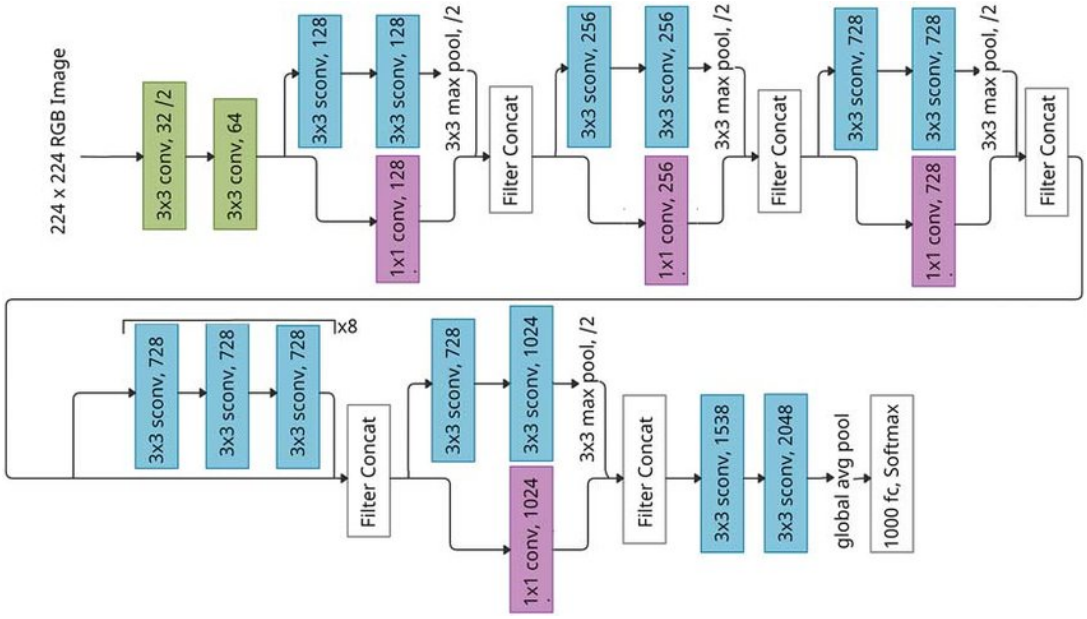


Figure 2.4: Xception architecture representation [Chollet 2017]

The Xception model [Chollet 2017] shown in figure 2.4 is a CNN architecture that was developed by François Chollet for image classification tasks. It is named ”Xception” because it was designed to be a ”deep learning with separable convolutions” architecture, which is a type of CNN that uses depthwise separable convolutions to bring down the number of parameters in the model while maintaining a similar level of performance.

The basic idea behind a separable convolution is to first apply a depthwise convolution, which convolves each input channel separately with a set of filters, and then apply a pointwise convolution, which combines the output of the depthwise convolution using a set of 1×1 filters. As a consequence, the model becomes more efficient as it employs fewer parameters than the conventional use of 3x3 or larger convolutional filters.

The Xception model is made up of a series of blocks, each of which contains a sequence of separable convolutional layers followed by a skip connection. The skip connection allows the model to bypass the convolutional layers and directly pass the input through to the output, which helps to improve the model's ability to preserve spatial information and reduces the risk of overfitting.

MobileNet

MobileNet [Howard *et al.* 2017] is a convolutional neural network (CNN) architecture that was designed to be efficient and lightweight, making it well-suited for use on mobile and embedded devices with limited computational resources. It is based on the idea of using ”depthwise separable convolutions” to build deep neural networks that are faster and more memory-efficient than traditional CNNs.

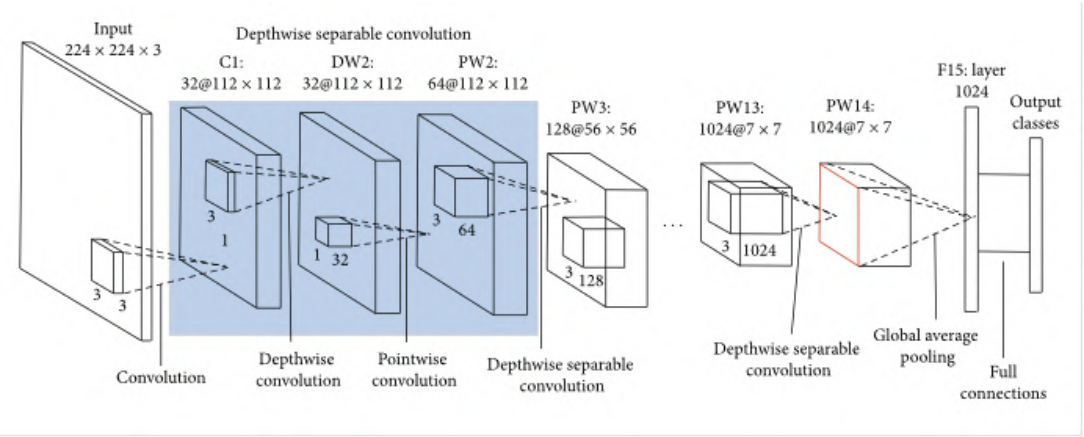


Figure 2.5: MobileNet architecture representation [Howard *et al.* 2017]

In a traditional CNN, a convolution operation typically consists of applying a set of filters to the input data, producing a set of output "feature maps". Each filter is a small matrix that is applied to the input data using a sliding window-like operation, and the resulting feature maps capture different aspects of the input data. Typically the convolution has an equal number of channels with the input, for example $3 \times 3 \times C$ for a 3×3 kernel operating on a C channel input.

In a depthwise separable convolution, the filters are applied separately to each input channel (e.g. each color channel in an image). This is called the "depthwise" part of the operation. Then, a separate set of filters (called "pointwise" filters) is applied to the resulting feature maps to combine the information across channels. This allows the model to learn more efficiently, since the depthwise filters can learn to capture local patterns within each channel, while the pointwise filters can learn to combine this information in a higher-level way.

MobileNet uses a combination of depthwise separable convolutions and regular convolutions to build a deep neural network. It also uses a number of other techniques to make the model more efficient, such as using lightweight "bottleneck" layers and using smaller filters in the early layers of the network

2.3.2 Classification confidence

Classification confidence [Athalye *et al.* 2018; Shafahi *et al.* 2019], refers to the level of certainty or probability that a machine learning model assigns to its predictions. In a classification task, the model assigns a label to each input based on a set of learned features, and the confidence score indicates how sure the model is that the predicted label is the right one. Confidence scores are often represented as a probability, with a value between 0 and 1, where 1 represents the highest level of confidence.

There are different ways to measure the effectiveness of adversarial attacks, but one commonly used measure is the confidence score of the adversarial examples, which is the probability assigned by the model to the misclassified label and the other is the

success rate, which is the proportion of inputs that are misclassified by the model after the attack.

2.3.3 Classification Loss Functions

Classification loss functions all take in logits and produce a probability distribution of classes which will be used for classification, such that if the output is below a certain confidence threshold, the prediction will be 0.

Cross-Entropy loss

The Cross-Entropy loss function (cross entropy plus sofmax) $sm : \mathbb{R}^d \to \mathbb{R}^d$

$$sm(\mathbf{z})_i = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T x_i + b_j}} \tag{2.6}$$

where N is the batch size and n is the number of classes. $x_i \in \mathbb{R}^d$ represents the features of the i^{th} sample of the y_i^{th} class. d is the feature dimension. $W_j \in \mathbb{R}^d$ is the j^{th} column of the weight $W_j \in \mathbb{R}^{d \times n}$ and $b_j \in \mathbb{R}^d$ represents the bias.

To clarify, the exponential function is applied to each element $z_i = W_{y_i}^T x_i + b_{y_i}$ of the input vector **z**, which the resulting values and the logits from the fully connected neural-network layer are normalized just by diving them with the summation of all exponential values. This normalization ensures that the elements of the output vector $sm(\mathbf{z})$ add up to a unit (1) just like a probability distribution.

Softmax alone is a function used to normalize logits to probabilities. Cross-entropy loss is calculated by taking the difference between the prediction and actual outputs, it is calculated as shown below.

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{2.7}$$

Where:

1. M - number of classes
2. log - the natural log
3. p - predicted probability observation o is of class c

ArcFace - Additive Angular Margin Loss

$$L_3 = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{s(cos(\theta_{y_i}+m))}}{e^{s(cos(\theta_{y_i}+m))} + \sum_{j=1,j\neq y_i}^{n} escos\theta_j} \tag{2.8}$$

In trying to optimize the feature embedding so that intra-class distances are minimized, and inter-class distances are maximized [Deng *et al.* 2019] fixes the bias to 0 and introduces the angle θ_j between weights Wj and feature x_j such that the logits (input vector) $\mathbf{z} = \|W_j\|\|x_j\|cos\theta_j$. The $\|W_j\| = 1$ individual weights are fixed by l_2 normalization. The features $\|x_j\|$ are also fixed to a hypersphere s by l_2. The normalizations are done to make the predictions solely θ dependent. An angular margin m is also introduced between the weights and features to amplify the intra-class closeness and inter-class looseness.

Ring Loss

$$L_R = \frac{\lambda}{2N}\sum_{N}^{i=1}(\|x_i\|_2 - R)^2 \tag{2.9}$$

where: R is the learnt target norm value, λ is the loss weight which enforces a trade-off between the primary loss functions. N and x_i represents the same as in Cross-Entropy and ArcFace.

Ring loss works with the Softmax, where Cross-Entropy loss will be the primary loss function. The loss is then minimized but subjected to a normalization constraint R. R is the scale constant to which features should be normalized. There the loss is minimized as follows:

$$minL_s(x) \;\; s.t. \|x\|_2 = R \tag{2.10}$$

The ring loss by [Zheng *et al.* 2018] provides better-informed gradients because the normalization is learnt, not forced. It stabilizes the feature norm for all classes, thereby rectifying the Cross-Entropy classification problems.

2.3.4 Metric Learning

Metric learning loss functions try to measure the distance between samples in the latent space they give a similarity score between the two samples. In Metric learning, care should be given to how sample mining is done during training. Mining is the process of choosing two or three samples, and the loss aims to increase/decrease the distance between those samples in the latent space. Typically, the metric loss functions aim to decrease the distance between images of the same identity, while increasing the distances between images of different identities.

Triplet Loss

Triplet loss [Hoffer and Ailon 2015] works over three samples, the anchor (a), the positive sample (p) and the negative sample (n). The margin, m, encourages the negative distance to be m units larger than the positive distance. $D_{a,p}$ and $D_{a,n}$ represent the distances between the samples in the latent space. The Triplet loss L_{tl} is defined as follows:

$$L_{tl} = \max(0, m + D_{a,p} - D_{a,n}) \tag{2.11}$$

Minimising the triplet loss means minimising the distance between similar sample $D(a, p)$, which is lower-bounded by 0, and maximising the distance between dissimilar samples $D(a, n)$, which can be arbitrarily large. The idea of the margin term is to make all positive samples (samples from the same class) closer together than any sample from a different class without collapsing to one point.

Triplet mining is essential in triplet loss to select good samples to train with. If the triplets are too easy, the network does not learn, but if the triplets are too hard, the training process becomes unstable. Hard positives are samples of the same class with the anchor but far in the latent space, and hard negatives are samples from a different class with the anchor but close together in the latent space. It was seen that selecting hard samples is unstable hence [Schroff *et al.* 2015] resorted to semi-hard samples, where the negative is not closer to the anchor than the positive, but which still have a positive loss. The mining of semi-hard samples is governed by the formula below:

$$d(a, p) < d(a, n) < d(a, p) + margin \tag{2.12}$$

In the original FaceNet paper, the authors used a CNN architecture with a multi-task learning objective as the backbone. The specific details of this architecture are not provided in the paper, but it is described as composed of a series of convolutional and fully connected layers. The primary focus of the FaceNet paper is on using the triplet loss function. However, it is worth noting that the FaceNet model can be implemented using various backbone architectures.

Contrastive Loss

The contrastive loss [Wang and Liu 2021] makes use of sample pairs which are an anchor (a) and positive (p) or an anchor (a) and negative (n) to minimize the distance between a and p and maximize the distance between a and n. In other words, the contrastive loss performs like the triplet loss but one by one rather than simultaneously. The margin m is also present, which will be the defining threshold for a sample to be positive or negative. The Contrastive Loss L_c is defined as:

$$L_c = y.D_{1;2}^2 + (1 - y)\max(m - D_{1;2}, 0) \tag{2.13}$$

where $D_{1,2}$ is the Euclidean distance seperation the two embeddings which can be any combination of positive sample and negative samples and m is still the margin same as in Triplet loss.

Quadriplet Loss

The aim of the quadruplet-loss by [Chen *et al.* 2017] is to push negative pairs and positive pairs even further apart with respect to different images. It is developed from the triplet loss, but it adds a term to it as follows:

$$L_{quad} = \max(0, m_1 + D_{a,p} - D_{a,n}) + \max(0, m_2 + D_{a,p} - D_{n_1,n}) \tag{2.14}$$

The first term is similar to that in the triplet-loss, the second term as well and D_{n_1} is a second negative. This is said to help make the biggest distances between images of the same class to be in all cases smaller than the smallest distance between images of different classes.

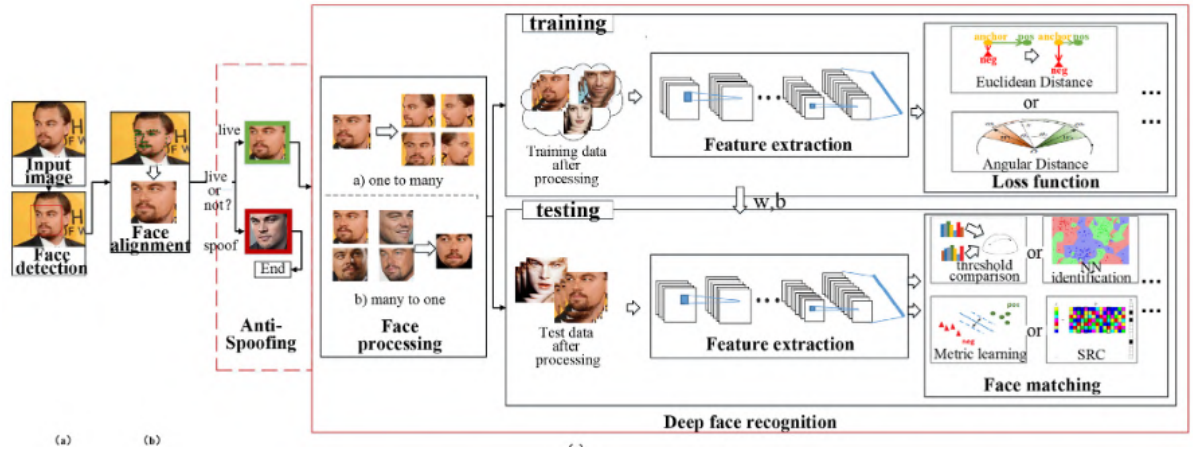


Figure 2.6: Generic Deep facial Recognition pipe line [Wang and Deng 2021]

Metric learning based classification

If you have a metric-learning-based facial recognition model, it only provides embeddings. If you want to perform recognition, you will need a classifier that can classify the embeddings produced by your model. The options for classifiers that will be discussed, which could be used for this purpose, are K-nearest neighbours (k-NN) and Support Vector Machines (SVMs).

The **k-NN** [Huang *et al.* 2014] classifier shown in algorithm 1is a simple and effective method for classification tasks. In the context of facial recognition, k-NN works by storing a set of labelled training examples (i.e. images of faces along with their corresponding identities) and using these examples to make predictions about the identity of a new face.

To classify a new face using k-NN, the classifier first computes the distance between the new face and each training example using a distance metric such as Euclidean distance in the latent space. It then selects the k training examples that are closest to the new face (where k is a hyperparameter of the classifier) and assigns the new face to the most common class among those k examples.

In a metric-learning-based classification system with a deep network, the k-nearest neighbour (k-NN) classifier is to be used as a post-processing step to make final predictions on test images based on the embeddings produced by the deep network.

SVMs [Hu *et al.* 2012] are a type of machine learning algorithm that can be used for classification tasks. In the context of facial recognition, SVMs construct a hyperplane in a high-dimensional feature space that separates the different classes of faces as well as possible. The hyperplane is chosen so as to maximize the margin between the closest training examples of different classes.

Algorithm 1 k-NN pseudocode [Guo *et al.* 2003]

procedure KNN(T, x)
    Set $S = \emptyset$
    for $i = 1$ to n **do**
        Calculate the distance between x and x_i using a distance metric such as Euclidean distance.
        Add (x_i, y_i) to S.
    end for
    Sort the elements of S in increasing order of distance from x.
    Select the top k elements of S.
    return the most common label among the selected elements.
end procedure

To classify a new face using an SVM, the algorithm projects the new face onto the feature space and assigns it to the class corresponding to the side of the hyperplane on which it falls.Support vector machines (SVMs) have shown potential in facial recognition tasks, particularly when the number of classes is limited and the decision boundary separating these classes is relatively uncomplicated.. However, SVMs can be sensitive to the choice of kernel function and other hyperparameters, and they may be less effective on large or complex datasets.

Triplet Mining

Triplet mining [Yu *et al.* 2018] is a technique that is used to learn a compact, embedding of low dimensional embedding of data points from a dataset by selecting triplets to assist training. A triplet is a set of three data points, consisting of an anchor point, a positive point, and a negative point. The goal is to learn a mapping from the data points to a compact embedding space such that the distance between the anchor and positive points is minimized, while simultaneously maximizing the distance between the anchor and negative points.

There are three different types of triplets that can be used in triplet mining, including:

1. **Hard triplets**: A set of triplets comprising an anchor point, a similar positive point, and a dissimilar negative point is employed, with the purpose of creating a discriminative embedding space through the use of hard triplets. This space ensures that similar points are positioned close to each other and dissimilar points are far apart

2. **Semi-hard triplets**: A collection of triplets is utilized, including an anchor point, a positive point that exhibits similarity with the anchor point, and a negative point that displays a certain degree of dissimilarity with the anchor point. Semi-hard triplets are commonly applied to construct an embedding space that possesses both discriminative capability and resistance to noise and data variability.

3. **Easy triplets**: A set of triplets is constructed, comprising an anchor point, a positive point that exhibits a high degree of similarity with the anchor point, and a negative point that displays a certain level of similarity with the anchor point. Easy triplets are frequently employed to develop an embedding space that is more robust to data variability and noise, albeit with a certain degree of sacrifice in discriminative ability.

These types of triplets are selected using two different methods, which are***Online Triplet Mining*** and ***Offline Triplet Mining*** [Tong and Li 2017].

1. In **online triplet mining**, triplets are generated and used for training in real-time, as the model is being trained. This means that the model is continuously presented with new triplets as it is learning, which allows it to adapt to the structure of the data and learn a more robust embedding space. Online triplet mining is often used when the dataset is very large or when it is not practical to pre-generate all of the triplets in advance.

2. In **Offline triplet mining**, all of the triplets are generated in advance and stored in a buffer, and then used to train the model in batch mode. This implies that during training, the model is exposed to a predetermined set of triplets and is unable to dynamically adjust to the data structure in real-time. Offline triplet mining is often used when the dataset is small enough to fit in memory, or when it is necessary to use a fixed set of triplets for training.

2.3.5 Adversarial Attacks

An adversarial attack is a result of carefully altering the original image, where the changes are imperceptible to humans yet capable of fooling a specific classifier. This could be performed by adding a small vector n to the original input-image x, i.e. x_{adv}= $x + n$, in this way the deep learning model F wrongly predicts an output for the tempered input x_{adv}. That way x_{adv} is an adversarial example. Rathgeb *et al.* [2020] describe a box-constrained optimization method for producing an adversarial example x_{adv} as follows:

$$\begin{aligned} \min_{x_{adv}} \| x_{adv} &- x \|_2 \\ \text{s.t. } F(x) &= l_{adv} \\ F(x) &= l \\ l &\neq l_{adv} \\ l_{adv} &\in [0, 1] \end{aligned}$$

where l and l_{adv} are respectively the output labels of x and x_{adv}. The susceptibility of CNN models to adversarial attacks was first demonstrated by [Huang *et al.* 2015] who introduced a small amount of noise to the input image. It was also shown that colour balance alterations are capable of lowering the accuracy of GoogleNet and VGG-Face models. The common attacks are as are mainly classification attacks since there have been FEW attacks for metric learning.

Fast Gradient Sign Method (FGSM)

Goodfellow *et al.* [2016] suggested the Fast Gradient Sign Method (FGSM) to computationally come up with an adversarial noise by solving the following problem in a fast and efficient manner:

$$\mathbf{n} = \epsilon \, \text{sign}(\bigtriangledown_{\mathbf{x}} L(x, y)) \tag{2.15}$$

where ϵ defines the magnitude of noise, $\text{sign}(\cdot)$ is the sign function, and $\bigtriangledown_{\mathbf{x}} L(\cdot, \cdot)$ represents the gradient of the cost function in the vicinity of the current state of the model's parameters regarding the matter of x. The produced adversarial example x_{n} is computed as $x + n$. The FGSM generates adversarial examples in a non-iterative manner by adjusting the gradient based on the sign of the gradient at each pixel in a single step . Miyato *et al.* [2018] suggested a similar way of generating noises and called it Fast-gradient L_2. The perturbation therefore is computed as follows:

$$\mathbf{n} = \epsilon \frac{\bigtriangledown_{\mathbf{x}} L(x, y)}{\parallel \bigtriangledown_{\mathbf{x}} L(x, y) \parallel_2} \tag{2.16}$$

The calculated gradient is normalized with its L_2-norm, as shown. Another option of using the L_∞-norm for normalization is there and it is called the Fast Gradient L_∞. Most authors and researchers classify these as one-step methods.

Randomized Fast Gradient Sign Method(RFGSM)

Randomized Fast Gradient Sign Method (RFGSM), is a variant of the Fast Gradient Sign Method (FGSM) developed by [Tramèr *et al.* 2017b]. It introduces randomness to the input data prior to running it through a machine-learning model. This is to avoid areas of non-smoothness in the loss function curve that occur when using the original input. It has also been found that gradients can poorly represent the loss function in such cases.

$$\mathbf{n} = (\epsilon - \alpha) \, \text{sign}(\bigtriangledown_{\mathbf{x}} L(x, y)) \tag{2.17}$$

Unlike the original FGSM, the RFGSM is iterative.

Basic Iterative Method (BIM)

BIM is an extension to the FGSM by [Kurakin *et al.* 2018]. The extension is through iteratively optimizing the pixel values of generated adversarial examples to make them more effective. For every single iteration, the pixel values were modified in a controlled way to prevent large changes from being made on each individual pixel. This helps to ensure that the generated adversarial image remains similar to the original image, while still being able to fool a machine learning model. It's described as follows:

$$x_{adv_0} \leftarrow \mathbf{x} \tag{2.18}$$

$$x_{adv_{n+1}} \leftarrow Clip_{x,\epsilon}\{x_{adv_n} + \alpha sign \bigtriangledown_{\mathbf{x}} L(x_{adv_n}, y)\} \tag{2.19}$$

In each iteration, the values of the generated adversarial image are controlled by a "clip function": (*Clip*), which limits the changes that can be made to the image, this makes

sure that all pixels are within the bounds of epsilon and the original maximum and minimum pixel values.

Projected Gradient Descent(PGD)

The PGD [Bai *et al.* 2020] method is suggested as a standard approach for large-scale optimization with constraints. When generating an adversarial example, PGD first starts the search for an adversarial example at a randomly chosen point within a defined area (the "lp norm sphere"). It then performs several iterations, similar to the BIM, to find an adversarial example. It is defined as follows:

$$x_{adv_{n+1}} = \Pi^{\epsilon}\{x_{adv_n} + \alpha sign \bigtriangledown_{\mathbf{x}} L(x_{adv_n}, y)\} \tag{2.20}$$

The Π is a projector function that maps the point reached in each iteration back to the non-trivial set. The Π is defined as:

$$\Pi(z) = \min_{x_{adv} \in P_x} \|x_{adv} - z\|_2 \tag{2.21}$$

where z is the point which the optimization happens and P_x is the non-trivial set.

Bai *et al.* [2020] introduces the same above-mentioned adversarial creation methods but this time specifically attacking the distance metric loss function. In place of the cross entropy loss function, there will be:

$$\frac{L(D(P, X))}{L(X)} \tag{2.22}$$

where $D(P, X)$ is the distance between two samples. [Bai *et al.* 2020] based his work on the assumption that metric learning models will be better robust against classification models' adversarial attacks because of the way they are trained hence there is a need to create adversarial examples for metric learning.

Randomised Projected Gradient Descent(RPGD)

Randomized Projected Gradient Descent (RPGD) is a variation of the original Projected Gradient Descent (PGD) algorithm proposed by [Madry *et al.* 2017]. It involves adding randomness to the input data before feeding it into a machine learning model. The aim is to prevent optimization from converging to local minima and, instead, explore the possibility of achieving the global optimum. It may also be that a single gradient descent is insufficient to solve the problem. The use of randomization helps to avoid getting stuck in a local minimum and allows the optimization process to reach the global optimum.

Carlini and Wargner attack (C & W)

The Carlini and Wargner attack (C & W) was introduced by [Carlini and Wagner 2017]. The mathematical formulation of the C & W attack involves defining an objective function that measures the distance between the original input and the adversarial example,

as well as the probability of the model making a mistake. The objective function can be expressed as follows:

$$J(x, y) = d(x, x') + c * \max(0, 1 - y' * y), \tag{2.23}$$

where:

- x is the original input
- y is the true label for the input
- x' is the adversarial example
- y' is the predicted label for the adversarial example
- $d(x, x')$ is a distance function that measures the similarity between the original input and the adversarial example
- c is a constant that controls the relative importance of the two terms in the objective function

The first term in the objective function, $d(x, x')$, measures the distance between the original input and the adversarial example and ensures that the adversarial example is close to the original input. The second term, $c * max(0, 1 - y' * y)$, measures the probability of the model making a mistake and is maximized when the model is most likely to make a mistake.

To generate an adversarial example, the C & W attack involves solving an optimization problem that involves finding the values of x' that minimize J(x, y). This can be done using optimization algorithms such as gradient descent or stochastic gradient descent.

2.4 Adversarial Defences

Adversarial defenses are used to defend models from adversarial attacks. These are ways of protecting models from misclassification in case of attacks. Several methods will be discussed, including Bayesian Uncertainties, Adversarial Training, Defense Distillation, Model Ensembles, and MagNet.

2.4.1 Bayesian Uncertainities

One prominent method is the use of Bayesian Uncertainties for detecting Adversarial examples. The Bayesian inference for deep learning learns a posterior distribution $p(w|D)$ over the weights, this is then used in predicting new data as shown in the equation below:

$$p(y|x, D) = \int p(y|x, w), p(w|D)dw \tag{2.24}$$

To make it suitable for deep learning models since there are huge numbers of parameters, [Theagarajan and Bhanu 2020] uses Bayes by backpropagation which is a variational inference technique to learn the posterior distribution of the weights of a CNN

from which the weights can be sampled in backpropagation. Through the utilization of this approach, an approximated distribution, denoted as $q_\theta(w|D)$, is obtained, where θ represents the distribution parameters. The aim is for this distribution to closely approximate the true posterior $p(w|D)$, as measured by the KL divergence. The optimal parameters will then be defined as:

$$\theta_{opt} = arg\min_{\theta} KL(q_\theta(w|D) \parallel p(w)) - \mathbb{E}_{q_\theta(w|D)}(logp(D|w)) + logp(D) \tag{2.25}$$

After the approx posterior distribution has learnt the Bayesian uncertainties are given by:

$$\text{Aleatoric}_{\text{uncertainty}} = \frac{1}{T}\sum_{t=1}^{T} diag(\hat{g}_t) - \hat{g}_t\hat{g}_t^T \tag{2.26}$$

$$\text{Epistemic}_{\text{uncertainty}} = \frac{1}{T}\sum_{t=1}^{T}(\hat{g}_t - \check{g})(\hat{g}_t - \check{g})^T \tag{2.27}$$

T is the number of samples drawn from the posterior distribution, $\check{g} = \frac{1}{T}\sum_{t=1}^{T}\hat{g}_t$ and $\hat{g}_t = f_{w_t}(x)$. After training the Bayesian CNN, average μ and standard deviation σ for the two uncertainties are calculated and thresholds are defined as follows:

$$t_1 = \mu(\text{Aleatoric}) - 3\sigma(\text{Aleatoric}) \tag{2.28}$$

$$t_2 = \mu(\text{Epistemic}) - 3\sigma(\text{Epistemic}) \tag{2.29}$$

If any or both of the uncertainties are greater than the corresponding thresholds of the input image then the image is classified as an adversarial example.

2.4.2 Adversarial Training

Adversarial training [Miyato *et al.* 2018] as shown in figure 2.7, involves using adversarial examples as part of the training data for the model. Specifically, the training process includes generating adversarial examples for each training example and then using both the normal and adversarial examples as inputs to the model during training. The model is then trained to correctly classify both the normal and adversarial examples.

By training on adversarial examples, the model learns to be more robust and less susceptible to adversarial attacks. This is because the model is exposed to a wider variety of inputs during training, including inputs that are similar to normal inputs but have small perturbations that might cause the model to make a mistake. As a result, the model becomes more resilient to these types of perturbations and is less likely to be fooled by adversarial examples when it is deployed in the real world.

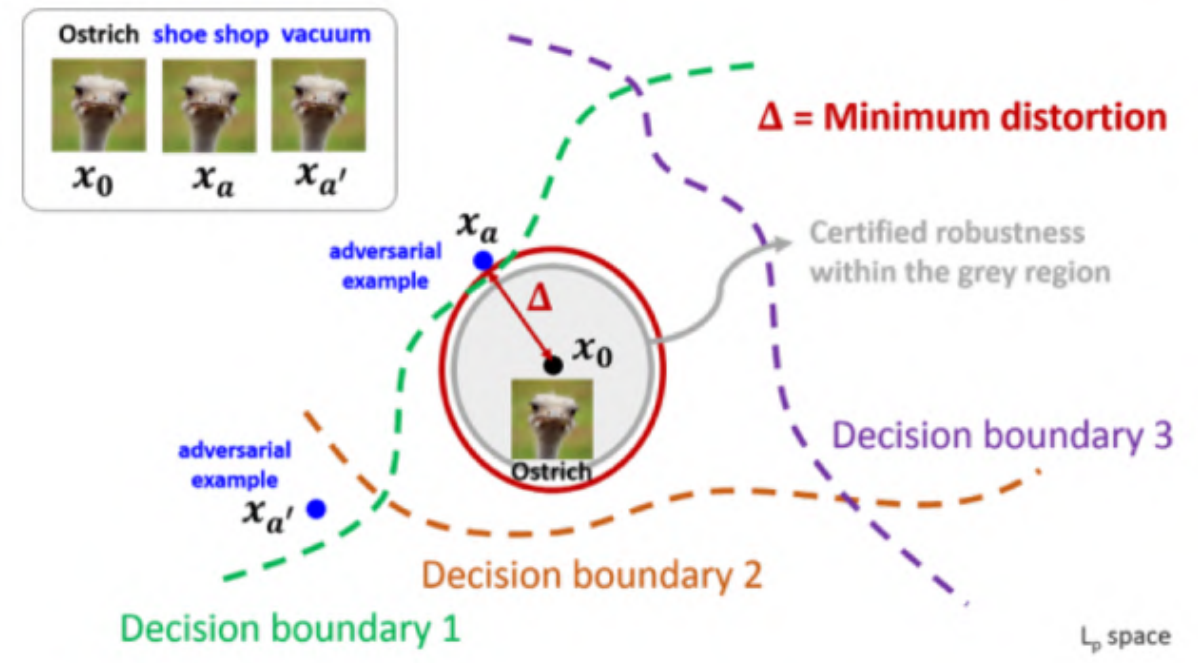


Figure 2.7: Representation of how adversarial training positively affects the model's decision boundaries. After training a model in infected data it learns boundaries which separate the true from the adversarial examples

2.4.3 Defence Distillation

Defence distillation [Elsayed *et al.* 2018] is a technique used to improve the robustness of a machine learning model, particularly a neural network, to adversarial attacks. The basic idea behind defence distillation is to train a smaller, "student" model to mimic the predictions of a larger, "teacher" model. The student model is then deployed in place of the teacher model and is less susceptible to adversarial attacks because it is smaller and simpler.

The training process for defence distillation involves two steps:

1. Training the teacher model: The teacher model is trained on a large dataset using standard supervised learning techniques.

2. Training the student model: The student model is trained to mimic the predictions of the teacher model. This is done by using the same inputs and labels as the teacher model, but using the predictions of the teacher model as the labels for the student model.

During training, the student model is exposed to the same type of input as the teacher model, including both normal and adversarial examples. However, because the student model is smaller and simpler than the teacher model, it is less susceptible to adversarial attacks. As a result, the student model is able to learn from the teacher model's predictions without being fooled by adversarial examples.

Once the student model has been trained, it can be deployed in place of the teacher model. Because the student model is smaller and simpler, it is less resource-intensive to use and may be more efficient and faster than the teacher model. Additionally, because the student model is less susceptible to adversarial attacks, it may be more robust and more reliable in practice.

2.4.4 Adversarial Regulation

Adversarial regularization [Zuts *et al.* 2018] is a technique used to improve the robustness of a machine learning model, particularly a neural network, to adversarial attacks. The basic idea behind adversarial regularization is to add an additional term to the objective function that is optimized during training, which penalizes the model for making predictions that are susceptible to adversarial perturbations.

In adversarial regularization, the objective function is modified to include an additional term that penalizes the model for making predictions that are susceptible to adversarial perturbations. Specifically, the objective function is modified to include a term that measures the model's sensitivity to adversarial perturbations. During training, the model is exposed to both normal inputs and adversarial examples, and the objective function is optimized to minimize the error on both types of inputs.

By minimizing the error on both normal and adversarial examples, the model is forced to learn a function that is not only accurate on normal inputs but also robust to adversarial perturbations. As a result, the model becomes more resistant to adversarial attacks and is less likely to be fooled by adversarial examples when it is deployed in the real world.

2.4.5 Model ensembles

In Model Ensembles [Zhang *et al.* 2018], the basic idea is to train multiple models and then use them in combination to make predictions. The models are trained independently and may have different architectures or be trained on different subsets of the training data. Model ensembles work against adversarial attacks by making it more difficult for the attacker to generate adversarial examples that fool all of the models successfully. This is because the models are trained independently and may make different errors, and by combining their predictions, the overall error rate can be reduced.

2.4.6 MagNet

MagNet [Meng and Chen 2017] is a technique used to defend against adversarial examples, it stands for 'Manifold-Guided Network', it is a two-step method that first detects and then corrects adversarial examples.

As shown in figure 2.8, the first step, detection, uses a two-part neural network, which is trained to classify whether an input is an adversarial example or not. The second step, correction, uses a technique called manifold-based correction, which aims to return the adversarial example to the closest point on the decision boundary of the original

classifier. This is done by projecting the adversarial example onto a lower-dimensional manifold and then back to the high-dimensional input space.

MagNet uses a modified version of autoencoder to achieve this. The encoder part of the autoencoder is used to project the input onto the manifold, and the decoder part is used to project it back to the high-dimensional input space. The encoder and decoder are trained to preserve the topology of the original data, making it hard for an attacker to find a new adversarial example on the manifold, after the correction step.

It is important to note that MagNet requires access to the original classifier during inference and it is computationally more expensive.

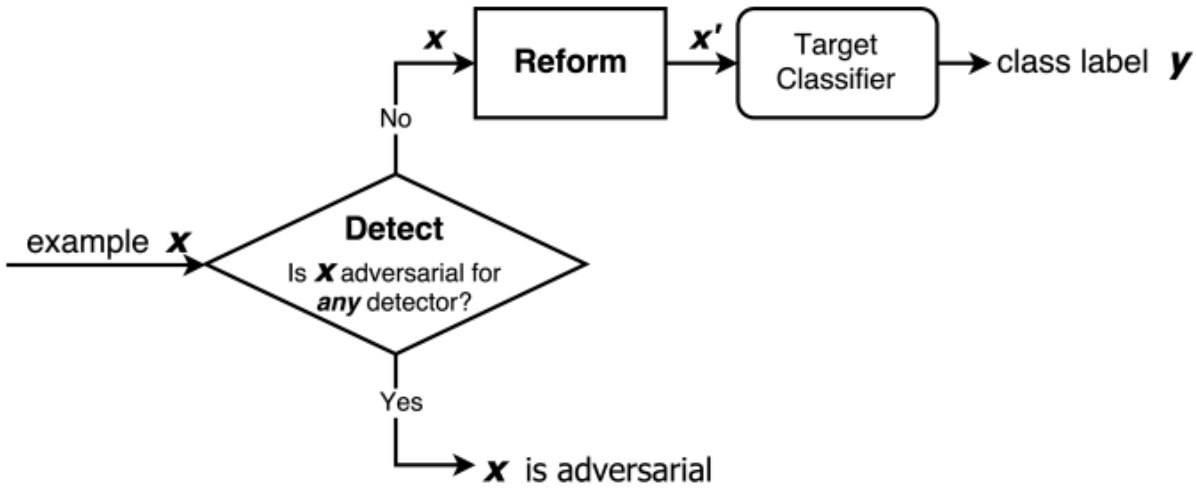


Figure 2.8: MagNet defense workflow

Chapter 3

Research Methodology

3.1 Research Hypothesis

Classification and metric learning are both commonly used techniques in facial recognition for the purpose of identification. This research focuses primarily on comparing the robustness of these approaches against adversarial attacks.

Research Hypothesis: Metric learning models provide more robustness against adversarial attacks than classification models.

3.2 Research Questions

This research aims to answer the following questions:

1. Which approach between metric learning and classification has the best face identification performance in terms of classification accuracy, false match rates and false non-match rates?

2. Are defenseless metric learning models more robust against the FGSM, RFGSM, BIM, PGD, RPGD and C&W adversarial attacks than facial recognition defenseless classification-based models?

3. How do the Defensive distillation, Feature squeezing, Adversarial training and Magnet defense methods perform on metric learning-based models and classification-based models against different adversarial attacks ?

4. Using model confidences rates from model probability scores, how do adversarial attacks perform against defended metric learning and classification FR models ?

3.3 Methodology

3.3.1 Architectures

The main aim of this research is to compare the difference in robustness between metric learning and classification. Different architectures commonly used in both metric learning and classification models will be selected. The highest-performing architectures from the networks listed in [Masi *et al.* 2018] will be chosen. The architectures which will be used in our research will be the following; VGGNET-16 [Parkhi *et al.* 2015], SENET [Kvak 2022], and ResNET [Ranjan *et al.* 2017].

3.3.2 Phase 1: Classification Models

There are a number of classification models in FR, as mentioned in Chapter 2.However, the discussion has not yet addressed the optimal approach for constructing a model that achieves high accuracy while also being resilient against adversarial attacks.

The first step in training classification models is data cleaning. It takes huge datasets to train good models, however, as data size increases, label noises also increase proportionally. The increase in label noise then counteracts the benefit of having a big data set. It then made it imperative to clean the data first [Brodley and Friedl 1999]. There are many data set label cleaning methods, but the methods are generally clustered into two groups: Non-graph based and Graph based. Non-graph-based methods are regarded as easier and straightforward to implement as done by: [Cao *et al.* 2018] and [Parkhi 2015] who use an SVM classifier to reject noisy data as outliers, [Yi *et al.* 2014] who uses a pre-trained FR engine to extract features then get one main picture which he uses to compare with others as a benchmark for acceptance or refusal. In this research, the FaceGraph method will be used, by [Zhang *et al.* 2020], to clean our data. FaceNet uses a GCN (Graph Convolutional Net), which is a transformation function that uses a normalized similarity score. The normalized similarity score, which will help make less similar neighbors (neighbors of the current node) provides aggregation information with fewer weights. This will make the less similar neighbors, when concatenated information is sent to the fully connected neural net, get a score below the threshold to be deemed as noise or not. The output data set will be used to train the designed FR models.

The aim of running many different models will be to get the best models in terms of classification scores. The classification scores which will be used are **FMR** (false match rate), **FNMR** (false non-match rate), and **CA** (Classification Accuracy). **FMR** is the rate at which the classifier falsely matches faces from different classes, **FNMR** is the rate at which the classifier falsely mismatch faces from the same class. The classifiers will be built on three loss functions: Cross-Entropy, and Arc face as they are the once widely used in facial recognition exercises.

$$FMR = \frac{FP}{FP + TN} \tag{3.1}$$

$$FNMR = \frac{FN}{FN + TP} \tag{3.2}$$

$$CA = \frac{TN + TP}{FN + FP + TF + TP} \tag{3.3}$$

The main focus will be placed on the analysis of classification confidence to assess the impact of adversarial examples on the model and evaluate its vulnerability.

3.3.3 Phase 2: Metric Learning Models

Metric learning is used mainly to determine the difference between samples' embeddings. This is done by learning a new metric that can efficiently reduce the distance separation between similar samples and increase the distance separation between samples that are not similar. In this paper, it was decided to define contrastive loss, and triplet loss under metric learning but this is also sometimes known as machine-learned ranking [Kulis and others 2013].

Using different loss functions will give us enough information to determine which of the selected loss functions and architectures work best together. The discussion has involved the consideration of selecting between hard positives pair mining and triplet mining. Initial experiments will be conducted to determine the most suitable method for pair and triplet mining.

Why not opting for Very Hard Positives and Negatives

For the loss functions that were listed, it will be theoretically beneficial to select very hard positives and negatives for good training and fast convergence [Meng *et al.* 2021]. One disadvantage mentioned by [Schroff *et al.* 2015], of poor training due to mislabelled and poor images becoming the greater percentage of the very hard positives and negatives is countered by the data set cleaning, which will be done to all datasets at the beginning. In this research, the utilization of semi-hard samples will be employed. Various thresholds were tested to identify the optimal values for defining easy, semi-hard, and hard samples.

This is one of the contributions of this research since many researchers use different methods. For example, [Wang *et al.* 2014] use an online pairwise relevance score by calculating the relevance of images relative to query images and choosing the less relevant pairs. Alternatively, in batch format sampling, [Wang *et al.* 2016] uses top k most violated matches.

3.3.4 Phase 3: Adversarial Attacks and Defences

In chapter 2,the different types of adversarial attacks and defences were discussed. There are types of adversarial attacks which are said to be mainly designed for classification and others for metric learning. To the best of our knowledge, there has never

been an in-depth comparison of how adversarial examples designed for classification affect metric learning models and how adversarial examples designed for metric learning affect classification models. This will be the main focus of the research.

Experiments will be designed to see how adversarial examples produced at different hyperparameters can fool our models and test the effect on the classification confidence of models when attacked with and without defences. This will help us understand the (1) relationship between the different attacks and defences of the two different learning paradigms and (2) the best defences mechanism for specific attacks.

Chapter 4

Experiments

4.1 Introduction

The scope of the research in Chapter 3 was introduced. The research process outlined is illustrated in figure 4.1. The first step is to select data sets to train our models on. This is followed by building the classification and metric learning models, which will be trained to evaluate their performance. Subsequently, adversarial attacks are generated using different methods and applied to the defenseless models. The models are then defended and re-attacked to evaluate their performance and assess their robustness.

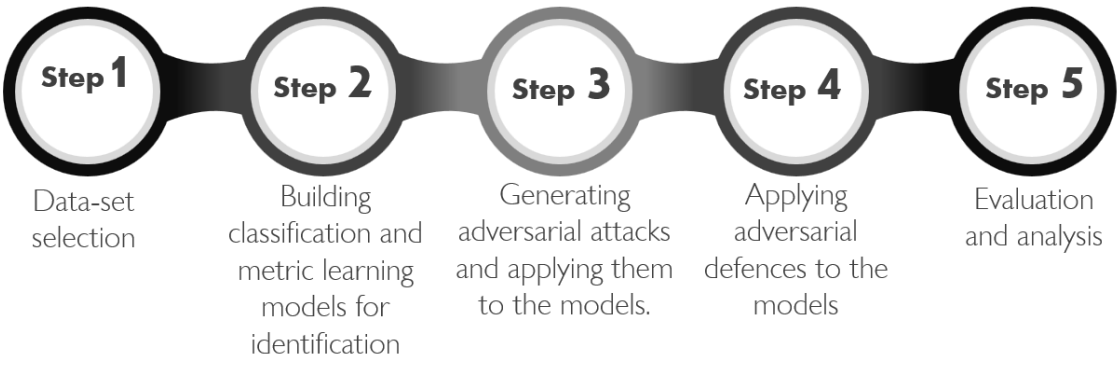


Figure 4.1: Research Experimental Evaluation Process

4.2 Data Sets

Two datasets were selected, with one designated for training the models and the other utilized for evaluation purposes. These dataset sets were selected based on availability, size, and also because they are used for similar studies by prominent researchers. These two datasets are The datasets are as follows:

- **Training**[Casia-WebFace]. The Casia-WebFace dataset [Yi *et al.* 2014] is a large-scale facial recognition dataset that was collected by the Institute of Automation, Chinese Academy of Sciences (CASIA) in China. It contains over 500,000 images of 10,575 individuals, with an average of around 45 images per person. The images were collected from the web, and the dataset contains a wide variety of facial poses, expressions, and lighting conditions. The dataset is large enough to train good models for a comprehensive investigation.
- **Testing** [LFW]. The LFW dataset [Huang *et al.* 2007] was created by researchers at the University of Massachusetts and the Technical University of Berlin. It is a collection of images of human faces that is commonly used for training and evaluating the performance of facial recognition algorithms. It contains more than 13,000 images of faces, each annotated with the name of the person depicted in the image. The images in the dataset were, collected from the internet and cover a wide range of variations in pose, lighting, and background.

The data sets are structured such that there is a main dataset directory that contains subdirectories of images, with each subdirectory representing a class. The names of the subdirectories correspond to the class names.

Data Exploration

Table 4.1: Casia-Webface and LFW dataset description

Dataset	Classes	Num_images	Img_dim
Casia-Webface	10 575	500 000	112 x 112
LFW	5749	13 000	250 x 250

As a data preprocessing step, data exploration was done on the Casia-Webface and LFW datasets in table 4.1 which allows us to understand the characteristics and patterns of the data-sets that are being used. It was found that the datasets are imbalanced, meaning that some identities/classes are significantly more or less represented than others, as represented in figure 4.2 and figure 4.3.Figures 4.2 and 4.3 are a representation of identity/class frequencies and the frequencies clearly decrease showing the imbalance. For the Casia-Webface dataset the ratio of the largest class and the smallest class is 396 $(786 : 2)$ and for the LFW dataset the ratio is 530 $(530 : 1)$.

It is mentioned by [He *et al.* 2009] that when training models on imbalanced datasets the model may be biased towards the more dominant classes which can lead to poor performance on the minority class, also known as class-imbalance problem.

4.3 Technical Requirements

All model training and adversarial image generation were done on cloud computing setup and high-performance computing environment because of their computational requirements. However model testing was done on a PC with AMD Ryzen 7 5700U

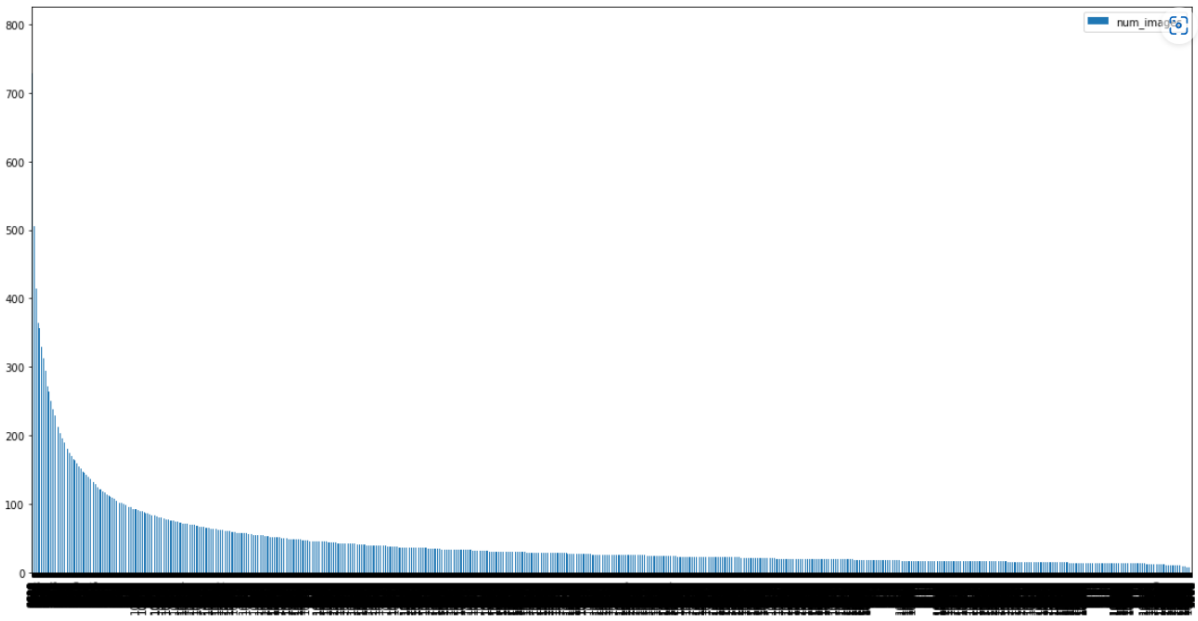


Figure 4.2: CASIA-Webface dataset class frequency. The highest being class 819 with a frequency 786, the lowest being class 9282 with a frequency 2.

with Radeon Graphics processor 1.80 GHz, 8.00 GB memory, and Windows 11 64-bit operating system.

4.4 Recognition models Experiments

For training our models, model training from scratch and transfer learning were explored. Transfer learning [Weiss and Torgo 2016; Pan and Yang 2009] is a machine learning technique where a model trained on one task is re-purposed on a second related task. It is particularly useful when the second task does not have a sufficient amount of training data. In transfer learning, the model is first pre-trained on the first task, and then the pre-trained weights are used as the starting point for training the model on the second task. This approach allows the model to learn more efficiently because it can utilize the knowledge it has gained from the first task, rather than starting from a completely random state. The best model ultimately came from using transfer learning however it was noted that all our models were over fitting if smaller datasets were used like Wider Face [Yang *et al.* 2016] and celebFace [Sun *et al.* 2014]. For training, the Casia-Webface was large enough to make the models not overfit, and transfer learning converged faster with better results as compared to training the models from scratch.

Classification Models

The model face identification/classification pipeline as shown in figure 4.4 starts with face detection where a face in the picture is detected. After detection, there is preprocessing which involves cropping, padding and alignment. Then the pre-processed

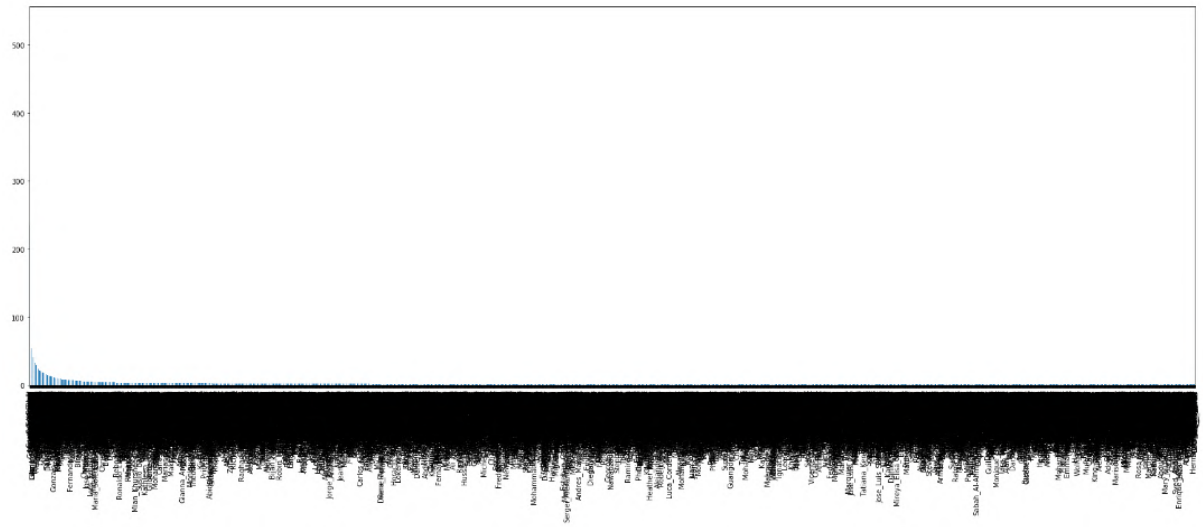


Figure 4.3: LFW dataset class frequency. The highest being class George W Bush with a frequency 530, the lowest being class Cristiano da Matta with a frequency 1.

input is then sent through a neural network for feature extraction. The extracted feature will be used for classification. In this research, the data sets being used are already face cropped so the only processing done was to pad the face images so as to have a size that fits into the models for transfer learning. The VGG16, SENet, ResNet50 and MobileNet models were used with three different loss functions which are Cross-Entropy, arcface loss and ring loss.

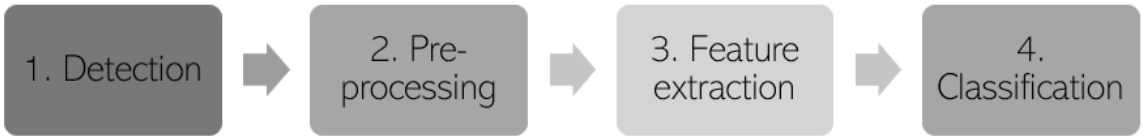


Figure 4.4: Face identification/classification pipeline

The summaries for the classification models used are found in Appendix A. Table 4.2 shows a summary of architectures models and loss functions used.

Metric learning Models

The metric learning model face identification pipeline is shown in figure 4.5. Just like classification models the first two steps are face detection and pre-processing before model training. In this research, the data sets are already cropped and aligned faces. The only pre-processing steps done were padding for transfer learning and sample mining.For triplet-loss models, the semi-hard triplets were mined as shown in fig 5.1 and

Table 4.2: Summary of models and classification loss functions used

Models	Loss Function
SENet	Cross-Entropy
	Arcface
	RingLoss
RESNET	Cross-Entropy
	Arcface
	RingLoss
MobileNet	Cross-Entropy
	Arcface
	RingLoss
VGG16	Cross-Entropy
	Arcface
	RingLoss

for contrastive-loss the sample pairs were mined as shown in figure 5.2. Just as in classification model training, training models from scratch did not require pre-processing as it was possible to feed the models inputs of any size as long as the input aspect ratio was 1:1.The VGG16, SeNet, ResNet50 and MobileNet models were used as the backbone models with triplet loss and contrastive loss functions. The k-NN was chosen as the classifier as it is a simple classifier which does not need to repeat the process of feature extractions and mapping in the latent space as done by the SVM. The summary of model architecture and loss function is shown in table 4.3

Table 4.3: Summary of models and metric learning loss functions used

Models	Loss Function	Classifier
SENet	Triplet loss	k-NN
	Contrastive loss	k-NN
RESNET	Triplet loss	k-NN
	Contrastive loss	k-NN
MobileNet	Triplet loss	k-NN
	Contrastive loss	k-NN
VGG16	Triplet loss	k-NN
	Contrastive loss	k-NN

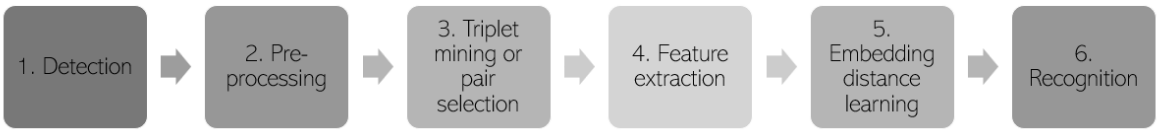


Figure 4.5: Metric Learning Face identification pipeline

4.5 Adversarial sample generations and model attack

The goal is to see how generated adversarial samples would cause a normal class to be misclassified and to what extent in facial recognition classification models and facial recognition metric learning-based models. Six(6) attacks were selected attacks as listed below:

1. Fast Gradient Sign Method attack (FGSM)
2. Randomized Fast Gradient Sign Method (RFGSM)
3. Binary Iterative Method (BIM)
4. Projected Gradient Descent (PGD)
5. Randomized Projected Gradient Descent (RPGD)
6. Carlini & Wagner Attack (C&W)

Adversarial examples of 50 images from a test set were generated using the above-listed methods and used to attack the classification and metric learning models in a bid to see how effective the generated attacks are on the two model types. The reported results are averages of confidences from the 50 adversarial images. Sample images are presented in figures 5.3, 5.4, 5.5, 5.6, 5.7, 5.8 The following hyper-parameters were used to generate the adversarial examples as shown in table 4.4:

Table 4.4: Attack Epsilon parameter values set

Attack	**Experiments**				**Constants per run**	
Method	**Run 1**	**Run 2**	**Run 3**	**Run 4**	Alpha	Iterations
FGSM	190/255	19/255	1.8/255	55/255	-	-
RFGSM	150/255	15/255	1.8/255	70/255	8/255	1
BIM	50/255	7.8/255	1/255	15/255	1/255	-
PGD	77/255	8/255	4/255	26/255	2/255	40
RPGD	40/255	8/255	1/255	15/255	2/255	40

In the process of selecting epsilon values for various adversarial attacks, a standard recommended value from the original paper is initially chosen [Madry *et al.* 2017; Kurakin *et al.* 2016; Goodfellow *et al.* 2014]. From this starting point, two additional values are randomly chosen, with one value above the recommended value and the other value below it, then a last set of random values is chosen. Table 4.4 presents values that are divided by 255 and, in some instances, are float values, which may seem counterintuitive. However, these values represent the maximum perturbation per pixel in the range [0, 255], so it is more convenient to express them in this format. It is seen from table 4.4 that some number of iterations is declared to either zero or one, and is determined automatically based on the algorithm. This approach also enables a comparison of the effect of naive methods on the results.

Table 4.4 omits the Carlini and Wagner (C&W) attack because it is distinct from other attack types and does not share any overlapping hyperparameters hence the hyparameters for the (C & W) attack are listed in table 4.5:

Table 4.5: Carlini and Wagner hyperparameters value set

Hyperparameter	Value
C	10
Binary search steps	2
Iterations	500
kappa	0
Learning rate	40/255

Kappa controls the trade-off between the confidence of the adversarial example and the distance (measured in some norm) between the adversarial example and the original input. In other words, a larger value of kappa means that the adversarial example will be more confident (i.e., the model will be more certain that the adversarial example is not the original input), but it will also be further away from the original input.
C is a hyperparameter that controls the relative importance of the confidence of the adversarial example and the distance between the adversarial example and the original input. A larger value of C means that the distance between the adversarial example and the original input is more important, and the confidence of the adversarial example is less important.

The process flow for the attack is shown in figure 4.6:

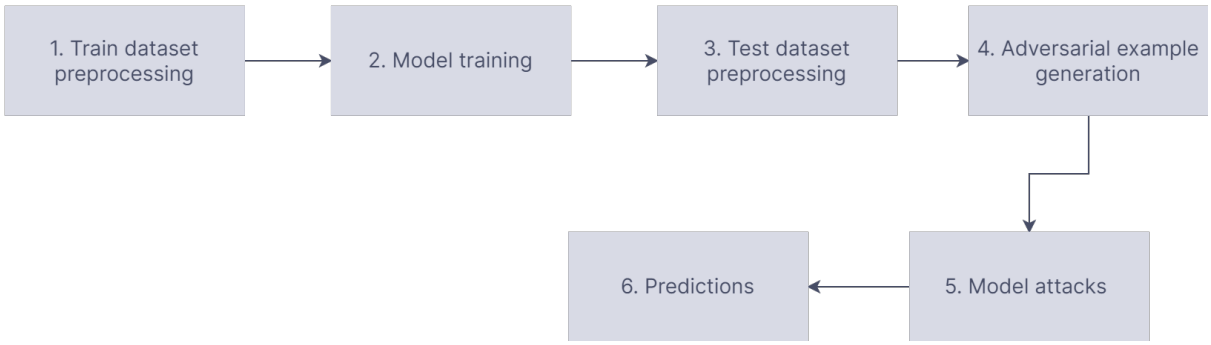


Figure 4.6: Pipe line for adversarially attacking face recognition models

4.6 Model Defence against adversarial attacks

In this section, the best and worst-performing models were taken from the classification-based models and the triplet-based models. The the adversarial defense methods were implemented. The following defense methods were used:

1. Defensive distillation
2. Feature squeezing
3. Adversarial training
4. Magnet

The dataset used was the aligned Casia-webface dataset.

Defensive distillation

Defensive Distillation is a technique that involves training a model twice. In the first training, the models were trained in the usual way, and the class probabilities were extracted from it. In the second training, these class probabilities were used as the target instead of the actual labels, which were used in the first training. The experimentation involved attempting to modify the loss functions; however, it was observed that such modifications did not yield any improvement. This approach was applied across all adversarial attacks.

Features squeezing

Feature squeezing is a technique that can be used to defend against adversarial attacks by reducing the dimensionality of the input space. The idea behind feature squeezing is to transform the input features in such a way that they are more difficult for an attacker to manipulate while still preserving the useful information for the model to make a correct prediction.
In the experiments, the following methods were used, **(1) Colour Bits Squeezing** using algorithm 2 and **(2) Spatial Smoothing- Local smoothing** using algorithm 3 which uses the mean kernel.

Algorithm 2 Color Bit Squeezing Algorithm

procedure COLORBITSQUEEZING
    img ← LoadImage(path/to/image.jpg)
    hsv_img ← RGBToHSV(img)
    for each channel ∈ {"Hue", "Saturation", "Value"} **do**
        for $i \in \{1, 2, \cdots, 7\}$ **do**
            hsv_img$[channel]$ ← *hsv_img*$[channel] * (2^i - 1)$
            hsv_img$[channel]$ ← round(hsv_img[channel])
            hsv_img$[channel]$ ← *hsv_img*$[channel]/(2^i - 1)$
        end for
    end for
    rgb_img ← HSVToRGB(hsv_img)
    SaveImage(rgb_img, path/to/output.jpg)
end procedure

Algorithm 3 Spatial Smoothing Algorithm (Mean Kernel)

procedure SPATIALSMOOTHING
    img ← LoadImage(path/to/image.jpg)
    kernel ← np.ones((3,3))/9
    smooth_img ← convolve(img, kernel)
    SaveImage(smooth_img, path/to/output.jpg)
end procedure

Adversarial training

The selection process involved choosing the best performing classification-based model and the best performing metric learning-based model. This decision was based on considerations such as computational requirements for adversarial training and the quantity of adversarial examples necessary to achieve a reasonably robust model.
After conducting a series of experiments, the following adversarial training parameters were determined and are presented in the table below 4.6:

Table 4.6: Adversarial training parameters

Parameter	**Values**
Learning rate	0.001
Ratio of adversarial examples to original images	0.5
Number of epochs	20

MagNet Implementation

Table 4.7: MagNet implementation parameters

Parameter	**Values**
Optimizer	Adam
Loss Function	Magnet Loss
Detector	Reconstruction error
Batch size	256
Epochs	250
Regularization	Noise

Chapter 5

Results and Discussion

5.1 Introduction

In this chapter, the obtained results from the experiments described in the previous section are presented. The initial segment focuses on discussing the results for facial recognition, considering metrics such as classification accuracy, False Match Rate, and False Non-Match Rate. Following that, an analysis of the observations derived from subjecting the models to different adversarial attacks is conducted. Lastly, the observations resulting from the implementation of various defense mechanisms on the models are examined.

5.1.1 Baseline performances

The results discussed will be from tables 5.1 and 5.2.

Classification based models

From the results in Table 5.1, it can be observed that the highest classification accuracy was achieved by RESNet with Arcface loss, with a score of 93.05%. This was followed by VGG16 with Arcface loss, with a score of 90.94%. Both these models performed significantly better than the other models in the comparison.

In terms of FMR, the best performance was achieved by RESNet with Cross-Entropy loss, with a score of 0.87%, followed by RESNet with Arcface loss, with a score of 0.63%. This suggests that the use of Arcface loss slightly improves the FMR.

Regarding FNMR, the best performance was achieved by RESNet with Arcface loss, with a score of 0.82%, followed by RESNet with Cross-Entropy loss, with a score of 1.01%. This suggests that the use of Arcface loss significantly improves the FNMR.

The results show that among the models evaluated, RESNet with Arcface loss achieved the best results for classification based models, with a classification accuracy of 93.05%, FMR of 0.63% and FNMR of 0.82%.

Metric learning based models

From the results in Table 5.2, it can be observed that the highest classification accuracy was achieved by RESNet with Triplet Loss, with a score of 99.96%. This was followed by VGG16 with Triplet Loss, with a score of 99.88%. Both these models performed significantly better than the other models in the comparison.

In terms of FMR, the best performance was achieved by RESNet with Triplet Loss, with a score of 0.1%, followed by VGG16 with Triplet Loss, with a score of 0.11%. This suggests that the use of RESNet tripelet and VGG16 Triplet Loss leads to a close performance in terms of FMR.

Regarding FNMR, the best performance was achieved by RESNet with Triplet Loss and VGG16 with Triplet Loss, with a score of 0.1%.. This suggests that the use of Triplet Loss leads to a similar performance in terms of FNMR.

The results show that among the models evaluated, RESNet with Triplet Loss achieved the best results, with a classification accuracy of 99.96%, FMR of 0.1% and FNMR of 0.1%.

Table 5.1: Classification models facial recognition evaluation

Model	Loss Function	Classification Accuracy %	False Match Rate (FMR) %	False Non_Match Rate (FNMR) %
SENet	Cross-Entropy	83.25	1.7	2
	Arcface Loss	83.68	1.2	2.3
RESNet	Cross-Entropy	91.02	0.87	1.01
	Arcface Loss	93.05	0.63	0.82
MobileNet	Cross-Entropy	82.39	0.9	0.96
	Arcface Loss	82.94	0.85	0.91
VGG16	Cross-Entropy	89.77	0.6	0.91
	Arcface Loss	90.94	0.6	1.02

Discussion Summary

When comparing the top performers of two types of models, those based on classification and those based on metric learning, it was found that ResNet with Triplet loss outperformed ResNet with Arcface loss. It is worth noting that ResNet50 was the optimal architecture and ResNet with Triplet Loss was the best facial recognition model among the models tested.

The results are in line with the properties of the ResNet-50. The 50-layer ResNet uses a bottleneck design for the building block. A bottleneck residual block uses 1×1 convolutions, known as a “bottleneck”, which reduces the number of parameters and matrix multiplications. This enables much faster training of each layer. This means it improves

the efficiency of deep neural networks with more neural layers while minimizing the percentage of errors.

On the good performance of metric learning models, There are a few reasons why metric learning loss functions give better facial recognition results compared to classification loss functions.

- Metric learning loss functions are designed to learn a distance metric that measures the similarity between two images. This is in contrast to classification loss functions, which are designed to learn a classifier that predicts the class label of an image. A distance metric is more suitable for face recognition because it can be used to compare the similarity of two images, even if they are of different people.
- Metric learning loss functions are able to learn more discriminative features than classification loss functions. This is because metric learning loss functions are able to learn features that are invariant to variations in pose, lighting, and other factors. Classification loss functions, on the other hand, are only able to learn features that are discriminative for the specific classes that they are trained on.
- Metric learning loss functions are more robust to noise and outliers. This is because metric learning loss functions are not directly affected by the labels of the training data. Classification loss functions, on the other hand, can be sensitive to noise and outliers in the training data.

Table 5.2: Metric learning models facial recognition evaluation

Model	Loss Function	Classification Accuracy %	False Match Rate (FMR) %	False Non_Match Rate (FNMR) %
SENet	Triplet Loss	93.66	0.9	1.03
	Constrastive Loss	94.77	0.75	0.93
RESNet	Triplet Loss	99.96	0.1	0.1
	Contrastive Loss	99.72	0.12	0.14
MobileNet	Triplet Loss	87.52	0.2	0.18
	Contrastive Loss	86.83	0.23	0.19
VGG16	Triplet Loss	99.88	0.11	0.1
	Contrastive Loss	99.45	0.17	0.12

5.1.2 Adversarial attacks on defenseless models

In this section, the effect of adversarial examples on our defenseless models was observed. These models were all trained and tested on the same data sets. Adversarial examples were generated for representative classes using different parameters to assess their ability to deceive the models and observe their impact on classification confidence. In the tables of results,tables 5.3, 5.4, 5.5, 5.6, 5.7, 5.8 for defenseless attacks and tables 5.9, 5.10, 5.11, 5.12, 5.13, 5.14, the green color (**was not fooled**) indicates that the model correctly classified the input adversarial example and the red color (**was fooled**) shows that the model mis-classified the input adversarial example

Fast Gradient sign Method attack

Our initial observation in table 5.3 was that the FGSM adversarial example/s generated at $\epsilon = 1.8/255$ was successful in fooling all of the models tested. The second notable

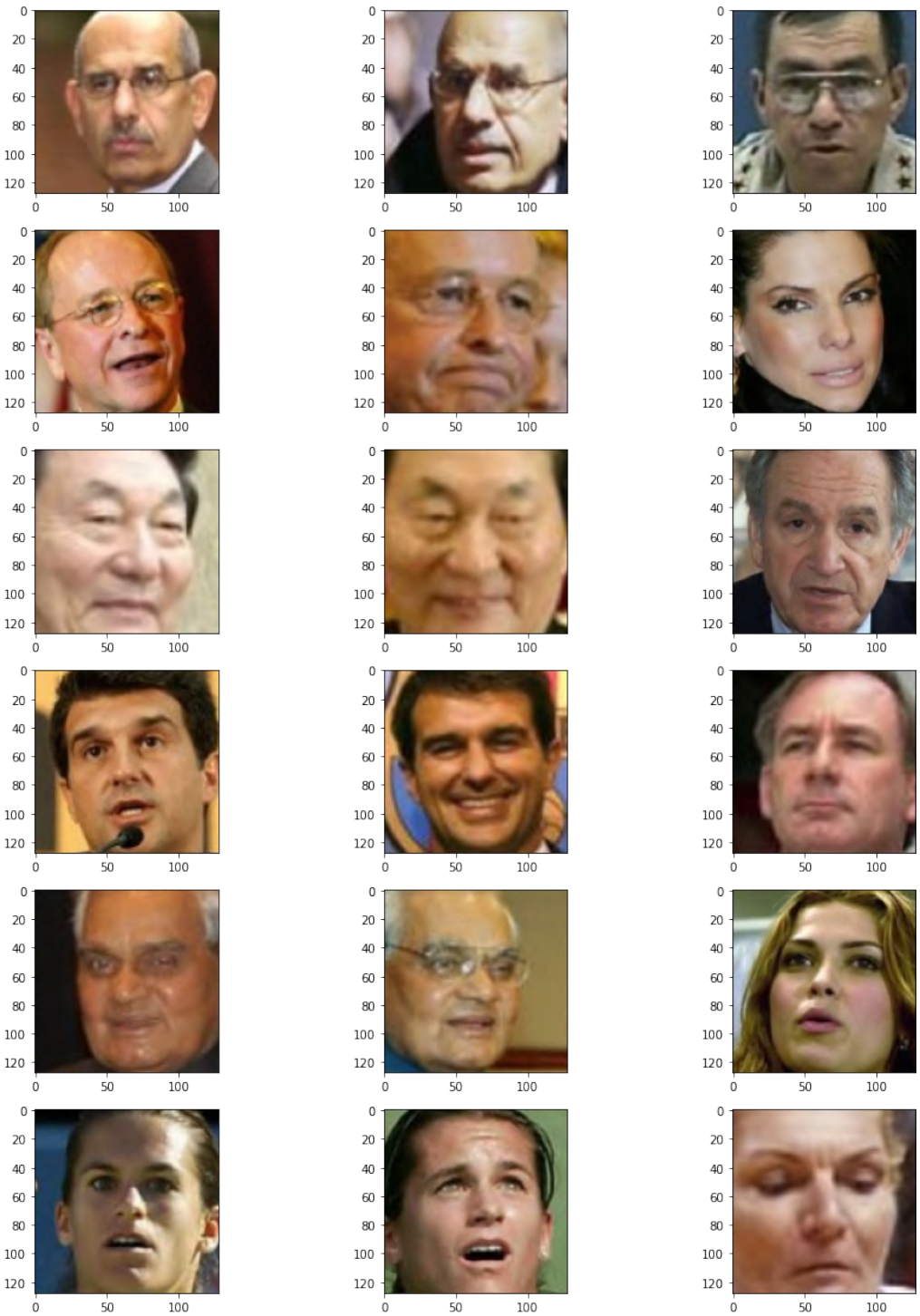


Figure 5.1: Sample of semi-hard triplets. The first two columns from the left contain anchors and positives and the last column has negatives

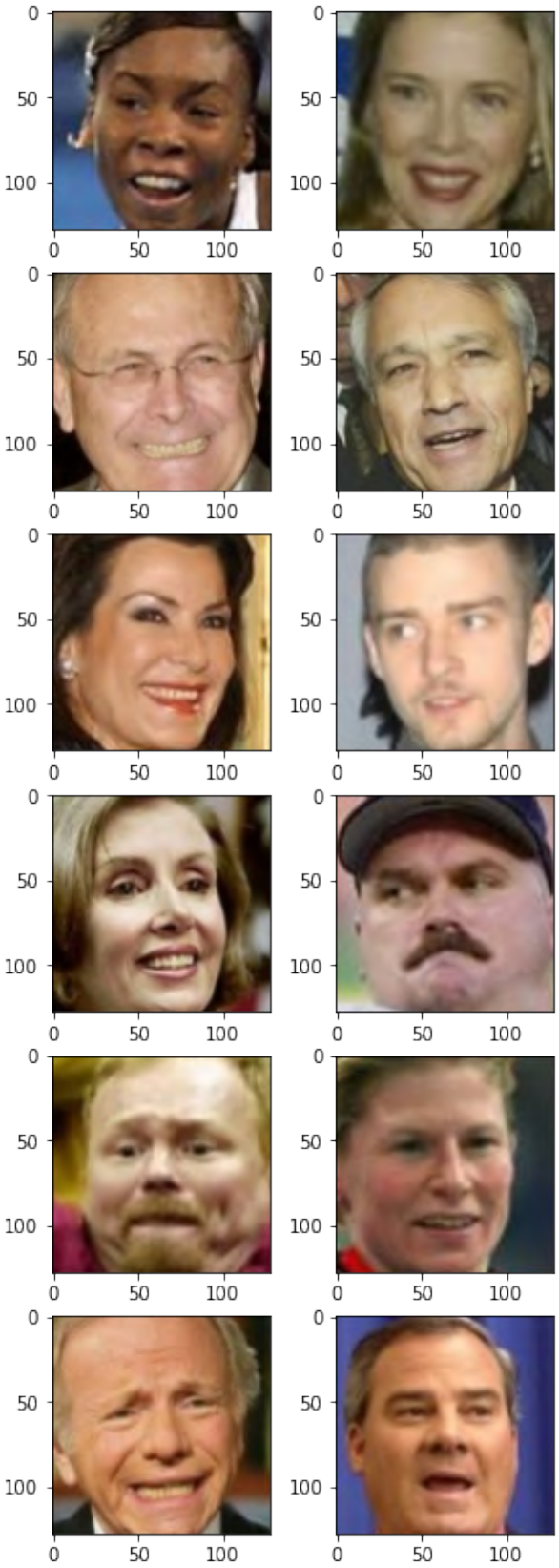


Figure 5.2: Samples of pairs for contrastive loss. For better results pairs from different classes were used.

observation is that, at $\epsilon = 19/255$ all Arcface and Cross-Entropy (classification) based models were fooled and all Triplet Loss and Contrastive loss (metric learning) based models were not fooled. Lastly all models managed to correctly classify the adversarial examples generated at $\epsilon = 1.8/255$ and $\epsilon = 190/255$.

Now considering the classification confidence rates it is seen that at $\epsilon = 1.8/255$, the ResNet with Triplet loss has the lowest confidence rate of 33.90% followed by VGG16 with triplet loss which has 33.91% confidence rate and MobileNet with Cross-Entropy has the highest confidence rate of 40.00% followed by by MobileNet with Arcface loss which has 39.91% confidence rate.

At $\epsilon = 190/255$ ResNet with Triplet loss has the highest confidence rate of 30.56% followed by ResNet with contrastive loss which has a confidence rate of 29.33% and MobileNet with Arcface loss has the lowest confidence rate of 19.49% followed by MobileNet with Triplet loss and Softmwax which have a confidence rate of 21.33%

Randomized Fast Gradient Sign Method Attack

From table 5.4, firstly, it is observed that the RFGSM adversarial example/s generated at $\epsilon = 1.8/255$, just as it was observed with FGSM attack, was able to fool all of the models tested. Again just like the FGSM result occurrences with RESNet models,both classification-based models, the Cross-Entropy and Arcface loss were fooled. The second outstanding observation is that at $\epsilon = 15/255$ all MobileNet models were fooled. Some surprising points that deviate from the trend seen from the FGSM results are also noted:

- The SENet with Cross-Entropy was not fooled but its Arcface version was fooled
- The VGG16 with Contrastive loss was unexpectedly fooled
- At $epsilon = 70/255$ the MobileNet with Triplet Loss was fooled.

Now considering the classification confidence rates, it is seen that at $\epsilon = 1.8/255$, the VGG16 with Triplet loss has the lowest confidence rate of 32.90% followed by VGG16 with Contrastive loss which has a confidence rate of 33.99% and MobileNet with Triplet loss has the highest confidence rate of 69.78% followed by MobileNet with Contrastive loss which has 63.35% confidence rate.

At $\epsilon = 150/255$, the ResNet with Triplet loss has the highest confidence rate of 57.21% followed by ResNet with Contrastive loss which has a confidence rate of 48.97% and MobileNet with Cross-Entropy loss has the lowest confidence rate of 16.12% followed by MobileNet with Arcface loss and Softmwax which have a confidence rate of 16.74%

Basic Iterative Method Attack

In table 5.5, initially, it is imperative to note that the epsilon (ϵ) values used are smaller compared to those used in the FGSM and RFGSM attacks. With that in mind, it is observed that the attack generated at $\epsilon = 4/255$ and $\epsilon = 7.8/255$ were successful in fooling all of the models tested. It is also noted that all MobileNet models with Cross-Entropy

and Arcface loss were fooled at all epsilons. The MobileNet with Contrastive Loss only managed to correctly classify the adversarial input generated at $\epsilon = 50/255$

For the classification confidence rates it is observed that at $\epsilon = 1/255$ the RESNet with Triplet loss has the lowest confidence rate of 30.94% followed by VGG16 with Triplet loss which has a confidence rate of 31.04%. It is important to note that the VGG16 with Contrastive-loss performed better that the RESNet with contrastive loss as the former has a confidence rate of 31.05% and the latter 33.43%. At $epsilon = 50/255$, the RESNet with Triplet and Contrastive loss has the highest confidence rates respectively.

Projected Gradient Descent Attacks

In table 5.6 a trend similar to the one seen in the FGSM attack is realized. At $\epsilon = 4/255$, all models were fooled and at $\epsilon = 8/255$ all classification-based models, i.e the Cross-Entropy and Arcface loss models were fooled and all metric learning-based models were not fooled. At $\epsilon = 77/255$ no model was fooled.

For the confidence rates, at the lowest epsilon that is, $epsilon = 4/255$ The lowest confidence rates are from the RESNet with Triplet loss and VGG16 with Triplet loss respectively with 33.37% and 33.41% respectively. The highest confidence rates are from the MobileNet with Cross-Entropy and Arcface. At $epsilon = 77/255$ the highest confidence rates are from RESNet with Triplet loss and Contrastive loss in that order and the lowest are from MobileNet with Contrastive loss with a confidence rate of 12.55%.

Randomised Projected Gradient Descent Attacks

In table 5.7 a similar trend is observed. At $\epsilon = 1/255$, all models were fooled and at $\epsilon = 8/255$ all classification-based models, i.e the Cross-Entropy and Arcface loss models were fooled and all metric-learning-based models were not fooled. At $\epsilon = 40/255$ no model was fooled.

For the confidence rates, at the lowest epsilon, that is, $epsilon = 1/255$ The lowest confidence rates are from the RESNet with Triplet loss and RESNet with Contrastive loss respectively with 30.27% and 30.37% respectively. The highest confidence rates are from the MobileNet with Cross-Entropy and Arcface. At $epsilon = 40/255$ the highest confidence rates are from RESNet with Triplet loss and Contrastive loss in that order and the lowest are from MobileNet with Cross-Entropy and Arcface loss.

Carlini and Wargner Attacks

C $\&$ W attacks were generated at one set of parameters and that fooled all models. In comparison with other adversarial attacks, the false confidence levels seem to be higher. The lowest is from the RESNet with Triplet Loss which has a confidence rate of 57.43% followed by RESNet with Contrastive Loss which has a confidence rate of 57.45%. The

Figure 5.3: One sample of FGSM adversarial attacks created per epsilon. The epsilon values appear as floats because matplotlib turns fractions in the titles to floats

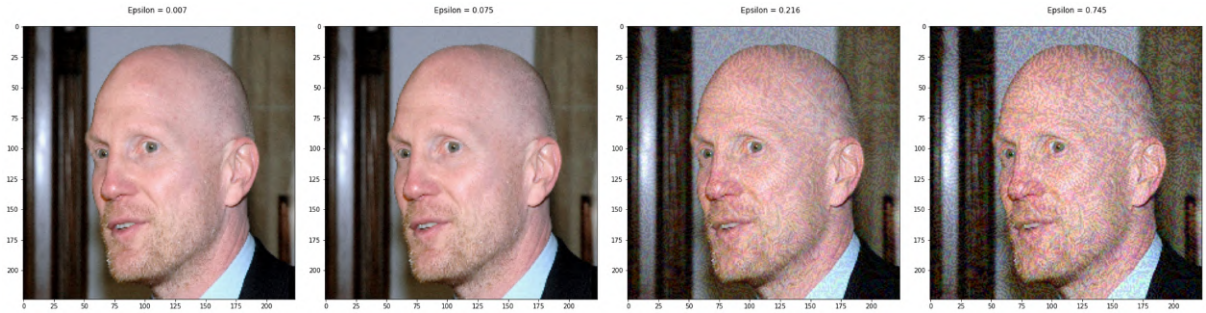


Figure 5.4: One sample of RFGSM adversarial attacks created per epsilon. The epsilon values appear as floats because matplotlib turns fractions in the titles to floats

highest is from MobileNet which has all its confidence rates above 60%. This seems to tell that the C $\&$ W is a stronger attack.

In summary, it is seen that as more disturbance is done on an image, models become more capable of detecting the image as adversarial. It is also observed that metric learning models generally have higher confidence when it is a true classification and a lower misclassification confidence rate than classification-based models.

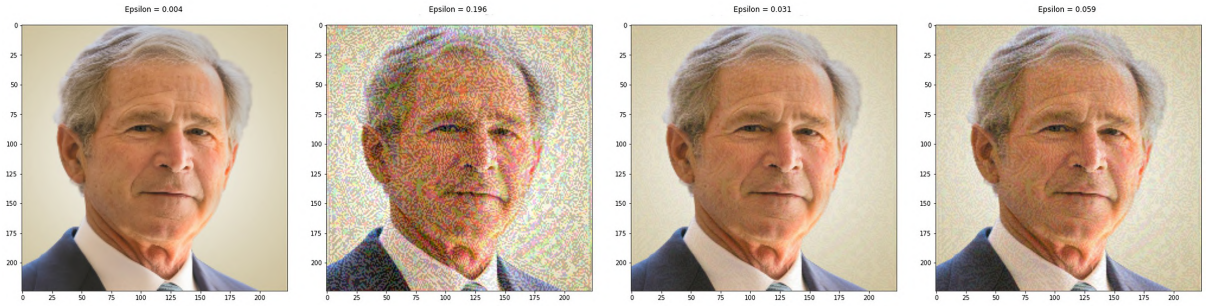


Figure 5.5: One sample of BIM adversarial attacks created per epsilon. The epsilon values appear as floats because matplotlib turns fractions in the titles to floats

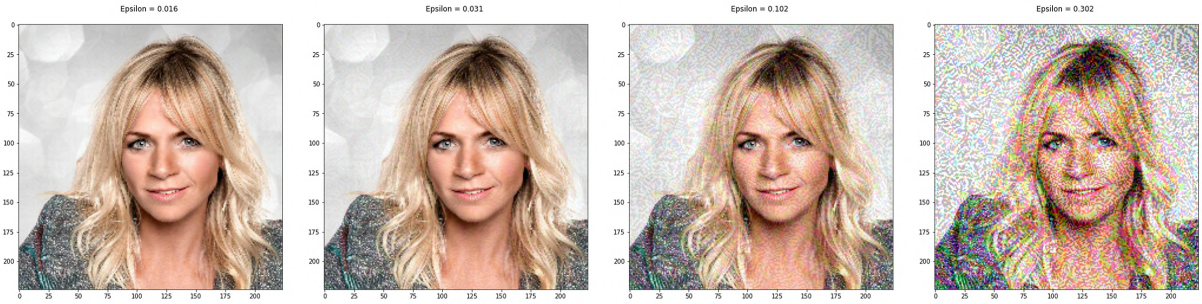


Figure 5.6: One sample of PGD adversarial attacks created per epsilon. The epsilon values appear as floats because matplotlib turns fractions in the titles to floats

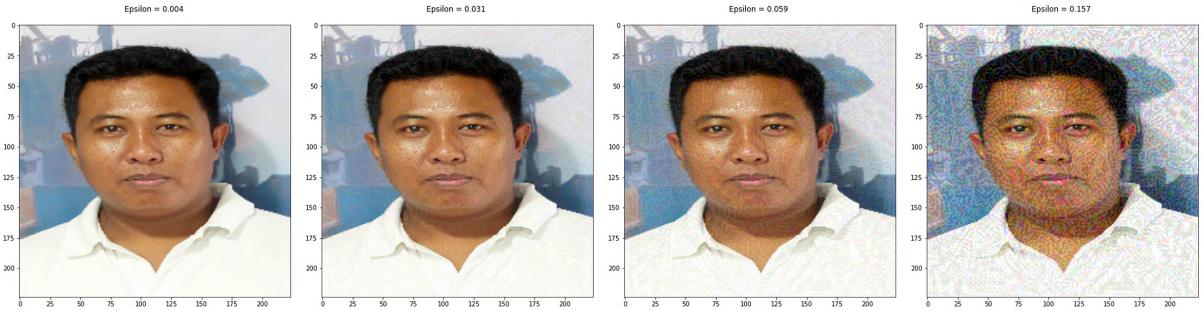


Figure 5.7: One sample of RPGD adversarial attacks created per epsilon. The epsilon values appear as floats because matplotlib turns fractions in the titles to floats

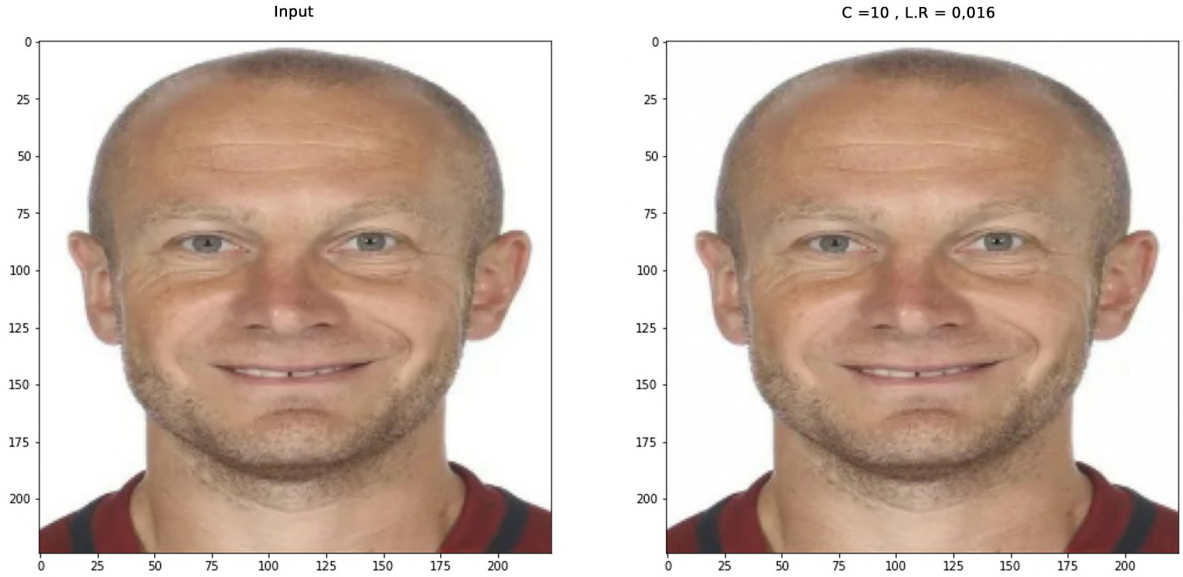


Figure 5.8: One sample of CW adversarial attacks

Table 5.3: FGSM Classification confidence % results per experimental epsilon. The green cells show correct classification and the red cells show wrong classification

		Classification Confidence % per Epsilon value				
Classification Model	**Loss Function**	**Original**	**1.8/255**	**19/255**	**55/255**	**190/255**
SENet	Cross-Entropy	78.03	37.33	15.44	15.95	22.44
	Arcface Loss	78.07	35.89	14.20	15.80	22.40
	Triplet Loss	78.0	35.72	20.26	16.07	23.60
	Contrastive Loss	78.0	35.95	21.02	16.09	23.54
RESNet	Cross-Entropy	88.72	35.97	13.19	15.83	26.43
	Arcface Loss	88.74	35.88	13.15	15.85	26.45
	Triplet Loss	90.34	33.90	20.08	16.40	30.56
	Contrastive Loss	89.97	34.05	19.24	16.86	29.33
MobileNet	Cross-Entropy	72.36	40.00	16.68	14.75	21.33
	Arcface Loss	72.45	39.91	16.65	14.76	19.49
	Triplet Loss	73.01	39.62	21.62	14.80	21.33
	Contrastive Loss	72.99	39.90	15. 57	14.86	21.36
VGG16	Cross-Entropy	88.00	35.99	13.20	15.96	26.07
	Arcface Loss	88.00	35.86	13.17	15.81	26.17
	Triplet Loss	89.36	33.91	20.04	16.29	28.77
	Contrastive Loss	88.60	34.10	19.37	16.22	28.36

Table 5.4: RFGSM Classification confidence % results per experimental epsilon. The green cells show correct classification and the red cells show wrong classification

		Classification Confidence % per Epsilon value				
Classification Model	**Loss Function**	**Original**	**1.8/255**	**15/255**	**70/255**	**150/255**
SENet	Cross-Entropy	78.03	43.72	9.44	24.36	31.91
	Arcface Loss	78.07	43.66	10.00	24.36	32.03
	Triplet Loss	78.0	36.22	20.43	25.11	33.76
	Contrastive Loss	78.0	35.95	21.02	24.09	33.54
RESNet	Cross-Entropy	88.72	39.22	18.78	24.83	26.43
	Arcface Loss	88.74	40.06	17.41	24.85	26.45
	Triplet Loss	90.34	37.38	10.77	28.80	57.21
	Contrastive Loss	89.97	34.24	28.64	28.00	48.97
MobileNet	Cross-Entropy	72.36	68.95	38.97	15.83	16.12
	Arcface Loss	72.45	56.23	38.97	15.90	16.74
	Triplet Loss	73.01	69.78	25.75	78.80	17.31
	Contrastive Loss	72.99	63.35	24.77	10.00	17.27
VGG16	Cross-Entropy	88.00	44.79	13.20	7.21	24.66
	Arcface Loss	88.00	40.26	5.04	10.47	24.59
	Triplet Loss	89.36	32.90	9.63	28.11	39.03
	Contrastive Loss	88.60	32.99	6.70	27.45	38.76

Table 5.5: BIM Classification confidence % results per experimental epsilon. The green cells show correct classification and the red cells show wrong classification

		Classification Confidence % per Epsilon value				
Classification Model	**Loss Function**	**Original**	**1/255**	**7.8/255**	**15/255**	**50/255**
SENet	Cross-Entropy	78.03	40.61	29.23	10.66	21.76
	Arcface Loss	78.07	40.47	28.93	10.27	21.89
	Triplet Loss	78.0	38.85	25.46	10.95	21.89
	Contrastive Loss	78.0	39.03	25.67	10.86	21.84
RESNet	Cross-Entropy	88.72	39.33	19.84	21.70	44.27
	Arcface Loss	88.74	39.21	16.75	23.34	46.81
	Triplet Loss	90.34	30.94	9.10	46.77	60.01
	Contrastive Loss	89.97	33.43	10.19	46.07	57.90
MobileNet	Cross-Entropy	72.36	59.77	51.25	43.69	29.45
	Arcface Loss	72.45	59.61	50.74	43.63	29.33
	Triplet Loss	73.01	53.00	37.68	10.29	34.17
	Contrastive Loss	72.99	58.64	49.12	9.00	12.12
VGG16	Cross-Entropy	88.00	39.01	20.13	19.24	40.77
	Arcface Loss	88.00	38.78	16.77	20.99	51.26
	Triplet Loss	89.36	31.04	10.93	37.21	55.72
	Contrastive Loss	88.60	31.05	11.74	20.41	31.22

Table 5.6: PGD Classification confidence % results per experimental epsilon. The green cells show correct classification and the red cells show wrong classification

		Classification Confidence % per Epsilon value				
Classification Model	**Loss Function**	**Original**	**4/255**	**8/255**	**26/255**	**77/255**
SENet	Cross-Entropy	78.03	37.26	15.67	15.93	22.39
	Arcface Loss	78.07	35.88	15.55	15.91	22.45
	Triplet Loss	78.0	35.70	20.31	25.11	23.76
	Contrastive Loss	78.0	35.96	21.16	24.09	23.54
RESNet	Cross-Entropy	88.72	35.99	13.23	15.76	26.54
	Arcface Loss	88.74	36.05	13.02	15.79	26.78
	Triplet Loss	90.34	33.37	20.12	16.80	31.24
	Contrastive Loss	89.97	34.20	20.09	16.56	31.00
MobileNet	Cross-Entropy	72.36	39.43	17.08	14.78	21.10
	Arcface Loss	72.45	39.35	16.56	14.25	20.37
	Triplet Loss	73.01	39.20	25.75	14.19	22.76
	Contrastive Loss	72.99	39.34	24.77	14.26	12.55
VGG16	Cross-Entropy	88.00	35.87	13.99	15.81	25.97
	Arcface Loss	88.00	35.83	13.37	15.82	25.99
	Triplet Loss	89.36	33.41	20.00	16.79	29.04
	Contrastive Loss	88.60	34.99	19.69	15.99	28.97

Table 5.7: RPGD Classification confidence % results per experimental epsilon. The green cells show correct classification and the red cells show wrong classification

		Classification Confidence % per Epsilon value				
Classification Model	**Loss Function**	**Original**	**1/255**	**8/255**	**15/255**	**40/255**
SENet	Cross-Entropy	78.03	37.23	15.33	14.99	22.47
	Arcface Loss	78.07	35.91	15.31	15.83	22.51
	Triplet Loss	78.0	35.66	20.24	26.08	29.46
	Contrastive Loss	78.0	35.98	21.78	25.43	29.34
RESNet	Cross-Entropy	88.72	34.60	13.21	15.89	26.73
	Arcface Loss	88.74	34.45	13.15	15.96	26.84
	Triplet Loss	90.34	30.27	20.34	16.97	31.67
	Contrastive Loss	89.97	30.37	20.11	16.99	31.32
MobileNet	Cross-Entropy	72.36	39.41	17.02	14.69	21.10
	Arcface Loss	72.45	39.29	17.01	14.82	21.29
	Triplet Loss	73.01	38.47	26.06	15.12	22.45
	Contrastive Loss	72.99	38.58	25.85	14.97	22.35
VGG16	Cross-Entropy	88.00	34.71	13.95	15.00	25.88
	Arcface Loss	88.00	34.60	13.62	15.31	25.31
	Triplet Loss	89.36	31.22	24.77	16.72	29.14
	Contrastive Loss	88.60	31.67	24.65	15.96	28.93

Table 5.8: C & W Classification confidence % results per experimental epsilon. The green cells show correct classification and the red cells show wrong classification.

		Classification Confidence % per Epsilon value	
Classification Model	**Loss Function**	**Original**	**C = 10 —— L.R = 4/255**
SENet	Cross-Entropy	78.03	57.22
	Arcface Loss	78.07	58.13
	Triplet Loss	78.0	57.94
	Contrastive Loss	78.0	58.01
RESNet	Cross-Entropy	88.72	57.09
	Arcface Loss	88.74	57.47
	Triplet Loss	90.34	57.43
	Contrastive Loss	89.97	57.45
MobileNet	Cross-Entropy	72.36	60.14
	Arcface Loss	72.45	60.56
	Triplet Loss	73.01	60.28
	Contrastive Loss	72.99	60.39
VGG16	Cross-Entropy	88.00	57.97
	Arcface Loss	88.00	57.63
	Triplet Loss	89.36	57.94
	Contrastive Loss	88.60	58.32

5.1.3 Adversarial Attacks on defended models

After implementing the discussed defense methods, the models were tested to see how they will perform. Only adversarial examples that successfully deceived all models for each type of adversarial attack were generated in this part of the experiments. Defensive distillation, Feature squeezing, Adversarial training and Magnet defense methods were used against all adversarial attacks. Figures 5.9, 5.10, 5.11, 5.12, 5.13 the confidences of all defense mechanisms were trended per models. In these graphs the x-axis is models and y-axis is confidences. The graphs were plotted to examine the overall trends in confidence levels among different defense mechanisms.
The most salient point to note is that the feature squeezing defense methods did not produce any defense and were deviant from the expected results, sometimes it was causing high confidence in wrong classification.

It is also noted that all other defense methods managed to make the models correctly classify the given adversarial example. This leads us to a discussion in which the confidence rates are compared.

Defenses against the FGSM at epsilon = 1.8/255

It is seen that from table 5.9, Defense distillation, Adversarial training, and Magnet defense managed to defend all models, more so with a high confidence rate. It is however noted that the average confidence rates of defense distillation are lower compared to what is seen from Adversarial training and Magnet Defense as shown in figure 5.9. Feature squeezing gave the least confidence rates but it is important to note that the confidence rates of feature squeezing are those for false classification.

Defenses against the RFGSM at epsilon = 1.8/255

From table 5.10, again Defense distillation, Adversarial training and Magnet defense managed to defend all models, more so with a high confidence rate. Just as with the FGSM, the average confidence rates of defense distillation are lower than those of Adversarial training and Magnet Defense as shown in figure 5.10.

Defenses against the BIM at epsilon = 1/255

From table 5.11, it is noted a similar trend with defenses of FGSM and RFGSM, as also shown in figure 5.10.

It is observed that for the Defensive Distillation method, the highest confidence rates are observed in the RESNet model with Triplet Loss, followed closely by the RESNet model with Contrastive Loss. The confidence rates for these models are recorded as 89.83% and 89.81% respectively.The Lowest confidence rates are from MobileNet with Cross-Entropy followed by MobileNet with Arcface, the confidence rates are 54.20% and 69.30% respectively.

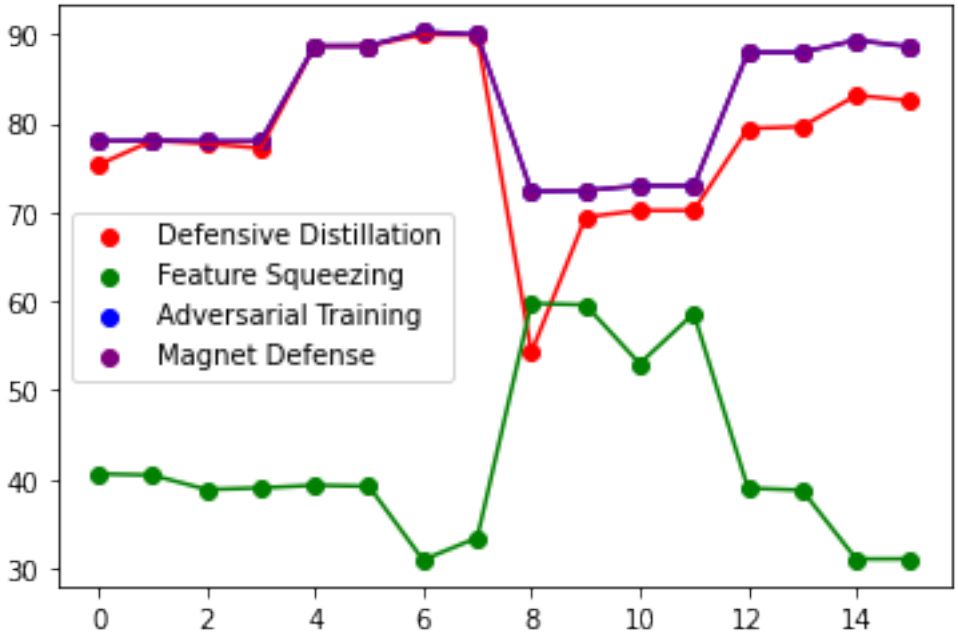


Figure 5.9: FGSM Trending of confidence rates of all models per defense mechanism. This is to depict which defense model gives a higher confidence rate.

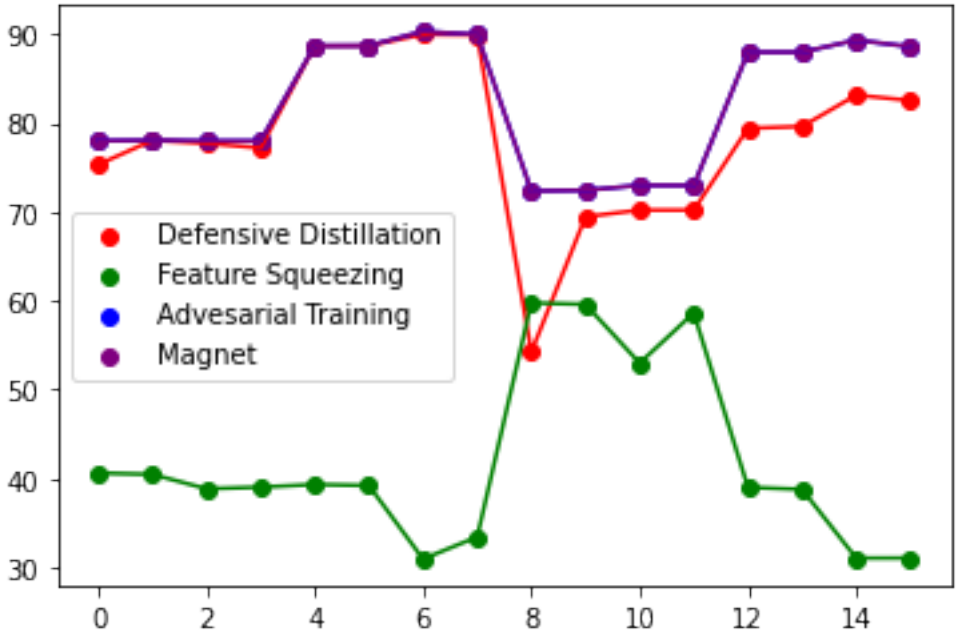


Figure 5.10: RFGSM Trending of confidence rates of all models per defense mechanism. This is to depict which defense model gives a higher confidence rate.

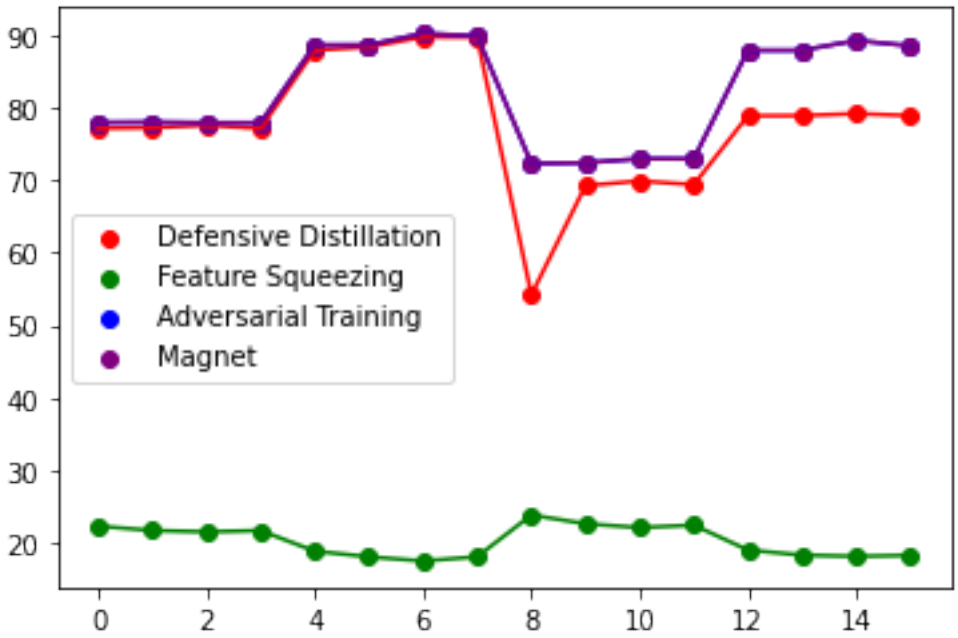


Figure 5.11: RFGSM Trending of confidence rates of all models per defense mechanism. This is to depict which defense model gives a higher confidence rate.

Defenses against the PGD at epsilon = 4/255

From table 5.12, as it is noted in trends from defenses against the FGSM, RFGSM, and BIM, the Defense distillation, Adversarial training, and Magnet defense were also able to defend all models. However, a decrease in confidence rates is observed for the Adversarial Training defense. The Magnet defense still maintains high confidence rates of true classification. This shown in fig 5.12.

It is also seen that the highest confidence rates for Defensive distillation are from the RESNet with Contrastive Loss followed by VGG16 with Triplet loss, the confidence rates are 88.79% and 85.00% respectively. The Lowest confidence rates are from MobileNet with Cross-Entropy followed by MobileNet with Arcface, the confidence rates are 44.95% and 50.20% respectively. For Adversarial training, the highest confidence rates are from the RESNet with Triplet lossfollowed by RESNet with Contrastive Loss, the confidence rates are 87.28% and 87.03% respectively. Deviant from the notable the Lowest confidence rates are from SENet with Cross-Entropy followed by SENet with Arcface, the confidence rates are 71.29% and 71.67% respectively.

Defenses against the RPGD at epsilon = 1/255

From table 5.13, as the trends above, the Defense distillation, Adversarial training, and Magnet defense were able to defend all models and the confidence rates for the Adversarial training defense had a drop which in this case is visibly lower than the confidence rate for the Defensive Distillation. The Magnet defense still maintains high confidence rates of true classification. This is shown in fig 5.13.

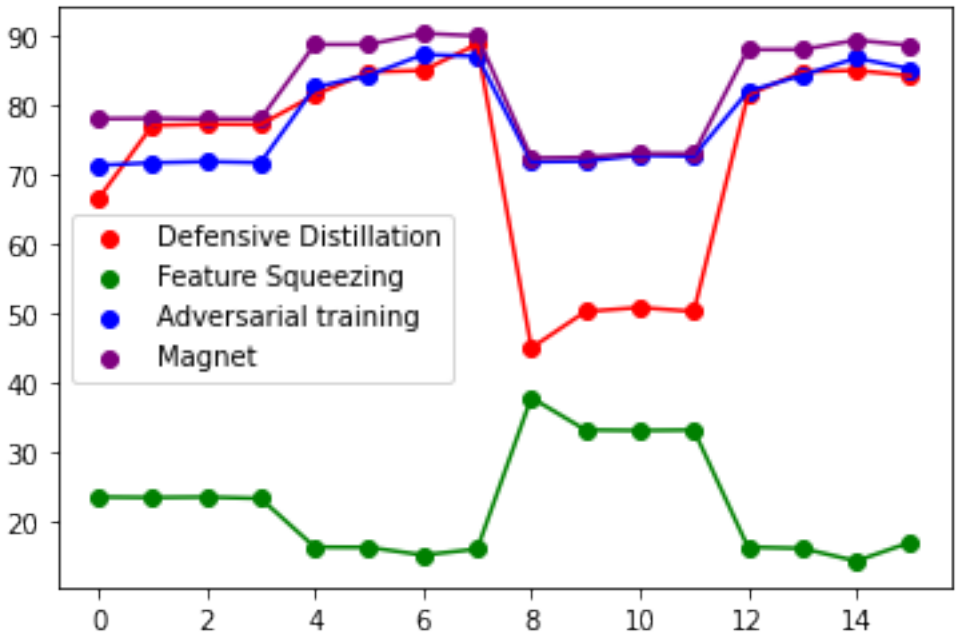


Figure 5.12: PGD Trending of confidence rates of all models per defense mechanism. This is to depict which defence model give a higher confidence rate.

Results in this experiment show a similar trend similar to those seen in the experiments defending again the PGD attacks, the highest confidence rates for Defensive distillation are from the RESNet with Contrastive Loss followed by RESNet with Triplet loss, the confidence rates are 88.81% and 84.92% respectively. The Lowest confidence rates are from MobileNet with Cross-Entropy followed by MobileNet with Arcface, the confidence rates are 45.01% and 50.24% respectively. For Adversarial training, the highest confidence rates are from the RESNet with Triplet loss followed by VGG16 with Triplet Loss, the confidence rates are 84.01% and 83.53% respectively.

Defense against Carlini and Wagner attack at C=10 and L.R = 4/255

From table 5.14, it is seen that all adversarial defense methods failed to defend against this attack. It is evident from the confidence rates that all MobileNET models were fooled the most followed by SENET models as they gave high confidence rates in misclassification. This is possibly because this attack takes into consideration both the misclassification rate and the perturbation level into the objective function, which was not often noticed in the previous attack strategies.

RESNet models had the lowest confidence rates in misclassification. The lower being from RESNET with Triplet loss and Contrastive loss, for adversarial training which had confidence rates of 17.02% and 17.19%. Same as for defensive distillation, the RESNet models had the highest confidence rates of 17.22% for RESNet with Triplet loss and 17.69 for RESNet with Contrastive Loss. MobileNet had the highest confidence rates for adversarial training with 65.49% for MobileNet with Cross-Entropy and 60.97% for MobileNet with Arcface. Even with the defensive distillation, it is seen that MobileNet

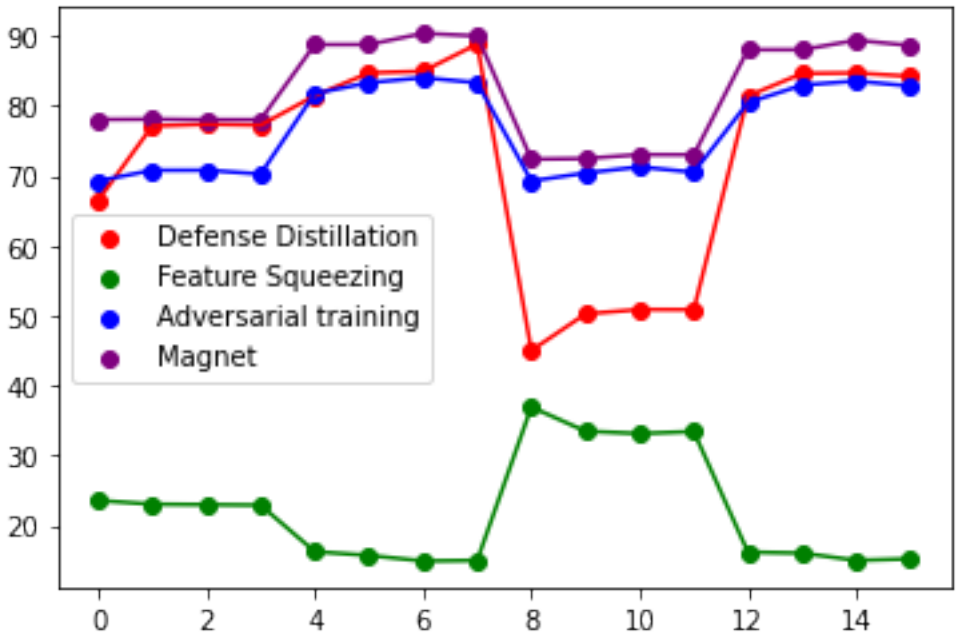


Figure 5.13: RPGD Trending of confidence rates of all models per defense mechanism. This is to depict which defence model give a higher confidence rate.

had the highest confidence rate with 68.79% for MobileNet with Cross-Entropy and 64.91% for MobileNet with Arcface.

It is evident in figure 5.14 that RESNet and VGG16 models have significantly lower confidence while SENet and MobileNet models have higher confidence in misclassification.

Discussion Summary

From the results it is apparent that better-performing models have a better ability to resist adversarial attacks. This is because better-performing models are able to learn more robust features from the training data. These features are less susceptible to small changes in the input data, such as those introduced by adversarial attacks.

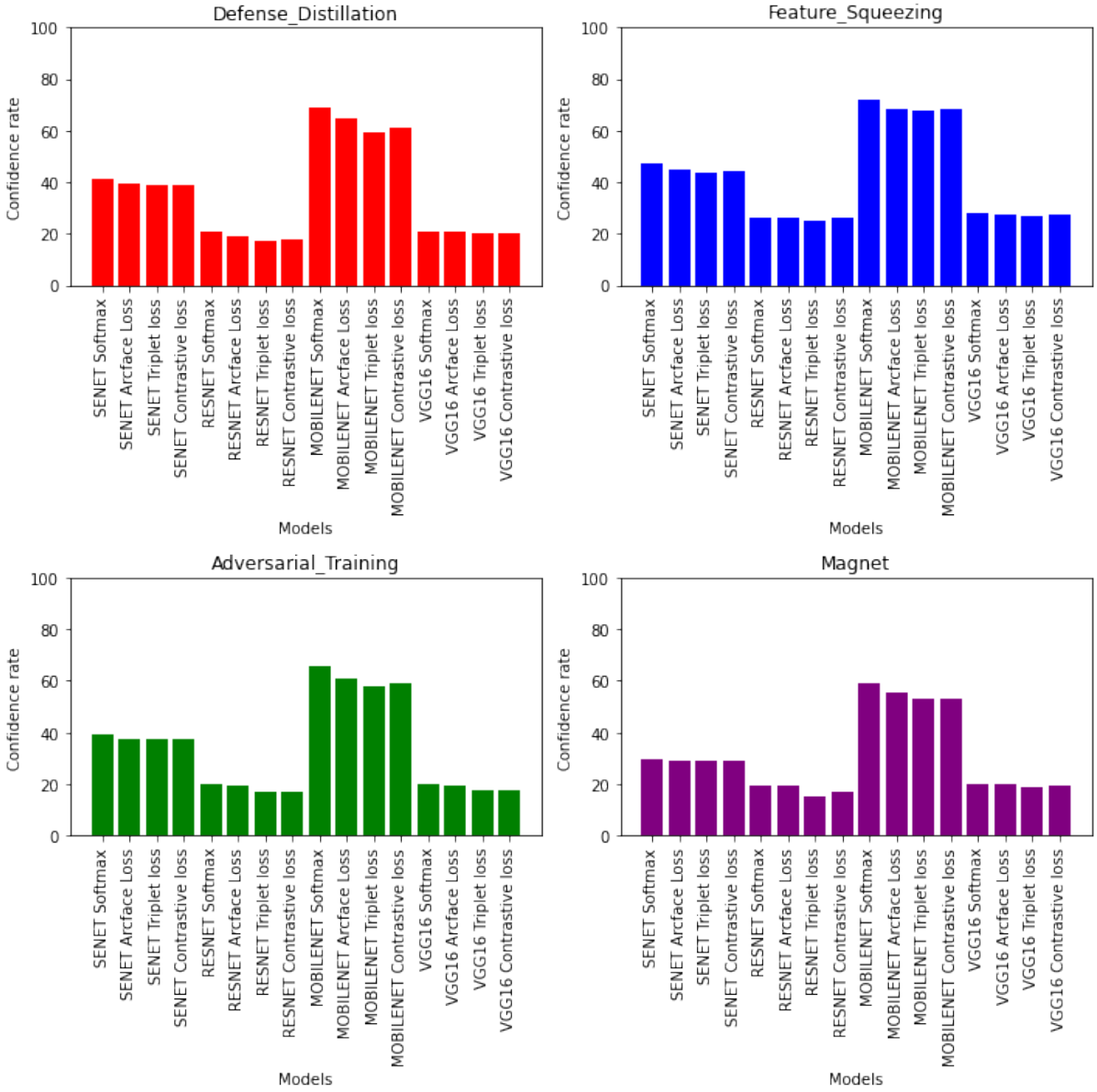


Figure 5.14: Carlini and Wargner attack bar graph of confidence rates per defense method.

Table 5.9: Classification confidence after defending models against the FGSM attack at epsilon 1.8/255

		Classification confidence after defenses for FGSM epsilon 1.8/255			
Classification Model	**Loss Function**	**Defense Distillation**	**Feature Squeezing**	**Adversarial Training**	**Magnet**
SENet	Cross-Entropy	75.34	40.61	78.03	78.03
	Arcface Loss	78.07	40.47	78.07	78.07
	Triplet Loss	77.74	38.85	78.0	78.0
	Contrastive Loss	77.21	39.03	78.0	78.0
RESNet	Cross-Entropy	88.70	39.33	88.73	88.73
	Arcface Loss	88.72	39.21	88.74	88.74
	Triplet Loss	89.99	30.94	90.34	90.34
	Contrastive Loss	89.94	33.43	89.97	89.97
MobileNet	Cross-Entropy	54.25	59.77	72.36	72.36
	Arcface Loss	69.47	59.61	72.45	72.45
	Triplet Loss	70.22	53.00	73.01	73.01
	Contrastive Loss	70.22	58.64	72.99	72.99
VGG16	Cross-Entropy	79.43	39.01	88.00	88.00
	Arcface Loss	79.63	38.78	88.00	88.00
	Triplet Loss	83.17	31.04	89.36	89.36
	Contrastive Loss	82.55	31.05	88.60	88.60

Table 5.10: Classification confidence after defending models against the RFGSM attack at epsilon 1.8/255

		Classification confidence after defenses for RFGSM epsilon 1.8/255			
Classification Model	**Loss Function**	**Defense Distillation**	**Feature Squeezing**	**Adversarial Training**	**Magnet**
SENet	Cross-Entropy	75.34	40.61	78.03	78.03
	Arcface Loss	78.07	40.47	78.07	78.07
	Triplet Loss	77.74	38.85	78.0	78.0
	Contrastive Loss	77.21	39.03	78.0	78.0
RESNet	Cross-Entropy	88.70	39.33	88.73	88.73
	Arcface Loss	88.72	39.21	88.74	88.74
	Triplet Loss	89.99	30.94	90.34	90.34
	Contrastive Loss	89.94	33.43	89.97	89.97
MobileNet	Cross-Entropy	54.25	59.77	72.36	72.36
	Arcface Loss	69.47	59.61	72.45	72.45
	Triplet Loss	70.22	53.00	73.01	73.01
	Contrastive Loss	70.22	58.64	72.99	72.99
VGG16	Cross-Entropy	79.43	39.01	88.00	88.00
	Arcface Loss	79.63	38.78	88.00	88.00
	Triplet Loss	83.17	31.04	89.36	89.36
	Contrastive Loss	82.55	31.05	88.60	88.60

Table 5.11: Classification confidence after defending models against the BIM attack at epsilon 1/255

		Classification confidence after defenses for BIM epsilon 1/255			
Classification Model	**Loss Function**	**Defense Distillation**	**Feature Squeezing**	**Adversarial Training**	**Magnet**
SENet	Cross-Entropy	77.21	22.24	78.03	78.03
	Arcface Loss	77.31	21.64	78.07	78.07
	Triplet Loss	77.63	21.45	78.0	78.0
	Contrastive Loss	77.20	21.63	78.0	78.0
RESNet	Cross-Entropy	88.01	18.78	88.73	88.73
	Arcface Loss	88.45	18.00	88.74	88.74
	Triplet Loss	89.83	17.42	90.34	90.34
	Contrastive Loss	89.81	18.01	89.97	89.97
MobileNet	Cross-Entropy	54.20	23.78	72.36	72.36
	Arcface Loss	69.30	22.57	72.45	72.45
	Triplet Loss	69.94	22.04	73.01	73.01
	Contrastive Loss	69.44	22.38	72.99	72.99
VGG16	Cross-Entropy	78.96	18.94	88.00	88.00
	Arcface Loss	78.97	18.23	88.00	88.00
	Triplet Loss	79.25	18.07	89.36	89.36
	Contrastive Loss	78.97	18.20	88.60	88.60

Table 5.12: Classification confidence after defending models against the PGD attack at epsilon 4/255

		Classification confidence after defenses for PGD epsilon 4/255			
Classification Model	**Loss Function**	**Defense Distillation**	**Feature Squeezing**	**Adversarial Training**	**Magnet**
SENet	Cross-Entropy	66.45	23.44	71.29	78.03
	Arcface Loss	77.04	23.37	71.67	78.07
	Triplet Loss	77.23	23.43	71.84	78.0
	Contrastive Loss	77.19	23.23	71.69	78.0
RESNet	Cross-Entropy	81.46	16.21	82.56	88.73
	Arcface Loss	84.77	16.17	84.31	88.74
	Triplet Loss	84.98	15.06	87.28	90.34
	Contrastive Loss	88.79	15.93	87.03	89.97
MobileNet	Cross-Entropy	44.95	37.78	71.80	72.36
	Arcface Loss	50.20	33.11	71.89	72.45
	Triplet Loss	50.78	33.04	72.74	73.01
	Contrastive Loss	50.21	33.10	72.63	72.99
VGG16	Cross-Entropy	81.44	16.21	81.97	88.00
	Arcface Loss	84.79	16.03	84.29	88.00
	Triplet Loss	85.00	14.20	86.78	89.36
	Contrastive Loss	84.21	16.90	85.21	88.60

Table 5.13: Classification confidence after defending models against the RPGD attack at epsilon 1/255

		Classification confidence after defenses for RPGD epsilon 1/255			
Classification Model	**Loss Function**	**Defense Distillation**	**Feature Squeezing**	**Adversarial Training**	**Magnet**
SENet	Cross-Entropy	66.47	23.57	69.25	78.03
	Arcface Loss	77.10	23.04	70.77	78.07
	Triplet Loss	77.34	22.98	70.78	78.0
	Contrastive Loss	77.22	22.91	70.23	78.0
RESNet	Cross-Entropy	81.43	16.26	81.75	88.73
	Arcface Loss	84.69	15.72	83.29	88.74
	Triplet Loss	84.92	14.92	84.01	90.34
	Contrastive Loss	88.81	15.01	83.28	89.97
MobileNet	Cross-Entropy	45.01	36.93	69.24	72.36
	Arcface Loss	50.24	33.47	70.38	72.45
	Triplet Loss	50.87	33.14	71.25	73.01
	Contrastive Loss	50.85	33.42	70.49	72.99
VGG16	Cross-Entropy	81.38	16.19	80.44	88.00
	Arcface Loss	84.61	16.05	82.94	88.00
	Triplet Loss	84.68	15.01	83.53	89.36
	Contrastive Loss	84.20	15.23	82.80	88.60

Table 5.14: Classification confidence after defending models against the C & W attack at C =10 AND L.R = 4/255

		Classification confidence after defenses for C & W. C =10 —— L.R = 4.255			
Classification Model	**Loss Function**	**Defense Distillation**	**Feature Squeezing**	**Adversarial Training**	**Magnet**
SENet	Cross-Entropy	41.53	47.21	38.98	29.44
	Arcface Loss	39.31	44.89	37.56	28.97
	Triplet Loss	39.06	43.58	37.24	28.93
	Contrastive Loss	39.09	44.07	37.51	28.95
RESNet	Cross-Entropy	20.73	26.44	19.81	19.66
	Arcface Loss	19.24	26.28	19.26	19.53
	Triplet Loss	17.22	25.15	17.02	15.21
	Contrastive Loss	17.69	26.39	17.19	16.74
MobileNet	Cross-Entropy	68.79	71.77	65.49	59.23
	Arcface Loss	64.91	68.34	60.97	55.71
	Triplet Loss	59.29	68.02	57.84	52.94
	Contrastive Loss	60.94	68.27	58.93	53.17
VGG16	Cross-Entropy	20.93	27.95	19.89	20.24
	Arcface Loss	20.62	27.41	19.31	20.12
	Triplet Loss	20.17	27.09	17.64	18.94
	Contrastive Loss	20.39	27.17	17.69	19.26

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this study, the effect of commonly used adversarial attacks and defenses was explored on different types of facial recognition techniques which were classified as classification models with Cross-Entropy and Arcface losses, the metric learning models with Triplet loss and Contrastive loss. As a result of this experiment, the following conclusions were drawn:

1. Of the models tested, the models with the RESNet50 architecture give better classification performance for the CASIA-Webface and LFW datasets. Follow by VGG16 and SENet models respectively. The least performance was from Mobilenet models.

2. Metric learning loss functions give better facial recognition results as compared to classification loss functions.As seen in all models, Triplet loss and Contrastive loss functions had better results than Cross-Entropy and Arcface. However the difference between Arcface and other metric learning loss functions is small.

3. There is a link between performance in classification and robustness against adversarial attacks. The confidence levels of metric learning models for RESNet and VGG16 performed better in terms of resisting the adversarial examples and also performed better when helped with defense methods. This talks mainly to their capability of separating close images of different classes on the classification manifold.

4. The Magnet defense method does not really work to improve the robustness of a model because it does not work with model knowledge, indicated by how the confidence rates of original images are similar to confidence rates after the defense of Magnet defence method.

5. Smaller epsilons produce adversarial examples that fool models better than large epsilons

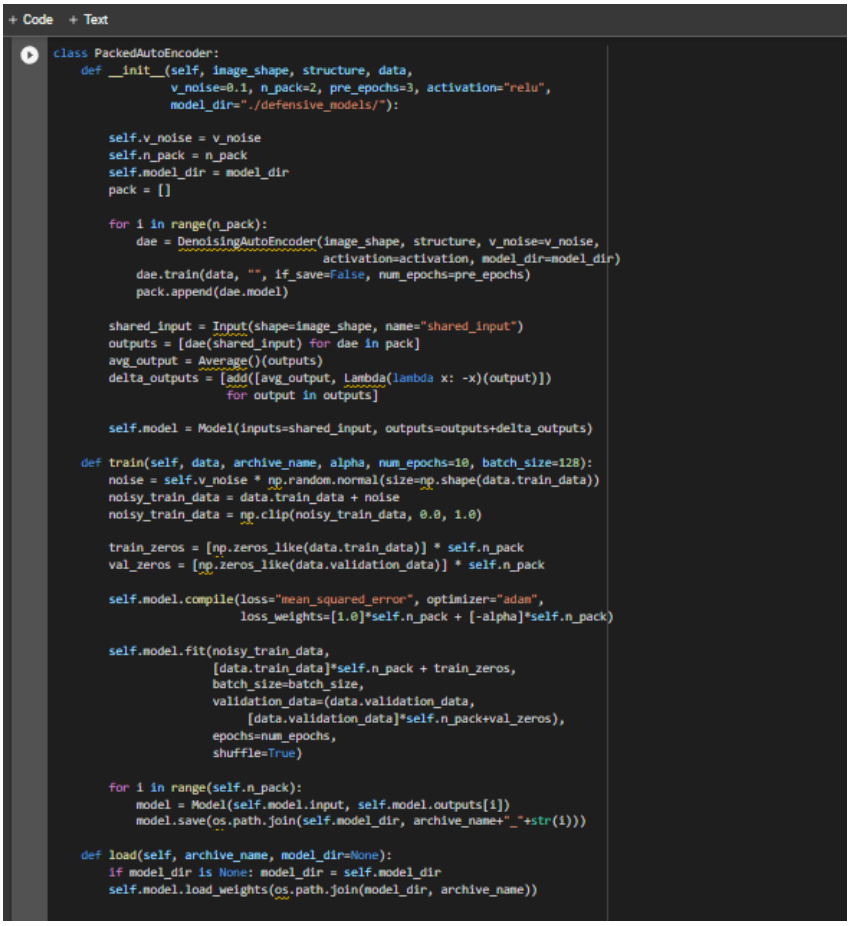
6. When fooled, better models like RESNet and VGG16 with metric learning loss functions give lower confidence rates compared to weaker models like MobileNet with Cross-Entropy, that is classification models in general.

6.2 Future Work

Future work includes testing these facial recognition models and adversarial attacks in real-time. It is important to see how models which work on security entrances can be spoofed or attacked and see which models or defense mechanisms are robust enough against spoofing. This will mainly be black-box attacks and defenses because it is generally difficult to have access to the cloud and servers where the models and training data for these systems are kept.

Appendix A

Magnet function in python

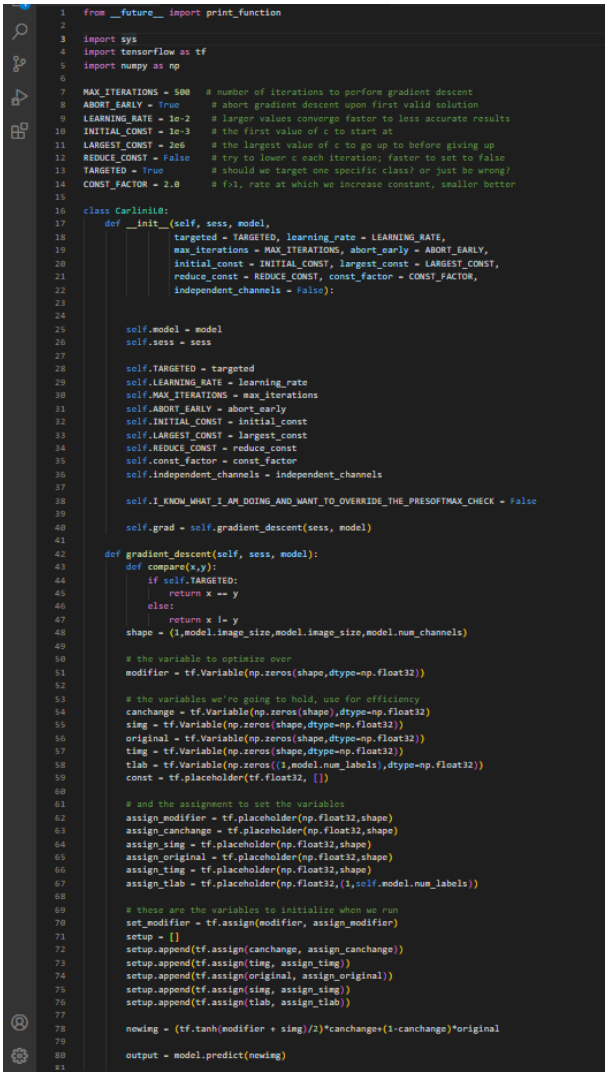


```
+ Code  + Text

class PackedAutoEncoder:
    def __init__(self, image_shape, structure, data,
                 v_noise=0.1, n_pack=2, pre_epochs=3, activation="relu",
                 model_dir="./defensive_models/"):

        self.v_noise = v_noise
        self.n_pack = n_pack
        self.model_dir = model_dir
        pack = []

        for i in range(n_pack):
            dae = DenoisingAutoEncoder(image_shape, structure, v_noise=v_noise,
                                       activation=activation, model_dir=model_dir)
            dae.train(data, "", if_save=False, num_epochs=pre_epochs)
            pack.append(dae.model)

        shared_input = Input(shape=image_shape, name="shared_input")
        outputs = [dae(shared_input) for dae in pack]
        avg_output = Average()(outputs)
        delta_outputs = [add([avg_output, Lambda(lambda x: -x)(output)])
                         for output in outputs]

        self.model = Model(inputs=shared_input, outputs=outputs+delta_outputs)

    def train(self, data, archive_name, alpha, num_epochs=10, batch_size=128):
        noise = self.v_noise * np.random.normal(size=np.shape(data.train_data))
        noisy_train_data = data.train_data + noise
        noisy_train_data = np.clip(noisy_train_data, 0.0, 1.0)

        train_zeros = [np.zeros_like(data.train_data)] * self.n_pack
        val_zeros = [np.zeros_like(data.validation_data)] * self.n_pack

        self.model.compile(loss="mean_squared_error", optimizer="adam",
                           loss_weights=[1.0]*self.n_pack + [-alpha]*self.n_pack)

        self.model.fit(noisy_train_data,
                       [data.train_data]*self.n_pack + train_zeros,
                       batch_size=batch_size,
                       validation_data=(data.validation_data,
                            [data.validation_data]*self.n_pack+val_zeros),
                       epochs=num_epochs,
                       shuffle=True)

        for i in range(self.n_pack):
            model = Model(self.model.input, self.model.outputs[i])
            model.save(os.path.join(self.model_dir, archive_name+"_"+str(i)))

    def load(self, archive_name, model_dir=None):
        if model_dir is None: model_dir = self.model_dir
        self.model.load_weights(os.path.join(model_dir, archive_name))
```

Figure A.1: Magnet function implementation in python

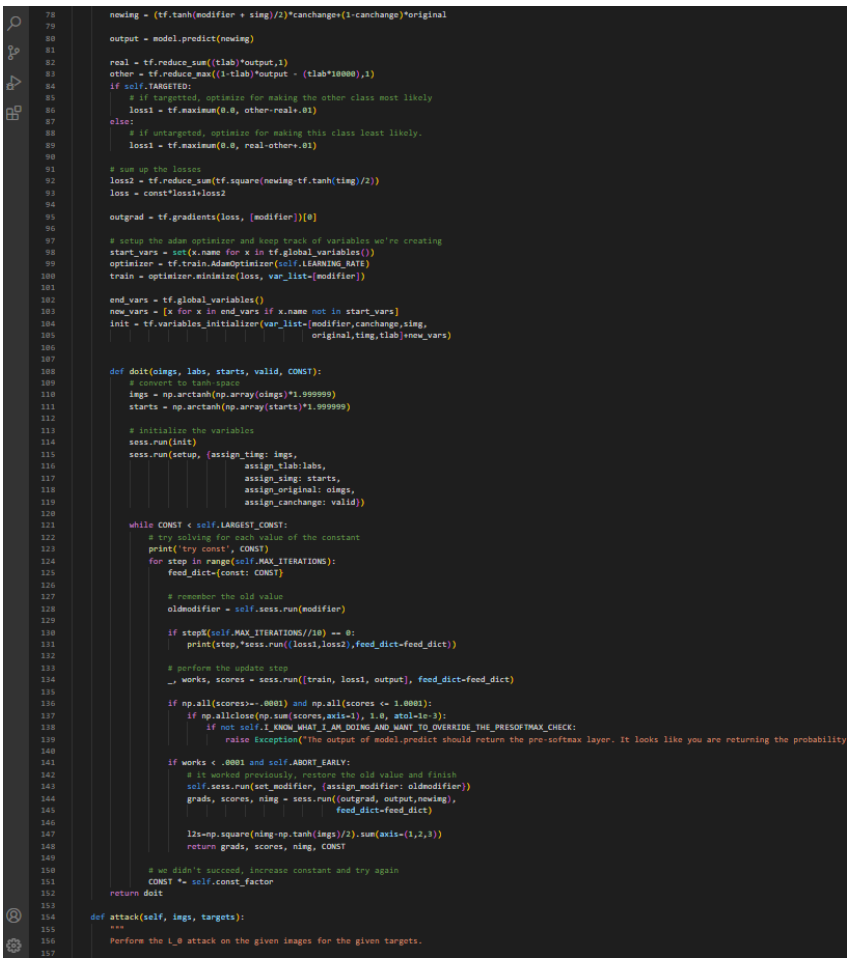
Appendix B

Carlini and Wargner



```python
from __future__ import print_function

import sys
import tensorflow as tf
import numpy as np

MAX_ITERATIONS = 500   # number of iterations to perform gradient descent
ABORT_EARLY = True     # abort gradient descent upon first valid solution
LEARNING_RATE = 1e-2   # larger values converge faster to less accurate results
INITIAL_CONST = 1e-3   # the first value of c to start at
LARGEST_CONST = 2e6    # the largest value of c to go up to before giving up
REDUCE_CONST = False   # try to lower c each iteration; faster to set to false
TARGETED = True        # should we target one specific class? or just be wrong?
CONST_FACTOR = 2.0     # f>1, rate at which we increase constant, smaller better

class CarliniL0:
    def __init__(self, sess, model,
                 targeted = TARGETED, learning_rate = LEARNING_RATE,
                 max_iterations = MAX_ITERATIONS, abort_early = ABORT_EARLY,
                 initial_const = INITIAL_CONST, largest_const = LARGEST_CONST,
                 reduce_const = REDUCE_CONST, const_factor = CONST_FACTOR,
                 independent_channels = False):


        self.model = model
        self.sess = sess

        self.TARGETED = targeted
        self.LEARNING_RATE = learning_rate
        self.MAX_ITERATIONS = max_iterations
        self.ABORT_EARLY = abort_early
        self.INITIAL_CONST = initial_const
        self.LARGEST_CONST = largest_const
        self.REDUCE_CONST = reduce_const
        self.const_factor = const_factor
        self.independent_channels = independent_channels

        self.I_KNOW_WHAT_I_AM_DOING_AND_WANT_TO_OVERRIDE_THE_PRESOFTMAX_CHECK = False

        self.grad = self.gradient_descent(sess, model)

    def gradient_descent(self, sess, model):
        def compare(x,y):
            if self.TARGETED:
                return x == y
            else:
                return x != y
        shape = (1,model.image_size,model.image_size,model.num_channels)

        # the variable to optimize over
        modifier = tf.Variable(np.zeros(shape,dtype=np.float32))

        # the variables we're going to hold, use for efficiency
        canchange = tf.Variable(np.zeros(shape),dtype=np.float32)
        simg = tf.Variable(np.zeros(shape,dtype=np.float32))
        original = tf.Variable(np.zeros(shape,dtype=np.float32))
        timg = tf.Variable(np.zeros(shape,dtype=np.float32))
        tlab = tf.Variable(np.zeros((1,model.num_labels),dtype=np.float32))
        const = tf.placeholder(tf.float32, [])

        # and the assignment to set the variables
        assign_modifier = tf.placeholder(np.float32,shape)
        assign_canchange = tf.placeholder(np.float32,shape)
        assign_simg = tf.placeholder(np.float32,shape)
        assign_original = tf.placeholder(np.float32,shape)
        assign_timg = tf.placeholder(np.float32,shape)
        assign_tlab = tf.placeholder(np.float32,(1,self.model.num_labels))

        # these are the variables to initialize when we run
        set_modifier = tf.assign(modifier, assign_modifier)
        setup = []
        setup.append(tf.assign(canchange, assign_canchange))
        setup.append(tf.assign(timg, assign_timg))
        setup.append(tf.assign(original, assign_original))
        setup.append(tf.assign(simg, assign_simg))
        setup.append(tf.assign(tlab, assign_tlab))

        newimg = (tf.tanh(modifier + simg)/2)*canchange+(1-canchange)*original

        output = model.predict(newimg)
```

Figure B.1: Carlini and Wagner function first screenshot piece



```
        newimg = (tf.tanh(modifier + simg)/2)*canchange+(1-canchange)*original

        output = model.predict(newimg)

        real = tf.reduce_sum((tlab)*output,1)
        other = tf.reduce_max((1-tlab)*output - (tlab*10000),1)
        if self.TARGETED:
            # if targetted, optimize for making the other class most likely
            loss1 = tf.maximum(0.0, other-real+.01)
        else:
            # if untargeted, optimize for making this class least likely.
            loss1 = tf.maximum(0.0, real-other+.01)

        # sum up the losses
        loss2 = tf.reduce_sum(tf.square(newimg-tf.tanh(timg)/2))
        loss = const*loss1+loss2

        outgrad = tf.gradients(loss, [modifier])[0]

        # setup the adam optimizer and keep track of variables we're creating
        start_vars = set(x.name for x in tf.global_variables())
        optimizer = tf.train.AdamOptimizer(self.LEARNING_RATE)
        train = optimizer.minimize(loss, var_list=[modifier])

        end_vars = tf.global_variables()
        new_vars = [x for x in end_vars if x.name not in start_vars]
        init = tf.variables_initializer(var_list=[modifier,canchange,simg,
                                                  original,timg,tlab]+new_vars)


        def doit(oimgs, labs, starts, valid, CONST):
            # convert to tanh-space
            imgs = np.arctanh(np.array(oimgs)*1.999999)
            starts = np.arctanh(np.array(starts)*1.999999)

            # initialize the variables
            sess.run(init)
            sess.run(setup, {assign_timg: imgs,
                             assign_tlab:labs,
                             assign_simg: starts,
                             assign_original: oimgs,
                             assign_canchange: valid})

            while CONST < self.LARGEST_CONST:
                # try solving for each value of the constant
                print('try const', CONST)
                for step in range(self.MAX_ITERATIONS):
                    feed_dict={const: CONST}

                    # remember the old value
                    oldmodifier = self.sess.run(modifier)

                    if step%(self.MAX_ITERATIONS//10) == 0:
                        print(step,*sess.run((loss1,loss2),feed_dict=feed_dict))

                    # perform the update step
                    _, works, scores = sess.run([train, loss1, output], feed_dict=feed_dict)

                    if np.all(scores>=-.0001) and np.all(scores <= 1.0001):
                        if np.allclose(np.sum(scores,axis=1), 1.0, atol=1e-3):
                            if not self.I_KNOW_WHAT_I_AM_DOING_AND_WANT_TO_OVERRIDE_THE_PRESOFTMAX_CHECK:
                                raise Exception("The output of model.predict should return the pre-softmax layer. It looks like you are returning the probability

                    if works < .0001 and self.ABORT_EARLY:
                        # it worked previously, restore the old value and finish
                        self.sess.run(set_modifier, {assign_modifier: oldmodifier})
                        grads, scores, nimg = sess.run((outgrad, output,newimg),
                                                       feed_dict=feed_dict)

                        l2s=np.square(nimg-np.tanh(imgs)/2).sum(axis=(1,2,3))
                        return grads, scores, nimg, CONST

                # we didn't succeed, increase constant and try again
                CONST *= self.const_factor
        return doit

    def attack(self, imgs, targets):
        """
        Perform the L_0 attack on the given images for the given targets.
```

Figure B.2: Carlini and Wagner function second screenshot piece

```
    def attack_single(self, img, target):
        """
        Run the attack on a single image and label
        """

        # the pixels we can change
        valid = np.ones((1,self.model.image_size,self.model.image_size,self.model.num_channels))

        # the previous image
        prev = np.copy(img).reshape((1,self.model.image_size,self.model.image_size,
                                     self.model.num_channels))

        # initially set the solution to None, if we can't find an adversarial
        # example then we will return None as the solution.
        last_solution = None
        const = self.INITIAL_CONST

        equal_count = None

        while True:
            # try to solve given this valid map
            res = self.grad([np.copy(img)], [target], np.copy(prev),
                            valid, const)
            if res == None:
                # the attack failed, we return this as our final answer
                print("Final answer",equal_count)
                return last_solution

            # the attack succeeded, now we pick new pixels to set to 0
            restarted = False
            gradientnorm, scores, nimg, const = res
            if self.REDUCE_CONST: const /= 2

            equal_count = self.model.image_size**2-np.sum(np.all(np.abs(img-nimg[0])<.0001,axis=2))
            print("Forced equal:",np.sum(1-valid),
                  "Equal count:",equal_count)
            if np.sum(valid) == 0:
                # if no pixels changed, return
                return [img]

            if self.independent_channels:
                # we are allowed to change each channel independently
                valid = valid.flatten()
                totalchange = abs(nimg[0]-img)*np.abs(gradientnorm[0])
            else:
                # we care only about which pixels change, not channels independently
                # compute total change as sum of change for each channel
                valid = valid.reshape((self.model.image_size**2,self.model.num_channels))
                totalchange = abs(np.sum(nimg[0]-img,axis=2))*np.sum(np.abs(gradientnorm[0]),axis=2)
            totalchange = totalchange.flatten()

            # set some of the pixels to 0 depending on their total change
            did = 0
            for e in np.argsort(totalchange):
                if np.all(valid[e]):
                    did += 1
                    valid[e] = 0

                    if totalchange[e] > .01:
                        # if this pixel changed a lot, skip
                        break
                    if did >= .3*equal_count**.5:
                        # if we changed too many pixels, skip
                        break

            valid = np.reshape(valid,(1,self.model.image_size,self.model.image_size,-1))
            print("Now forced equal:",np.sum(1-valid))

            last_solution = prev = nimg
```

Figure B.3: Carlini and Wagner function third screenshot piece

References

[Almadan and Rattani 2021] Ali Almadan and Ajita Rattani. Towards on-device face recognition in body-worn cameras. In *2021 IEEE International Workshop on Biometrics and Forensics (IWBF)*, pages 1–6. IEEE, 2021.

[Angelova *et al.* 2005] Anelia Angelova, Yaser Abu-Mostafam, and Pietro Perona. Pruning training sets for learning of object categories. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 494–501. IEEE, 2005.

[Athalye *et al.* 2018] Alex Athalye, Logan Engstrom, Andrew Ilyas, and Kunal Kwok. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[Bai *et al.* 2020] Song Bai, Yingwei Li, Yuyin Zhou, Qizhu Li, and Philip HS Torr. Adversarial metric attack and defense for person re-identification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(6):2119–2126, 2020.

[Ballester and Araujo 2016] Pedro Ballester and Ricardo Araujo. On the performance of googlenet and alexnet applied to sketches. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[Brodley and Friedl 1999] Carla E Brodley and Mark A Friedl. Identifying mislabeled training data. *Journal of artificial intelligence research*, 11:131–167, 1999.

[Cao *et al.* 2018] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *2018 13th IEEE international conference on automatic face & gesture recognition (FG 2018)*, pages 67–74. IEEE, 2018.

[Carlini and Wagner 2017] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

[Chen *et al.* 2017] Weihua Chen, Xiaotang Chen, Jianguo Zhang, and Kaiqi Huang. Beyond triplet loss: a deep quadruplet network for person re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 403–412, 2017.

[Cheng *et al.* 2020] Zhiyi Cheng, Xiatian Zhu, and Shaogang Gong. Face re-identification challenge: Are face recognition models good enough? *Pattern Recognition*, 107:107422, 2020.

[Chollet 2017] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1251–1258, 2017.

[Coe and Atay 2021] James Coe and Mustafa Atay. Evaluating impact of race in facial recognition across machine learning and deep learning algorithms. *Computers*, 10(9):113, 2021.

[Croak 2021] Marian Croak. *A decade in deep learning, and what's next*, Nov 2021.

[Deng *et al.* 2019] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4690–4699, 2019.

[Elsayed *et al.* 2018] Gamaleldin Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alexey Kurakin, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial examples that fool both computer vision and time-limited humans. *Advances in neural information processing systems*, 31, 2018.

[Foote 2022] Keith D. Foote. *A brief history of deep learning*, Mar 2022.

[Gao *et al.* 2007] Wen Gao, Bo Cao, Shiguang Shan, Xilin Chen, Delong Zhou, Xiaohua Zhang, and Debin Zhao. The cas-peal large-scale chinese face database and baseline evaluations. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(1):149–161, 2007.

[Goodfellow *et al.* 2014] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[Goodfellow *et al.* 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[Gross *et al.* 2010] Ralph Gross, Iain Matthews, Jeffrey Cohn, Takeo Kanade, and Simon Baker. Multi-pie. *Image and vision computing*, 28(5):807–813, 2010.

[Guo *et al.* 2003] Gongde Guo, Hui Wang, David Bell, Yaxin Bi, and Kieran Greer. Knn model-based approach in classification. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings*, pages 986–996. Springer, 2003.

[Guo *et al.* 2016] Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *European conference on computer vision*, pages 87–102. Springer, 2016.

[He *et al.* 2009] Hui He, Enrique A Garcia, Ya-Qin Li, and Nitesh V Chawla. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.

[He *et al.* 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[He *et al.* 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[Heisele *et al.* 2007] Bernd Heisele, Thomas Serre, and Tomaso Poggio. A component-based framework for face detection and identification. *International Journal of Computer Vision*, 74(2):167–181, 2007.

[Hoffer and Ailon 2015] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*, pages 84–92. Springer, 2015.

[Howard *et al.* 2017] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. In *CVPR*, 2017.

[Hu *et al.* 2012] X Hu, Z Zeng, and D Zhang. Discriminative sparse coding for face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2961–2968, 2012.

[Hu *et al.* 2017] Lanqing Hu, Meina Kan, Shiguang Shan, Xingguang Song, and Xilin Chen. Ldf-net: Learning a displacement field network for face recognition across pose. In *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, pages 9–16. IEEE, 2017.

[Hu *et al.* 2018] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[Huang *et al.* 2007] Gary Huang, Mahdi Mattar, H. Lee, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[Huang *et al.* 2008] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

[Huang *et al.* 2014] Jiwen Huang, Weiyang Xu, Dapeng Chen, and Hai Li. Labeled faces in the wild: A survey. *arXiv preprint arXiv:1411.7766*, 2014.

[Huang *et al.* 2015] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. *arXiv preprint arXiv:1511.03034*, 2015.

[Kim *et al.* 2017] Ye-Hoon Kim, Bhargava Reddy, Sojung Yun, and Chanwon Seo. Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. In *ICML 2017 AutoML Workshop*, pages 1–8, 2017.

[Klein and Celik 2017] R. Klein and T. Celik. The Wits Intelligent Teaching System: Detecting student engagement during lectures using Convolutional Neural Networks. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2856–2860, Sep. 2017.

[Krizhevsky *et al.*] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks (alexnet) imagenet classification with deep convolutional neural networks (alexnet) imagenet classification with deep convolutional neural networks.

[Kulis and others 2013] Brian Kulis et al. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364, 2013.

[Kurakin *et al.* 2016] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[Kurakin *et al.* 2018] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

[Kvak 2022] Daniel Kvak. Visualizing coatnet predictions for aiding melanoma detection. *arXiv preprint arXiv:2205.10515*, 2022.

[Liu and Jin 2021] Jia Liu and Yaochu Jin. Multi-objective search of robust neural architectures against multiple types of adversarial attacks. *Neurocomputing*, 453:73–84, 2021.

[Liu *et al.* 2015] Jingtuo Liu, Yafeng Deng, Tao Bai, Zhengping Wei, and Chang Huang. Targeting ultimate accuracy: Face recognition via deep embedding. *arXiv preprint arXiv:1506.07310*, 2015.

[Ma *et al.* 2012] Bingpeng Ma, Yu Su, and Frédéric Jurie. Bicov: a novel image representation for person re-identification and face verification. In *British Machive Vision Conference*, pages 11–pages, 2012.

[Machiraju *et al.* 2021] Harshitha Machiraju, Oh-Hyeon Choung, Pascal Frossard, Michael Herzog, et al. Bio-inspired robustness: A review. *arXiv preprint arXiv:2103.09265*, 2021.

[Madry *et al.* 2017] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[Masi *et al.* 2018] Iacopo Masi, Yue Wu, Tal Hassner, and Prem Natarajan. Deep face recognition: A survey. In *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*, pages 471–478. IEEE, 2018.

[McCulloch and Pitts 1943] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[Mehdipour Ghazi and Kemal Ekenel 2016] Mostafa Mehdipour Ghazi and Hazim Kemal Ekenel. A comprehensive analysis of deep learning based representation for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 34–41, 2016.

[Meng and Chen 2017] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 135–147, 2017.

[Meng *et al.* 2021] Qiang Meng, Shichao Zhao, Zhida Huang, and Feng Zhou. Magface: A universal representation for face recognition and quality assessment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14225–14234, 2021.

[Miyato *et al.* 2018] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

[Nech and Kemelmacher-Shlizerman 2017] Aaron Nech and Ira Kemelmacher-Shlizerman. Level playing field for million scale face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7044–7053, 2017.

[Pan and Yang 2009] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2009.

[Parkhi *et al.* 2015] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In Xie Xianghua, Mark W Jones, and Gary K. L. Tam, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 41.1–41.12. BMVA Press, September 2015.

[Parkhi 2015] OM Parkhi. Vedaldi a zisserman a. *Deep face recognition BMVC*, 1(3):6, 2015.

[Phillips *et al.* 2000] P Jonathon Phillips, Hyeonjoon Moon, Syed A Rizvi, and Patrick J Rauss. The feret evaluation methodology for face-recognition algorithms. *IEEE Transactions on pattern analysis and machine intelligence*, 22(10):1090–1104, 2000.

[Raji and Fried 2021] Inioluwa Deborah Raji and Genevieve Fried. About face: A survey of facial recognition evaluation. *arXiv preprint arXiv:2102.00813*, 2021.

[Ranjan *et al.* 2017] Rajeev Ranjan, Carlos D Castillo, and Rama Chellappa. L2-constrained softmax loss for discriminative face verification. *arXiv preprint arXiv:1703.09507*, 2017.

[Ranjan *et al.* 2018] Rajeev Ranjan, Swami Sankaranarayanan, Ankan Bansal, Navaneeth Bodla, Jun-Cheng Chen, Vishal M Patel, Carlos D Castillo, and Rama Chellappa. Deep learning for understanding faces: Machines may be just as good, or better, than humans. *IEEE Signal Processing Magazine*, 35(1):66–83, 2018.

[Rathgeb *et al.* 2020] Christian Rathgeb, Pawel Drozdowski, and Christoph Busch. Makeup presentation attacks: Review and detection performance benchmark. *IEEE Access*, 8:224958–224973, 2020.

[Samaria and Harter 1994] Ferdinando S Samaria and Andy C Harter. Parameterisation of a stochastic model for human face identification. In *Proceedings of 1994 IEEE workshop on applications of computer vision*, pages 138–142. IEEE, 1994.

[Schroff *et al.* 2015] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[Seal *et al.* 2015] Ayan Seal, Suranjan Ganguly, Debotosh Bhattacharjee, Mita Nasipuri, and Consuelo Gonzalo-Martin. Feature selection using particle swarm optimization for thermal face recognition. In *Applied computation and security Systems*, pages 25–35. Springer, 2015.

[Shafahi *et al.* 2019] Alhussein Shafahi, Eric Wong, Alex Raff, J Z Kolter, and Tom Goldstein. Adversarial training for free! *arXiv preprint arXiv:1902.01557*, 2019.

[Simonyan and Zisserman 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2014.

[Stanley and Miikkulainen 2002] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[Sun *et al.* 2014] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation from predicting 10,000 classes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1891–1898, 2014.

[Szegedy *et al.* 2013] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[Tang and Wang 2002] Xiaoou Tang and Xiaogang Wang. Face photo recognition using sketch. In *Proceedings. International Conference on Image Processing*, volume 1, pages I–I. IEEE, 2002.

[Theagarajan and Bhanu 2020] Rajkumar Theagarajan and Bir Bhanu. Defending black box facial recognition classifiers against adversarial attacks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 812–813, 2020.

[Tong and Li 2017] Eric Tong and Jian Li. Triplet loss and online triplet mining in tensorflow. *arXiv preprint arXiv:1703.07737*, 2017.

[Touvron *et al.* 2020] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy: Fixefficientnet. *arXiv preprint arXiv:2003.08237*, 2020.

[Tramèr *et al.* 2017a] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[Tramèr *et al.* 2017b] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[Wang and Deng 2021] Mei Wang and Weihong Deng. Deep face recognition: A survey. *Neurocomputing*, 429:215–244, 2021.

[Wang and Liu 2021] Feng Wang and Huaping Liu. Understanding the behaviour of contrastive loss. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2495–2504, 2021.

[Wang *et al.* 2014] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1386–1393, 2014.

[Wang *et al.* 2016] Liwei Wang, Yin Li, and Svetlana Lazebnik. Learning deep structure-preserving image-text embeddings. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5005–5013, 2016.

[Weiss and Torgo 2016] Karl Weiss and Lu'ıs Torgo. A survey of transfer learning. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, pages 177–190. Springer, 2016.

[Yang *et al.* 2016] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5525–5533, 2016.

[Yi *et al.* 2014] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014.

[Yu *et al.* 2018] Baosheng Yu, Tongliang Liu, Mingming Gong, Changxing Ding, and Dacheng Tao. Correcting the triplet selection bias for triplet loss. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 71–87, 2018.

[Zhang *et al.* 2018] Huan Zhang, Yixuan Li, Duane Boning, and Cho-Jui Hsieh. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*, 2018.

[Zhang *et al.* 2020] Yaobin Zhang, Weihong Deng, Mei Wang, Jiani Hu, Xian Li, Dongyue Zhao, and Dongchao Wen. Global-local gcn: Large-scale label noise

cleansing for face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7731–7740, 2020.

[Zheng *et al.* 2018] Yutong Zheng, Dipan K Pal, and Marios Savvides. Ring loss: Convex feature normalization for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5089–5097, 2018.

[Zhong *et al.* 2021] Yaoyao Zhong, Weihong Deng, Jiani Hu, Dongyue Zhao, Xian Li, and Dongchao Wen. Sface: Sigmoid-constrained hypersphere loss for robust face recognition. *IEEE Transactions on Image Processing*, 30:2587–2598, 2021.

[Zuts *et al.* 2018] Andriy Zuts, Ilya Loshchilov, Francis Bach, and Yann LeCun. Adversarial regularization for deep learning. In *International Conference on Learning Representations*, 2018.