

**Big Data Content Analytics**

**Face Recognition**

Gkliati Argyro P2821808

Konsolaki Eleni P2821815



September, 2019

**Contents**

[**Description** 3](#_Toc18774873)

[**Mission** 3](#_Toc18774874)

[**Data** 3](#_Toc18774875)

[**Methodology** 5](#_Toc18774876)

[**Results** 9](#_Toc18774877)

[**Members/Roles** 10](#_Toc18774878)

[**Bibliography** 10](#_Toc18774879)

[**Time Plan** 10](#_Toc18774880)

[**Contact Person** 10](#_Toc18774881)

[**Comments** 11](#_Toc18774882)

# **Description**

The aim of this project is to implement deep learning techniques in order to build a face recognition model. Facial recognition is a way of recognizing a human face through technology. A facial recognition system uses biometrics to map facial features from a photograph or video. It compares the information with a database of known faces to find a match. Facial recognition can help verify personal identity, but it also raises privacy issues.

First, we will train the algorithm. We will not use a standard CNN (Convolutional Neural Network) but a Siamese Neural Network, which needs only a handful of images from each person. To do so, we need to use a dataset with the facial images of the people we want to recognize. We need to also set an ID (it may be a number or the name of the person) for each image, so the algorithm will use this information to recognize an input image and give you an output. Images of the same person must have the same ID.

# **Mission**

Face recognition, while initially a form of computer application, it is now typically used as access control in security systems, such as in airports. During the past few years and especially after 9/11, significant progress has been made in developing and using face recognition systems in airports as US, Canadian and British Governments sponsored evaluations and challenge problems to ameliorate the error rate and thus improve systems’ performance.

Our target is to use this data- images- to implement face recognition as part of an airport’s check in process. Our model will be trained to identify similarity of the images taken at an airport gate (check in phase) with an existing database of images captured at an earlier point. This database could be images from sources like passport or another document.

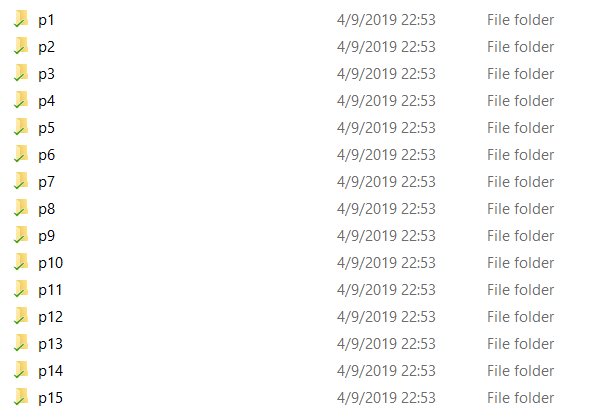
So, next time you go to the airport you might experience something different as part of the security process: A machine scanning your face to recognize you and eventually verify you.

# 

# **Data**

Selecting an appropriate dataset is the most important aspect. The dataset needs to contain enough valid labeled images in each class to allow the neural network to learn every label. There is no magic threshold for number of images per class. Deep Convolutional Neural Networks require a lot of labelled data, which is not feasible sometimes. On the other hand, Οne Shot Classification aims to solve this problem, as it requires only one training example for each class (only one image per person in our case). The Labeled Faces in the Wild is a database of face photographs designed for studying the problem of unconstrained face recognition. The data set contains more than 13,000 images of faces collected from the web. Specifically, there are 5749 folders with 13233 images. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set. Due to the Data Privacy Policy we replaced the names of the folders with P(1) to P (5749) in order to be anonymous, since they are not necessary for the purpose of this project. You can browse the initial dataset from here: <http://vis-www.cs.umass.edu/lfw/#deepfunnel-anchor>.

Unfortunately, due to the computing power, we were forced to train our model with only 100 images. You may see below a sample of the trained images.









# **Methodology**

In order to train the model, we need to split the dataset to train and test. The first one is used to train the model and the second to provide an unbiased evaluation of a final model fit on the training dataset.

**Why Siamese Model?**

If we want to build our face recognition model using a Convolutional Neural Network (CNN) from scratch, then we need many images of different people for training the network and attaining good accuracy. But apparently, such a process would be time consuming and of high cost. Normally, it is quite difficult to have many images for many different people. Thus, it is not feasible to build a model using a CNN or any deep learning algorithm, unless we have sufficient data points. So, in these kinds of scenarios, we can resort to a sophisticated one-shot learning algorithm such as a Siamese network, which can learn from fewer data points.

One-shot learning is a technique where we learn from only one training example per class.

**How does Siamese networks work?**

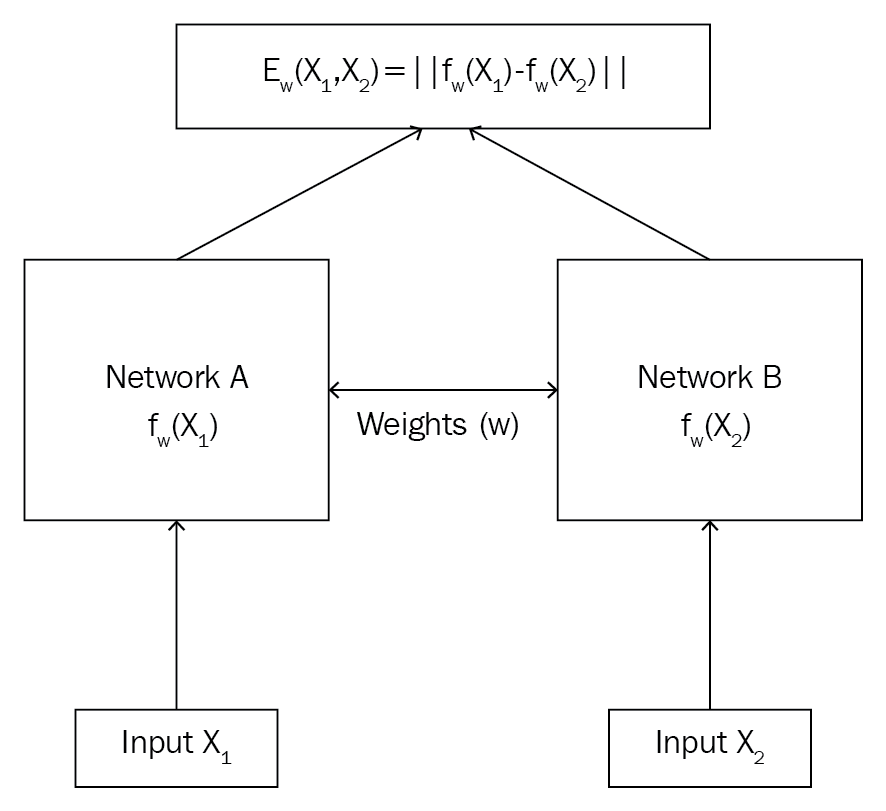
A close up of a logo

Description automatically generated

But how do siamese networks work? Siamese networks basically consist of two symmetrical neural networks both sharing the same weights and architecture and both joined together at the end using some energy function, E. The objective of our siamese network is to learn whether two input values are similar or dissimilar. Let’s say we have two images, X1 and X2, and we want to learn whether the two images are similar or dissimilar.

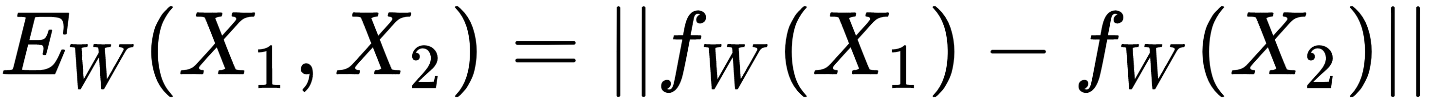
Siamese networks are not only used for face recognition, but they are also used extensively in applications where we don’t have many data points and tasks where we need to learn similarity between two inputs. The applications of siamese networks include signature verification, similar question retrieval, object tracking, and more. We will study siamese networks in detail in the upcoming section.

**Architecture of Siamese networks**

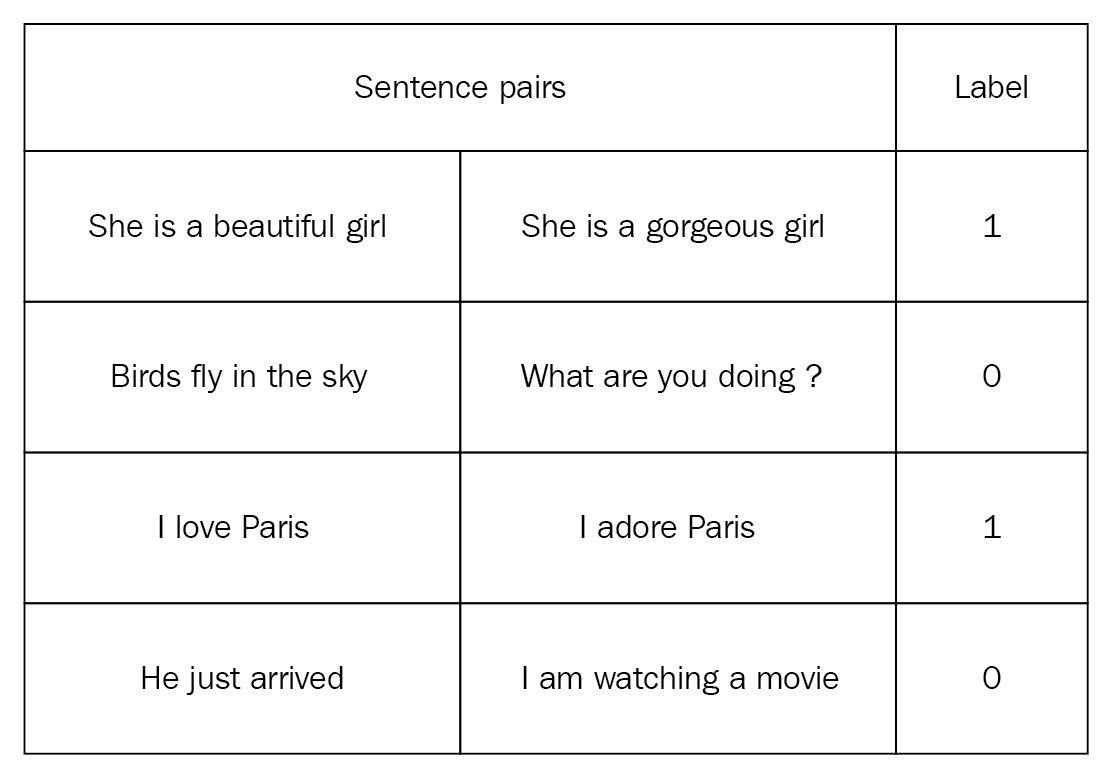


As you can see in the preceding diagram, a Siamese network consists of two identical networks both sharing the same weights and architecture. Let’s say we have two inputs, X1 and X2. We feed our input X1 to Network A, that is, fw(X1), and we feed our input X2 to Network B, that is, fw(X2). As you will notice, both of these networks have the same weights, w, and they will generate embeddings for our input, X1 and X2. Then, we feed these embeddings to the energy function, E, which will give us similarity between the two inputs.

It can be expressed as follows:



The input to the siamese networks should be in pairs, (X1, X2), along with their binary label, Y ∈ (0, 1), stating whether the input pairs are a genuine pair (same) or an imposite pair (different). As you can see in the following table, we have sentences as pairs and the label implies whether the sentence pairs are genuine (1) or imposite (0):



a Siamese network learns by finding similarity between two input values using identical architecture. It is one of the most commonly used few-shot learning algorithms among tasks that involve computing similarity between two entities. It is powerful and robust and serves as a solution for a low data problem.

**The process of building the Model**

In order to build our model, we used Tensorflow and Keras which operates over Tensorflow. Keras offers the option to use the Sequential structure to build the model. By using the Sequential form, the layers are getting stacked linearly and in order (from input to output). While constructing the model, we should specify the input shape.

We compiled the model using:

1. Loss function (the objective that the model tries to minimize each time): We used the binary cross entropy as our siamese network is tasked with binary classification.

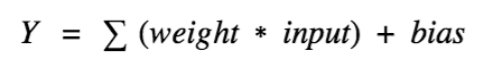
2. Optimizer: we used Adam optimizer, an adaptive learning rate algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.

3. Metrics (the function used to judge the performance of our model): Since, it is a binary classification problem we used binary accuracy.

Convolution layers: The convolution layer is the main building block of a convolutional neural network. It comprises of a set of independent filters. Each filter is independently convolved with the image and we end up with n (n represents the number of filters).

Pooling layer: Its function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network. Pooling layer operates on each feature map independently. The most common approach used in pooling is max pooling. The FC is the fully connected layer of neurons at the end of CNN. Neurons in a fully connected layer have full connections to all activations in the previous layer.

Activation function: An artificial neuron calculates a “weighted sum” of its input, adds a bias and then decides whether it should be “fired” or not. So, consider a neuron.



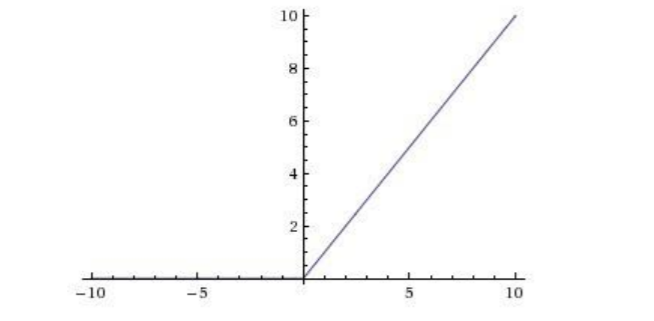
Now, the value of Y can be anything ranging from -inf to +inf. The neuron really doesn’t know the bounds of the value.

We decided to add “activation functions” in order to check the Y value produced by a neuron and decide whether outside connections should consider this neuron as “activated” or not.

ReLu

In our case the activation function that we used is the ReLu function, A(x) = max(0,x)

The ReLu function gives an output x if x is positive and 0 otherwise.



First, ReLu is nonlinear, is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. That is a good point to consider when we are designing deep neural nets.

Our model consists of 10 layers. Below, all these layers are analyzed:

Prior to our layers we reshape the input data to feed into the 2D convolutional layers.

➢ First Layer (Convolutional)

The first layer is convolutional, and its input are images of (128x128), the convolution is 32 filters (4x4) using ReLU activation function and the output are 32 tensors (128x128).

➢ Second Layer (Pooling)

The second layer is pooling where the input is 32 tensors (128x128) from the previous layer and the output are 32 tensors again but with (2x2) sizing. That is because pooling layer is used to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network, and hence to also control overfitting.

➢ Third Layer (Dropout)

In this layer the input is 32 tensors with rate=0.4. The rate was used to prevent overfitting. So, in this layer we drop out 40% of input units.

➢ Fourth Layer (Convolutional)

The fourth layer used is again convolutional, with input 32 tensors (2x2) from the previous layer. The convolution is using 64 (3x3) filters and ReLu activation function. The output is 64 tensors(2x2).

➢ Fifth Layer (Pooling)

In the fifth layer we used our last pooling layer with input 64 tensors using 1 pooling filter and the output is 64 tensors (2x2).

➢ Sixth Layer (Dropout)

In this layer the input is 64 tensors with rate=0.3. In this layer we drop out 30% of input units.

➢ Seventh Layer (Flatten)

In this layer Flatten was used. Flatten is our regularizer.

➢ Eighth Layer (Dense)

In this step we used dense function with activation function ReLu. This step returns the output tensor with the same shape as inputs except the last dimension is of size units (units=128).

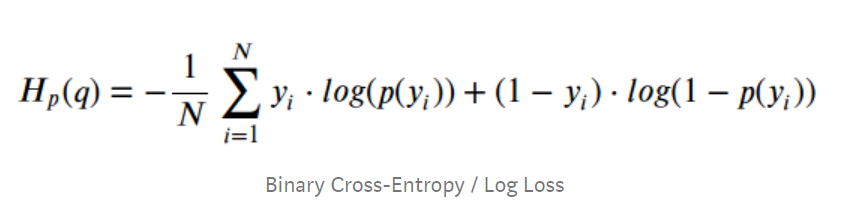
➢ Ninth Layer (Dropout)

In this layer the input is 64 tensors with rate=0.5. In this layer we drop out 50% of input units.

➢ Tenth Layer (Dense)

In our last layer we used dense again but with Sigmoid and unit=64. Sigmoid function was used as the similarity score is squished between 0 and 1.

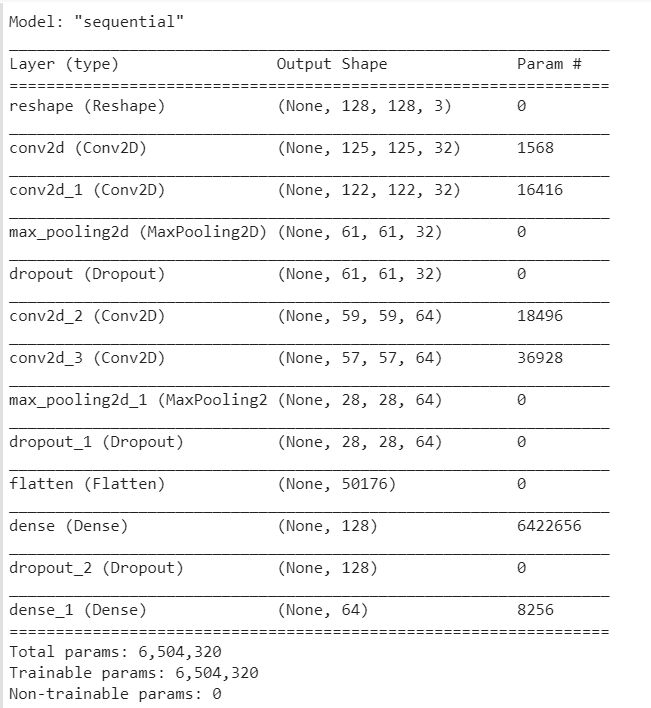
Finally, we compile the model using Adam optimizer and loss function binary crossentropy. The equation for categorical cross entropy is



where y is the label (1 and 0) and p(y) is the predicted probability of the point being 1 for all N points.

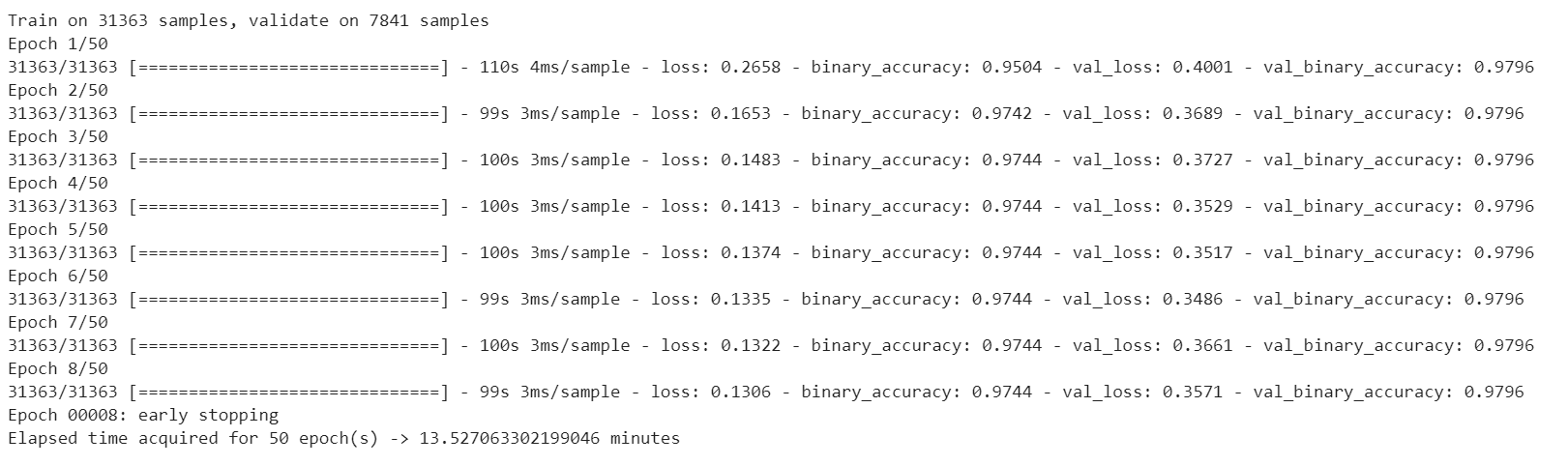
**Parameters**

The parameters of our Siamese network model are 6,504,320 with no nontrainable. Our hyper parameters used for choice activation were the function ReLu and Sigmoid. The batch size was 256, the training samples were 100 and the epochs were 50.



**Summary of the Model**

At the extract below, we can see the loss and accuracy for each epoch. In the first few epochs we can easily track the impressive improvements that slow as epochs go by.



# **Members/Roles**

|  |  |  |
| --- | --- | --- |
| **Member** | **Background** | **Role** |
| Gkliati Argyro | Business Economics & Finance | Responsible for the business case, the dataset,  the implementation of the code and the methodology |
| Konsolaki Eleni | Business Economics & Finance | Responsible for the business case, the dataset,  the implementation of the code and the methodology |

# **Bibliography**

1. <https://medium.com/swlh/advance-ai-face-recognition-using-siamese-networks-219ee1a85cd5>
2. <https://towardsdatascience.com/one-shot-learning-with-siamese-networks-using-keras-17f34e75bb3d>
3. <https://www.humanrecognitionsystems.com/product/mflow/>
4. <https://en.wikipedia.org/wiki/SmartGate>

# **Time Plan**

* Searching for the appropriate dataset (4 days)
* Studying tutorials regarding Neural Networks and specifically Siamese Neural Networks (1 week)
* Methodology & Code (3 weeks)
* Report & Presentation (1 week)

# **Contact Person**

* Gkliati Argyro [agklia@aueb.gr](mailto:agklia@aueb.gr)
* Konsolaki Eleni elenikonsolaki@gmail.com

# **Comments**