

## INTERIM REPORT

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

---

# Deep Learning For Facial Expression Analysis

---

*Author:*

Tom Bartissol (CID: 00824562)

Date: June 14, 2017

# Contents

<b>1</b>	<b>Introduction</b>	<b>4</b>
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	Facial Expressions . . . . .	5
2.1.1	Facial Action Coding System (FACS) . . . . .	5
2.1.2	Interpreting emotions . . . . .	6
2.2	Facial Expression Analysis . . . . .	7
2.3	Databases . . . . .	8
2.4	Deep Learning . . . . .	8
2.4.1	Deep Convolutional Neural Networks . . . . .	8
2.4.2	Optimisers . . . . .	12
2.4.3	Activation functions . . . . .	13
2.4.4	Models . . . . .	13
<b>3</b>	<b>EmotionNet Database</b>	<b>13</b>
3.1	Downloading . . . . .	13
3.1.1	Reading the xlsx file . . . . .	14
3.1.2	Downloading the image . . . . .	14
3.1.3	Storing the image . . . . .	15
3.2	Converting to TFRecords . . . . .	15
3.3	Extending TF-Slim datasets . . . . .	16
3.4	Annotating Valence and Arousal . . . . .	16
<b>4</b>	<b>Action Unit Classification</b>	<b>17</b>

---

4.1	VGG 16 . . . . .	17
4.2	ResNet . . . . .	17
<b>5</b>	<b>Valence and Arousal Regression</b>	<b>17</b>
5.1	VGG 16 . . . . .	17
5.2	ResNet . . . . .	17

# 1 Introduction

The ability to analyse human facial expressions is an active area of research with exciting near-future real-world applications ranging from law enforcement to advertising (measuring how positively or negatively people respond to an ad). It would also be at the core of any system capable of intelligent Human-Computer Interaction (HCI). Indeed, for such interactions to be life-like, the Computer should be able to recognise human emotions which are expressed through multiple channels, an important one being facial expressions. *Interpreting* the recognised facial expressions into the six *basic emotions* [2] (see Table 2) would allow the Computer to identify human emotions to some extent.

Facial Expressions result from the contraction of a facial muscle or a group of facial muscles and the visually perceptible changes due to such contractions have been codified by Ekman and Friesen into the well-known Facial Action Coding System (FACS) [3] which provides labels, facial *Action Units* (AUs), for the actions of a muscle or group of muscles. For instance, AU 12 corresponds the lip corner puller and if it is active at the same time as AU 6, which corresponds to the cheek raiser, then the subject is smiling and this could be interpreted as happiness.

We are therefore concerned with two problems (i) identifying these AUs and (ii) estimating continuous emotion dimensions: valence (how positive or negative the emotion is) and arousal (how intense) instead of discrete emotions such as the six basic emotions. Since changes in muscular activity are not instant but last between 250ms and 5s [4], we are interested in data with a temporal dimension, that is videos. More specifically, whereas the majority of previous work has been conducted on data sets captured in constrained environment and/or using acted or posed facial expressions [10], we are interested in the so called "*in-the-wild*" videos, which are recorded in different lighting conditions with different poses and most accurately represent spontaneous facial expressions.

To achieve this, we will use end-to-end deep learning models, namely ResNet, VGG and ResNet-Inception. Each model will be trained/fine tuned to jointly solve problems (i) and (ii) on data taken in-the-wild. Finally we will evaluate these models on existing and established benchmarks and databases (not necessarily in-the-wild).

## 2 Background

### 2.1 Facial Expressions

#### 2.1.1 Facial Action Coding System (FACS)

The human head having a finite number of muscles, the number of visually perceptible changes that can be caused by the contraction or relaxation of one or more of these muscles, that is the number of facial expressions, is finite. We can therefore taxonomise these facial changes into a coding system. Although there exists a few such coding systems [8], the most popular and used coding system is the FACS [3]. FACS breaks down each visually perceptible change into facial *Action Units* (AUs) which roughly corresponds to the contraction or relaxation of individual facial muscles. The activation of one or more AU creates a facial expression. Some examples of AUs and their associated facial are listed in Figure 1.

On top of this taxonomy, FACS also provides an intensity score, A-E (maximum-minimum), to rate how pronounced each AU is in a facial expression (see Table 1). For instance AU 12A would indicate that the lip corners are slightly pulled whereas AU 12E would indicate that they are maximally pulled.

Finally, AUs do not activate instantly. Indeed, an AU is firstly in a neutral state, then, when it starts activating, muscles contract and the AU is said to be in an *onset* phase, once the muscles are contracted and the AU is at its peak, it is said to be in an *apex* phase, finally, when the muscles start to relax and the AU starts to disappear, the AU is said to be in an *offset* phase before returning to normal. So the activation of an AU generally follows the following pattern: neutral - onset - apex - offset - neutral.

Intensity	description
A	Trace
B	Slight
C	Marked or Pronounced
D	Severe or Extreme
E	Maximal

**Table 1:** The scale for measuring the intensity with which an AU is activated












Upper Face Action Units					
AU 1	AU 2	AU 4	AU 5	AU 6	AU 7
					
Inner Brow Raiser	Outer Brow Raiser	Brow Lowerer	Upper Lid Raiser	Cheek Raiser	Lid Tightener
*AU 41	*AU 42	*AU 43	AU 44	AU 45	AU 46
					
Lid Droop	Slit	Eyes Closed	Squint	Blink	Wink
Lower Face Action Units					
AU 9	AU 10	AU 11	AU 12	AU 13	AU 14
					
Nose Wrinkler	Upper Lip Raiser	Nasolabial Deepener	Lip Corner Puller	Cheek Puffer	Dimpler
AU 15	AU 16	AU 17	AU 18	AU 20	AU 22
					
Lip Corner Depressor	Lower Lip Depressor	Chin Raiser	Lip Puckerer	Lip Stretcher	Lip Funneler
AU 23	AU 24	*AU 25	*AU 26	*AU 27	AU 28
					
Lip Tightener	Lip Pressor	Lips Part	Jaw Drop	Mouth Stretch	Lip Suck

Figure 1: Facial Action Units

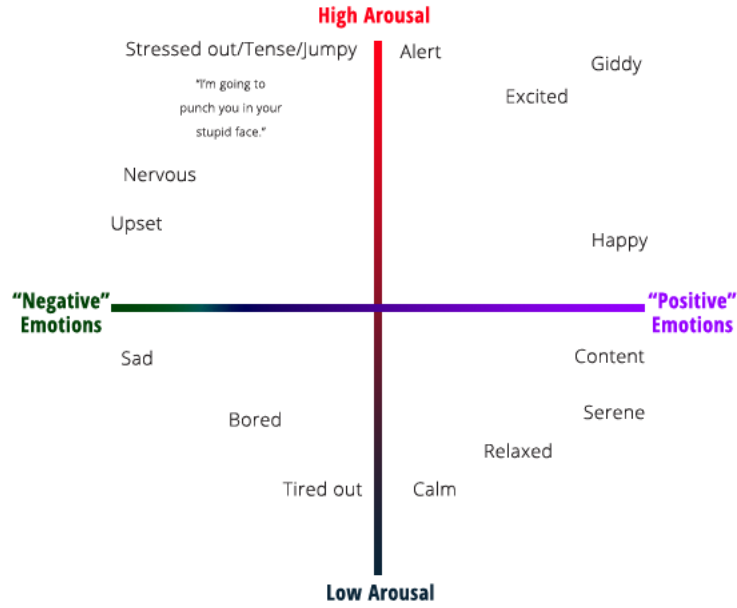
### 2.1.2 Interpreting emotions

**Discrete case** Emotions can roughly be broken down into six *basic emotions* [2], seven if we count the neutral emotion. We can then interpret a facial expression into one of these seven categories. Note that this is only an interpretation as emotions are expressed through multiple channels such as body language or voice and facial expressions are only one of these channels.

**Continuous case** This is the case in which we are interested. Emotions can be represented using two continuous dimensions:

- **Valence:** characterises the attractiveness or aversiveness of an emotion, that is respectfully how positive or negative the emotion is.
- **Arousal:** characterises the intensity of the emotion

We can therefore represent emotions in a two dimensional coordinate system using their valence and arousal values, see Figure 2



**Figure 2:** Plotting emotions according to their valence and arousal

Emotion	AUs
Neutral	0
Happiness	6, 12
Sadness	1, 4, 14
Surprise	1, 2, 5B, 26
Fear	1, 2, 4, 5, 7, 20, 26
Anger	4, 5, 7, 23
Disgust	9, 15, 16

**Table 2:** The 6 basic emotions (plus neutral) and the corresponding Action Units that are generally activate when the emotion is present

## 2.2 Facial Expression Analysis

In the early days, facial expression analysis was restricted to recognising the six basic emotions. Furthermore, computer vision techniques were used to extract features from input images and then classify them instead of using the *end-to-end* deep learning techniques we will use in this project. As indicated in [10], the interested reader is directed towards [6, 7] for a thorough overview of these early techniques.

## 2.3 Databases

We direct the reader towards [10] for a complete survey of the main databases available for facial expression analysis. We are interested in databases containing images or videos taken in-the-wild and annotated with facial action units. A such, we will use the EmotioNet [1] database which contains 1,000,000 in-the-wild images of emotions. These images were downloaded from the Internet by using specific combinations of keywords and were then automatically annotated with AUs and AU intensities as well as emotion categories by using a novel computer vision algorithm presented in [1]. We also want to annotate valence and arousal, so we will be using the newly available Aff-Wild database [10] which contains video taken in-the-wild (from YouTube) that have been manually annotated with valence and arousal.

For evaluation purposes, we will be using two databases:

- **DISFA** [5]: videos of young adults watching video clips intended to elicit certain emotions. Although these videos are not in-the-wild since they were all recorded with the same stereo camera in similar lighting conditions, each frame has been manually annotated with AUs and their respective intensities.
- **AMFED** [6]: videos of people reacting to 3 chosen amusing Super Bowl commercials. The people were recorded using their laptop's web camera so the lighting conditions differ from one video to another. Furthermore, each frame was carefully annotated with a subset of facial action units.

## 2.4 Deep Learning

We use deep learning models in order to, given an image of a facial expression, recognise which AUs are active and which aren't as well as estimate the valence and arousal of the given facial expression. We use these models as they have proved to be able to learn to represent abstract data features (e.g a smile) thanks to multiple computational layers

### 2.4.1 Deep Convolutional Neural Networks

In particular, since our task focuses on images (i.e. 2/3-dimensional data), we will be using a particular type of deep learning models called deep convolutional neural networks (DCNNs). DCNNs consist of a succession of convolutional and pooling layers followed by fully connected layers.



The first block of layers, the convolutional and pooling layers, act as a feature extractor. They take as input a raster image  $\in \mathcal{R}^{n \times m}$  where  $n$  is the image height and  $m$  the image width, either in greyscale ( $\in \mathcal{R}^{1 \times n \times m}$ ) or in RGB colours ( $\in \mathcal{R}^{3 \times n \times m}$ ) (where 3 is the depth of the image, i.e. the number of channels) and processes it to extract features which will constitute a meaningful representation of the image in a lower dimension, i.e. a vector.

**Convolutional Layers** Convolutional layers are the core components of DCNNs. They are where the most heavy computations take place and act as feature extractors by applying a set of spatially small learnable kernel filters to the input image. These filters could for instance have a size of  $7 \times 7 \times D$  where the first and the second 7 correspond to the height and the width of the filter respectively and  $D$  corresponds to the depth (either 1 or 3 if we are working with RGB input images).

Each filter is applied by “sliding” it over the input image and performing a convolution at every pixel of the image between the filter and the portion of the image that lies directly beneath it. Mathematically, this gives the following: suppose we have an input image  $I \in \mathcal{R}^{W \times H}$  and a filter  $F \in \mathcal{R}^{k \times k}$  such that  $I(a, b)$  denotes the pixel on row  $a$  and column  $b$  of  $I$  (assume the same notation for  $F$ ). Then the convolution operation  $C(.,.)$  executed at pixel  $(a, b)$  of the image is

$$C(a, b) = \sum_{i=1}^k \sum_{j=1}^k F(i, j) \times I(a - i, b - j) \quad (1)$$

small

Therefore, as we slide a filter over the image, we produce a 2D response map that corresponds to the convolved features at each spatial location of the input image, see Fig. 3. Intuitively, each kernel filter learns to detect specific features in the input such as a corner or a blob of colour

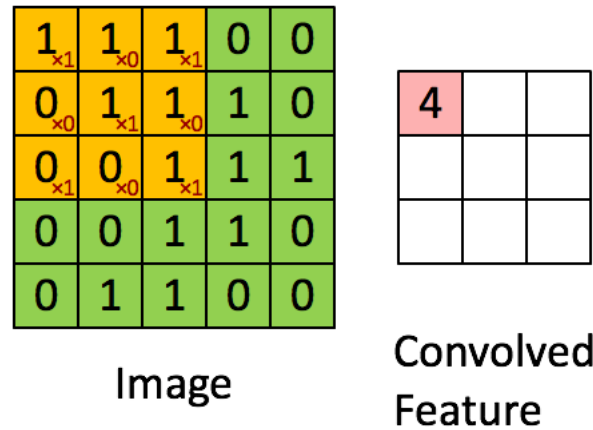
In the case of a 3 dimensional input volume ( $Width \times Height \times Depth$ ), each filter will also be 3 dimensional with the depth dimension  $D$  matching that of the input. Therefore each filter will produce  $D$  response maps which are summed together with a bias term to produce a 2 dimensional overall response map for that filter.

Finally, when using a  $k \times k$  kernel on an  $n \times m$  image, the response map will have a smaller size of  $(n - l) \times (m - l)$  where  $l = \frac{k-1}{2}$  ( $k$  is usually an odd number). While we do want to perform dimensionality reduction before going through the fully connected layers, this means that the kernel might omit some important fea-

tures that are at the edges of the image. To solve this problem, it is common practice to add  $l$  layers of zero-padding on each side of the image so that the kernel also covers the edges of the image and the response map has the same dimension as the input.

To sum up, here are the inputs and the outputs of a typical convolutional layer with  $N$  kernel features of size  $k \times k$  applied using a stride of  $s$  on an image with a padding of  $p$ :

1. **In:** an image with size  $W \times H \times D$  where  $W$  is the width of the image,  $H$  the height and  $D$  the depth
2. **Out:** a volume with size  $W_{out} \times H_{out} \times N$  where  $W_{out} = \frac{W-k+2p}{s}$ ,  $H_{out} = \frac{H-k+2p}{s}$  and  $N$  is the number of filters used at that layer



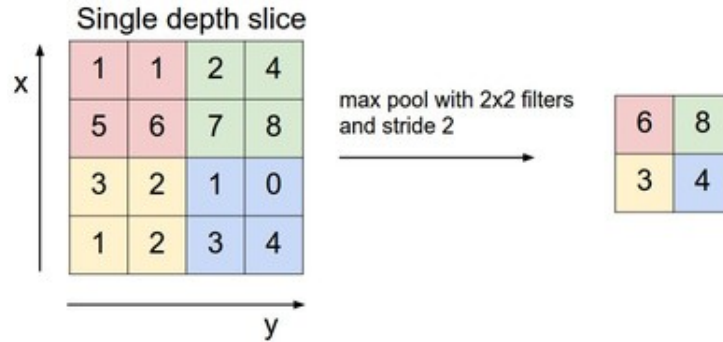
**Figure 3:** Example of a convolution operation at a single spatial location. The orange square represents the kernel filter. Note that no padding has been applied here which is why the convolved feature (i.e. the response map) is smaller.

Source: [http://deeplearning.stanford.edu/wiki/images/6/6c/Convolution\\_schematic.gif](http://deeplearning.stanford.edu/wiki/images/6/6c/Convolution_schematic.gif)

**Pooling Layers** As mentioned in the above paragraph, convolutional layers can perform dimensionality reduction if no zero-padding is added to the input image. However this job generally falls to pooling layers.

As their name suggests, pooling layers group pixels together to reduce them to a single pixel. There are several ways to determine the value of the output pixel and the most common one is maximum pooling which consists in outputting the pixel in the group that has the highest value. Since this is just a static operation,

there are no learnable parameters involved. The only parameters are the spacial extent  $p$  (i.e. the width and height) of the pooling region/filter and the stride  $s$  which defines how many pixels to skip until applying the next pooling operation. For instance, a pooling layer with  $p = 2$  (2x2 filters) and  $s = 2$  will divide the width and height of the input image by 2 (it does not change the depth) as can be seen in Fig 4.



**Figure 4:** example of a single max pooling operation with a  $2 \times 2$  pooling filter and a stride of 2

Source: <http://cs231n.github.io/assets/cnn/maxpool.jpeg>

To sum up here are the inputs and outputs of a pooling layer:

1. **In:**  $k$  images of size  $n \times m \times d$  where  $n$  is the width,  $m$  the height and  $d$  the depth of the image
2. **Out:**  $k$  responses of size  $\left(\frac{n-p}{s} + 1\right) \times \left(\frac{m-p}{s} + 1\right) \times d$  where  $p$  is the width of the square pooling filter and  $s$  the stride size

**Fully Connected Layers** Fully connected layers constitute the second block of the

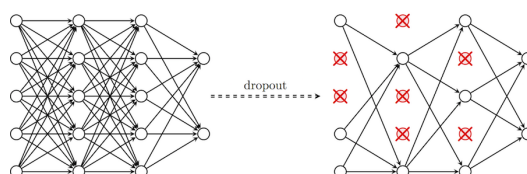
Unlike convolutional layers, which are locally connected (each pixel in the response map is locally connected to a sub-region of the input), fully connected layers have their outputs connected to *every* input.

As such, the activations can be simply calculated via matrix multiplication between the inputs and their associated weights and adding a bias term before passing all that to an activation function (e.g. ReLu, Sigmoid, Softmax).

It is worth noting that we can easily replace a fully connected layer by an equivalent convolutional layer; all that must be done is to match the size of the kernel filters to the size of the input image or vector. For instance, in the VGG 16[CITE] network, the final convolutional layer outputs 512 feature maps of size

$7 \times 7$  which are passed on to a fully connected layer with 4096 neurons. Then we can replace this fully connected layer by a convolutional one with 4096 filters of size  $7 \times 7$

**Dropout Layers** Dropout layers[CITE] are a simple yet powerful regularisation technique to prevent, overfitting. The key idea is to randomly block some neurons from connecting to the next layer during training see Fig. 5. This prevents groups of neurons from co-adapting too much and improves the flexibility of the network which decreases overfitting.



**Figure 5:** example dropout layers: the crossed neurons are blocked from sending their inputs to the next layer

Source: <https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/figures/drop.png>

### 2.4.2 Optimisers

#### Stochastic Gradient Descent

#### SGD with Nesterov Momentum

#### Adagrad

#### RMSProp

#### Adam

#### 2.4.3 Activation functions

#### 2.4.4 Models

The reader is referred to [10] (paragraph 3) for a review of existing deep learning methodologies for facial expression recognition "in-the-wild".

We use two pre-existing and pre-trained models, namely **VGG** [9] and **ResNet** [CITE]. These were fine-tuned for specific tasks on our database (a subset of the EmotioNet database[CITE])

**VGG 16** The VGG 16 network was developed by Karen Simonyan and Andrew Zisserman from Oxford University's Visual Geometry Group and was the runner-up in ILSVRC 2014. The "16" signifies that there are 16 trainable (weight) layers: 13 convolutional layers and 3 fully connected layers.

It's originality comes from the fact that it uses small kernel filters ( $3 \times 3$ ) at each convolutional layer which are convolved with every pixel of the input image (i.e. they use a stride of 1). Furthermore, they increase the number of filters as we progress through the convolutional layers. The idea behind this is to get the first layers to detect abstract features such as lines and get the following layers to refine on that feature.

**ResNet**

## 3 EmotioNet Database

### 3.1 Downloading

In order to use less space, the authors of the EmotioNet database do not offer a direct download of the around 25k images. Instead, they provide an xlsx file containing one line per image which consists of 61 columns: the first column corresponds to the URL of the image and the next 60 columns correspond to the 60 AUs in ascending order. To download the database, one must therefore read this file line by line, get the image at the specified URL and store it in a file.

### 3.1.1 Reading the xlsx file

Python does not natively support the reading of xlsx files, and even though third party packages such as **openpyxl** by Eric Gazoni and Charlie Clark exist, we chose to convert the xlsx file containing the URLs and labels to a CSV file as these are easily readable in Python. One problem arose using this method, some URLs (less than 15) contained commas, meaning that the URL was split up into two or more columns which shifted the index of the AU values, thus giving an invalid label to the associated image. To remedy to this situation, we simply deleted all the commas from the URLs.

### 3.1.2 Downloading the image

Once we have retrieved the URL, we can get the raw image over the Internet by using the **urllib** library. This did not come without some complications as 2,760 images were not downloaded due to the errors listed below:

- HTTP 404 (Not Found) error: 1,726 URLs
- HTTP 400, 401, 403, 410 errors: 205 URLs
- HTTP 500, 502, 503, 504 errors: 102 URLs
- HTTP 301, 302 errors: 20 URLs
- Name or service not known: 500 URLs
- Connection timed out: 109 URLs
- Connection reset by peer: 17 URLs
- Connection refused: 16 URLs
- No route to host: 10 URLs
- Certificate verify failed: 14 URLs

The *Name or service not known* errors can be caused by the lack of an internet connection which can occur when downloading via Wi-Fi. Because of this, some valid URLs will not be accessed to download an image. To solve this problem, we keep track of which images remain to be downloaded by taking all the URLs and removing those that were already downloaded as well as those that led to an error and launch the download script again. To keep track of the erroneous URLs, each

time an URL leads to an error, it is stored along with its error in a text file and to construct the list of already downloaded images, we just list the files present in the specified download directory.

### 3.1.3 Storing the image

Storing the fetched raw image implies creating a file and more importantly naming it in such a way that using this name, we can retrieve the associated label (AU values) from the `xlsx` file. We initially used an URL parser to extract the query component of the URL which should consist of the name of the image (e.g. `image.jpeg`). However, it turns out that this query component can also have a parameter component (e.g. `”?size=200x200”` in `HTTP://www.foo.com/bar/image.jpeg?size=200x200`). So converting the URL into a file name based solely on the query component of the URL meant that two different URLs/images could lead to the exact same file name when we require this `url_to_filename` function to be a bijection in order to be able to retrieve the associated label from the `xlsx` file.

We therefore abandoned this method and just used the URL with a couple of preprocessing steps as file name. The preprocessing steps consisted in removing full stops from the URL and replacing forward slashes with underscores as they would otherwise be understood as directories by the file system. Additionally, we determine the type of the ray image (e.g. `png`, `jpeg`, `gif` ...) and append it to the file name. Finally, file names cannot be longer than 250 chars on most systems so we truncate the beginning of URLs that violate this limit.

## 3.2 Converting to TFRecords

TFRecords is the standard recommended file format to store data that will be used by a TensorFlow network or Graph. It is based on Google’s Protocol Buffers which are language and platform agnostic object serialisation mechanisms.

Converting a data set to a set of TFRecord files is no easy task, but thankfully TensorFlow provides a script, `build_image_data.py`, that does just that and only needed a few modifications to adapt it to the nature of our data set. Indeed, this script provided by TensorFlow was used to convert the images of the CIFAR-10[CITE] database and relied on a special organisation of the data to assign labels: one eponymous subdirectory should be created for each image category (label) and all the images of that category should be placed into that subdirectory (e.g. and image of a cat would be in the `/dataset/cat/` subdirectory). However this mechanism

does not apply in our case since it assumes that the categories are mutually exclusive (one image cannot be categorised as a cat and a dog at the same time) to partition the data set into subdirectories, whereas in our case, an image can have multiple active AUs at once.

So we had to change the way the script assigned labels to images by constructing a dictionary of URLs to labels using the `xlsx` data file and then using the file name of the image to look up its label in this dictionary. We also had to change the way the label was stored in the `TFRecord` as it now consist of an array of `int64` rather than a single `int64`. Finally, the script converts andy PNG image to JPEG for consistency, however, it does not handle other image types such as GIF so we added the necessary logic to convert GIFs to JPEG images.

The script then takes care of multi-threading for efficiency and outputs a training and a validation `TFRecord` file, both split into a number of user-specified shards.

### 3.3 Extending TF-Slim datasets

Having downloaded the database and converted it to `TFRecord` format, we now extend the TF-Slim datasets to incorporate ours. This has many advantages as TF-Slim then takes care of creating a data provider for our models and more importantly, it allows us to easily reuse TF-Slim’s generic training and evaluation scripts.

In order to do this, we created a new file in `/slim/datasets/` called `images.py` which returns an instance of a TF-Slim Dataset class, customised for our dataset. We then add this instance to the `dataset_factory` file’s dictionary. Our dataset can now be accessed using the `dataset_factory.get_dataset()` method which only requires the name of the dataset, the path to the actual data and the name of the split to use (one of ‘train’ or ‘validation’). This means that we can easily change data sets, should we add some in the future and makes our training and evaluating system more modular.

### 3.4 Annotating Valence and Arousal

As mentioned in section [CITE], this database was annotated by human annoators on Action Unit presence/absence only. However, we are also intereseted in valence and arousal values as it is believed that the presence or absence of AUs is correlated to the valence and arousal of the emotion that can be interpreted from a facial



expression.

As such, one of the major tasks, was to first annotate the 21,500 images that were downloaded with both valence and arousal. This was done using a tool available for research purpose only and developed by J. Kossaifi, G. Tzimiropoulos, S. Todorovic and M. Pantic. This tool was designed to annotate videos, not large image data sets, so we first divided the 21,500 images into 21 videos of 1000 frames and a 22nd video of 500 frames. You then load up a video and start annotating the frames for valence. Once all the frames of the video are annotated with valence, we annotate them with arousal and move on to the next video.

## 4 Action Unit Classification

This task was performed on the full data set

### 4.1 VGG 16

### 4.2 ResNet

## 5 Valence and Arousal Regression

### 5.1 VGG 16

### 5.2 ResNet

## References

- [1] C. Fabian Benitez-Quiroz, Ramprakash Srinivasan, and Aleix M. Martinez. Emotionet: An accurate, real-time algorithm for the automatic annotation of a million facial expressions in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5562–5570, 2016. pages 8
- [2] Paul Ekman and Wallace V. Friesen. Constants across cultures in the face and emotion. *Journal of personality and social psychology*, 17(2):124, 1971. pages 4, 6
- [3] Paul Ekman, Wallace V. Friesen, and Joseph C. Hager. Facial action coding system (facs). *A technique for the measurement of facial action*. Consulting, Palo Alto, 22, 1978. pages 4, 5
- [4] B. Fasel and Juergen Luetttin. Automatic facial expression analysis: a survey. *Pattern Recognition*, 36(1):259–275, 1 2003. pages 4
- [5] S. Mohammad Mavadati, Mohammad H. Mahoor, Kevin Bartlett, Philip Trinh, and Jeffrey F. Cohn. Disfa: A spontaneous facial action intensity database. *IEEE Transactions on Affective Computing*, 4(2):151–160, 2013. pages 8
- [6] Daniel McDuff, Rana Kaliouby, Thibaud Senechal, May Amr, Jeffrey Cohn, and Rosalind Picard. Affectiva-mit facial expression dataset (am-fed): Naturalistic and spontaneous facial expressions collected. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 881–888, 2013. pages 7, 8
- [7] Maja Pantic and Leon J. M. Rothkrantz. Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1424–1445, 2000. pages 7
- [8] RL Schiefelbusch. *Handbook of methods in nonverbal behavior research*. Klaus R. Scherer & Paul Ekman (Eds.). Cambridge University Press, 1982., volume 4. Cambridge Univ Press, 1982. pages 5
- [9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. pages 13
- [10] Stefanos Zafeiriou, Athanasios Papaioannou, Irene Kotsia, Mihalis A. Nicolaou, Guoying Zhao, E. Antonakos, P. Snape, G. Trigeorgis, and S. Zafeiriou. Facial affect in-the-wild: A survey and a new database. In *International Conference on Computer Vision*. pages 4, 7, 8, 13