

## **AGE AND GENDER CLASSIFIER**

### **I- Definition**

---

#### **I.1- Project Overview**

Deep Learning is a recent advance in the field of Machine Learning which allows or even outperforms human-like performances. It is applied to perform several tasks in Artificial Intelligence, Self-Driving Car, Robotics, etc.

Deep Learning is also used in computer vision where nowadays it outperforms human vision in several domains such as age and gender guessing from pictures.

Some researches are still done on the subject. We could relate for example a recent very interesting research work of Gil Levi and Tal Hassner from the Open University of Israel (Age and gender classification using convolutional neural networks), in the field of Computer Vision and Pattern Recognition (CVPR), in June 2015. They performed a separate Age and Gender Classification using Convolutional Neural Networks. Their process was built around the Adience benchmark, a public Age and Gender Classification Dataset which assembles 26,580 photos of 2,284 subjects.

My main goal with this project is to unify the prediction process in one step for both predictions, age, and gender on the same Adience benchmark. For each image of this benchmark, I should generate a unique label containing its age and gender. But this time I will perform the work using a different Deep Neural Network architecture, trying to get or outperform their accuracies [**86.8±1.4** for Gender and **84.7±2.2 (50.7±5.1 of exact accuracy)** for Age].

At the end of the project, I should also provide some simple and easy to use tools for dataset pre-processing on the Adience benchmark in Python, considering different Machine Learning Frameworks requirements and process unification too. And, as an option, users should be able to install the application on their smartphone and perform age and gender classification on unseen faces.

#### **I.2- Problem Statement**

Nowadays Deep Learning technology is used in Computer Vision to provide or outperform human-like level for Age and Gender classification. This could be very **useful for Identification and Authentication processes for many infrastructure systems**.

We have a need to secure our infrastructures so that only trusted persons could have access.

On the other hand, we also know that smartphones are expanded through the world. Everybody has a smartphone and a solution which includes it should be very practical in term of accessibility and easiness. So why not use it for Age and Gender classification?

We think **the combination of both, Deep Learning technologies for Age and Gender classification, and smartphones should bring a significant revolution in the domain of Identification**. For example using a simple smartphone, we should control the access to a room (a bar for example) to teenagers or children.

So, with this project, using the public Adience benchmark, I am going to create an app which can perform Age and Gender classification using latest high-level computing technologies such as Fully-Connected Networks in a unique step, trying to outperform current known accuracies. During the process, the intended input should be target faces images and as output, we should have a label with the correct information. All bring into a useful application for Identification and Authentication purposes in many infrastructure systems.

The three (03) main steps for this work will be **data transformation** applied to the Adience Benchmark to make it adapted to the model, **model implementation** with a suitable Fully-Connected Network Architecture, and the **model fine-tuning** to make it more reliable.

### I.3- Metrics

Our work is based on the **Adience benchmark**.

The **Adience benchmark** is a dataset of 26,580 photos of 2,284 subjects, also available with separate age and gender labels annotations.

As Evaluation Metrics we will use:

**1- The *Loss* and *Accuracy* of our benchmark model (Age and Gender Classification Model) on the *Unified Training Set* (computed from the *Adience benchmark*).**

**1.1==>** We could consider the **Loss** as the difference between an estimation made by the model ( $\hat{f}$ ) and its true value in the dataset ( $f$ );  $\mathcal{L}(f, \hat{f}) = \left\| f - \hat{f} \right\|_2^2$ , representing the price paid for inaccuracy of predictions in Machine Learning problems. So we expect the Loss to be very small, as small as possible.

In Deep Learning, using weights ( $\omega$ ) and biases ( $b$ ) applied on the input vector:

**Loss ( $\mathcal{L}$ ) = AVERAGE CROSS ENTROPY**

$$\mathcal{L} = \frac{1}{N} \sum_i D(S(\omega X_i + b), L_i),$$

$S$  = A Softmax function which turns scores into probabilities (**the predictions**)

$L_i$ : Local minima which will become the One-Hot Label (**the true values in the dataset**)

$D$  = **Derivative** or **Distance**

**Cross Entropy** = Distance between the two probabilities vectors ( $S$  and  $L_i$ ) :  $D(S, L)$

The **Training Loss** is a measure of the distance average over the entire training set for all inputs and all the labels that are available.

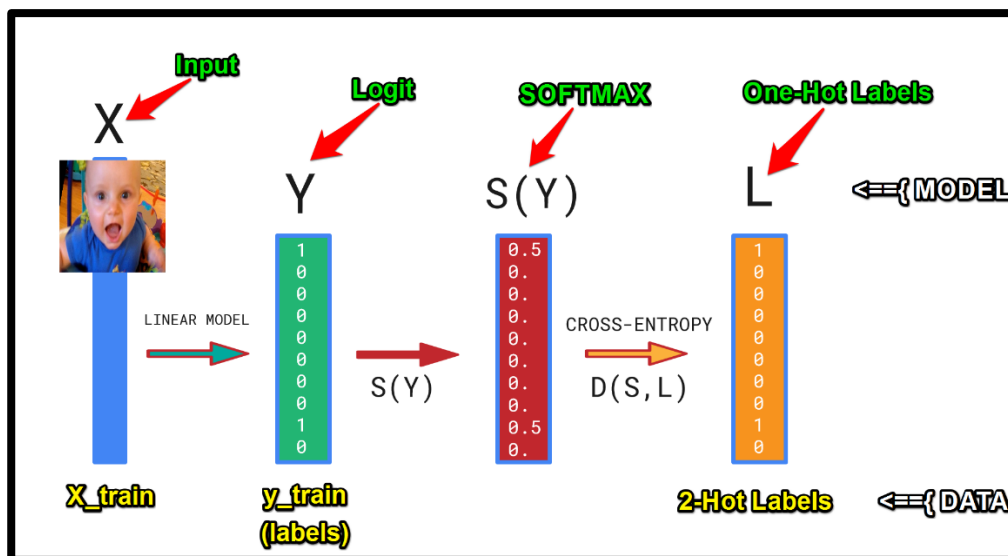


Figure 1: The **Training Loss Process** computation on the Unified Training Set

There are several types of Loss functions according to the Machine Learning task to perform (regression, classification, decision tree, etc.). But **as we are working on a Supervised Classification problem we will only be interested in Loss functions for classification: Square loss, Logistic loss, Hinge loss, Cross-entropy loss**. Indeed, these losses work better on classification problems.

**1.2==>** We could consider the **Accuracy** as the number of correct predictions made by the model among a set of input data (for example the *Unified Training Set* images for the *training accuracy*).

$$\text{Accuracy} = \frac{\text{Number of well predicted One - Hot Labels}}{\text{Total number of images in the Unified Training Set}}$$

**2- The Accuracy of our Age and Gender Classification Model on the *Adience benchmark* and also on *other unseen face images*.** This is performed with a trained model, using Unified Test Set and other images.

Indeed, during the process:

- first, we will evaluate and validate the performance of our benchmark model (*Age and Gender Classification Model*) on its capacity to produce **low Loss** and **high Accuracy** during the training process (How well it learns?). The model learns only and only if the losses decrease toward zero and the accuracy increase toward 1 over all the process. Constant values with low accuracy mean no learning.
- then, we will use the **validated benchmark model** (the **trained model**) to evaluate its Accuracy on the **Adience benchmark** and also on **other unseen face images**.

## II- Analysis

### II.1- Data Exploration

As we know Computer Vision and Deep Learning tasks need data to be well performed. In our case too, we will need data for the age and gender classification.

The dataset to use should have these characteristics:

- enough data for each age class to predict (at least hundreds of images per class)
- enough data for each gender class to predict (at least hundreds of images per class)

If possible, the dataset used should provide as many as possible age classes for more precision. The better use case should be to provide data for each age as a class (for 0, 1, 2, up to 100 years old people for example).

But the collection process is very hard because of:

- availability of people,
- accurate information on them,
- the amount of data required (several dozens of thousand),
- data collection cost and time.
- dataset verification before its publication. An error on the dataset labels could lead to biases on the final application. So it needs a lot of precaution in the building process.

Fortunately, some research groups started to collect and make this kind of dataset publicly available through their research projects.

The biggest dataset we found during our research is the [Adience benchmark](#), a public Age and Gender Classification Dataset of the [Open University of Israel](#) (Face Image Project).

The **Adience benchmark** is a dataset of 26,580 photos of 2,284 subjects, also available with separate age and gender labels annotations.

It provides:

---> **height (8) age classes** = ['(0-2)', '(4-6)', '(8-12)', '(15-20)', '(25-32)', '(38-43)', '(48-53)', '(60- )']

---> **two (2) gender classes** = ['m', 'f'] or ['0', '1']

Below the details about the content of the Adience Benchmark:

Gender\Class	0-2	4-6	8-13	15-20	25-32	38-43	48-53	60-	Total
Male	745	928	934	734	2308	1294	392	442	<b>7777</b>
Female	682	1234	1360	919	2589	1056	433	427	<b>8700</b>
Both	<b>1427</b>	<b>2162</b>	<b>2294</b>	<b>1653</b>	<b>4897</b>	<b>2350</b>	<b>825</b>	<b>869</b>	<b>16477</b>

**Table 1:** Breakdown of the AdienceFaces benchmark into the different Age and Gender classes.

The labels of the images have been made available in separate files, through an [Age and gender Data Preparation Code in Python by Gil Levi](#) in their [Age and gender classification using convolutional neural networks](#) paper. These (.txt) files are available in the **AgeGenderDeepLearning/Folds/train\_val\_txt\_files\_per\_fold** directory and had been prepared for a different goal; separated training process of age or gender.

In our work, we will need to bring them together in a **Unified Dataset**, to have both information (age and gender) for a unique prediction of age and gender on each single image.

In the [Adience Benchmark dataset repository](#), the images have been made available into two sets:

- A non-aligned set: **faces.tar.gz** (1.2Go)
- An aligned set: **aligned.tar.gz** (2.6Go).

Both datasets have the same content (images), and the difference is only the aligned operation on images with the **aligned.tar.gz** dataset.

## II.2- Exploratory Visualization

The images are stored in the **faces.tar.gz** and **aligned.tar.gz** as .JPG images.

In each dataset repository, images are filled in separate directories as below.

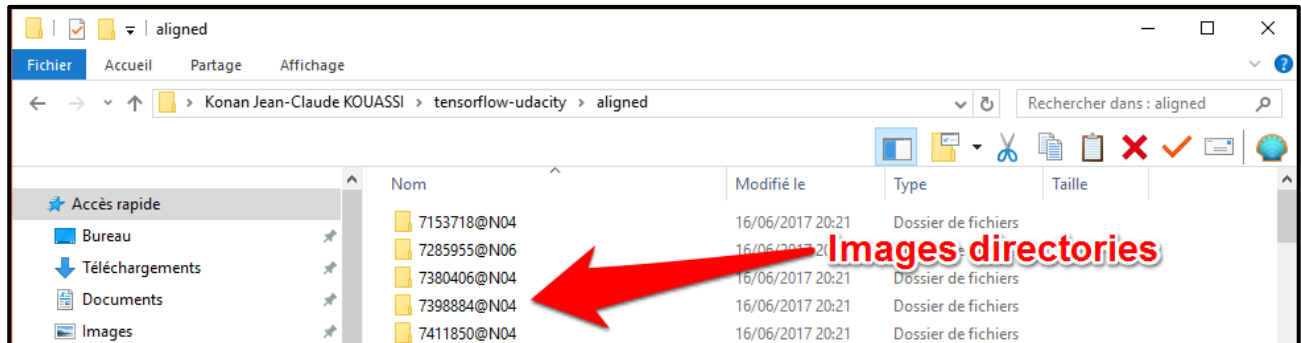


Figure 2: aligned-images-directories

And then in each directory, the .JPG images are directly available with their ids as a name.

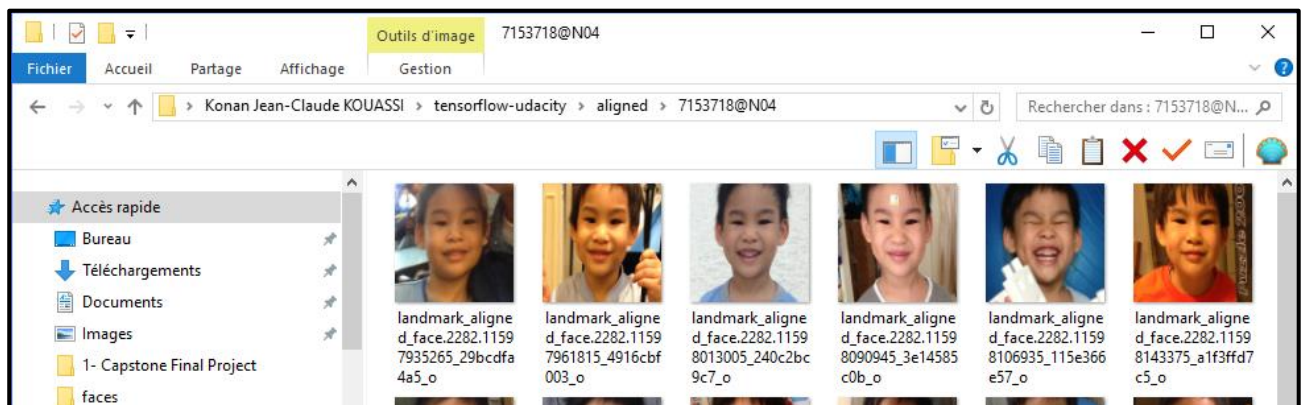


Figure 3: aligned-jpg images location

But, we will also need the **.txt files** which provide useful information about the raw images (their labels).

We could use those available in the [Adience Benchmark dataset repository](#) (the original txt files: **fold\_0\_data.txt**, **fold\_1\_data.txt**, **fold\_2\_data.txt**, **fold\_3\_data.txt** and **fold\_4\_data.txt**), or those available in the [AgeGenderDeepLearning/Folds/train\\_val\\_txt\\_files\\_per\\_fold](#) directory which had been prepared for separated training process of age or gender.

The figures below show an example of the content of these files.

fold_0_data - Bloc-notes													
Fichier	Edition	Format	Affichage	?									
user_id	original_image	face_id	age	gender	x	y	dx	dy	tilt_ang	fiducial_yaw_angle	fiducial_score		
30601258@N03	10399646885_67c7d20df9_o.jpg	1	(25, 23)	f	0	414	1086	1383	-115	30	17		
30601258@N03	10424815813_e94629b1ec_o.jpg	2	(25, 32)	m	301	105	640	641	0	0	94		
30601258@N03	10437979845_5985be4b26_o.jpg	1	(25, 23)	f	2395	876	771	771	175	-30	74		
30601258@N03	10437979845_5985be4b26_o.jpg	3	(25, 32)	m	752	1255	484	485	180	0	47		
30601258@N03	11816644924_075c3d8d59_o.jpg	2	(25, 32)	m	175	80	769	768	-75	0	34		
30601258@N03	11562582716_dbc2eb8002_o.jpg	1	(25, 23)	f	0	422	1332	1498	-100	15	54		

Figure 4: fold\_0\_data.txt file content (original txt files)

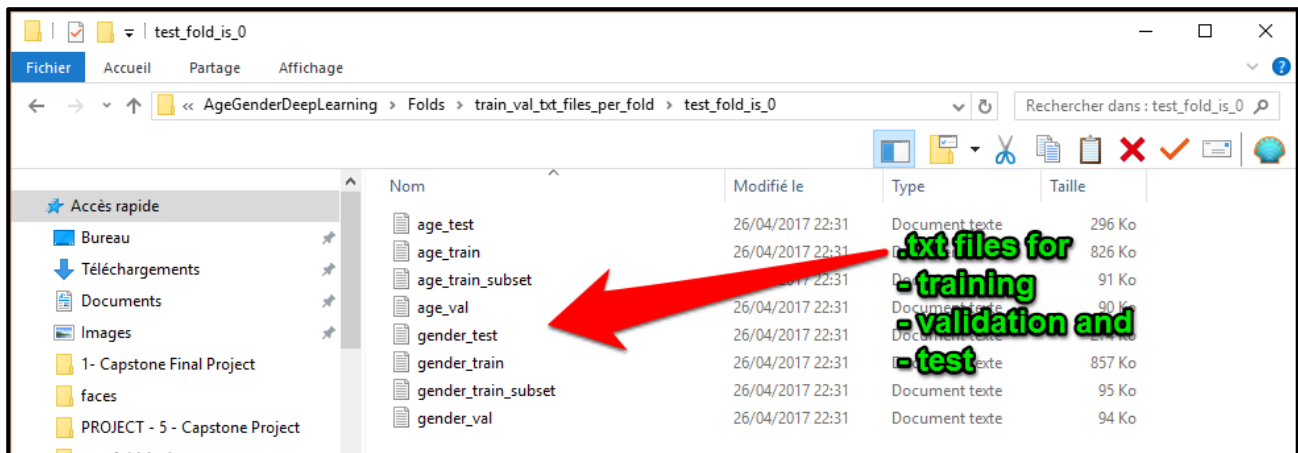


Figure 5: age and gender .txt files for training, validation and test.



Figure 6: age-train txt file content

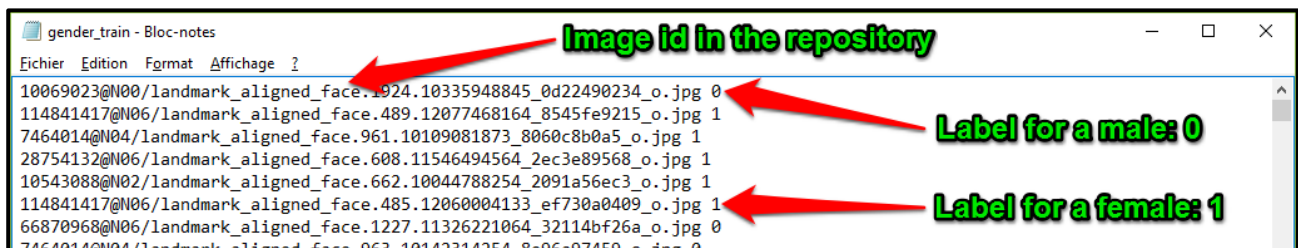


Figure 7: gender-train txt file content

So, for the fusion process (dataset unification) we could either run a script to extract relevant information from the **original txt files** (*user\_id*, *original\_image*, *face\_id*, *age* and *gender*) or from the **age and gender .txt files for training, validation and test** (*Image\_id*, *age class label* and *gender class label*).

### II.3- Algorithms and Techniques

It is possible to use several algorithms and techniques to perform Computer Vision and Pattern Recognition tasks with Deep Learning. But in our specific use case of age and gender classification, we will use those which are more suitable to the available dataset (Adience Benchmark) and also to age and gender recognition in images tasks.

**1-** We intend to perform the **Data Unification** through a python script which will bring together into a dictionary each **image full path** as the key, and the **age class and gender** as its value.

**2-** For the **training and testing data split**, we will use the **sklearn train\_test\_split** tool to split the Unified Dataset into training and testing datasets. But this tool accepts only lists, numpy arrays, scipy-sparse matrices or pandas dataframes as input. So, we will need to transform the Unified Dataset dictionary into two lists: one list of images full path as features (**age\_gender\_feature\_list**), and another list of age class and gender as targets or labels (**age\_gender\_label\_list**).

If we need to provide a **Validation dataset**, we will use this time the model estimators tools to provide a subset of the Training set as a Validation set.



We could also compute several pre-processing operations to perform on the Unified dataset to make it more suitable to the network model. We could perform for example:

==> **Histogram equalization**: it is the process of modifying the intensities of the image pixels to enhance the contrast. It makes images look nice so that the algorithms could better detect the edges. But this process is heavy, it consumes a lot of memory. You may prefer to compute it on the training batches, instead of the whole dataset directly (it will make the training process slower than without this operation).



Figure 8: Histogram equalization on an image of the unified dataset

==> **Label binarizing**: it transforms the labels for training into binary values, and then *inverse\_transform* the prediction labels into their correct format.

==> **min max scaling**: applied to the features, it changes their range for a better level separation. It scales and translates each feature individually such that it is in the given range on the training set, i.e. between zero and one.

==> **rgb to yuv(or ycbcr)**: *ybcr* is the color space which is commonly used by video codecs, it is sometimes incorrectly called "YUV". It returns a ndarray for fast array-processing. This format will make the data ready for video purposes.

These operations could be performed either before or during the training process.

Another useful technique consists in **providing a visualization of the training and test sets**. This could help us to have a clear view of the ratio of age and gender classes in them, and so take suitable decisions for improvement.

For example for a sample of **200 images**, we could make the following visualization using 33% of the data for the *test\_size* with the sklearn *train\_test\_split* tool (*len X\_train*= 134, *len X\_test*= 66).

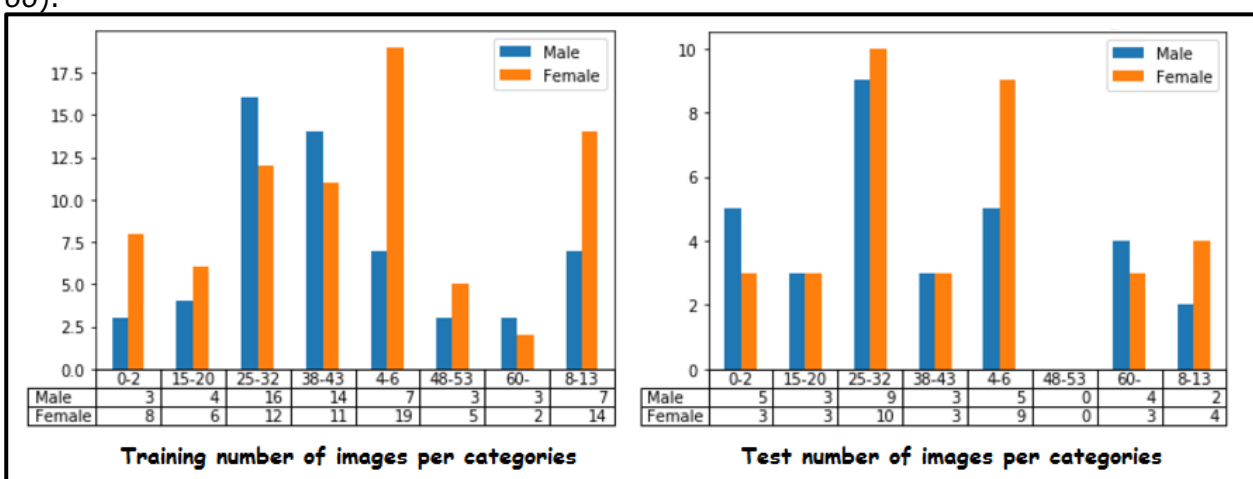


Figure 9: The number of images per categories for both the training and test set

Among all observations, we could easily see that there is no data for the '48-53' age class in the test set.

So, we could improve it, running on a greater sample of images or on the whole dataset as shown below, to ensure that each category of image has enough data to learn (at least several hundred):

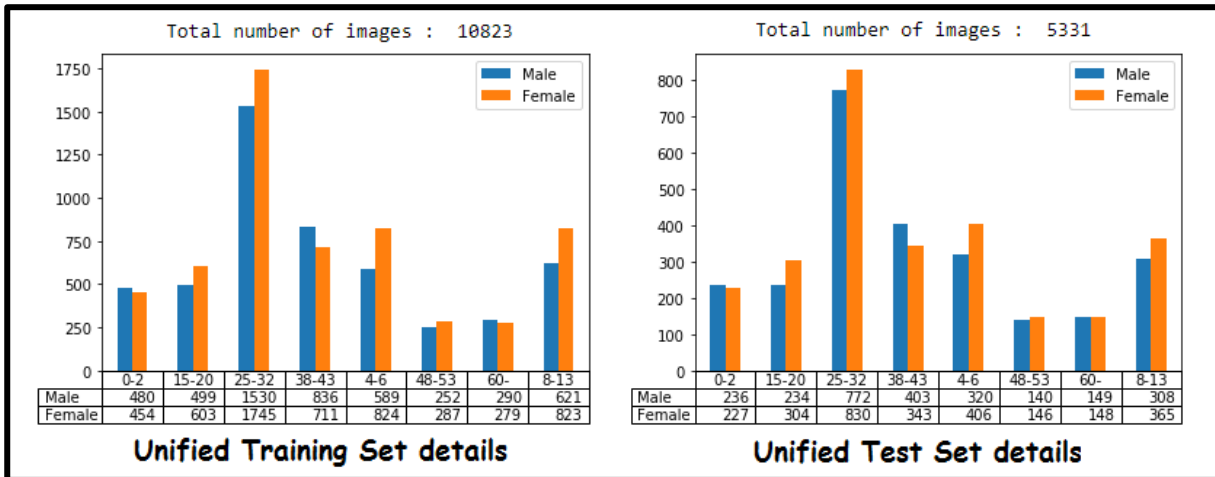


Figure 10: Unified training and test sets details for the whole Unified dataset

**3-** For the **Network Model**, we will use a **Fully-Connected Network (FCN)** in a unique step for prediction. Indeed, this network could provide a pixel-wise precision on images as all the nodes of adjacent layers are fully connected. So, all information will be kept in the network nodes, and this should be useful for age classification task which should require more details to differentiate age classes and gender on images.

#### II.4- Benchmark

The goal of this project is to build an Age and Gender Classifier and then deploy the final improved model into a mobile application to allow live Age and Gender Classification on faces.

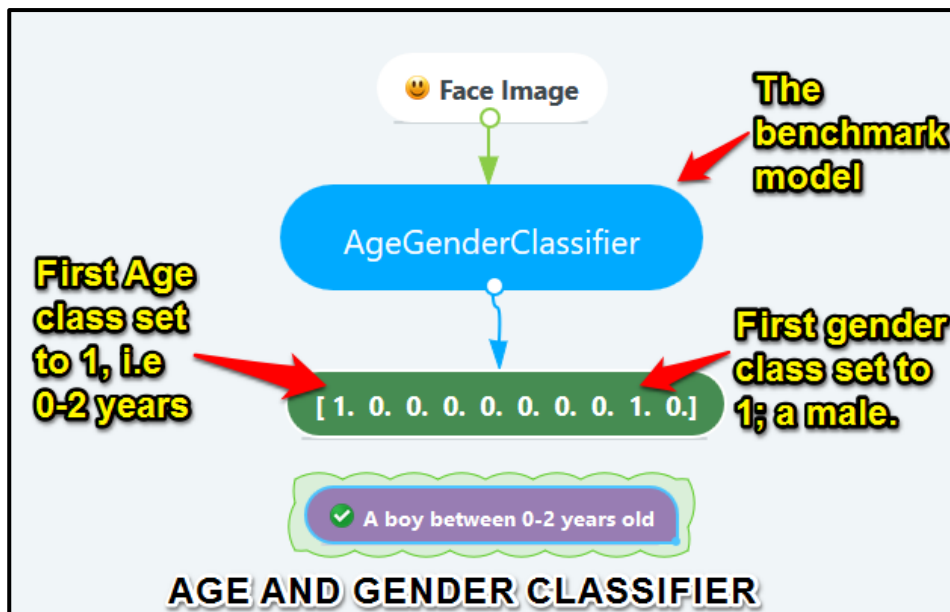


Figure 11: The benchmark model (Age and Gender Classification Model)

**1-** As shown in Figure 11, raw face images are sent to the benchmark model as input. Then they will be pre-processed into a numpy array to meet the classifier requirements. Any image should be transformed, no matter its original form (an image stored on the computer, taken by a smartphone or via a video, etc.), into a numpy array and sent to the classifier input.

**2-** The model itself (Age Gender Classifier) consist of a **Fully-Connected Artificial Neural Network (FCN)** which takes the images as input and output a 10-dimensional One-Hot Label (8 digits for the age classes and the last 2 digits for the gender classes).

**Note:** Here we keep the appellation One-Hot Label because it is done for each class (age or gender) separately, but the final result should be a 2-Hot Label. So, the expected result is the 10-dimensional label (2-hot label).

This model will be transformed through a script into a friendly User-Machine-Interface, which should deal with the cases where the result is not a 2-hot label. If the result is a 1-hot label or 3-hot label, or all digits come to zero, or anything else which is not a 2-hot label, the script should ask the user to retry with another image or prompt him that it is not able to give the appropriate age and gender on the image ("Undefined" for example).

**3-** Two frameworks will be used to create two Fully-Connected Artificial Neural Networks (FCNs); one with **TFLearn** and the other with **Tensorflow Keras**. Both admit numpy array and lists as input for the training process. Doing so, we will be able to compare their influences on the results and increase the model performance. The results with one framework could help improve the other and vice-versa.

It is also possible to use **Tensorflow DNNClassifier** or **DNNRegressor** which are also adapted for classification tasks, but in our case, they will require more data pre-processing operations, as they work only with dictionaries as input. So we would need to recreate dictionaries of training and test dataset after the split operation.

However, they have the advantage to produce a few line of code for the network model and easy visualization tools.

### III- Methodology

#### III.1- Data Preprocessing

First, we should download the Adience Dataset [HERE](#) (*faces* or *aligned* compressed files). We should also clone the [AgeGenderDeepLearning](#) repository. Both files should be extracted and made available at the same location.

Then, to unify the Adience benchmark according to our goals we created a function (***age\_gender\_per\_image***) to perform dataset unification. This function ***generates an age and gender dictionary*** of all images which have both information (age and gender) available in the Adience Benchmark. The keys of the dictionary are the full path name of the images in the Adience benchmark, and the age class and gender class are the tuples values.

Indeed, the ***age\_gender\_per\_image*** function uses all the txt files in the directory presented in II.2 (Exploratory Visualization): `[./AgeGenderDeepLearning/Folds/train_val_txt_files_per_fold]`. In each fold we have these files: {*age\_test.txt*, *age\_train.txt*, *age\_train\_subset.txt*, *age\_val.txt*, *gender\_test.txt*, *gender\_train.txt*, *gender\_train\_subset.txt*, *gender\_val.txt*} (Figure 5).

The age and gender dictionary returned by the ***age\_gender\_per\_image*** is called ***age\_gender\_dic***.

And, using the ***age\_gender\_dic***, we created a new ***One\_Hot\_Labels*** dictionary which also has the full path name of the images in the Adience benchmark as keys, and uses ***as values*** the 10-dimensional ***One-Hot Labels*** created from the ***age\_gender\_dic*** values (from the age class and gender tuples).

The ***One\_Hot\_Labels dictionary*** is considered as the ***Unified DataSet*** as it has for each image, its age and gender information stored as a unique one-hot label (***2-hot label***).

The size of the Unified Dataset is **16156**. *It contains only unique images that have both information, age and gender available in the Adience Benchmark.* We get the same size from the *aligned* and *non-aligned* datasets of this Benchmark.

**Note: it is possible to keep the age and gender classes, instead of transforming them into a 10-dimensional vector. In this use case, the *age\_gender\_dic* is the Unified DataSet.**

Using a python script, we could have a breakdown of the Unified Dataset benchmark into the different Age and Gender classes:

Gender\Class	0-2	4-6	8-13	15-20	25-32	38-43	48-53	60-	Total
Male	716	909	929	733	2304	1239	392	439	<b>7661</b>
Female	681	1230	1188	907	2575	1054	433	427	<b>8495</b>
Both	<b>1397</b>	<b>2139</b>	<b>2117</b>	<b>1640</b>	<b>4879</b>	<b>2293</b>	<b>825</b>	<b>866</b>	<b>16156</b>

**Table 2:** Breakdown of the *Unified DataSet benchmark* into the different Age and Gender classes



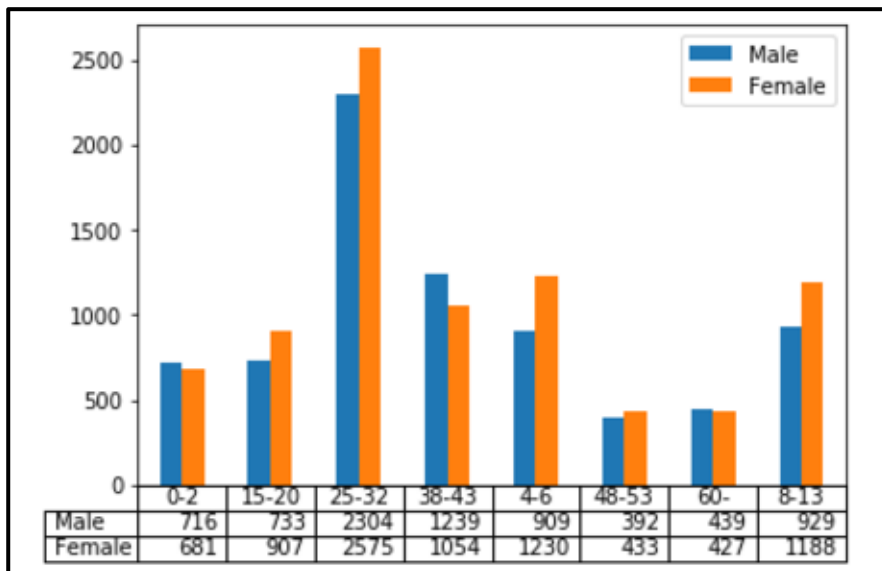


Figure 12: Unified Dataset benchmark plot

Finally, the Unified Dataset will be used to produce **age\_gender\_feature\_list** and **age\_gender\_label\_list** which are the inputs for the **Training and Test Sets** creation, using the sklearn *train\_test\_split* tool. This tool accepts only *lists*, *numpy arrays*, *scipy-sparse matrices* or *pandas dataframes* as input.

If we need to provide a **Validation dataset**, we will use this time the model estimators tools to use a part of the Training set or even the Test set as a Validation set.

The full code is available on my Github [HERE](#).

### III.2- Implementation

Here we created two Fully-Connected Artificial Neural Networks (FCNs); one with TFLearn and the other with Tensorflow Keras. Both admit numpy array and lists as input for the training process. They also have advanced APIs and awesome visualization tools to easily perform Professional Deep Learning tasks. Let's note that Keras and TFLearn are now both integrated into the Tensorflow Framework, as a proof of their efficiency.

Running the model on two different frameworks, we will be able to compare their influences on the results and better improve the model performance.

For the installation of these frameworks, follow the links below:

- [TFLearn installation](#)
- [Tensorflow installation](#)

#### III.2.1- Model Building

The Fully-Connected Artificial Neural Network (FCN) to implement has the following architecture:

- 1 Layer of 1024 nodes
- 1 Layer of 512 nodes
- 1 Layer of 256 nodes
- 1 Layer of 128 nodes
- 1 Layer of 32 nodes
- 1 Layer of 10 nodes

The last layer of 10 nodes is coupled with a **Softmax** activation.

**Note: if we choose tuples (age class, gender) as output, we should set the last layer number of nodes to 2. The final architecture should be a bit different, with respect to the improvement operations during the model tuning.**

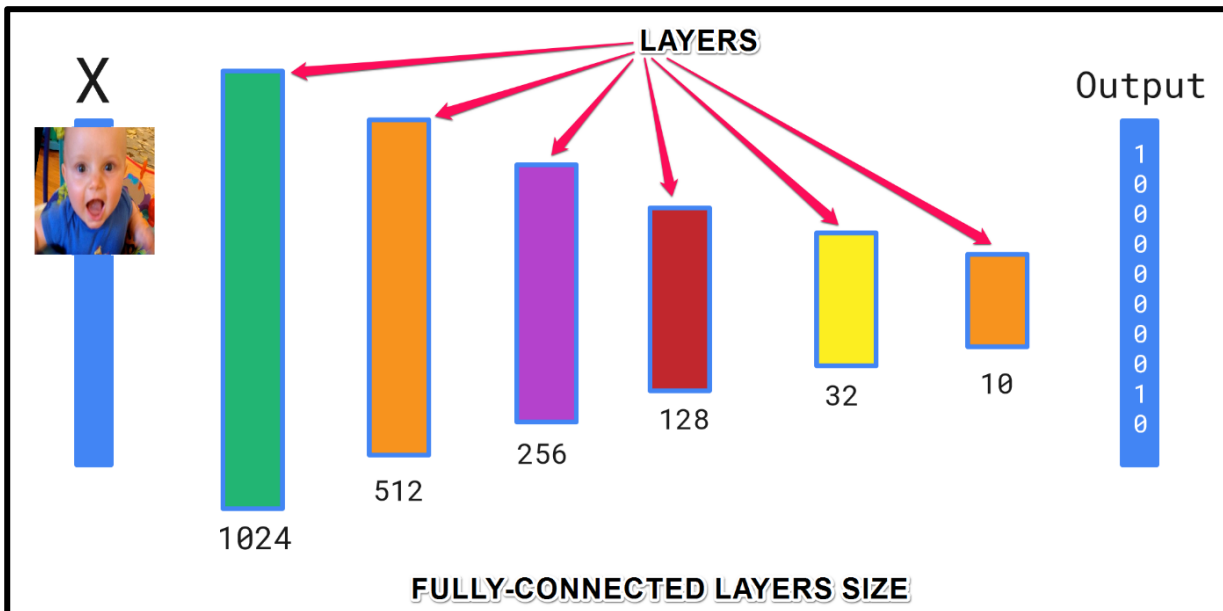


Figure 13: Our Fully-Connected Artificial Neural Network (FCN) Architecture

Each layer color represents the information about the input stored in. The information included in one layer are always relative to the previous one.

In another word, each layer has information about the previous one and gives its own information to the next one, and so, the last layer could recursively have information on the first layer.

Each layer is a function of all elements (nodes or units) of the previous layer.

A function of recursivity between two (2) layers  $f$  and  $g$  where  $x$  represents the elements of the layer before  $g$  should be:  $F = f \circ g = f(g(x))$ .

For an FCN each value of  $x$  (nodes or units) is linked to all values (nodes or units) in  $g$  and vice versa.

For a CNN some values (nodes or units) in  $g$  would not be linked with some values of  $x$  (nodes or units of previous layer to  $g$ ).

You could have a simulation view of how neurons (nodes) are connected and work together on <http://playground.tensorflow.org/>.

### III.2.2- Model implementation

Once we have a defined architecture of the FCN, the network model could be built independently with each framework APIs (*TFlearn* or *Tensorflow Keras*).

- ✓ With **TFlearn**, an example of FCN is given at [http://tflearn.org/getting\\_started/](http://tflearn.org/getting_started/), more clearly in the *Weights persistence* part.
- ✓ With **Tensorflow Keras**, an example of FCN is given [HERE](#).

Using Keras, Fully-Connected layers are the Dense Layers.

And the Tensorflow layers required to perform the example above are:

- `tensorflow.contrib.keras.layers.Input`
- `tensorflow.contrib.keras.python.keras.layers.core`
- `tensorflow.contrib.keras.python.keras.models`

### III.2.3- Model evaluation

The two frameworks have the same methods' names for *evaluation* and *prediction* (*evaluate* and *predict* methods).

But evaluation with *TFlearn* returns only Accuracy scores, while with *Tensorflow Keras*, additionally to the Accuracy we also have the Loss.

### III.2.4- Model visualization

Here Loss and Accuracy are our Evaluation Metrics.

- ✓ With *TFlearn*: we use *tensorboard* to visualize the learning curves.

**tensorboard --logdir='tflearn\_logs'** (in a terminal, from the directory where 'tflearn\_logs' is)

- ✓ With *Tensorflow Keras*: as *tensorboard* is not included in the version 1.1 of *Tensorflow*, we printed the learning curves using the training history.

```
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Tensorboard is available with *Tensorflow Keras* in the Tensorflow 1.2 branch, some updates are still ongoing with this branch, and it will be worth to follow and try it.

**At this point, we should have a working Age-Gender Classifier.**

The full code is available on my Github [HERE](#).

### III.3- Refinement

Both frameworks provide useful tools to fine-tune the models using **network definition** (activation for example), **estimators' definition** (regression for example) and the **training process** (the fit method).

So, the elements to consider for improvement, to increase the accuracy with each framework are provided with the following checklist:

- ✓ **The number of epoch:** increase this number could give to the model more time to learn better.
- ✓ **The optimizer functions** (Adam, RMSProp, Adadelta, SGD, etc.).
- ✓ **The activation functions** (sigmoid, relu, softplus, hard\_sigmoid, softsign, etc.).
- ✓ **The loss functions** (categorical\_crossentropy, mean\_squared\_error, squared\_hinge, sparse\_categorical\_crossentropy, hinge, mean\_squared\_logarithmic\_error, etc.).
- ✓ **The regularization functions** (L2 regularization, l1\_l2, max-norm, etc.).
- ✓ **The initializers** (he\_normal, he\_uniform, Xavier normal initializer, Xavier uniform initializer, LeCun uniform initializer, etc.).
- ✓ **The model:** when building the model we thought to another version of the output which could require less computation. The idea is to output a couple (a, g) at the end of the network. Where a is the age range [0:7] and g, the gender (0 or 1). So the output and labels of **Figure 13** would become (0, 0). 0 for the first class of age [0-2 years] and 0 for the gender as a male. But even if it does not clearly appear, we have still 10 classes (8 possibilities for age and 2 for gender).

The computation cost during the learning process is:

$$\text{- First model: } \left( \overbrace{\{0, 1\}}^2, \overbrace{\{0, 1\}}^2, \dots, \overbrace{\{0, 1\}}^2 \right) \Rightarrow [0. 0. 0. 0. 0. 1. 0. 0. 1. 0.] \Rightarrow 2^{10} = 1024$$

$$\text{- Second model: } \left[ \overbrace{\{0, 1, 2, 3, 4, 5, 6, 7\}}^{8 \text{ possibilities}}, \overbrace{\{0, 1\}}^2 \right] \Rightarrow (3, 1) \Rightarrow 8 * 2 = 16$$

So, we could try the second model and fine-tune our implementation. If we pick the second model, let's note that it is still possible to switch between the two kinds of output using a simple function (generate a 10-dimensional 2-hot label from a tuple).

- ✓ **The size of the unified dataset:** the Adience benchmark presents two datasets, non-aligned faces dataset (1.2Go) and aligned dataset (2.6Go). We could check the difference in accuracy between the non-aligned dataset and the aligned one for the same image ids.  
We could also **increase the sample of training data** to improve the accuracy.  
Here are some Datasets which could be used to perform this task.

A specific script should be written for each dataset to add its content to the Adience benchmark (i.e. the Unified dataset):

- **FaceTracer Database**

<http://www.cs.columbia.edu/CAVE/databases/facetracer/>

15,000 faces in the database

- **Color FERET Database**

<https://www.nist.gov/itl/iad/image-group/color-feret-database>

Contains 1564 sets of images for a total of 14,126 images that includes 1199 individuals and 365 duplicate sets of images.

- **Karolinska Directed Emotional Faces (KDEF)**

<http://www.emotionlab.se/resources/kdef>

A set of totally 4900 pictures of human facial expressions of emotion.

- **Chicago Face Database**

<http://faculty.chicagobooth.edu/bernd.wittenbrink/cfd/download/download.html>

Version 2 of the database includes high-resolution photographs of 597 male and female targets of varying ethnicity. File size is approx. 1.5 GB.

- **Aging mind Stimuli Face Database**

<http://agingmind.utdallas.edu/download-stimuli/face-database/>

A database of 575 individual faces ranging from ages 18 to 93.

Another tool used for data increasing during the training process is **Data Augmentation**. Each framework has its own embedded methods to perform real-time data augmentation on the inputs, while your selected hardware (CPU or GPU or TPU) is performing model training ([TFLearn Data Augmentation](#) or [Tensorflow Data Augmentation](#)). More precisely, once Data Augmentation is defined, it will be applied to the batches of images during the training process.

Run training on large datasets and networks could take weeks on CPU and several hours or days with GPU. The initial network size [1024 - 512 - 256 - 128 - 32 - 10] was too heavy for our computing environment.

Indeed, we performed the training on two platforms:

- [Microsoft Azure Notebooks Preview](#) which provides a 4Gb memory on CPU. Here, we were able to download all the dataset and make the tests on hyper-parameters with a sample of data (at least 5000 images). But if there is a histogram equalization we reduce it to 1000 images when the operation is performed before the training process. If the equalization is performed on images batches during the training process we could keep the initial number of images.  
The training on this platform with a sample of data helped us see how things works and so make best choices for parameters.
- [Floyd](#). This platform offers us (at this time: June 2017) 100 hours of free 1 GPU, with a delay of 24 hours max for each instance (4 Cores, 61G of Memory, 12G GPU Memory, 100G HD). So we were able to run here the training on the whole dataset.  
We also noted that, despite its great capacities, the histogram equalization can't be applied to the whole dataset before the training. Perform it during the training process seems to be the best technique, even if it makes the training slow.

So, for more efficiency, we finally reduced the layers size to [64 - 32 - 32 - 16 - 16 - 10] for the first model (*OneHot Label model*) and to [32 - 32 - 16 - 16 - 10 - 2] for the second model (*Tuples model*).

These network sizes allowed us to get suitable accuracy without improvement (around 50% for the first model and around 90% for the second model).

So the improvement process tools could still be applied to them to increase the accuracies.

Below a visualization for the *network graphs for both models*.

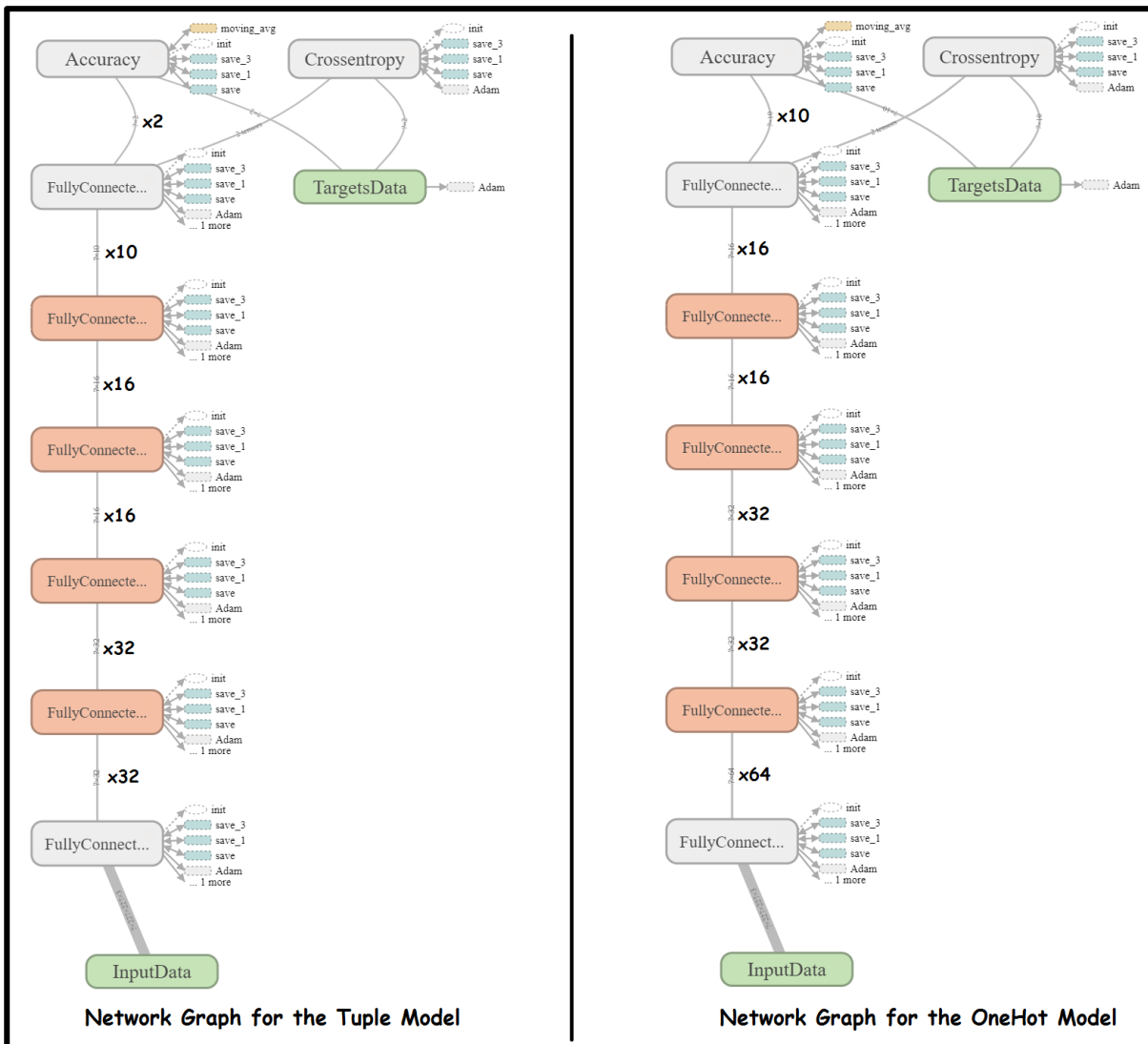


Figure 14: Models' Network Graphs

## IV- Results

### IV.1- Model Evaluation and Validation

#### IV.1.1- Before the improvement

==> **Tests done on a sample of 7000 images, 1000 epochs and a batch size of 128:**

##### - **With tuples:**

The training losses evolve from **2** to **1.23**.

The training accuracies increase from **0.90** to **0.9629**.

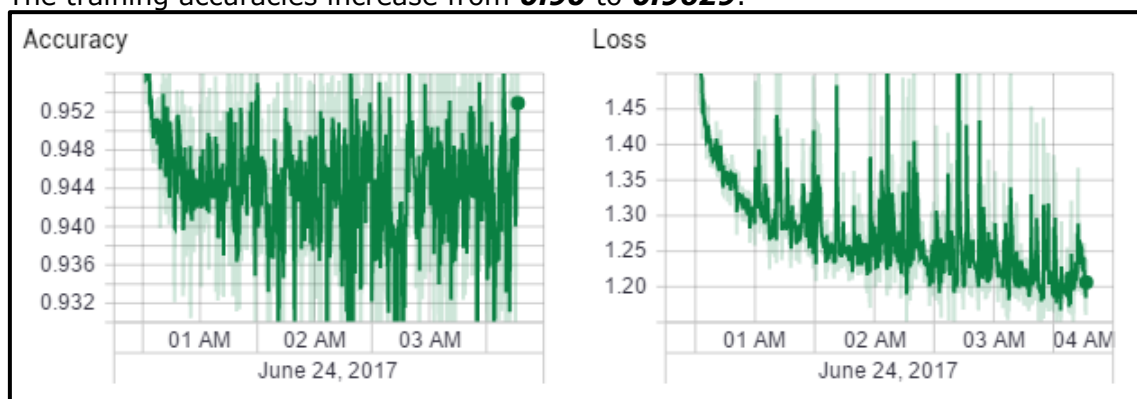


Figure 15: Tuples Model on Tensorboard observation with TFLearn



- **With 2-hot labels:**

The training losses evolve from **4.80** to **1.74398**.

The training accuracies increase from **0.0000** to **0.4651**.

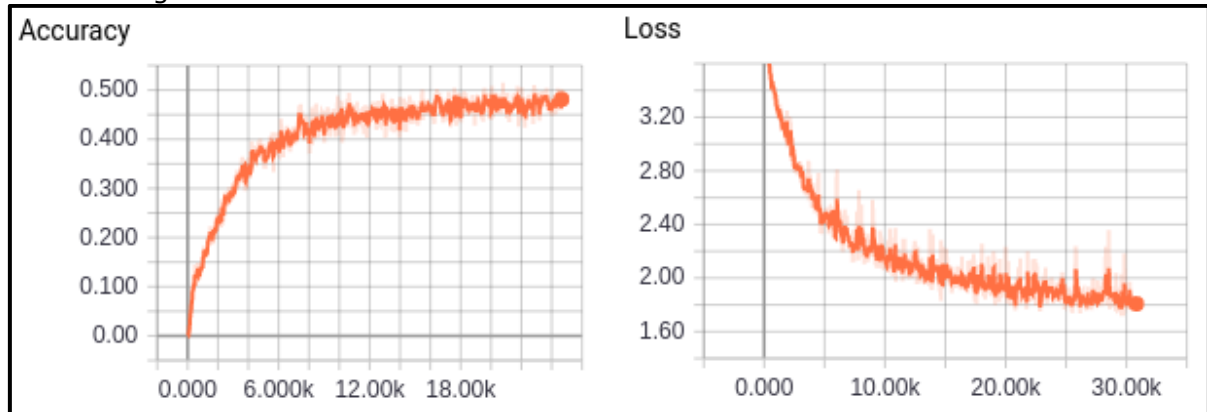


Figure 16: OneHot Model on Tensorboard observation with TFLearn

We can see that both models learn well up to 1000 epochs. But **the tuples model has already 96% of accuracy** for only 1000 epochs while **the 2-hot labels model turns around 50%**.

#### IV.1.2- For the improvement

As explained in *III.3- Refinement*, for the improvement we dealt with the following aspects:

- For the network:

We set an **activation='relu'** and a **regularizer='L2 or max-norm'** for each layer to ensure a uniform distribution of the weights in each layer. Max-norm regularization prevents the network weights to grow very large. The **max\_value** should be between 3 and 4.

We set a **dropout of 0.8** for the first hidden layer adjacent to the input layer. The dropout operation keeps the probabilities between layers. A value of 0.8 increases the chance for the network to threshold low input values of a neuron to 0 and its high input values to 1. It is also a typical value for real-valued inputs (image patches or speech frames). We also set a **dropout of 0.5** each time the number of nodes changes from a layer to another. This value of 0.5 seems to be close to optimal for a wide range of networks and tasks. It ensures an equal probability for the network when turning values (neuron's output) to 0 or 1. And the dropout is a [Simple Way to Prevent Neural Networks from Overfitting](#). Perhaps, a model with high accuracy could suffer from Overfitting.

- For the estimator: optimizer='**adam**', loss='**categorical\_crossentropy**', learning\_rate=0.001.

Here according to our choice, it is proven that for the dropout, using high learning rate (**0.1** or **0.01**) and/or momentum (between **0.95** and **0.99**) significantly speed up learning.

- For the trainer (fit function): **validation\_set=0.3**

- For the number of epochs: We decided at least to double it (**higher or equal to 2000**). This should help the model to learn better.

- For the amount of data: The first run was performed on a sample of 7000 images. Now for the improvement, we will run it on the whole dataset (16156 images).

#### **==> Tests done on the whole dataset (16156 images), 2000+ epochs and a batch size of 128:**

Regarding the results without improvement, we are tempted to choose to work on the model with the highest accuracy when running on a sample of 7000 images and 1000 epochs. But the second model learning curve seems to be steadier and could lead to a better generalization if we can improve it.

So we tried to improve both models.

- **With tuples:**

The losses evolve from **2** to **1.2**.

The accuracies increase from **0.90** to **0.972**.

- **With 2-hot labels:**

The losses evolve from **4.50** to **1.63**.

The accuracies increase from **0.0000** to **0.60**.

Applying the improvement plan above with dropout, which is a cutting-edge technique for this kind of task, **we observed that the dropout effectively prevents overfitting (training and testing curves very close)**, but in our case it kills the learning process (the learning breaks the accuracy down up to 100 epochs, i.e. at about 2000 steps). We could also note that observations in the plan for dropout have been done on millions of images (*600,000 images for the training set*), while we have just about 10,000 images in our training set.

Talking about the framework, TFLearn does not perform max-norm regularization, we used L2. But Tensorflow Keras allows it, with required learning rate and momentum configuration using SGD optimizer.

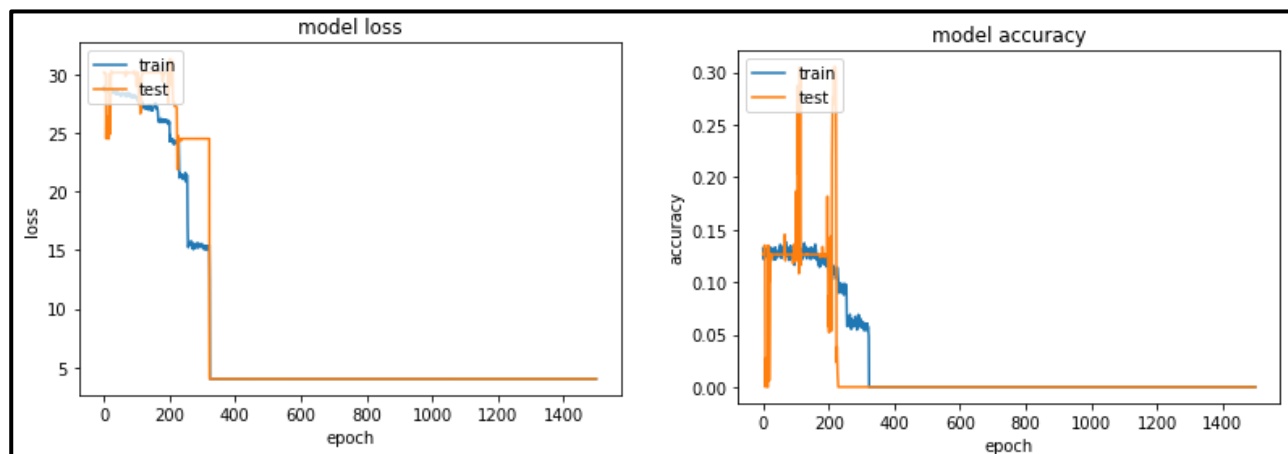


Figure 17: Dropout effect observation with Tensorflow

*So, we concluded that perhaps the dropout deletes some useful information for the network to learn the difference between images.*

So, pursuing the improvement process, **we assert that increase the number of layers and nodes (units) would help the model to have more detailed information in the Fully-Connected Layers and so improve the accuracy.** For example, the following architecture: [128 - 32 - 32 - 16 - 16 - 10]. And, of course, we should also increase the number of training data (Data Augmentation) and epochs.

**And this approach worked perfectly (as reported above).**

### Through experimentation,

For the OneHot model: we found that **linear** activation and **categorical\_crossentropy** loss speed up the learning process and increase the training, validation and test accuracies.

For the tuples model: we found that **relu** activation, **categorical\_crossentropy** loss, **L2** regularizer and **Adam** optimizer speed up the learning process and increase the training, validation and test accuracies.

Finally, we have a better result with the whole dataset in comparison to the first sample.

### => Output function for result interpretation

Algorithms have their internal metrics to compare images and labels. The results are mostly in the form of computed probability values for classes. So once we have a good accuracy, we should apply a function on the prediction results for each model, to make it understandable.

Below an overview of the algorithmic steps of implementation of this kind of function:

- **With 2-hot labels:**

- Put the highest number of the first 8 classes to 1 and others to 0.
  - Put the highest number of the 2 latest classes to 1 and the other to 0.
  - Return them into an understandable format.

- **With tuples:**

- Verify through experimentation on the training set, each age class prediction values range returned by the model. Indeed, the predictions performed in the same age class with both genders (male and female) cover a certain range (or interval of prediction values).
  - So according to the defined ranges, return the correct class for age.

The gender class is much easier; set the values above 0.5 to 1 and the others to 0. And at the end return them in an understandable format of age and gender.

The hardest part with this model is that each loss function has its own metric, so the ranges will automatically change with the loss function we choose. And if we want to experiment all the loss functions on the model, the experimentation (ranges detection) will take a while.

**Let's note that a high accuracy makes the ranges stable.**

With the 2-hot labels model, ranges are already done by the separated classes.

An example of output function for result interpretation will be available on my [GitHub](#).

## IV.2- Justification

After our implementation, according to the results, we could affirm that:

- **With the 2-hot labels model:**

*Training accuracy = 60%, Test accuracy=58%, Final Loss =1.63.*

The 2-hot labels model is good as it reaches the **exact accuracy of the Age estimation results on the Adience benchmark (see Table 3)** of the [Age and gender classification using convolutional neural networks paper](#) (i.e.  $50.7 \pm 5.1$ ).

It is very steady regarding the learning curves. This model also facilitates the results interpretation, as its classes are already separated in independent ranges.

That's said it could generalize well on unseen images.

Of course, we have already reached our defined goal with this model (*reach existing accuracy, the exact accuracy*), but we are sure that with more computation (*several days or weeks of training time*) we should reach or outperform the *1-off accuracy* of the *paper's table 3* too.

- **With the tuples model:**

*Training accuracy = 97.27%, Test accuracy= 96.12%, Final Loss =1.2.*

This model reaches the accuracy of 96% for both training and validation sets. So, it is very accurate. This means that the predictions ranges should be well separated according to each class of age and gender. And we should find a precise range of prediction values for each class.

Finally, we decided to keep both models. The 2-hot label model has a lower accuracy but is very steady to well generalize everywhere. It could be useful in stochastic environments. The tuple model has a very low computational cost and a high accuracy. It would run well on devices with weak computational resources.

**With both models we have reached our goal of trying to get or outperform the existing accuracies, in a unique step, using Fully-Connected Networks only.**

## V- Conclusion

---

### V.1- Free-Form Visualization

Using an FCN only on a Unified Dataset was a great idea as it allows us to reach an accuracy around 96% for the Tuple model and around 60% for the OneHot model.

Indeed, during our implementation, we found that the FCN structure, which put all units in relation and so makes available all information about the input data, helps the model in age and gender detection. Any part of information could be useful to make the difference between all classes to learn.

And this work, allow us to perform Age and Gender Classification using one network, instead of separate age and gender networks.

Below, we could have a view of how the models performed predictions on unseen images.

As explain in **IV.1- Model Evaluation and Validation**, at the **Output function for result interpretation** section, a simple python function applied to the output (*probabilistic values*) should return the predicted labels in their correct form, and/or in a more understandable format for a human.

The following examples are taken from the OneHot model, where the prediction ranges are already well separated (to facilitate the output interpretation process).

Indeed, the output has this form:

`['(0, 2)', '(4, 6)', '(8, 12)', '(15, 20)', '(25, 32)', '(38, 43)', '(48, 53)', '(60, 100)', 'm', 'f']`, where the model returns a probabilistic value for each class; the age class with the highest value is set to 1, the gender class with the highest value is also set to 1, and all other values are set to 0.

- **On a picture from the test set:**

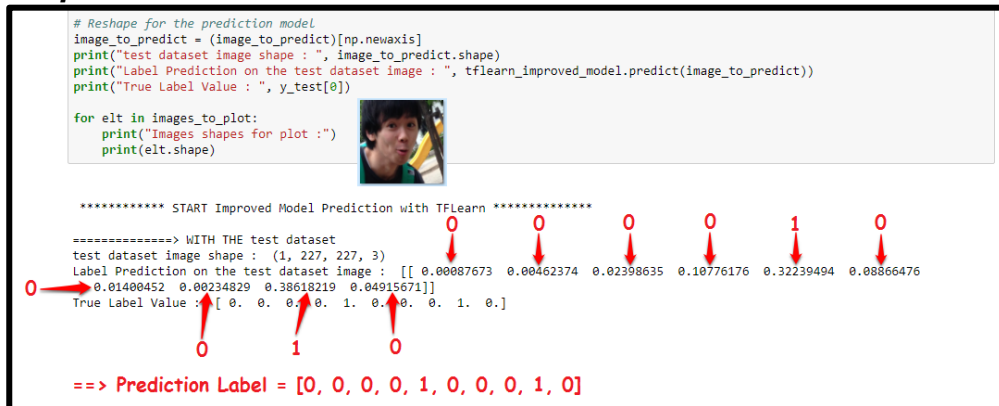


Figure 18: Label prediction on an image from the test set

In this picture, the model made a prediction on an unseen image taken from the test set. And it matches with the true label of this image in the test set:

- True Label = `[0., 0., 0., 0., 1., 0., 0., 0., 1., 0.]` ; **a male between [25 – 32] and/or under 38.**

- **On my own picture:**

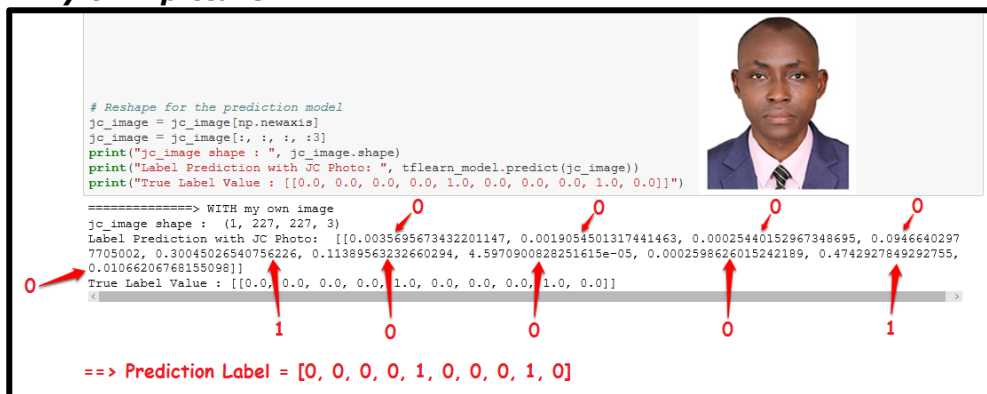


Figure 19: Label prediction on my own face image

In this picture, the model made a prediction on an unseen image (*my own photo*). And it matches with the true label of this image:

- True Label = `[0., 0., 0., 0., 1., 0., 0., 0., 1., 0.]` ; **a male between [25 – 32] and/or under 38.** Here it is a photo of my 34 years old.

- **A case of misclassification:**

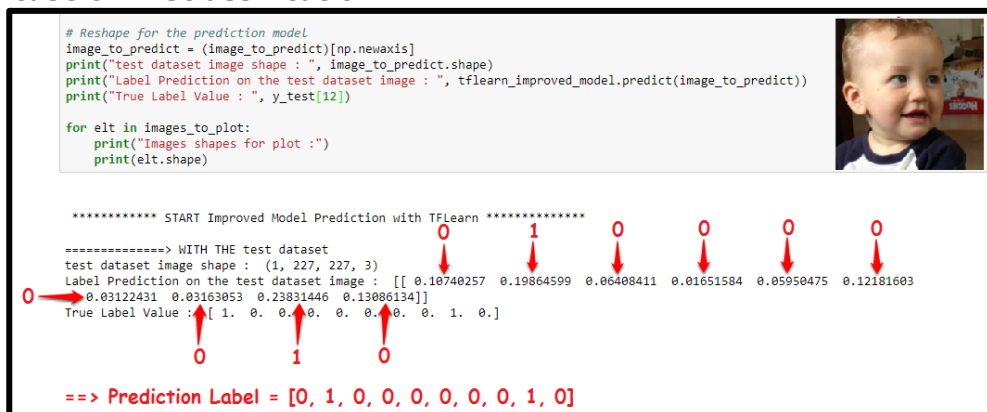


Figure 20: Label prediction with misclassification on an image from the test set

In this picture, the model made a prediction on an image taken from the test set. But this time it mismatches with the true label of this image in the test set: the gender of the image is correct (a male) while for the age, the predicted class is the class of age just after the true one.

- True Label = [1., 0., 0., 0., 0., 0., 0., 0., 1., 0.] ; **a male between [0 – 2] and/or under 4**. While the prediction gives a male between 4 and 6.

#### - On a non-human picture:

```
# Reshape for the prediction model
other_image = other_image[np.newaxis]
print("other_image shape : ", other_image.shape)
print("Label Prediction with another image: ", tflearn_improved_model.predict(other_image))
print("True Label Value : Here we have a picture of fishes")

=====> WITH a non-face image
other_image shape : (1, 227, 227, 3)
Label Prediction with another image: [[ 0.03521736  0.22467338  0.22530656  0.04183876  0.07477461  0.08766549
  0.01592951  0.00569209  0.1441547  0.14474757]]
True Label Value : Here we have a picture of fishes

==> Prediction Label = [0, 0, 1, 0, 0, 0, 0, 0, 0, 1]
```




Figure 21: Label prediction on my own face image

In this picture, the model made a prediction on an unseen image of a non-human, precisely on a picture of fishes. And the prediction label is : [0., 0., 1., 0., 0., 0., 0., 0., 0., 1.]. But here, as we put only the highest probabilistic value to 1, note that for the gender both values are very close (**0.1441547** and **0.14474757**).

So, the model predicts **a female between [8 – 12] and/or under 15**.

However, we know that it is not a human face picture there.

Surely, this guess of the model is done using the similarities between the fishes' picture characteristics and those it learned from human faces during the training process. More generally, the model should compare any non-human picture characteristics to those of the human images it learned, and then select the closest as result.

#### ==> The Output function for result interpretation

A result interpretation function is available for the OneHot Model. The others will follow according to the roadmap. Below an overview of that:

```
***** START Improved Model Prediction with TFLearn *****

=====> WITH THE test dataset
test dataset image shape : (1, 227, 227, 3)
Label Prediction on the test dataset image : [[0.0060229613445699215, 0.06429971754550934, 0.056978482753038406, 0.00266051851
21297836, 0.009781187400221825, 0.0019561832305043936, 0.0009656562469899654, 1.656175300013274e-05, 0.011111949570477009, 0.84
6206784248352]]
True Label Value : [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  1.]
Images shapes for plot :
(227, 227, 3)

In [48]: prediction_label = tflearn_improved_model.predict(image_to_predict)
interpret_label_prediction(prediction_label)

===== START PREDICTION INTERPRETATION =====

Predicted label in 2-hot format : [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  1.]
Predicted label interpretation : [0 1 0 0 0 0 0 0 0 1]
We have a female between 4-6 and/or under 8.

===== END OF PREDICTION INTERPRETATION =====
```




Figure 22: An example of the output function display for the OneHot model

The full code is available on my [GitHub](#), via the IPython notebooks.

## V.2- Reflection

In this project, we have successfully implemented an Age and Gender Classifier using the latest cutting-edge technology to reach and outperform the accuracies provided by other state-of-the-art methods. Indeed, we have been able to provide two models applied to a hand-crafted Unified Dataset, which perform well on this kind of task.

The challenge was first finding a suitable network model and a dataset to perform such work, and then fine-tune it to reach the expected precision. So we found the FCN useful because it



keeps all information about the input into the layers and so could help to better differentiate images.

The fine-tuning part was less evident because it needed more experimentations to fit the model even if there are previous works about fine-tuning methods. And these operations take a lot of time (*several hours or days of training on GPU*). That's to say, perhaps someone could beat our results through experimentation of some parameters. We expect more accuracy for the reliability of the application. So do not hesitate to share your contribution sending us a pull request, if you outperform it (your GitHub name will appear as a contributor).

Another important point to relate is that the frameworks (tools) we used to train and evaluate the dataset could have an influence on the result. For example, the use of a Tensorflow DNNClassifier or a Tensorflow Keras has different requirements. The dataset structure should be a dictionary for the DNNClassifier, and a list for the Keras branch. We have also seen how during the improvement, TFLearn has no *max-norm regularizer*, it is available only with Keras (including its Tensorflow's branch).

So it would be interesting to evaluate all the possibilities and compare what accurate improvement there is for each use case, and what specific tool use for a particular task. ***This shows how work with several frameworks helps in the choice of the best model. And Tensorflow, fortunately, integrates platforms such as TFLearn and Keras inside its framework.***

The final model we found has a pretty good accuracy, and it should be (improved and) used in a general setting to solve professional age and gender problems.

We also provided the required resources to allow anyone interested in, to turn this application into a mobile app. A demo will be available on my [GitHub](#).

### V.3- Improvement

We are proud of our Age and Gender Classifier Benchmark as it performs well and reaches our goals (*get or outperform the existing accuracies, in a unique step, using FCNs only*).

But, it always exists some other challenges to beat, especially concerning fraud detection in the field.

- Hand-crafted pictures with tools like Photoshop could lead to false age and gender detection.
- Real life cases of lifting, aesthetical surgery make it difficult for a human to find the real age of people.

Could technology overcome that?

How could we improve our model to take into account these aspects and make it more reliable for Identification and Authentication processes?

Perhaps, we could train the model to also make it learn with datasets which contain this kind of pictures or let work in parallel a network which solves this problem, so that both give a more confident result.

Anyway, future research work in the domain should be implied in solving this problem.

On the other hand, it would be worth to apply the same techniques we used for this project on other types of ANN such as GAN (Generative Adversarial Network), as it is a recent field always in exploration, mainly the Wasserstein GAN which produces accurate results.

And also propose a universal and easy method for deployment of notebook implementations on a Mobile App.

### V.4- Acknowledgments

We are very thankful to [Udacity](#) for this effective Machine Learning Engineer Program. We have learned a lot of things and ready to go and apply them to professional and personal objectives projects.

This capstone project allowed us to use tools provided by the MLND (Machine Learning Engineer Nanodegree Program) for the Machine Learning Process, especially:

Supervised Classification, Data Exploration (recreate a new Unified dataset using the Adience benchmark), Performance Metric (Loss and Accuracy Scores), Training and Testing Data Split (sklearn train\_test\_split tool), Analyzing Model Performance (Tensorboard and Training History Visualization), Optimization - Model Tuning (optimizer, activation, loss and regularization

functions, number of epoch, etc.), Training computational cost (Big-O complexities of common algorithms used in Computer Science), etc.

We also thank the authors of the research paper which inspired our work, [Age and gender classification using convolutional neural networks](#), and the [Open University of Israel](#) for the [Adience Benchmark](#) which has been very useful for us to perform this work.

## V.5- References

- 1- [Machine Learning Engineer Nanodegree Program](#) - UDACITY
- 2- [Age and gender classification using convolutional neural networks](#)  
(The research paper) | Gil Levi, Tal Hassner
- 3- The [Adience benchmark](#),  
a public Age and Gender Classification Dataset of the [Open University of Israel](#) (Face Image Project)
- 4- [Face Aging with Conditional Generative Adversarial Networks](#)  
| Grigory Antipov, Moez Baccouche, Jean-Luc Dugelay
- 5- <https://www.tensorflow.org>
- 6- <http://tflearn.org>
- 7- <https://keras.io>
- 8- [https://en.wikipedia.org/wiki/Loss\\_functions\\_for\\_classification](https://en.wikipedia.org/wiki/Loss_functions_for_classification)
- 9- <http://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/>
- 10- TFLearn, an example of FCN (in the Weights persistence part)
- 11- TFLearn, an example of Highway FCN
- 12- Tensorflow Keras, an example of FCN
- 13- TFLearn example of test-validation-monitors
- 14- TFLearn basic fine-tuning
- 15- [A Simple Way to Prevent Neural Networks from Overfitting](#)
- 16- TFLearn Data Augmentation
- 17- Tensorflow Data Augmentation
- 18- Keras Data Augmentation
- 19- Detailed Data Augmentation example with Keras 2.0 API
- 20- MoviePy
- 21- [Deploying a TensorFlow model to Android](#)
- 22- [Creating Custom Model For Android Using TensorFlow](#)
- 23- [TensorFlow for Poets 2: Optimize for Mobile](#)
- 24- [Full implementation code and documentation](#)

"Intellectuals solve problems, geniuses prevent them."

"The difference between stupidity and genius is that genius has its limits."

[Albert-Einstein](#)