

**ARTIFICIAL INTELLIGENCE PROGRAMMING PROJECT**

**Report 5 –Implementation   
and Analyzing**

– Hanoi, November 2021 –

**Table of Contents**

[V. Implementation and Analyzing 3](#_heading=h.30j0zll)

[1. Implementation 3](#_heading=h.1fob9te)

[1.1. Model selection 3](#_heading=h.3znysh7)

[1.2. Datasets preparation](#_heading=h.2et92p0) 4

[1.3. Model training](#_heading=h.tyjcwt) 6

[2. Analyzing](#_heading=h.3dy6vkm) 8

[2.1. Performance](#_heading=h.1t3h5sf) 7

[2.2. Areas of improvement](#_heading=h.2s8eyo1) 8

# V. Implementation and Analyzing

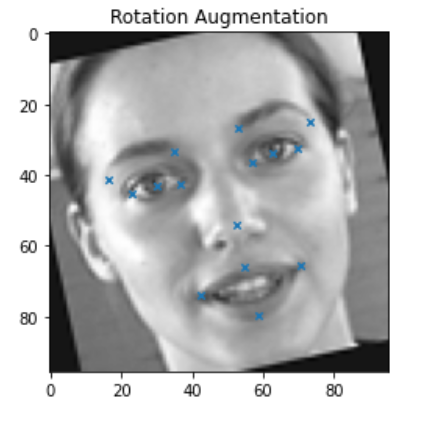
## 1. Implementation

### 1.1. Model selection

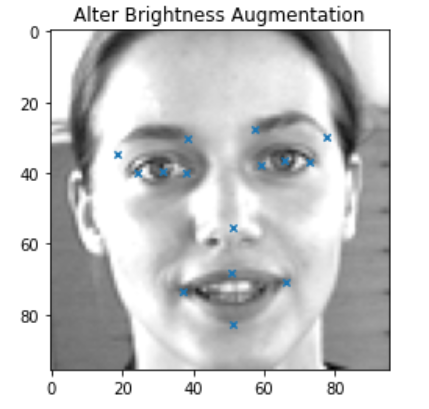
* For solving facial keypoints problems, first, we apply the Inception model and after that is a transfer learning model for improvement areas.

### 1.2. Datasets preparation

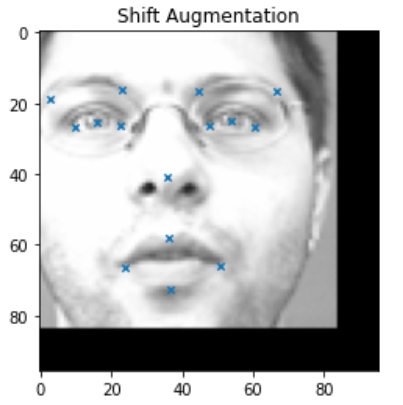
* There are 7049 images in the training dataset but the issue is there are 68% data that are null-value. Consequently, after analysing the normal distribution the data set will be categorized into two different data frame:
  + clean\_train\_file = train\_file.dropna() will be used for data augmentation
  + train\_file: replace the NaN entries with the distribution mean.
* After dropping the null-value, the data augmentation method will be applied using the clean\_train\_file to increase the number of training examples thus improving the model performance.
* Rotation augmentation: the images will be rotate with angle is set to be 12 so the rotation\_angle = [12]:



* Change brightness example:



* Shift augmentation: the images will be shift with the pixel\_shift = [12]

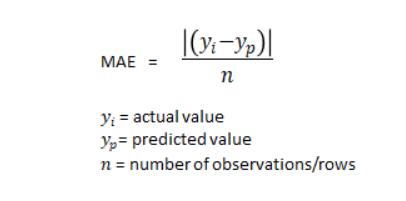


* After data augmentation, the training data will have 24099 images.

### 1.3. Model training

Our model has an architecture below with an special block “Inception-like” :

* **An inception-like block** : has 4 parameters respectively: [input, number of filter size [1,1], number of filter size [3,3], number of filter size [5,5] ] . After each filter is a Batch Normalization layer which helps normalize the filter input. The filters with size [1,1]; [3,3] or [5,5] are all fixed the size of the output compared to the input with parameter ‘ padding = “same” ’ and using the ReLU activation function to remove all values < 0. After filter and BatchNormalization layer is a Max pooling layer with filter size [2,2] and strides = [2,2], fixed output. Finally the second, the third BatchNormalization layer and the Max pooling layer are concatenated together in the Concatenate layer.
  + **Batch Normalization layer**: Batch normalization is a technique to standardize the inputs to a network, applied to either the activations of a prior layer or inputs directly. It can accelerate training, in some cases by halving the epochs or better, and provides some regularization, reducing generalization error.
  + **Parameter ‘ padding = “same” ’** : it applies padding to the input image so that the input image gets fully covered by the filter and specified stride.It is called SAME because, for stride 1 , the output will be the same as the input.
* **The input layer** : a 96x96x1 matrix corresponding to the width, height and channel of the photo.
* **Convolution layer 1:** an inception-like block with 4 parameters (input\_im, 64, 64, 32). It takes the input layer as input and it has: 64 filters size [1,1]; 64 filters size [3,3]; 32 filters size [5,5]. Output matrix is [96,96,97].
* **Max Pooling layer 1:** filter size [3,3] and strides = [2,2], fixed output. Output matrix is [48,48,97].
* **Convolution layer 2:** an inception-like block with 4 parameters (layer 1, 64, 64, 32). It takes the input layer as input and it has: 64 filters size [1,1]; 64 filters size [3,3]; 32 filters size [5,5].Output matrix is [48,48,193].
* **Max Pooling layer 2:** filter size [3,3] and strides = [2,2], fixed output. Output matrix is [24,24,193].
* **Convolution layer 3:** an inception-like block with 4 parameters (layer 2, 96, 96, 64). It takes the input layer as input and it has: 96 filters size [1,1]; 96 filters size [3,3]; 64 filters size [5,5]. Output matrix is [24,24,353].
* **Max Pooling layer 3:** filter size [3,3] and strides = [2,2], fixed output. Output matrix is [12,12,353].
* **Convolution layer 4:** an inception-like block with 4 parameters (layer 3, 96, 128, 64). It takes the input layer as input and it has: 96 filters size [1,1]; 128 filters size [3,3]; 64 filters size [5,5]. Output matrix is [12,12,545]
* **Max Pooling layer 4:** filter size [3,3] and strides = [2,2], fixed output. Output matrix is [6,6,545].
* **Convolution layer 5:** an inception-like block with 4 parameters (layer 4, 128, 256, 128). It takes the input layer as input and it has: 128 filters size [1,1]; 256 filters size [3,3]; 128 filters size [5,5]. Output matrix is [6,6,929]
* **Global Average Pooling layer:** a pooling operation designed to replace fully connected layers in classical CNNs. The idea is to generate one feature map for each corresponding category of the classification task in the last mlpconv layer. Instead of adding fully connected layers on top of the feature maps, only take the average of each feature map, and the resulting vector is fed directly into the [softmax](https://paperswithcode.com/method/softmax) layer. Therefore this layer is used to reduce the spatial dimensions of a three-dimensional tensor. Output matrix is [929].
* **Flatten layer:** The matrix in the previous floor will be spent flat so it changes the size of this floor into 929.
* **Dense layer:** A fully connected layer has 1024 units using the l2(weight decay) regularizer function = 0.02 applied to the kernel weights matrix.
* **Dropout**: This floor does not change the size of the previous floor but only impacts on the training process when randomly turns off the units of the previous floor with the RATE = 0.2. This is a technical in control (regularization) model to minimize overfitting and increase the accuracy of forecasting and training speed.
* **Output layer:** A fully connected layer has 30 units equivalent to the facial keypoints
* For the learning rate measurement and overfit, avoiding the callbacks function in Keras will be applied. A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.
  + **EarlyStopping**: monitors the performance of the model for every epoch on a held-out validation set during the training, and terminates the training conditional on the validation performance. In this function, we use validation loss as a performance measure to terminate the training with the mode = ‘min’
  + **ReduceLROnPlateau**: This callback monitors a quantity and if no improvement is seen for a 'patient' number of epochs, the learning rate is reduced. We chose the ‘val\_loss’ as a monitor for reducing the learning rate if the val\_loss of 5 epochs in a row does not improve with the parameter ‘patience=5’.
* For the metric used for evaluating the model, “MAE” will be used with the form:



## 2. Analyzing

### 2.1. Performance

* To evaluate the model’s performance, we compare loss and MAE (Mean Absolute Error) after a certain number of epochs (100). The curves of decreasing loss are shown as follows in Fig.1. Red solid curves represent training loss while navy dashed curves are validation loss. Both training loss and validation loss are decreased as the number of epochs increase.
* There is a problem that after 70 epochs, the result shows that the model is having the slight overfit problem.

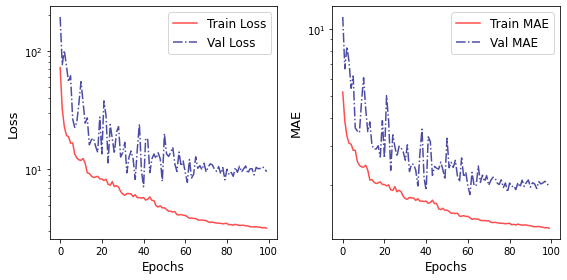


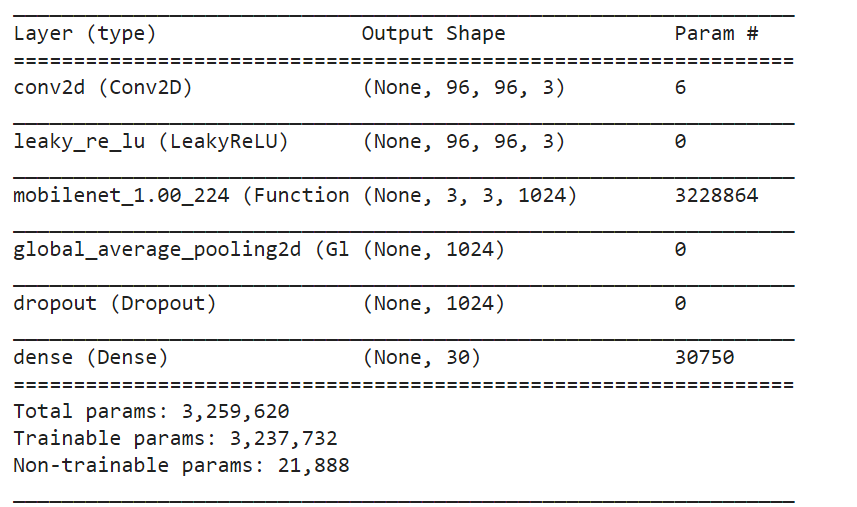
Fig 1:

### 2.2. Areas of improvement

* In the areas of improvement, we decided to use the Transfer learning method instead of using a large number of layers in the Interception-like model. This will decrease the number of epochs to train and reach the desired result, speeding less time than the traditional method.

**2.2.1. Model building**

* In model building, the model will still be based on the convolutional neural network, with the first layer being the input layer which is Conv2d. It takes 3 filters with the size [1,1] since the input image has the size 96x96. The padding parameter will be ‘same’ since we want to preserve the spatial dimensions of the volume such that the output volume size matches the input volume size.
* After the convolution layer, the LeakyReLu will be added to the model with the param alpha = 0.1
* Pretrained-model will be added after the LeakyReLu layer. The pretrained-model we chose is MobileNet has the input shape similar to the input image and truncated top layer through the parameter ‘include\_top=False’.
* After that, the GlobalAveragePooling2D() layer will be applied to the model in order to return the average of all the values from the portion of the image covered by the Kernel.
* The dropout layer will also be added to the model to deactivate some neural inorder to avoid overfit. The params ‘0.1’ represent the percent of neural deactivation.
* The dense layer represents the fully connected layer so that all of the unit of the previous layer has connected to the unit of this layer. The number ‘30’ represents that there are 30 units in this layer.
* Model summary:



**2.2.2 Performance of Transfer learning model**

* For fine tuning the parameters, 10 epochs will be compiled first and analysed before training with larger epochs. The train\_loss and the validation\_loss are decreasing as the number of epochs increase, so basically there is no sign of an overfitting phenomenon.
* After training 13 epochs the EarlyStopping was activated to avoid overfitting since 3 consecutive epochs do not improve the result. Overall, the Transfer-learning model result is better than the Interception block and reduces the epochs and training time. Furthermore, the gap between the validation set and training set is equal, so that the transfer-learning model operates stable.
* The final loss function is 2.11022 in the training set and 2.6815 on the validation set while the MAE on the training set is 1.0827 and 1.282 on the validation set.
* Model loss function and MAE of the model after 13 epochs of training:

