Age Estimation: A Convolutional Neural Network Based Regression Approach

Proposed to: Hamdi Dibeklioglu

Emre Doğan
Department of Computer Engineering
Bilkent University
Ankara, Turkey
emre.dogan@bilkent.edu.tr

Hamdi Alperen Çetin
Department of Computer Engineering
Bilkent University
Ankara, Turkey
alperen.cetin@bilkent.edu.tr

***Abstract*—This document is the technical report of CS559 Deep Learning Course Homework. The main purpose is to build a CNN (*Convolutional Neural Network*) based regression model to accurately estimate ages of people from their facial images.**

Keywords—convolutional neural networks, age estimation, regression

I. Introduction

In the last decade, Convolutional Neural Network (CNN) has achieved promising results in computer vision tasks due to its self-learning power and ability of working with very large data. In our task, we employ CNN models to estimate human ages from given their face images. Different types of CNN architectures are employed and compared in our study. Another important point regarding deep network models is the process of validating hyperparameters. We applied a set of values for each hyperparameter to result with the best model.

This paper is organized as follows. Section 2 describes the background and previous studies that we took advantage of. Section 3 gives brief information about our dataset. Section 4 describes our model. Section 5 gives the implementation details. In section 6, experiments regarding hyperparameters of the model are shared. Section 7 describes the performance of our finally tuned model. Finally, Section 8 presents our conclusions.

II. Dataset

With In our study, we used a modified version of UTKFace Dataset[1]. The original dataset includes samples of people with an age range from 0 to 116 years old. It also has some additional information such as age, gender, and ethnicity. But for our case, the dataset includes people aged in a range of 0 to 80 with no additional information.

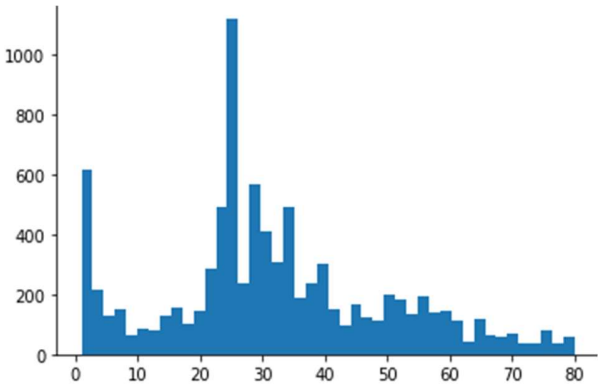


Figure 1. Histogram of Age Distribution in Our Data

Our dataset consists of 5400 training, 2315 validation and 1159 test samples. The histogram of age distribution of our dataset is available in Figure 1. While reading the dataset, we normalized images dividing them by 255 to get them in between 0 and 1.

III. Our Model

Through this homework, we implemented different architectures to observe differences in terms of performance. At the beginning we tried a basic structure as 3 convolutions, 1 max pool, 1 convolution, 1 max pool and 2 fully connected layers. The second convolution was 1x1. 1x1 convolution and the max pools was for decreasing the number of parameters. In the first experiments, we reached a MAE value between 7 and 8.

After some research, we changed our model. All the models start with convolutions then continue with fully connected layers. To keep the number of parameters in a reasonable range, we need to reduce the parameters before fully connected layers.

[1] http://aicip.eecs.utk.edu/wiki/UTKFace

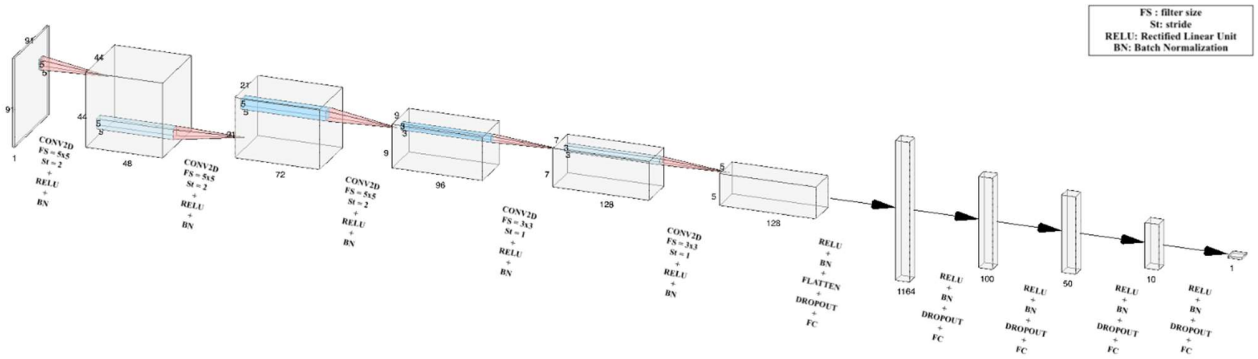


Figure 2. Structure of Our Model.

There are three different mostly used ways to decrease the number parameters: pooling, 1x1 convolution with less filters and convolution with an enough amount of stride.

For example, let's say we have a layer with 40x40 dimensions. Applying 2x2 max pooling with stride 2 or applying 8 filters 1x1 convolution with stride 1 will result with a 20x20x32 layer. Using 32 filters 5x5 convolution with stride 2 will produce a 19x19x32 layer. At the end, all of them will end up with a layer having less parameters.

In our model, we employ the last way by using 5x5 convolutions with stride 2 at the beginning and using 3x3 convolutions with stride 1 towards the end of convolutions. Figure 2 shows the overall architecture of our model. At the output of each convolution, there is ReLU activation.

In our first model, there is a max-pool layer after each con2d layer. But in this model, we do not use any max-pool operation at all. While investigating different studies on CNN architectures, we noticed that some studies do not use max-pooling layer, or any other pooling methods. In [1], Kuen et al. proposed a new idea '*DelugeNets*' in which information across many layers is propagated with greater flexibility compared to ResNets by using convolution with stride instead of pooling. Similarly, Springenberg et al.[2] question the necessity of different components in the pipeline and find that max-pooling can simply be replaced by a convolutional layer with increased stride without loss of accuracy. With an inspiration from these studies, we replaced pooling layers with convolution layers. By adjusting stride values of convolution layers, reduction of input size through the network can be done, similar to max-pooling.

Another discussion on the architecture is the location of Batch Normalization and ReLU layers. There are several discussion posts on this topic online. After reading all these discussions, we decided to use ReLU before Batch Normalization, just like in the original paper. We think that batch normalization before ReLU is not a very meaningful option. Because, conv2D layer may result with some negative features which should be truncated by ReLU. When ReLU is applied after batch normalization layer, these values cannot be truncated properly and these negative features will be forwarded through the next layers. Not to have this problem, we applied batch normalization after our activation function, ReLU.

After 5 convolutional layers, we added fully connected layers. Instead of using just a layer with a high number of nodes, we used multiple fully connected layers with a smoothly decreasing number of nodes. At the end, we had 5 fully connected layer as in Figure 2. Before each fully connected layer, there is a dropout layer. Dropout rate is explained in the Expreiments part.

We used the output value of the last fully connected layer to minimize the loss function. Then, we round and clip the output of the last fully connected layer for the values larger than 80 and smaller than 1 while calculating the train MAE and validation MAE.

IV. Experiments

As required in the description of homework, there are some methods & hyperparameters to explore so that we can improve our model. Effects of each improvement will be discussed in this section step by step.

In the MAE graphics, red color represents the train dataset and blue lines represent the validation dataset.

We used 12GB Tesla K80 GPU from Google Colaboratory[2] for our experiments.

A. Batch Normalization

Batch normalization is the idea of normalizing layer weights just like input values. Beyond its contribution to training speed, it helps to solve two important problems regarding deep networks:

1. Vanishing & Exploding Gradient:

Due to very large and very small weight values, the signal proceeding through the network may converge to 0 or diverge through infinity. By normalizing each layer, we avoid having such kind of a problem.

2. Covariance Shift:

It is simply the bias of the model through the distribution of input features. If an algorithm learns a mapping from X to Y, the

[2] https://colab.research.google.com/

model may fail when a new X with different distribution of features is tested.

Beyond these two problems, batch normalization also;

- reduces overfitting
- lets us to use higher learning rates
- allows each layer to learn more independent of other layers.

For our model, we conducted an experiment to observe the effect of batch normalization using 0.9 momentum for moving mean and moving variance while training. Other parameters are kept constant and given below so that the effect of batch normalization can be observed in a better way: learning rate 0.01, batch size 100, number of epochs 100, no drop-out or L2 regularization.

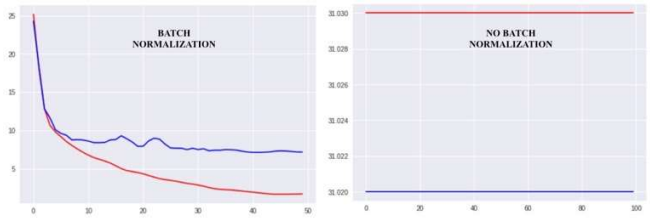


Figure 3. MAE Graphs of the model using batch normalization (left) and the model not using batch normalization (right)

Mean Average Error (MAE) values for models with and without batch normalization layers can be observed in Figure 3.

Without batch normalization, our model gives the same loss at each epoch. It is because the gradients are vanishing, and it predicts age '1' for all images. Indeed, the gradient is clearly vanishing but as we clipped all values smaller than 1 to 1, we are getting predictions of '1' for all instances. 31.030 and 31.020 are the exact mean absolute error for train and validation datasets when predicting all '1'.

B. Adam Optimizer & Early Stopping

In our model, we used Adam Optimizer. In this section, tuning process of its hyperparameters (*batch size, learning rate, number of iterations*) and early stopping method will be investigated.

Batch Size:

Batch size is an important choice depending on the memory size of our GPU, in our case 12 GB. When we tried to feed the whole training data at once, we got a memory explosion error. So that, we tried mini-batch approach with different batch sizes. Figure 4. MAE Values for Different Batch Sizes.illustrates MAE results of our experiments with different batch sizes. As expected, small batch sizes such as 10 was not a good fit for batch normalization approach. For batch normalization to affect the model in a good way, the number of samples must not be that small. We used batch sizes up to the value of 1350, that our memory can manage at a time, and observed that a batch size between 100 and 200 is a good choice for our final model. To be able to feed more instances at each time, we stick with 200 as our final batch size.

Learning Rate:

Learning rate is another important hyperparameter effecting the model's both speed and accuracy. For this reason, we started with a very small number (0.0001) and observed loss function as learning rate is increased. MAE values for different learning rate values can be found in Figure 5. MAE Values for Different Learning Rate Values.

For quick experiments, using learning rate 0.01 would be a good choice as it reaches a result in a faster way. But, for a more reliable learning process, we are using 0.001 as our final learning rate.

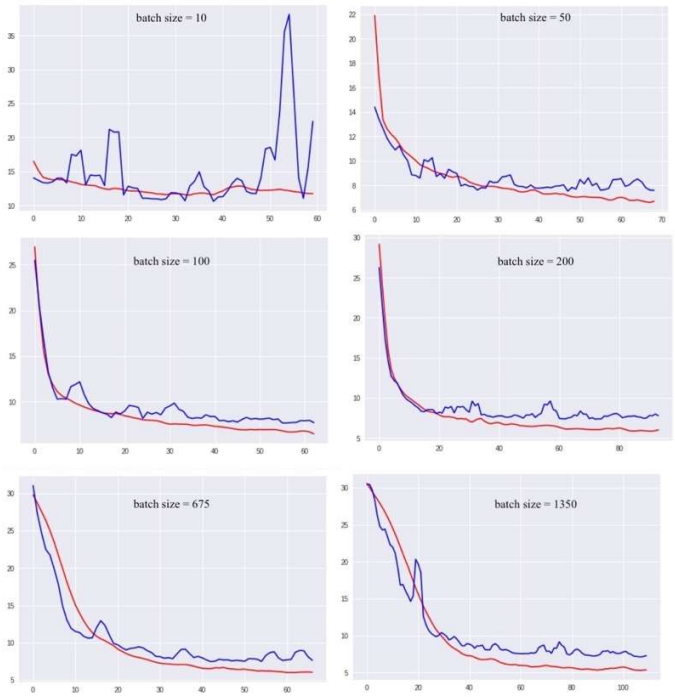


Figure 4. MAE Values for Different Batch Sizes.

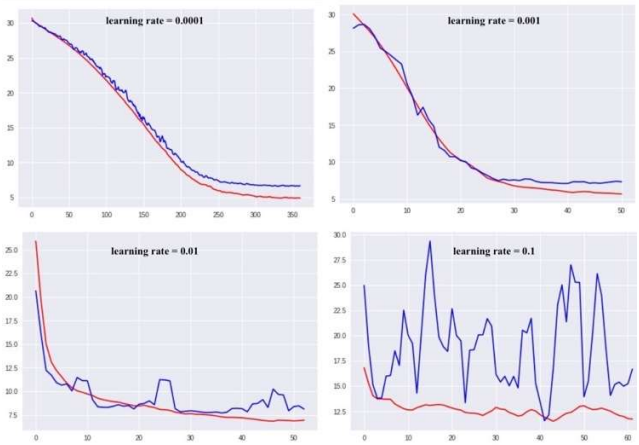


Figure 5. MAE Values for Different Learning Rate Values.

Number of Iterations & Early Stopping:

We implemented early stopping method manually. We set a threshold value for the number of epochs without finding a better validation MAE. When the number of epochs without a better loss reaches this threshold, the training process of the model terminates and saves the last seen best model. In case of not reaching this threshold, there is a parameter for the maximum number of epochs to terminate the model eventually.

The required number of iterations depends on the other hyperparameter selections. For example, it needs to be larger for a smaller learning rate. At this point, we are not deciding a number for iteration count. We will use a high number for the maximum number of epochs with a reasonable early stopping number. For the experiments we use 20 as the early stopping count, but for the final tuned model it needs to be higher such as 100.

C. Loss Function

To be able to make a controlled experiment successfully, other hyperparameters such as learning rate, batch size and methods such as regularization, batch normalization are kept constant.

As a loss function, we used Mean Absolute Error(MAE), Mean Square Error(MSE) and Huber Loss. The resulting error graphs for these 3 different losses are given in Figure 6. Comparison of errors 3 different loss functions used in our model:MAE (upper), MSE(middle), Huber Loss(lower).

In MSE, any error in estimation is punished with its square. In Huber, punishment is second order polynomial within a range. Beyond this range, punishment becomes linear. In MAE, the punishment is always linear.

After the experiments on loss function, MSE gave the best results. So, we decided to use MSE loss as we are already used to it from our previous machine learning knowledge

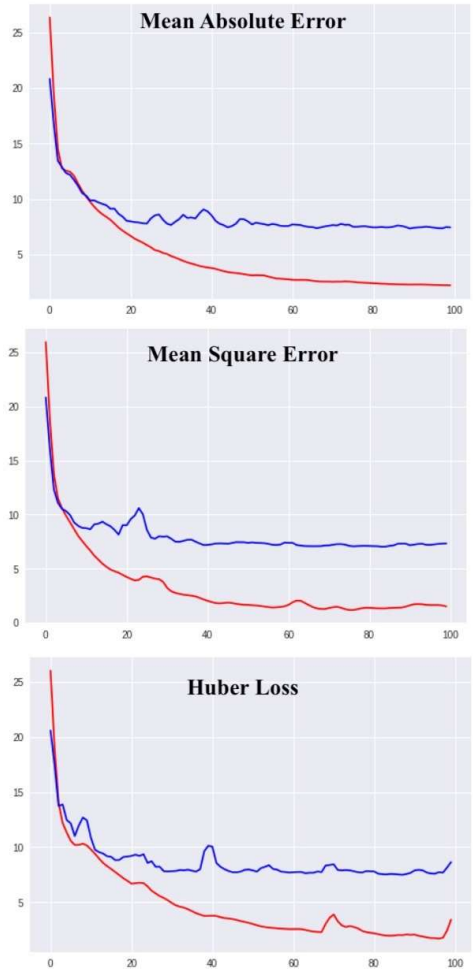


Figure 6. Comparison of errors 3 different loss functions used in our model:MAE (upper), MSE(middle), Huber Loss(lower).

D. Initialization Type

Initialization step can be seen as an easy and non-critical task. But a bad initialization might lead some serious problems such as vanishing and exploding gradients. For Gaussian initialization, the same problem is valid.

To solve this issue, Xavier initialization proposes to assign weights with random values that are not "too small" and not "too large". Xavier initialization tries to decrease the difference of variance of input & output weights. Training and validation MSE values are available in Figure 7.

Although Xavier seems a better option than Gaussian initialization, we could not observe an obvious difference between these two. So, we decided to continue with Tensorflow's default option, Xavier.

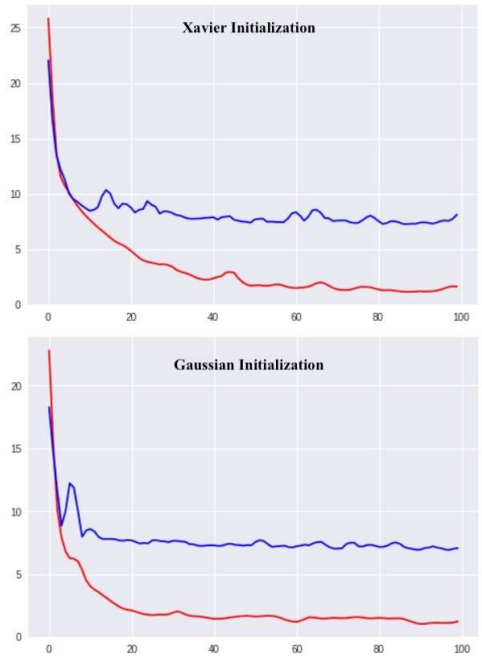


Figure 7. MAE Results for Different Initialization Methods.

E. Regularization (Dropout & L2)

- ***Dropout Regularization:***

Beyond its regularization effect, Dropout lets us combine different versions of our model, just like an ensemble learning.

Table 1. MAE Values for Different Dropout Keep Probabilities.

Keep Probability	Training MAE	Validation MAE
0.4	6.93	7.05
0.5	6.70	6.86
0.6	6.76	7.02
0.7	6.83	6.91
0.8	6.63	6.70
0.9	7.01	7.07

We set a dropout layer before each fully connected layer. Training and validation MAE values for different dropout keep probabilities are available in Table 1. MAE Values for Different Dropout Keep Probabilities.Experiment setup is as following: Loss type MSE, learning rate 0.01, batch size 100

The epoch-loss graphics of different dropout keep rates are in Figure 8. MAE Values for Different Dropout Keep Probabilities. The main difference between the previous model and the new model with dropout layers is that the validation loss is decreasing faster than the train loss. It is seen at every graphic in Figure 8. MAE Values for Different Dropout Keep Probabilities. But after some rate around 0.8, it resembles the previous model, because it more becomes the previous model when the keep rate increases.

In the experiments, 0.5 and the 0.8 gave the best loss values. But, 0.8 doesn't seem well on the graphics. So, we are stick with 0.5 dropout rate.

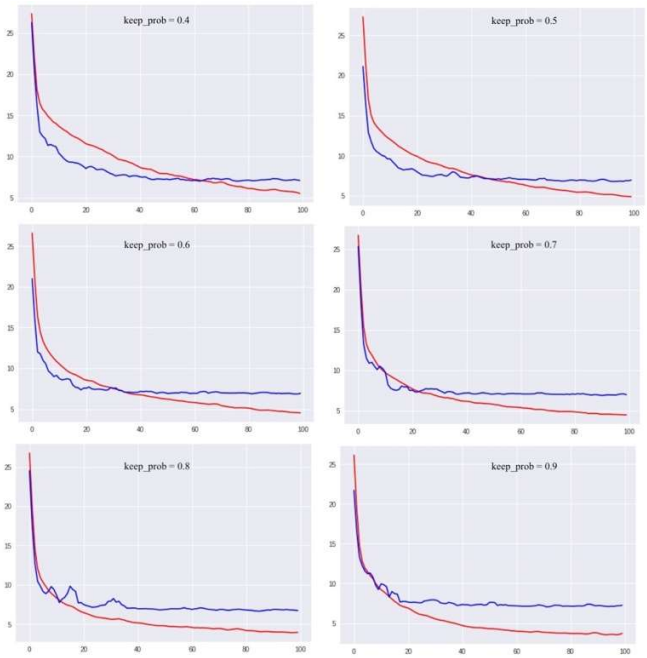


Figure 8. MAE Values for Different Dropout Keep Probabilities.

- ***L2 Regularization:***

L2 Regularization, also known as weight decay, is another way of overcoming the overfitting problem in deep neural network models by adding an extra term to the loss function. It forces the weights to decay towards zero, but not exactly zero.

For CNN models, L2 regularization must be tuned very carefully to take advantage of it. For this reason, we applied different values of λ as L2 regularization parameter to observe differences. MAE graphs through iterations for different λ values can be seen in Figure 9. MAE Values for Different L2 constants. Experiment setup is like following: loss type MSE, learning rate 0.01, batch size 100, dropout keep 0.8.

It is obvious that λ = 0 case results with a smoother MAE graph whereas L2 regularization leads some fluctuations through epochs, as expected from the addition to the loss function.

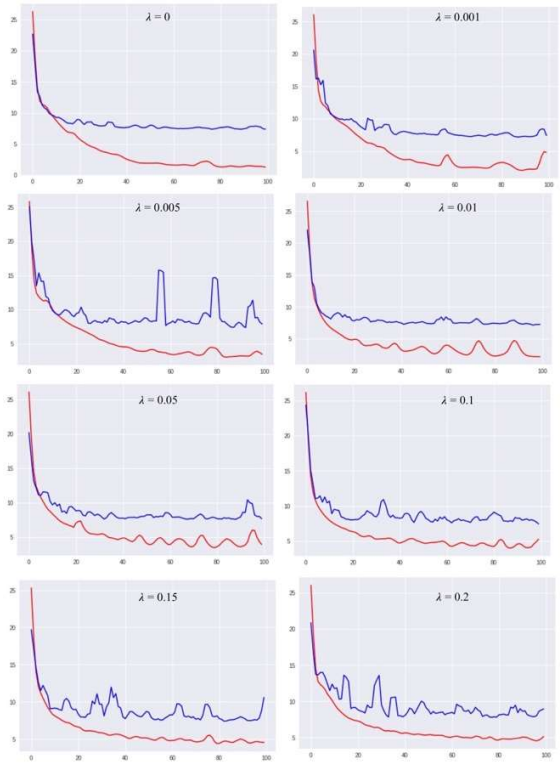


Figure 9. MAE Values for Different L2 constants.

L2 regularization helps to overcome the overfitting problem as can be seen from the graphs. Although there are some undesired fluctuations, the resulting MAE difference between training and validation data is becoming smaller meaning that overfitting does not occur at all. To prevent from overfitting problem, we used λ = 0.01 value in our final model.

V. Final Model and Test Results

After all the experiments, we started at some point to try our model changing hyperparameters a bit, and we ended up with the following parameter:

Loss function: MSE
Learning Rate: 0.0001
Dropout Keep Rate: 0.6
L2 Regularization: 0.01
Batch Size: 200

The learning rate and the dropout keep rate values are different then the values from the experiments section. When we try to use all the parameter all together, their behavior is changing. So, the values are needed to be fixed. The last version of the hyperparameters is as given above. The results of the final version of the model is shown in Figure 10.. The best validation loss was at the epoch 1972. **Validation MAE was 6.419, test MAE was 6.486.**

After tuning hyperparameters with experiments, we tried to linearly scale all images to have zero mean and unit variance before feeding the model with images. In Figure 11, there are graphics for the results with the standardized images. The best validation loss was at the epoch 1593. Validation MAE was 6.356 and test MAE 6.745.

The tensorflow graph of the model is in Appendix.

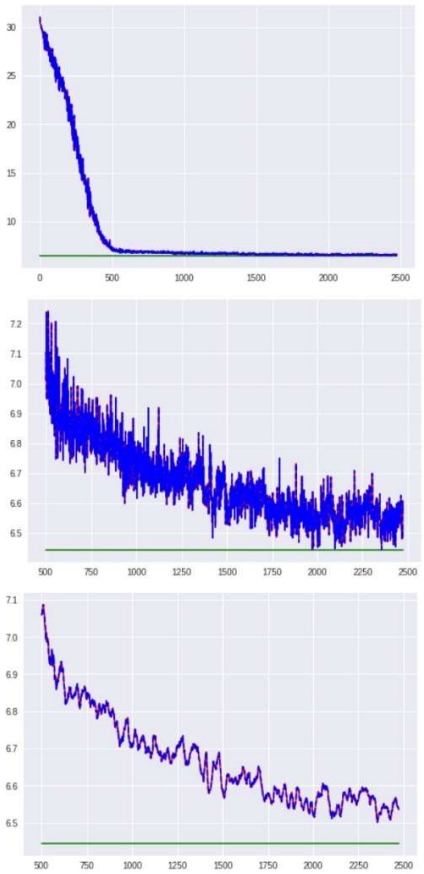


Figure 10. Last model without image standardization: all epochs (upper), epochs after 500(middle), smooth version of epochs after 500(lower).

VI. Tensorflow Functions

Here is the list all the functions that we used from tensorflow while implementing our model:

tf.placeholder()
tf.reshape()
tf.contrib.layers.xavier_initializer()
tf.initializers.truncated_normal()
tf.contrib.layers.l2_regularizer()
tf.contrib.layers.conv2d()
tf.layers.batch_normalization()
tf.contrib.layers.flatten()
tf.contrib.layers.dropout()
tf.contrib.layers.fully_connected()
tf.round()
tf.cast()
tf.clip_by_value()
tf.losses.absolute_difference()
tf.losses.mean_squared_error()
tf.get_collection()
tf.contrib.optimizer_v2.AdamOptimizer()
tf.group()

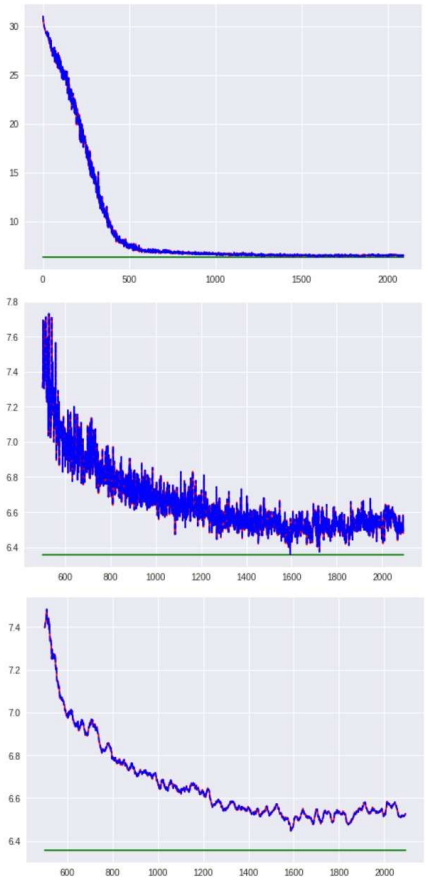


Figure 11. Last model with image standardization: all epochs (upper), epochs after 500(middle), smooth version of epochs after 500(lower).

VII. Conclusion

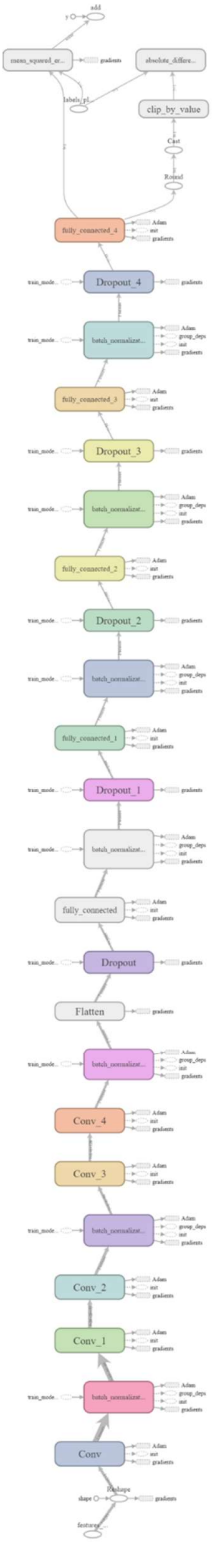
Consequently, we implemented a CNN based regression approach for age estimation problem. In our case, many different CNN architectures have been employed. Finally, we decided to implement a model consisting of all convolution layers and no pooling. In the experiments, different methods and hyperparameters were tuned as loss function MSE, learning rate 0.0001, dropout keep rate 0.6, L2 regularization scale 0.01 and batch size 200. At the end, we got validation **MAE 6.419** and test MAE **6.486**.

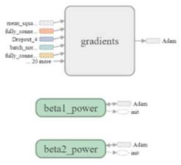
References

[1] J. Kuen, X. Kong, G. Wang, and Y. P. Tan, "DelugeNets: Deep Networks with Efficient and Flexible Cross-Layer Information Inflows," *Proc. - 2017 IEEE Int. Conf. Comput. Vis. Work. ICCVW 2017*, vol. 2018–January, pp. 958–966, 2018.

[2] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net," pp. 1–14, 2014.

VIII. APPE


add
y
mean_squared_er...
gradients
absolute_differe...
labels_pl
clip_by_value
Cast
Round
fully_connected_4
Adam
init
gradients
train_mode...
Dropout_4
batch_normalizat...
group_deps
fully_connected_3
Dropout_3
fully_connected_2
Dropout_2
fully_connected_1
Dropout_1
fully_connected
Dropout
Flatten
Conv_4
Conv_3
Conv_2
Conv_1
Conv
ExpandDims
shape
features


mean_squa...
fully_conne...
Dropout_4
batch_nor...
fully_conne...
20 more
gradients
Adam

beta1_power
Adam
init

beta2_power
Adam
init


Conv
beta1_power
beta2_power
gradients
batch_nor...
17 more
Adam
group_deps


Adam
batch_nor...
batch_nor...
batch_nor...
batch_nor...
3 more
group_de...


Conv
Conv_1
Conv_2
Conv_3
Conv_4
34 more
init


train_mod...
batch_nor...
batch_nor...
batch_nor...
Dropout
batch_nor...
7 more