Age Estimation from Facial Image

İdil Sülo
Department of Computer Engineering
Middle East Technical University
Ankara, TURKEY 06800
Email: idil.sulo@ceng.metu.edu.tr

***Abstract*—This paper presents a brief analysis of age estimation systems based on deep learning methods through a large dataset. Age Estimation systems are used to determine the age of the person in a given query image. This estimation is done by evaluating semantic contents of the query image. Because of the difficulty in revealing the semantic contents of the images, feature vectors are used as a higher level representataion to overcome this problem. By the help of these feature vectors, age estimation can be formulated as a learning problem which has the goal of matching the image representation with the age of the person in that image.**

***Index Terms*—Age Estimation, Deep Learning, PyTorch**

I. Introduction

At the basis of the languages we speak, how a person is addressed is mostly influenced by the characteristics that rely on gender and age of that person. For instance, an older person is often addressed more formally than a younger one. Furthermore, different words and grammar rules are reserved in languages to address different people. Through this phenomenon, the ability to estimate these individual traits from facial appearances lies at the heart of the social interactions of the people. As the roles of computers in daily life grows day by day, and the interaction between the computers and human increases, it is expected from the computerized systems to be able to do the same with similar accuracy and effortlessness. [1]

Given a face image, the goal of age estimation is to predict the actual age of the person in the image from the visual appearance of the face in that image. In this paper, we discuss this problem through deep learning techniques. [2] To evaluate the semantics of the images, features vectors are used as a higher level representation. Resnet18 is used to extract these feature vectors. Then, the feature vectors are given as input to a fully-connected neural network which is constructed with rectified linear unit (ReLU) as nonlinearity function in between layers. The neural network is trained with RMSprop optimizer, and mean squared error loss function is used to minimize the difference between the actual and the estimated age. Mean squared loss function is given as follows:

$$L = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

II. Implementation

The procedure that is used for the implementation of the age estimation system is as follows:

1) We first start by loading the feature vectors which are extracted using Resnet18 model. Each of the feature descriptors corresponds to a numpy array with size 512. Then, we also load the ground truth information, and convert both of them to tensors.
2) Next, we create a train and validation dataset by the help of these tensors, and create two dataloaders as *trainloader* and *validationloader*.
3) After this step, we define our neural network class by inheriting it from the *nn.Module*. We define the constructor, and the forward method to feed our inputs to the neural network. We also create a *net* variable which is an object of this class, and check whether a pickle file named *net.pkl* that consists of this *net* object exists in the same directory with the Python script. If there is such a file, then we load it to the *net* object and use it for computation afterwards.
4) There is indeed a need for a loss function to compute the difference between the prediction and the target. We define our *criterion* using MSELoss which is the mean squared error loss function. It is important to note that the lower the loss is, the better the model.
5) Optimization algorithms are also needed for our model to update the weight and bias values for the neural network. For this purpose, we choose the *optimizer* as RMSProp.
6) Then, we start looping over our dataset. For each epoch, we get the data with help of *trainloader*, and seperate this data as *inputs* and *labels*. Next, we make our labels with the same shape as the output. We zero the parameter *gradients*, compute our *outputs* with the help of our *net* object, compute the loss, and call *loss.backward()* to backpropagate. By backpropagating, we simply calculate the gradient of the error function with respect to the neural network's weights. The neural network can be seen as a graph, and the *backward()* function computes the sum of gradients of given tensors with respect to the graph leaves. Lastly, we call *optimizer.step()* to update the parameters. We also print the running loss at each epoch to have an idea about the drop of the MSELoss values.
7) Then for each data in *validationlader*, we simply extract the predicted age value, and save into a numpy array. We also save this numpy array as a file for further use.

III. Experiments and Results

Through this section, we present our observations in the sense of a controlled experiment where the set of hyperparameters that we keep constant through the experiment is shown with the example structure given below:

shuffle = True
number of epochs = 100
number of hidden layers = 2
learning rate = 0.0001
momentum = 0.8
batchsize = 50

A. Hyperparameters

1) Shuffle Parameter: For our experiments, we have started experimenting with one layered neural network model. First, we set the *shuffle* parameter for the *dataloader* to *True* for train dataset, and we discovered that when we increase the number of epochs, accuracy of the train does not increase in a consistent way. The data values for this experiment is shown below:

TABLE I
Train Accuracy with *shuffle=True*

Number of epochs	Train Accuracy
100	0.3948
200	0.4
300	0.395
400	0.4066
500	0.4016

Since this *shuffle* parameter, reshuffles the data at every epoch when it is set to *True*, we observed that the accuracy values changed in a misleading way. Therefore, we decided to set this parameter to *False* for the rest of our experiments.

2) Number of Epochs: In our experiments, we observed that the number of passes done through the training set, effects the accuracy values. One pass of the whole data set is called an epoch. Therefore, different number of epochs lead to different accuracy values, and in general, when the number of epochs increases, the accuracy of the predictions done by the neural networks increases as well. This can be shown by the table below with the parameters that we keep constant throughout the experiment:

shuffle = False
number of hidden layers = 0
learning rate = 0.0001
momentum = 0.8
batchsize = 50

TABLE II
Train Accuracy, Validation Accuracy and Number of Epochs

Number of epochs	Train Accuracy	Validation Accuracy
100	0.5942	0.56
200	0.6328	0.577
300	0.6310	0.5775
400	0.6472	0.5775
500	0.6494	0.58

3) Learning Rate: Learning rate is one of the most important hyperparameters for training deep neural networks. In the graph shown below, we observe changes in loss for 4 different learning rate values. It can also be seen that the model does not improve for the learning rate values 0.1 and 0.01, since for these values, there is no change in the MSE loss for different number of epochs.

shuffle = False
momentum = 0.9
batchsize = 64

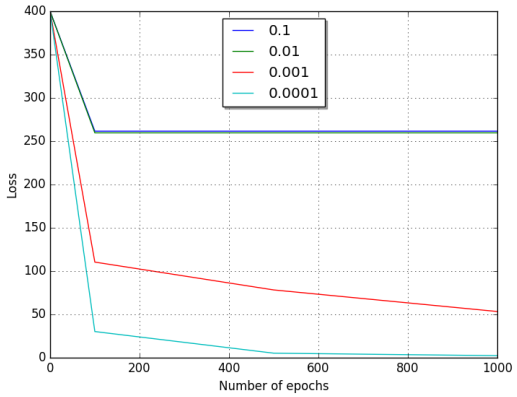

Fig. 1. Change in loss value for each learning rate

From the above graph, the best configuration for the learning rate is 0.0001 because of the fact that the MSE loss value converges to minimum for this learning rate. Therefore, 0.1 and 0.01 are high learning rates, and 0.0001 is a good learning rate. It should also be noted that the train and validation accuracy values do not change for the learning rates 0.1 and 0.01. This can be seen with the table below:

TABLE III
LEARNING RATES, NUMBER OF EPOCHS, TRAIN ACCURACY AND VALIDATION ACCURACY

Learning rate	Number of epochs	Train Accuracy	Validation Accuracy
0.1	100	0.3864	0.3665
0.1	500	0.3864	0.3665
0.1	1000	0.3864	0.3665
0.01	100	0.3864	0.3665
0.01	500	0.3864	0.3665
0.01	1000	0.3864	0.3665
0.001	100	0.6648	0.5795
0.001	500	0.7214	0.575
0.001	1000	0.7958	0.5845

By considering the increase in the accuracy values in this table, we can summarize that decrease in the learning rate results in better accuracy values for both train and validation set. Also, higher learning rates result in a point where the model does not improve anymore, and get stuck in the same loss and accuracy values.

4) Batch Size: Batch size is another hyperparameter that both effects the accuracy and the performance of the neural network. We can define batch size as the number of samples that are going to be used through training in order to make one update to the model parameters which is the number of samples per batch to load. For *torch.utils.DataLoader*, this parameter is set to 1 by default. Below, we present a graph showing the experiments with 3 different batch size values and a table showing the train and validation accuracies.

$$\textit{shuffle} = \text{False}$$
$$\textit{number of hidden layers} = 2$$
$$\textit{learning rate} = 0.0001$$
$$\textit{momentum} = 0.9$$

TABLE IV
LEARNING RATES, NUMBER OF EPOCHS, TRAIN ACCURACY AND VALIDATION ACCURACY

Batch size	Number of epochs	Train Accuracy	Validation Accuracy
16	100	0.6604	0.4945
16	500	0.994	0.5965
16	1000	0.9986	0.5865
32	100	0.7824	0.5825
32	500	0.9392	0.544
32	1000	0.9696	0.5845
64	100	0.8236	0.572
64	500	0.9364	0.5735
64	1000	0.9434	0.5705

From these experiments, we observed one important thing which is that the lower MSE loss values do not always lead to higher validation accuracy values. Although lower MSE loss values can be seen with small batch sizes, there is a huge gap in between the validation accuracies. For instance, when the batch size is 16, MSE loss for training converges to 1.804 for 1000 epochs, but the corresponding train and validation accuracies are calculated as 0.9986 and 0.5865, respectively. When the batch size is set to 64, MSE loss converges to 4.688 for 1000 epochs, and the corresponding train and validation accuracies are 0.9434 and 0.5705 which means the accuracy values increase when the batch size decreases. However, the opposite situation is observed for 100 epochs. For the batch size of 16, corresponding train and validation accuracies are calculated as 0.6604 and 0.4945, but for the batch size of 32 these values are observed as 0.7824 and 0.5825.

So, when we observe that MSE loss converges to a much more smaller value for a model with a given batch size, we cannot directly say that we will also observe higher accuracies for that model since the other hyperparameters that we mention through our experiments also affect these results in relation with each other.

Another important point about the choice of batch size is that it has a huge affect on the performance of the model. We can summarize the change in performance with respect to each batch size value by the table below:

TABLE V
PERFORMANCE AT EACH EPOCH VERSUS BATCH SIZE

Batch size	Time
16	1.470sec
32	0.613sec
64	0.45sec

By looking the data given above, we can conclude that the performance drops when the batch size decreases.

5) Number of Layers: Through the experiments, it is observed that changing the number of layers also effects the accuracy and the performance of the predictions done by the neural network. First of all, as we increase the number of layers, performance of the neural network drops significantly. For instance, the computation of the loss at each epoch takes approximately 0,0626 seconds when the number of layers is equal to 0, whereas the same computation takes approximately 0,3515 seconds for the case with 3 hidden layers. In other words, training a neural network with 3 hidden layers takes 6 times more time compared to a neural network with 0 hidden layers.

Secondly, the decrease in the mean squared loss value changes with the number of layers used in the model. It is known that lower MSE loss values leads to a better trained model. From the graph given below, it can be observed that when we increase the number of layers, the final MSE loss that we get decreases. Therefore, the higher number of layers result in a better trained model.

$$\textit{shuffle} = \text{False}$$
$$\textit{number of epochs} = 500$$
$$\textit{learning rate} = 0.0001$$
$$\textit{momentum} = 0.8$$
$$\textit{batchsize} = 50$$

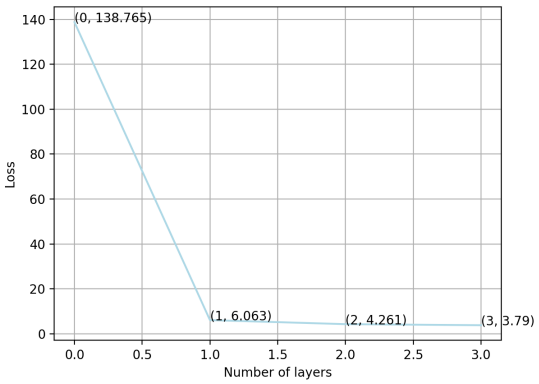

Fig. 2. MSE Loss value for each number of layers

In the plot below, we also present a comparison of the number of layers and the corresonding train and validation accuracies.

shuffle = False
number of epochs = 300
learning rate = 0.0001
momentum = 0.8
batchsize = 50

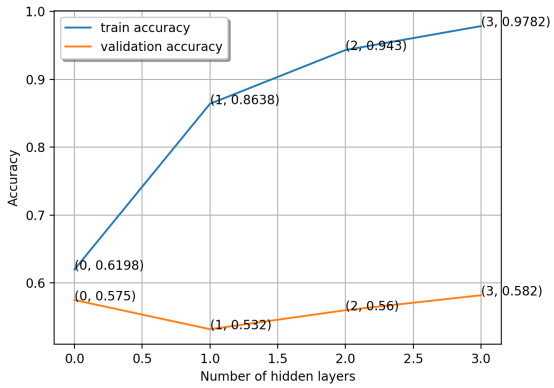

Fig. 3. Train and validation accuracies for each number of layers

From this graph, we can make the observation that an increase in the number of layers result in better train accuracies and better validation accuracies, in general. The best configuration of layer numbers is 3 hidden layers for this neural network.

Aside, we should note that the chosen batch size is not the best one for this model since it is good practice to choose the batch size as a power of 2.

6) Layer Size: Layer size is another factor which affects the accuracy of the neural network model. Below we present a table to compare the MSE loss value, train accuracy and validation accuracy with respect to different layer size configurations with the corresponding hyperparameter structure.

shuffle = False
number of epochs = 100
number of hidden layers = 3
learning rate = 0.0001
momentum = 0.8
batchsize = 50

TABLE VI
EFFECT OF DIFFERENT LAYER SIZES ON ACCURACY VALUES

Layer pattern	Train accuracy	Validation accuracy
512,256,32,16	0.7042	0.545
512,256,64,32	0.7768	0.556
512,256,128,64	0.788	0.561

From the table above, it can be observed that a uniform distribution of the layer sizes affects the accuracy of the neural network in a positive way for both validation and train accuracy.

B. Training and Validation Loss History

Below we present a plot for a moderate configuration of the neural network model with the corresponding hyperparameter structure:

shuffle = False
number of epochs = 1000
number of hidden layers = 3
learning rate = 0.0001
momentum = 0.9
batchsize = 50

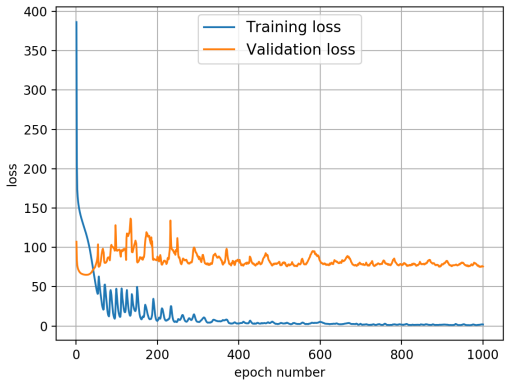

Fig. 4. Change in validation and train loss value at each epoch

From this plot, it can be diagnosed that both the train and the validation loss decreases and stabilizes around the same point. Therefore, our model is an example of a good fit where the performance of the model is good on both the train and validation sets.

C. Best, Moderate and Worst Estimates for Validation Set

In this subsection, we present our best, moderate and worst estimates for the validation set with their corresponding train accuracies and hyperparameter configurations.

1) Best Estimate for Validation Set:

shuffle = False
number of epochs = 10000
number of hidden layers = 3
learning rate = 0.0001
momentum = 0.8
batchsize = 50

Train Accuracy: 1.0
Validation Accuracy: 0.624

2) Moderate Estimate for Validation Set:

shuffle = False
number of epochs = 400
number of hidden layers = 2
learning rate = 0.0001
momentum = 0.8
batchsize = 50

Train Accuracy: 0.9738
Validation Accuracy: 0.5820

3) Worst Estimate for Validation Set:

shuffle = False
number of epochs = 100
number of hidden layers = 2
learning rate = 0.1
momentum = 0.9
batchsize = 64

Train Accuracy: 0.3864
Validation Accuracy: 0.3665

D. Analysis of Self Portrait

In this subsection, we experimented with a image from real life. The experiment consists of two parts which are predicting the age of the same person with and without glasses on. The images that were used are given below:

Fig. 5. Image of the person with glasses

Fig. 6. Images of the person without glasses

Our model predicts the age of both images as 36.3664 for both images whereas the actual age of the person is 21. So, there is a difference between the predicted and the actual ages.

IV. Conclusion

In this paper we have observed the implementation and the execution of a deep neural network with RMSProp optimizer and mean squared error loss function. Through the experiments, we have seen that several hyperparameters affect the accuracy of the neural network. We have examined the changes in the accuracy and MSE loss values with respect to parameters like number of layers, number of epochs, learning rate and batch size.

References

[1] E. Eidinger, R. Enbar and T. Hassner, "Age and Gender Estimation of Unfiltered Faces," in IEEE Transactions on Information Forensics and Security, vol. 9, no. 12, pp. 2170-2179, Dec. 2014. doi: 10.1109/TIFS.2014.2359646

[2] Xing, J., Li, K., Hu, W., Yuan, C., & Ling, H. (2017). Diagnosing deep learning models for high accuracy age estimation from a single image. Pattern Recognition, 66, 106-116. doi:10.1016/j.patcog.2017.01.005