Age Estimation from Facial Image

Ismail Hakki Kocdemir, *Computer Engineering Department, METU*

Abstract—In this report, I present my try to estimate age of an individual from facial images using features extracted by pretrained ResNet18 network. For this, I built several architectures with different number of hidden layers and different number of hidden neurons, and trained them up to their best as much as I could; in order to demonstrate the effects of those numbers on accuracy performance. I also had the best performer estimate my age from several pictures with different angles and occluding accessories.

Index Terms—computer vision, age estimation, neural network

1 Introduction

GUESSING the people's age from their facial features is a difficult task for us, humans, partially because there are not enough indicators in our face (or there are ones we can not catch) that changes in correlation with each years passing, or because the look of the face might be inconsistent with the age by a considerable margin, or simply because you can be trying to be polite. Except for being polite, those reasons that make estimation difficult are likely to be valid when training a neural network to do it. Hence, throughout this report, I will explain the steps to my attempt to tackle the problem with a simple set of neural networks and try to answer the questions of whether the look is good enough indicator that can outperform the human performance, assuming that the features extracted with pretrained ResNet18 has sufficient capability for that.

2 Neural Networks

A neural network is basically a mathematical function that transforms a given input into a an output in a desired form. It is hierarchically structured in a sense that each layer in the network transforms the previous layer's output by multiplying them with a weight set and feed it to the next layer. A layer consists of neural units that calculates an output from N dimensional data in previous layer into a real number. The number of units in a layer determines the dimensionality of the input to the next layers' units. Without any addition, this way of transformation is only linear, which is not suitable for problems requiring complex non-linear relation between input and the output. Non-linearity is realized by so-called activation functions applied to each neuron's output, which also acts as a threshold for firing of the neuron given the inputs.

As shown in [1] and [2] feed forward neural networks are universal approximators. It is shown that, with 2 layers, they can represent any continuous function; and with 3 layers, they have the capacity to represent any arbitrary function given the right set of weights.

We find those weights by propagating the error in the output of the network trough the layers backwards by taking chain of derivatives to see the effect of the each parameter on the final error, and changing their value accordingly so that error decreases.

In this simple scenario, as the universal approximation theorem states, going deeper with the number of layers does not bring considerable benefits. However, with more complex problems that requires recognition of really abstract features on top of each other, from an image that is just a bunch of numbers for pixel values (such as recognizing a car from different angles); deeper networks with special structures(e.g. CNN's) helps with the convergence to correct set of parameters, by providing a better starting point for the network to stack its transformations that detects more complex structures (structures of structures) as it goes deeper.

3 Problem and Methodology

As mentioned before, in this problem, I try to estimate an individual's age from a facial image using feed forward, fully-connected neural networks. To do that, we need to be able to detect the face in the image and extract the indicators in the face that can contribute to a better estimation. This is really hard to do from scratch requiring great ability of abstraction. Hence, we use a version of ResNet[3], which is a deep convolutional neural network architecture proposed by Microsoft Research team and won the ImageNet object detection competition in 2015, in order to directly jump to looking for relation between age and facial features embedded in the outputs of the pretrained ResNet18 on a given image.

Since we have the inputs extracted from the images in the given dataset and the ground truth values, which are actual ages of the people in the images, what we left with is comping up with a best feed forward network architecture and train it on the dataset. For this, I tested 4 architecture choices in terms of number of hidden layers: No hidden layer with direct mapping from features to age; and 1,2 and 3 hidden layers. For each hidden layer, I experimented with different values for number of hidden neurons.

As optimization method, I used mini-batch gradient descent with RMSprop. RMSprop adapts the learning rate for each parameter based on their past gradients (Pytorch implementation extends this with momentum). I have trained all architectures with several set of optimization parameter settings including learning rate, momentum and regularization. Additionally, I have made use of early stopping by

watching the improvement of the model on the whole data dataset (using both validation loss function and zero-one loss with the threshold given in the problem description) after every epoch to prevent overfitting and not bother with finding optimum number of epochs. I have made intermediate saves over improved accuracies. I set patience for stopping to 30 epochs.

4 Experiments and Results

Here, I will present the details of the each experiments and results with given dataset. I will start with 0 hidden layer, go up to 3 hidden layer and finish with self-portrait analysis. Before demonstrating the results I would like to restate that the objective function while training was $MeanSquareError$ (MSE) between target values and outputs. For accuracy, we are taking the ratio of outputs within 10 radius of the target value, to all outputs. To summarize, from no hidden layer to 1 hidden layer, I have seen significant improvement. After 1 hidden layer, with each extra layer, accuracy improved much less. MSE Loss rather showed small increases with extra layers.

4.1 No Hidden Layers

Since there was no option for number of neurons, I simply trained several networks with the following hyperparameters which results in 27 experiments in total :

Learning rate: $1e{-}2, 1e{-}4, 1e{-}7$
Momentum: $0, 0.5, 0.9$
Regularization:$1e{-}1, 1e{-}3, 1e{-}5$

The results were really close where no network performed significantly better than the rest. As seen in figure 1, with patience of 30 epochs, it took 95 epochs to finish the training with no sign of obvious overfitting (maybe a little bit towards the end) to reach accuracy of 59.3% and 197.71 MSE loss. All other loss histories showed similar patterns apart from the different fluctuations stemming from the stochasticity in mini-batches used in the training. One thing I must mention here as well is that, smaller fluctuations with validation data are because of the larger batch size I used in the validation set. This difference is not present in the experiments with more layers since I used closer batch sizes for both. To clarify how close the results are, the lowest performer model obtained 58.8% accuracy and all validation accuracies fluctuated in very similar range.

4.2 One Hidden Layer

With one hidden layer, I obtained significantly better accuracies. I have tried following values resulting in total of 72 experiments:

Hidden layer dimension: $[512], [1024]$
Learning rate: $1e{-}2, 1e{-}4, 1e{-}7$
Momentum: $0, 0.5, 0.7, 0.9$
Regularization:$1e{-}1, 1e{-}3, 1e{-}5$

With addition of non-linearity, accuracy climbed to 64%'s. In table 1, we can see that best models with both

Fig. 1: Vertical axis: MSE, Horizontal axis: Number of epochs. Loss history of best model with no hidden layers: 59.3%.

1024 and 512 hidden sizes produces very close accuracies and MSE losses. Also in figure 3 and 2 we can see loss for validation and training batches go hand in hand except for the number of epochs. The model with 1024 neurons gets its last improvement around 50th epoch while the other one gets around 40th epoch given the 30 epoch patience for early stop. This might be because of more number of parameters slowing down the convergence. We can also see the slight over-fit after some point close to where improvements stop.

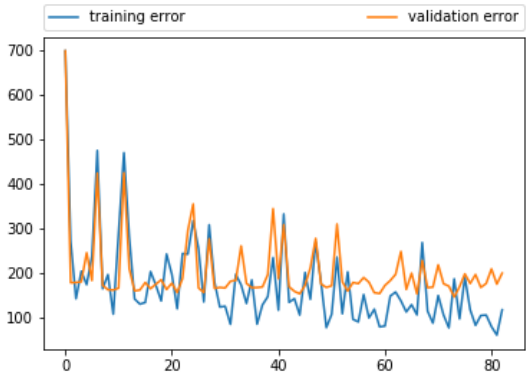

Fig. 2: Vertical axis: MSE, Horizontal axis: Number of epochs. Loss history of best model with 1 hidden layer(1024 neurons): 64.5%.

4.3 Two Hidden Layers

Two hidden layers did not make much difference in terms of accuracy and MSE Loss on validation set. Here is the hyper-parameter set I have tried:

Hidden layers: $[512, 256], [512, 512], [1024, 512]$
Learning rate: $1e{-}2, 1e{-}4, 1e{-}7$
Momentum: $0, 0.5, 0.7, 0.9$
Regularization:$1e{-}1, 1e{-}3, 1e{-}5$

As it was the case that I mentioned between 1 and 0 hidden layer networks, 2 hidden layer networks generally

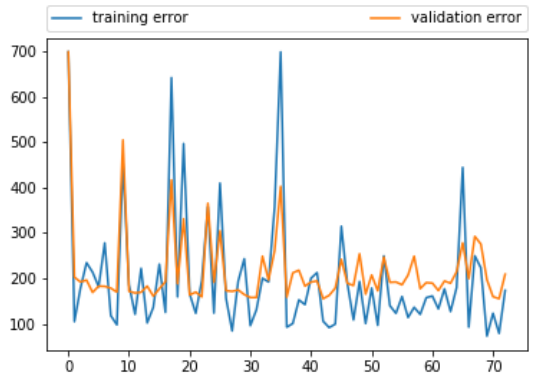

Fig. 3: Vertical axis: MSE, Horizontal axis: Number of epochs. Loss history of best model with 1 hidden layer (512 neurons): 64.3%.

took longer time to converge (More than 100 epochs.). The loss patterns can be seen in the figures 4 and 4. Again as table 1 suggests, accuracies are too close.

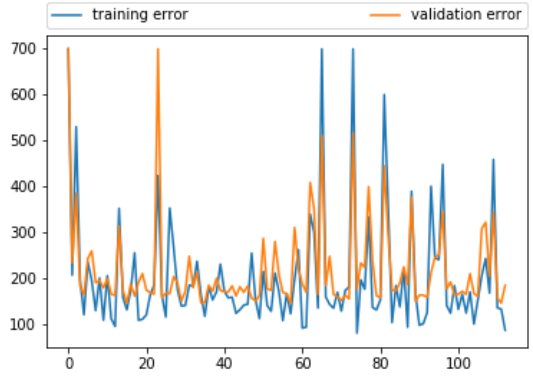

Fig. 4: Vertical axis: MSE, Horizontal axis: Number of epochs. Loss history of 2 hidden layer model ($512 - 512$, $1e-4$ learning rate, no momentum): 64.25%.

4.4 Three Hidden Layers

Three hidden layers resulted in slightly better accuracies but worse validation MSE loss values. I have experimented with the following settings:

Hidden layers: $[512, 256, 256]$, $[1024, 512, 256]$ $[1024, 1024, 512]$
Learning rate: $1e{-}2, 1e{-}4, 1e{-}5, 1e{-}7$
Momentum: $0, 0.5, 0.7, 0.9$
Regularization:$1e{-}1, 1e{-}3, 1e{-}5$

As can be seen in figures 6 and 7, losses fluctuate around the same values as in previous architectures. Same figures also suggest that last improvements take place at around epochs between 60 and 70.

4.5 Discussion

I want to immediately make clear that I saved loss history from mini-batches which caused too big fluctuations

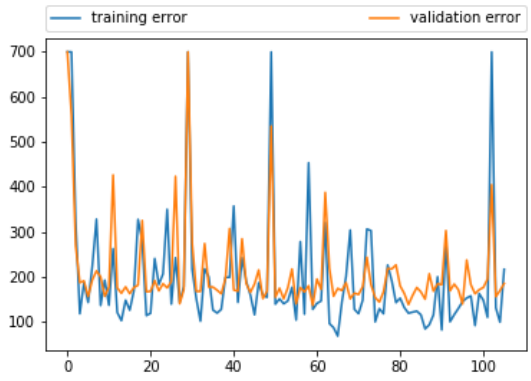

Fig. 5: Vertical axis: MSE, Horizontal axis: Number of epochs. Loss history of 2 hidden layer model ($512 - 512$, $1e-7$ learning rate, 0.5 momentum): 64.6%.

Fig. 6: Vertical axis: MSE, Horizontal axis: Number of epochs. Loss history of 3 hidden layer model ($1024 - 512 - 256$, $1e-4$ learning rate, No momentum): 65.1%.

in plots that made seeing the pattern in the learning difficult, or maybe even made the plots pointless. I wanted to change it when I noticed it, but it was late and I wanted to keep it as is for the sake of being consistent.

Thus, please see the extra example I did just to demonstrate this in figure 8 where the history of losses is extracted on full dataset. According to verbose output, model ceases to improve at epoch 35 in accuracy and epoch 30 in MSE loss; and training stops at epoch 65. We can see consistent decreasing trend in the training loss and slight curvature with a minimum around epoch 30 in the validation loss after which over-fitting starts (though not so obvious). It was really rare that a network improved after around 10 consecutive epoch without improvements, so probably a patience value of 10 would have been a better option/sufficient for this problem.

Other than that issue, the intuition behind the worst performance of no hidden layer networks with relatively large difference might be that without non-linearity, models' capacity are very limited and they easily converge to their global minimum for the objective regardless of the hyper-parameters. Without being sure, if I had not used a robust optimizer as RMSprop that automatically adapts the learn-

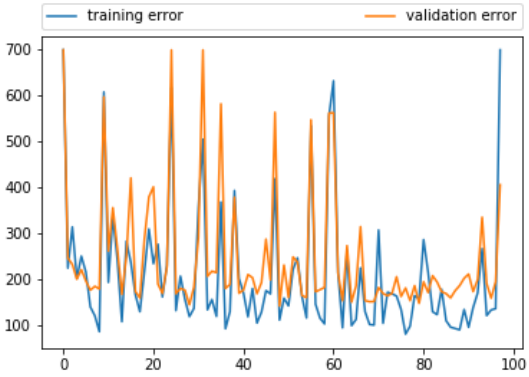

Fig. 7: Vertical axis: MSE, Horizontal axis: Number of epochs. Loss history of 3 hidden layer model ($512 - 245 - 256$, $1e-7$ learning rate, 0.9 momentum): 65.05%.

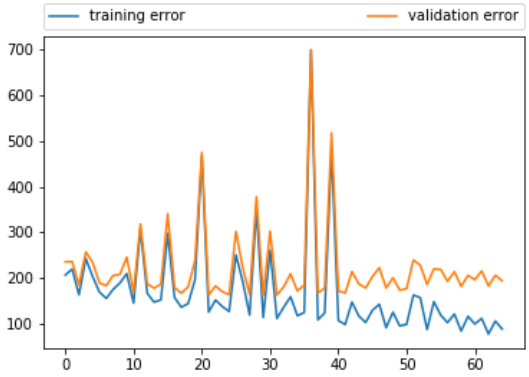

Fig. 8: Vertical axis: MSE, Horizontal axis: Number of epochs. Loss history of 1 hidden layer model (512, $1e-4$ learning rate, No momentum): 64.2%.

TABLE 1: Samples of models with best accuracies and MSE Losses.

No	Hidden Size	MSE Loss	Accuracy
1	1024	162.89	0.645
2	512	164.16	0.643
3	512-512	172.26	0.646
4	512-512	165.76	0.6425
5	1024-512-256	175.86	0.651
6	512-256-256	170.79	0.6505

ing rate and such, I could have encountered with issues on convergence to that minimum.

I want to also point out that MSE losses of 1 hidden layer networks in table 1 are two smallest I got among the all models, meaning that more layers produced slightly larger losses, although accuracies improved. Nonetheless, I am not sure if the differences are significant enough to relate this increase to the number of layers.

Secondly, we are also uninformed of the distribution in the square losses that each model produces, except for the mean. Hence, although it can be insightful, we can not simply choose the best model on MSE loss. Closer to that insight about distribution, we have our accuracy. But since their accuracies are so close; it is hard to rely on them. For that reason, I tend to declare the winner according to squared loss more than accuracy in our case.

As a side note, I did not try the extreme choices in all those network architecture and hyper-parameter to see how we can deliberately under-fit or over-fit the data. I rather stayed within the limits of common settings that I observed in on-line community and aimed for settings with relatively better results while experimenting.

4.6 Self Portrait Analysis

I have tested 6 networks in table 1 with 4 pictures of myself as can be seen in figure 9. On picture (a), interestingly, all networks guessed my age to be above 46 even though I assumed before that it would turn out to be youngest since it looks cleanest without beard. I guess, in training set, age is not correlated with the beard that much (results with pictures (c) and (d) agrees on that with their lower estimations) and I also look a little white

(a) $46.34, 47.69, 49.94$
$50.35, 55.75, 50.35$

(b) $34.62, 30.25, 35.84$
$36.95, 33.87, 36.95$

(c) $43.28, 47.09, 47.49$
$45.19, 33.56, 45.19$

(d) $28.03, 26.12, 30.95$
$27.38, 28.22, 27.38$

Fig. 9: Ages produced by best networks. Order same as in table 1.

haired due to lightning. I find guesses in picture (b) to be reasonable enough for each network because of the wrinkles and misleading style of the glasses that makes my eyes look like a grandfather's.

As for picture (c), when compared with (a) results would be inconsistent if my reference were human judgment. However, when compared with (b), general increase in all estimations are reasonable since I look mature and older due to more wrinkles in the forehead, receding hairline etc.

In picture (d), all models estimates to be the youngest among four. This might be because of more healthy skin (reddish) and plenty of clearly seen brown-blond hair. I think any real person would not disagree with estimations in (d).

In general, given the accuracy around 64.5% with the provided metric, results are not very surprising. As a sign of

consistency, we see that all of the networks' estimations are in the same order from youngest to oldest: (d), (b), (c), (a); with one exception: network 5 estimates (c) to be younger than (b) (by a large margin compared to others).

5 Conclusion

In this report, I have tested several architectures of neural networks to estimate age of a person from the features extracted by ResNet18 network. I could train networks that correctly estimated 65% of people within 10 radius of their correct age. I found more 1,2 and 3 hidden layers to be almost equal performers when trained properly. Non linear model were also be able to find some correlations but loss values and accuracy were worse compared to the other architectures with hidden layer. All in all, it is hard to say if the not-so-good results are due to that features from ResNet are not good encoders of facial features, or that results are at the limit of what facial features can tell about the age, or that I could not make us of those features at a 100%.

References

[1] Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.

[2] Kurt Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks", Neural Networks, 4(2), 251257.

[3] He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2015). Deep Residual Learning for Image Recognition. 7.