Age Estimation By Looking At Face Features

Ertugrul Kara - 2099125

Abstract— **In this assignment, I aimed to estimate a person's age by only looking at a single image of the person. Firstly, we extract features from this image. Then the extracted face features fed into a feed-forward network to output an estimate of age of the person.**

I. INTRODUCTION

Nowadays computer vision is in our lives and using deep learning methods in computer vision problems such as face recognition, image retrieval, image classification, landmark recognition etc. is common over traditional methods. This comes from the tremendous success of deep learning in recent years with novel developments. Age estimation is one of the problems that waits to be solved and using deep learning to do so is not a bad idea. Knowing age of a person would make our job easier when processing images. To give a simple example, when a person takes a selfie, if we knew her age then the camera could apply filters and enhancements more accurately.// To solve age estimation problem there are couple of options. First one that comes to mind is using end-to-end learning which probably makes use of convolutional neural networks and let the network handle the feature extraction, regression itself. But in this one, we will use Resnet18 pretrained network to extract features from the image. Resnet18 network is a residual convolutional network and one can see from the filters of a convolutional network that, first layers of the network generally learns to recognize edges -aka interest points- etc. When we pass face images through our pretrained resnet18 network, with the high success at image classification of resnet, we can say and hope that discriminative features from the faces will be extracted.
After this step, rest of the problem is similar to decoder part of the autoencoder method. We have feature vector instead of the image itself, which is efficient to play with compared to images, we can train a network to estimate age of the person appearing in image by looking at the features. I have conducted sets of experiments and tried different architectures by changing hyper-parameters and empirically tested which would be more accurate and tried to reason why it is so.
For feature extraction and training, testing the decoder part, we are using PyTorch library. Citations of the libraries that I have used to calculate the results can be found in the references section.

II. Architecture Experiments

I have tested 4 basis models which conducted more experiments on top of them. They are 0 hidden layers, that is there is no hidden layer between face features and output. There is only 1 weight matrix. Others are 1, 2 and 3 hidden layer versions.
I have also experimented with different layer sizes, batch sizes, learning rates. I have used RMSprop optimizer with momentum and L2 regularization. Because adding momentum to optimizer, allows us to -hopefully- overcome local minima's and makes it easier to reach lower local minima's. Adding momentum increases the accuracy of the model and actually this optimizer somewhat converges to ADAM optimizer. And adding L2 regularization is good practice and have it's benefits to generalize. Also, L2 regularization term is also a hyper-parameter to tune. I have tested several different values such as 1e-5, 1e-4, 1e-3.
I have implemented early stopping and model checkpointing. When validation loss is improved, current model is saved. Also when validation loss does not improve over some number of epochs, that I am stopping training to prevent overfitting. As a loss function we are using mean squared error loss and as an non-linearity function between layers, we are using rectified linear unit which is proven to be successful when compared to other non linearity functions such as sigmoid. I have also kept track of the loss history for both train and validation sets during training. I thought this is a good idea because otherwise it would be hard to understand what is actually going on or if the model started to overfit etc.
First parameters that I have tried is setting a overall limit to my experiments by setting parameters set apart from each other and see which end performed better. I have set learning rate to 1e-2, 1e-4 and 1e-5. batch size to 16, 32 and 128. Set the layer size to 20, 200 and

1000. Therefore for the first round, I had 90 different setups. Setting hidden layer size to 3 and layer size to 1000 leads to overfitting very fast. Since our features are in the shape [1, 512] for a single image, it might not be a good idea to put this many neurons for estimation.

From my experiments and history plots, I can clearly see that decreasing batch size increases the fluctuation in the confidence of network. So for the rest of my experiments I will choose batch size of greater than 64 would be perfect. From my previous experience and knowing that mini-batch implementation is converges anyways, I choose batchsize of 96 for speeding up a little and prevent fluctuation. Detailed graphs can be found in my IPython notebook.

As we can see from this figure, when layersize is

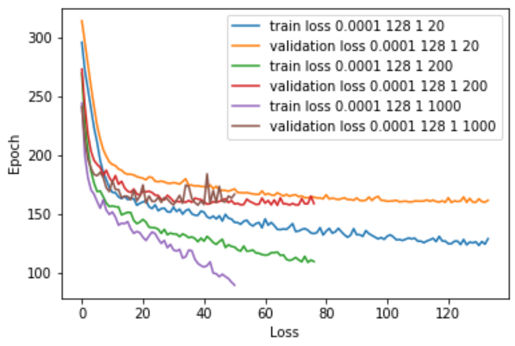

Fig. 1. 1 Layer setup with different number of layer sizes and loss plot. Setup: LearningRate: 1e-4, BatchSize: 128, LayerNumber:1, LayerSize: 20, 200 and 1000

1000, network overfits to data. other approaches looks similar.

A. *Layer Number: 0*

This model did not improved well even when I set the early stopping to 20 epochs. Best I have achieved on validation set is MSE loss of 168. So, I moved on to 1 layer option.

B. *Layer Number: 1*

For smaller layer sizes that I have tried such as 32, 128; network could net reached the accuracy of 1024 setup. Best accuracy for this experiment is reached by 1024 neurons with learning rate of 1e-5 with validation loss of 153.. And validation loss of 512 neurons was 154. With 64 neurons however, best loss I could get was 160. Therefore, for the next experiments, I've decided to set first layer size high. Also, Learning rate of 1e-5

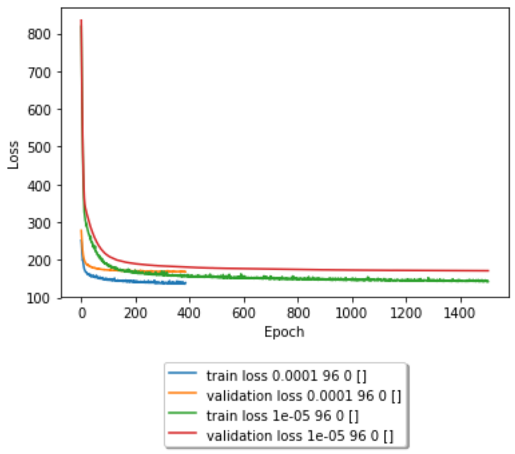

Fig. 2. 0 Layer setup with loss plot. Setup: LearningRate: 1e-4, 1e-5, BatchSize: 96, LayerNumber:0, LayerSize: 0

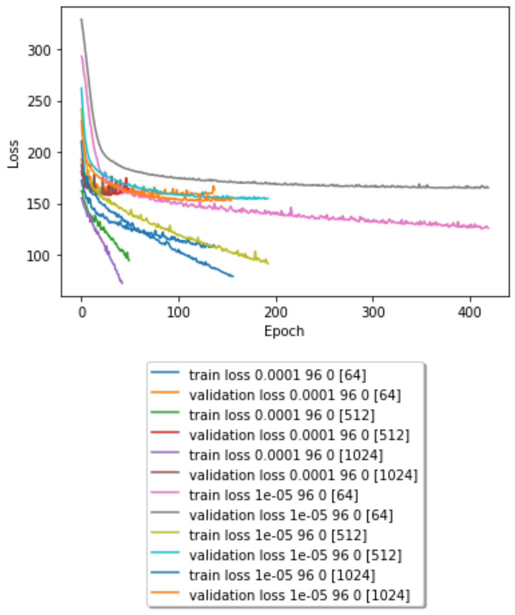

Fig. 3. 1 Layer setup with loss plot. Setup: LearningRate: 1e-4, 1e-5, BatchSize: 96, LayerNumber:1, LayerSize: 64, 512 and 1024

seems to reach lower loss values so I sticked to that with trading time.

C. *Layer Number: 2*

Best result that I could get was with 512 size at first layer and 64 at the second hidden layer.
When I try to interpret the results, decreasing the layer size when going deeper seems to give better results. Having too low layer size in the first layer gives poor results. Having a large number of neurons in the second hidden layer also seems to cripple the accuracy. I have done separate tests that I did not added the graphs here

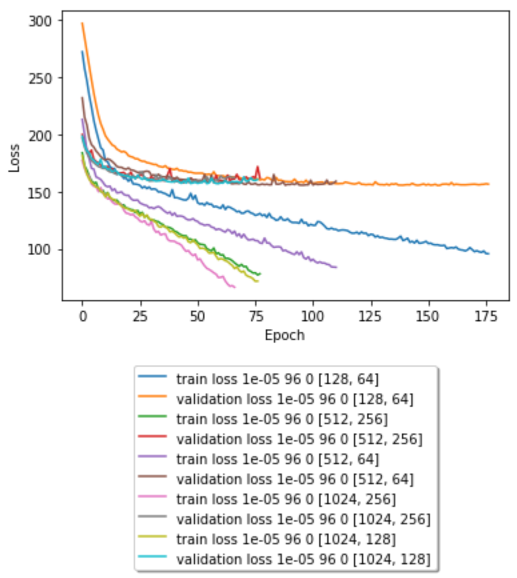

Fig. 4. 2 Layer setup with loss plot. Setup: LearningRate: 1e-5, BatchSize: 96

that can be found in IPython Notebook, that agrees with my findings. Having large size in first layer increases performance.

D. Layer Number: 3

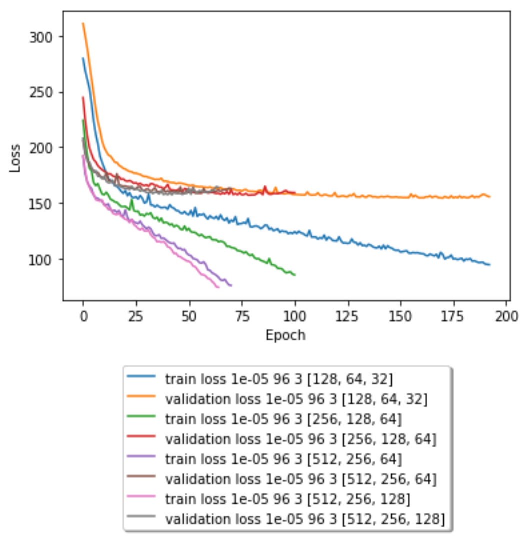

Fig. 5. 2 Layer setup with loss plot. Setup: LearningRate: 1e-5, BatchSize: 96, LayerSizes: [128, 64, 32], [256, 128, 64], [512, 256, 64], [512, 256, 128]

For first experiments with 3 layers, I have tested with lower number of sizes. For the first layer, I have tested sizes of 128, 256 and 512. Best loss value I have got is surprisingly with the setting of 128 at the first layer and 64 and 32 for the others. This had loss of 154.5.
When I have increased the total number of neurons, the loss increased. But I wanted to further investigate this and tested with 1024 neurons in the first hidden layers.

For another set of experiments -I have excluded the graphics because it was becoming a long paper-, I have 1024 size at the first layer. Best loss I have got is with 64 at the second layer and 32 at the last. This is similar to the previous test. With this setup, I had loss value of 156.1.
Also, from the graphs, we can see that with increasing number of neurons, network overfits within 50 epochs. Actually, in this set of experiments, the only setup that does not overfits is the one with 1024, 64, 32 size.
Therefore, having this many training data and 512 features at the input layer, since our data is not excessive, it might be a better idea to not increase layer sizes too much.

E. Final Network

After having an idea about the fit network for this problem, I have tried several setups. As I have guessed,

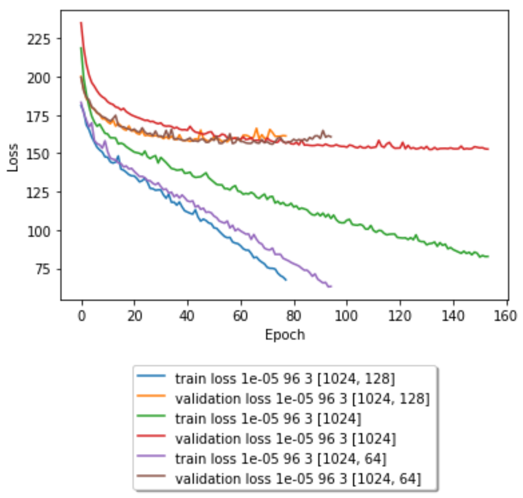

Fig. 6. 2 Layer setup with loss plot. Setup: LearningRate: 1e-5, BatchSize: 96, LayerSizes: [1024, 128], [1024], [1024, 64]

adding more and more does not produce better results. So I will stick with the network with 1 layer with size 1024. Since our input is 512 wide, it would be enough to grasp the idea from the input and reason with it. With 1 layer and size 1024, I had 152.45 loss while with size 512, I had 154.28 loss.

III. My Self Portrait

For the final experimenting, I have tested my network with 3 pictures of me. One of it was plain me. One was with my glasses and the last one with my sunglasses. Luckily, I have a system to compare the results. My phone is Xiaomi Mi 5s and it has age estimation embedded in default camera app. While taking my pictures ages found by my phone respectively was; 33, None, None. My phone was not able to detect my face with glasses.
However, the network I have trained found my ages from the pictures respectively; 29.4, 30.6, 27.4. This seems accurate except the version with glasses since I am 22 years old. When I wear my glasses, everyone agrees that I look older. So, I am somewhat satisfied with the results however I would expect higher accuracy overall rather than 62.75 percent. And I believe this may be caused by the feature extraction progress. Features extracted from people with sunglasses could be younger -as it is expected, younger people wear glasses that is more like as mine.- And people with normal glasses are probably older so the network learned to classify these features as older.

If I should discuss my best networks best, moderate and worst estimations; the worst is validation image number 43. My network classifies it to be 36 years old but the ground truth is 111. I believe this is a noisy classification data. This may be included to prevent overfitting. The best guess is for validation image number 1061. The ground truth is 31 years old and the estimation of my network is 33.997. This is the regular picture of a man. To discuss a moderate guess, I have chose a estimation with error of 28.4. My network estimated the age of validation image with index 1997 as 34.6 years old. The ground truth is 63 and this is clearly a not correct estimation. I believe this could be because of hat, colour hair (which mostly exists in younger people) and microphone. These occlusions led to broken estimation.

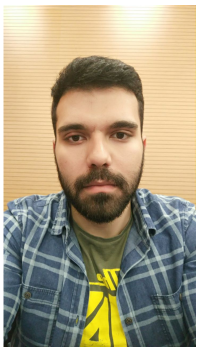
Fig. 7. Me

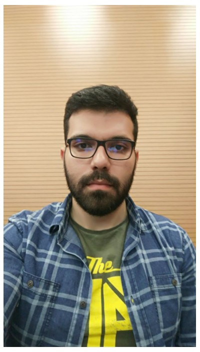
Fig. 8. Me with glasses

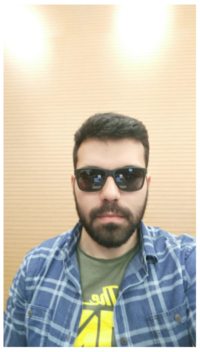
Fig. 9. Me with sunglasses

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. CVPR 2016.

[2] Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam. Automatic differentiation in PyTorch. NIPS-W 2017.