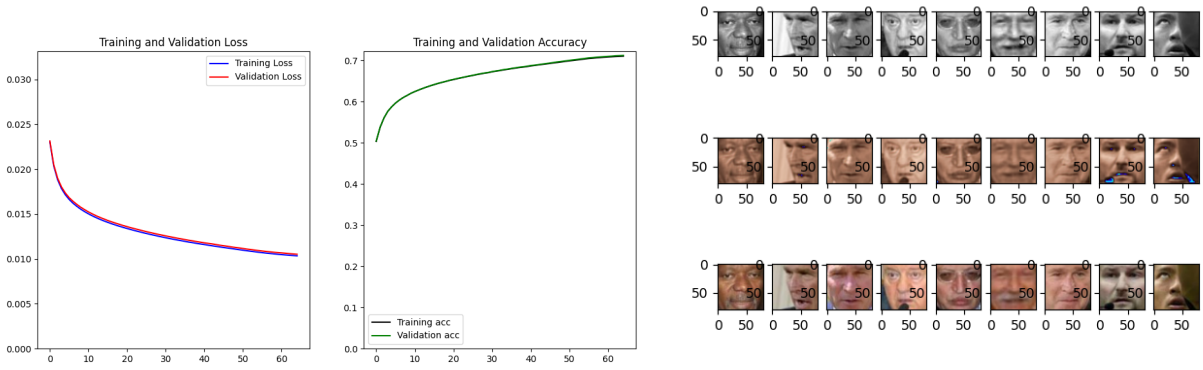
Face Colorization with CNN Based Architectures

Alperen Bitirgen

1 Baseline Architecture

It would be useful to try to create a controlled experimental environment in order to more clearly examine the effect of the parameters to use. And for this, I set default values for the model. The things I paid attention to when choosing the default values were that I wanted the model not to be too deep or shallow, and to have as large an area as possible for the convolution operations. In this way, I planned that I could have a good enough core value and that I would improve our model by making step-by-step changes on this value. In our investigation, I used the default architecture as 2 convolution layers, kernel size of 5, kernel count of 4, and learning rate of 0.0070. Then by changing the each value step by step only and keeping the rest unchanged, I discussed the effect of each parameter below:

Description/Change	#conv_layer	kernel_size	#kernel	learning_rate	#epoch	loss	accuracy
default	2	5	4	0.0070	65	0.01052	0.71211
#conv_layer	4	5	4	0.0070	98	0.01008	0.71619
kernel_size	2	3	4	0.0070	41	0.01067	0.70657
#kernel	2	5	8	0.0070	49	0.01012	0.72307
learning_rate(large)	2	5	4	0.0700	100	0.00837	**0.75520**
learning_rate(small)	2	5	4	0.0007	80	0.01535	0.63207
our preference	2	5	8	0.0700	51	0.00838	**0.75580**



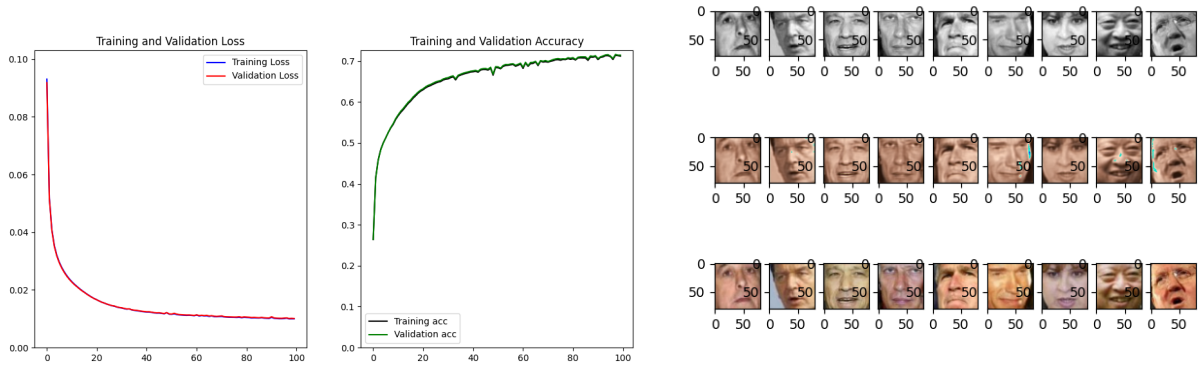
(a) Loss and accuracy tracking on training and validation data (b) Model input, prediction and ground

Figure 1: Using default parameters.

1.1 Effect of the number of convolution layers

Description/Change	#conv_layer	kernel_size	#kernel	learning_rate	#epoch	loss	accuracy
default	2	5	4	0.0070	65	0.01052	0.71211
#conv_layer	4	5	4	0.0070	98	0.01008	0.71619

I notice that when I increase the number of convolution layer, I observe small increase (0.004) in accuracy and small decrease (0.0005) in loss on validation dataset. In each layer, filters convolve with the image and creates an activation map. When number of layer increase, layer of activation functions also increase and our network become more complex. One can think that this is always good for learning process like more learning; however, it depends on the task and data. For example, when network increase, convergence may gets harder. With similar logic, according to above table, epoch number increases with increase in number of convolution layer. This shows that our model is struggling to optimize. Therefore, even if epoch number considerably increase, accuracy have very small increase compared to default configuration. So, I decided that it is not worth increasing the number of layers for this very small increase in accuracy against big increase in the number of epochs.



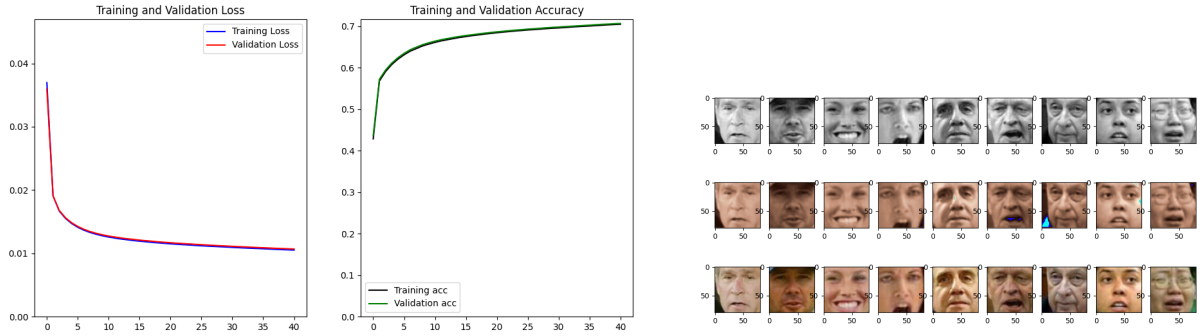
(a) Loss and accuracy tracking on training and validation data (b) Model input, prediction and ground

Figure 2: Using greater number of convolution layers than default.

1.2 effect of the kernel size(except the last conv layer

Description/Change	#conv_layer	kernel_size	#kernel	learning_rate	#epoch	loss	accuracy
default	2	5	4	0.0070	65	0.01052	0.71211
kernel_size	2	3	4	0.0070	41	0.01067	0.70657

I noticed that when I shrink our kernel size, it reaches the local minimum value faster. Of course, I experienced a decrease in our accuracy value in return. Frankly, I expected that I would get a result like this here, as a result, our loss calculation is one and the transaction cost is in a kind of trade-off mechanism. When the kernel size increased the number of parameters to train increases with the rate of the quadratic formula. Therefore, when the kernel shrinks to size, I can reach the local minimum with less cost. However, when the number of epochs is limited, I see an increase in our loss value and a decrease in our accuracy value due to the inability to extract detailed information compared to the default values.



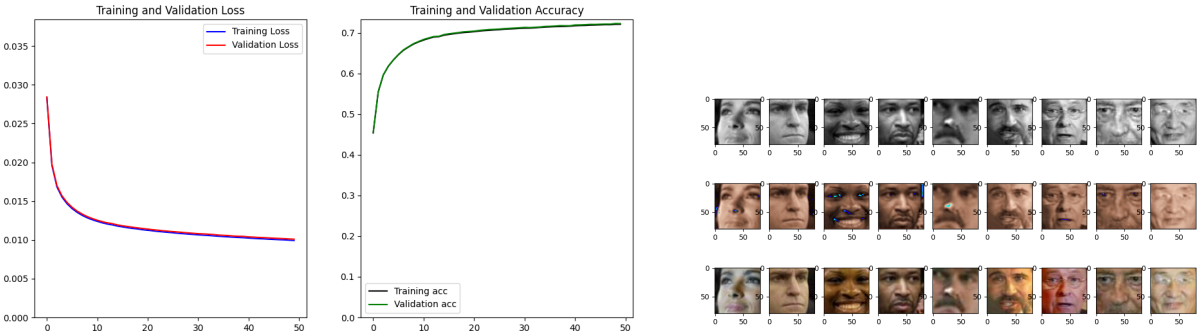
(a) Loss and accuracy tracking on training and validation data (b) Model input, prediction and ground

Figure 3: Using smaller kernel size than default.

1.3 effect of the number of kernels(except the last conv layer

Description/Change	#conv_layer	kernel_size	#kernel	learning_rate	#epoch	loss	accuracy
default	2	5	4	0.0070	65	0.01052	0.71211
#kernel	2	5	8	0.0070	49	0.01012	0.72307

I notice that when I increase the number of kernel, the accuracy also increases and loss decreases. This result is actually expected because increasing in the number of kernels means that convolution operation is done by more and different filters. This increases the robustness. Moreover, number of kernel determine the out channel shape. For example, when I increase the number of kernels from 4 to 8, I use 8 different kernel to convolve and I obtain 8 different output channel. Also, number of epoch decrease. This means more kernel facilitates optimization.



(a) Loss and accuracy tracking on training and validation data (b) Model input, prediction and ground

Figure 4: Using greater number of kernels than default.

1.4 effect of the learning rate by choosing three values: a very large one, a very small one and a value of your choice

Description/Change	#conv_layer	kernel_size	#kernel	learning_rate	#epoch	loss	accuracy
default	2	5	4	0.0070	65	0.01052	0.71211
learning_rate(large)	2	5	4	0.0700	100	0.00837	**0.75520**
learning_rate(small)	2	5	4	0.0007	80	0.01535	0.63207
our preference	2	5	8	0.0700	51	0.00838	**0.75580**

I notice that when I increase the learning rate, our model accuracy increase considerably (0.043). This is the best result among our controlled experiments. However, number of epoch increase and this may means optimization is

getting hard. Actually when I examine the Figure 9, at the 50'th epoch validation loss is almost converge but our delta value is very low to stop experiment(will be explained in part 3). So, I can say that it also optimize well.
On the other hand, when I decrease the learning rate, accuracy decrease considerably (0.08) and number of epoch increase by 15. Also, when I examine the Figure 11, I can clearly see fluctuation in loss and accuracy values. This means the danger of local min. In fact, if I decrease the learning rate more such as 0.0001, it can be confined to the local minimum and I will probably less accuracy and more loss.

I wondered and I made one more experiment to see whether it is confined to the local minimum. I obtain the result for lr = 0.0001 as follows:

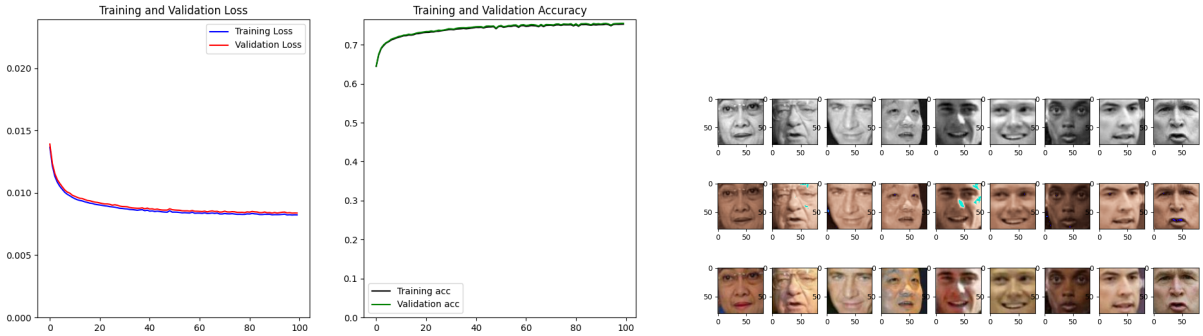
- Accuracy: 0.51590
- Losses: 0.02219

Loss increases by 0.1 and accuracy decrease by 0.1. Also I see the loss as Figure 13. It is a smooth graph different from the Figure 11. This also shows that it is confined to local min.

Our preference is as follows:

- #conv_layer = 2
- kernel_size = 5
- #kernel = 8
- learning_rate = 0.07

I don't change the #conv_layer from default because increase in #conv_layer lets to hard optimization and convergence. It is not worth in decrease accuracy 0.004 compared to increase in #epoch by 33. Kernel size 5 gives better accuracy (0.01) compared to kernel size 3 and it is worth to increase in #epoch. Therefore, I keep 5 as kernel size. I change the #kernel from 4 to 8. It improve our accuracy as 0.11 and it also optimize quickly. Finally, I change the learning rate from learning rate of base model (0.007) to 0.07. It improves considerably for both loss and accuracy. Although #epoch increase in controlled experiment, accuracy increase is very high;therefore , I decided that our new learning rate should be 0.07. With all these changes in our last experiment to see the combination results, I obtain the best accuracy up to now and #epoch also less than default. This is very good result and it showed that our analysis make quite sense.



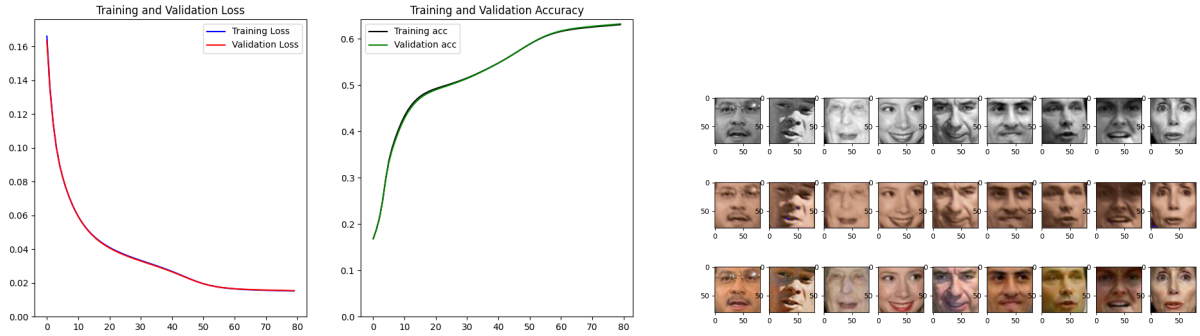
(a) Loss and accuracy tracking on training and validation data (b) Model input, prediction and ground

Figure 5: Using a very large learning rate.

2 Further Experiments

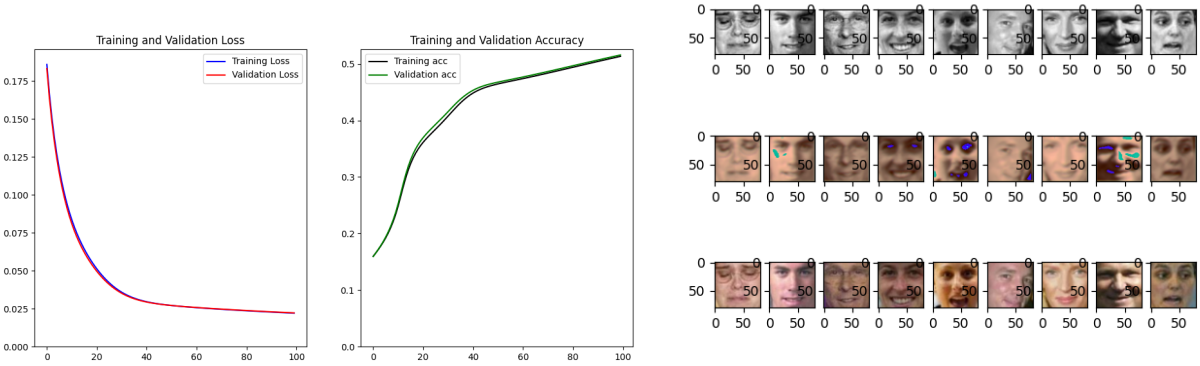
2.1 add a batch-norm layer (torch.nn.BatchNorm2d) after each convolution layer

According to our experiment, When I add the batch norm, I observe a decrease in accuracy and an increase in loss. The main feature of the batch norm is accelerating. It make the training faster. If I examine the loss and accuracy



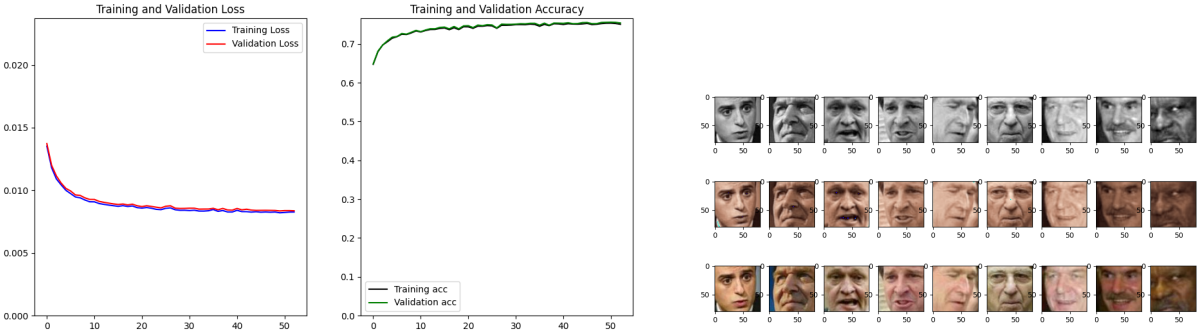
(a) Loss and accuracy tracking on training and validation data (b) Model input, prediction and ground

Figure 6: Using a very small learning rate.



(a) Loss and accuracy tracking on training and validation data (b) Model input, prediction and ground

Figure 7: Using a small learning rate.



(a) Loss and accuracy tracking on training and validation data (b) Model input, prediction and ground

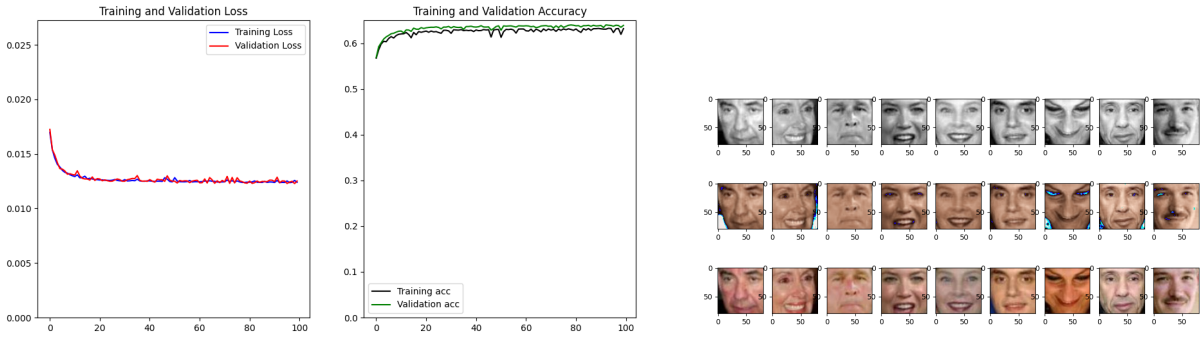
Figure 8: Using the parameter to continue with.

plots, I can see that loss decrease very fast and accuracy converge very fast. Speed is very beneficial for a model but accuracy decrease 0.11 and this is a very essential value. Therefore, I decided not to keep it in our network. Also, there are noises on the image (cyan colors on the image). Noises result from the RGB output of the network which includes more than 1 and less than -1. In the hw3utils.py file, there is a transform function

Description/Change	#conv_layer	kernel_size	#kernel	learning_rate	batch_norm	tanh	#epoch	loss	accuracy
initial	2	5	8	0.07	False	False	51	0.00838	0.75580
batch_norm	2	5	8	0.07	True	False	93	0.01235	0.64015
tanh	2	5	8	0.07	False	True	100	0.00840	0.75435
batch_norm & tanh	2	5	8	0.07	True	True	98	0.01249	0.63520
tanh & #kernel = 16	2	5	16	0.07	False	True	94	0.00829	**0.75830**

transforms.functional.to_pil_image(tensor/2+0.5)

This function transforms our network output to an image file. If our all output value are between [-1,+1], this function transform all value to [1,0]. Also, python image library convert [255,0] or [1,0] values to image. If the values are out of this ranges, image include noises. When I examine our network output in debug mode, I see that there are some values more than 1 and less than -1. This is the reason of the noises.

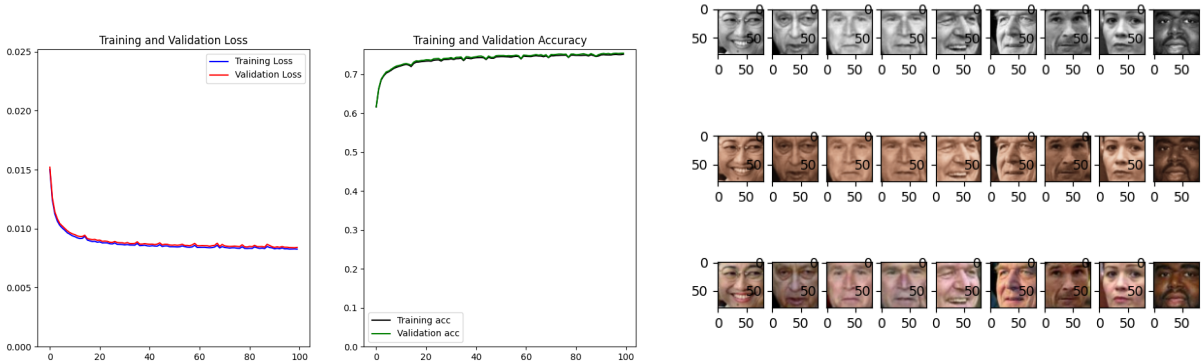


(a) Loss and accuracy tracking on training and validation data (b) Model input, prediction and ground

Figure 9: Applying Batch-Norm after each convolution layer(except the last one) of best Baseline model.

2.2 add a tanh activation function after the very last convolution layer

According to our experiment, benefit of Tanh at the vert last convolutional layer is that it maps all values to [-1,1] and this removes the noise which explained one question above. Thanks to Tanh, I can obtained the noiseless image. Also, accuracy are almost same with our preference from baseline model and there are no noise anymore. Therefore, I decided to keep it in our network.

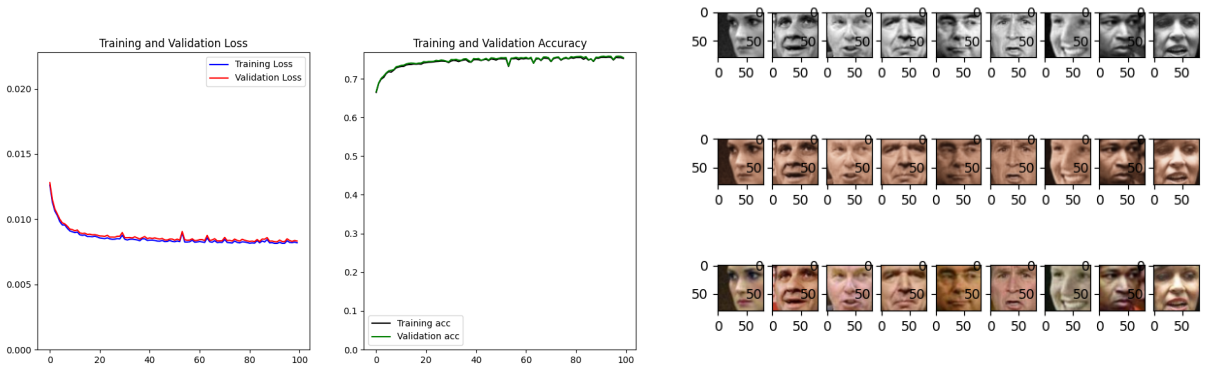


(a) Loss and accuracy tracking on training and validation data (b) Model input, prediction and ground

Figure 10: Applying Tanh activation after the last convolution layer of best Baseline model.

2.3 set the number of channels parameter to 16

It increases the accuracy and decreases the loss as seen in the table below 2'nd part (Furter Experiments). Increasing in the number of kernels means that convolution operation is done by more and different filters. I observe the same effect when number of kernel increases from 4 to 8.



(a) Loss and accuracy tracking on training and validation data

(b) Model input, prediction and ground

Figure 11: Using 16 kernels and applying Tanh activation after the last convolution layer of best Baseline model.

3 Best Configuration

Using the best model that I obtain:

- #conv_layer = 2
- kernel_size = 5
- #kernel = 16
- learning_rate = 0.07
- batch_norm = False
- tanh = True

3.1 Strategy for automatically chosen number of epochs

I take validation loss into account while chosen number of epoch. I have two parameter for this operation. First is the patience and second is delta. Delta is 0.00005 and patience is 10. This means validation loss change are less than Delta during last 10(patience) epochs

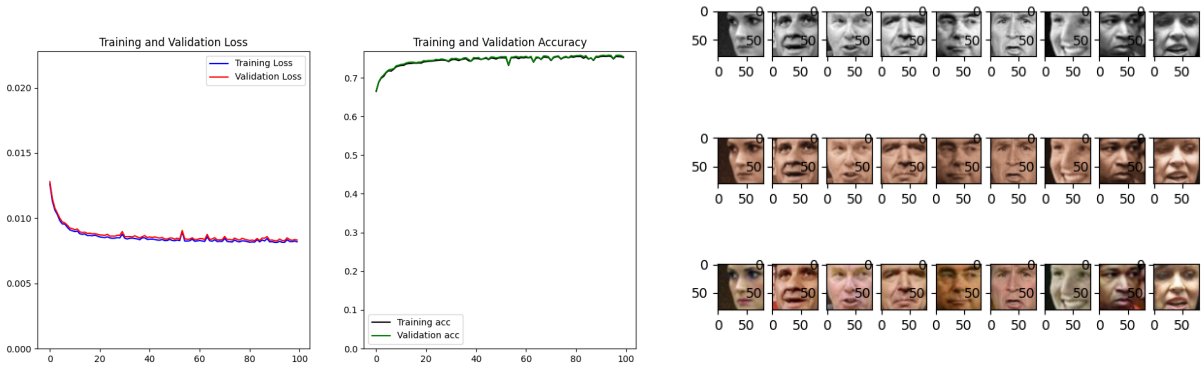
3.2 The advantages and disadvantages of the model, potential ways to improve the model

- Advantages of our model:

  I add tanh at the end of the model. Therefore, it removes the noise point by squashing the out to [-1,1]. Our learning rate is not too low therefore, I prevent our model from the local minimum point. Our model is not too deep, I use 2 conv layer, it increases the our speed and accuracy is almost same with more deep(4 conv layer) model.

- Disadvantages of our model:

  Number of kernel is 16 in our model. This cause it to run slow.



(a) Loss and accuracy tracking on training and validation data

(b) Model input, prediction and ground

Figure 12: Using the best parameter configurations.

- To improve our model:

  Loss function and optimization methods can be changed. Also more deep network increase the baseline model in our controlled experiment (I don't choose it because of high number of epoch with low accuracy increase).

4 Results on the Test Set

I arrange this part in a user-defined way, any model can be used to obtain outputs for the test images. Outputs of the input images given model are saved as 'estimations_test.npy', however, running the test function for a specific model is sufficient to get the *out* directory and the output images in it. In order not to cause an incomplete explanation, the functions in the code I write can be run through the CLI, and thanks to the generic structure of the function that provides this operation, the user only needs to write the '**python template test-best-only**' command if the model checkpoint already exists in the 'checkpoints' directory. If the model is not already in the directory, or deleted somehow, user can type in **python template improved-best true**. Which will then train the best model I obtain from scratch then save to for later use. The last parameter ***true*** stands for enabling the test operation, such that, after the training that model will be used to obtain the outputs and save them('estimations_test.npy' and).

For further notice about the accepted commands and various run-able commands I added a README file.

5 Additional Comments and References

5.1 U-Net

When I tried to improve our model, I wondered how a different model perform. Then, I make one more experiment with U-Net. I obtain 0.01 accuracy increase and 0.0002 loss decrease compared to our model.

What is the advantage of U-Net?
The architecture contains two paths. First path is contracting path and second path is expansive path. The contracting path follows the typical architecture of a convolutional network. At each down sampling step, the number of feature channels is doubled. Every step in the expansive path consists of an up sampling of the feature map and convolution layers halves the number of feature channels. Also, Model makes concatenation with the correspondingly cropped feature map from the contracting path. This concatenation is one of the major enhancement of the U-Net because it transfer the information from up sampling part to to higher resolution layers.

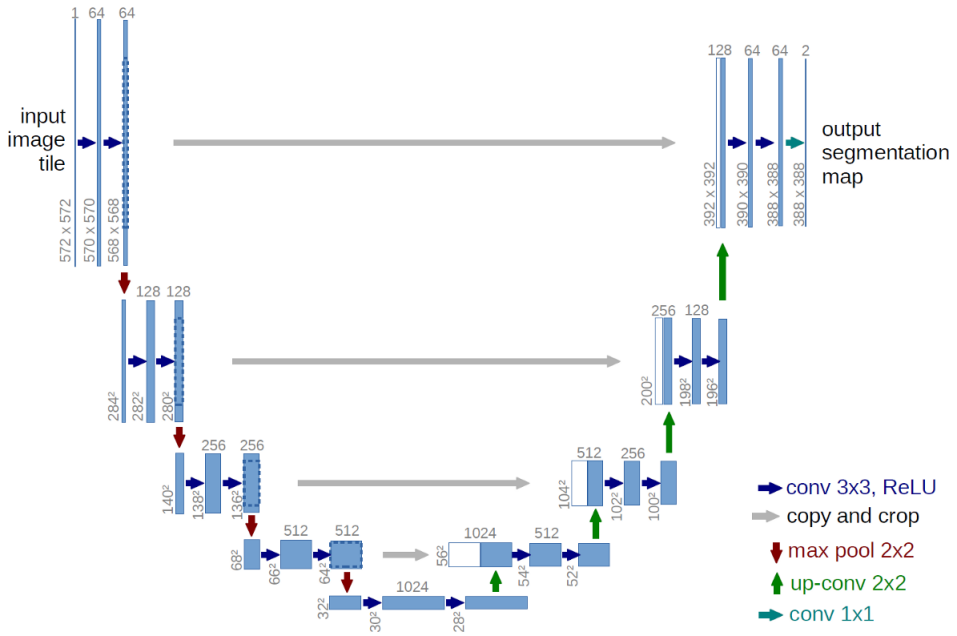
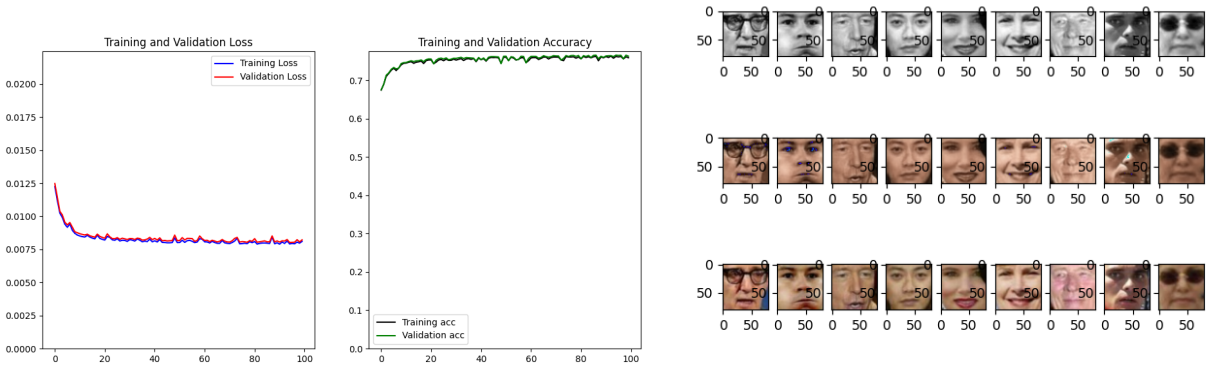


Figure 13: U-Net Architecture

Our results are as follows
Accuracy: 0.7658194010416679

Losses: 0.008069421112537384
Finished Epoch: 87



(a) Loss and accuracy tracking on training and validation data (b) Model input, prediction and ground

Figure 14: Using U-Net architecture.

5.2 Weird-Net

Another method I wondered and tried to improve our model was to go through the convolution with different numbers of channels through the input and process them separately, then concatenate them as combinations and continue the process in that way.

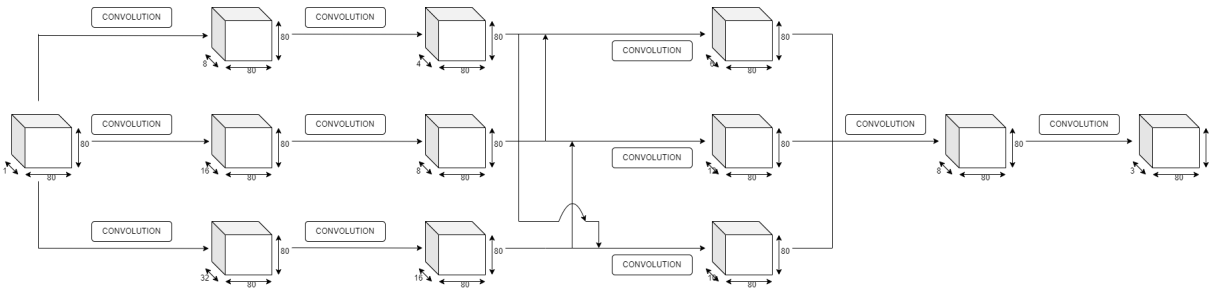
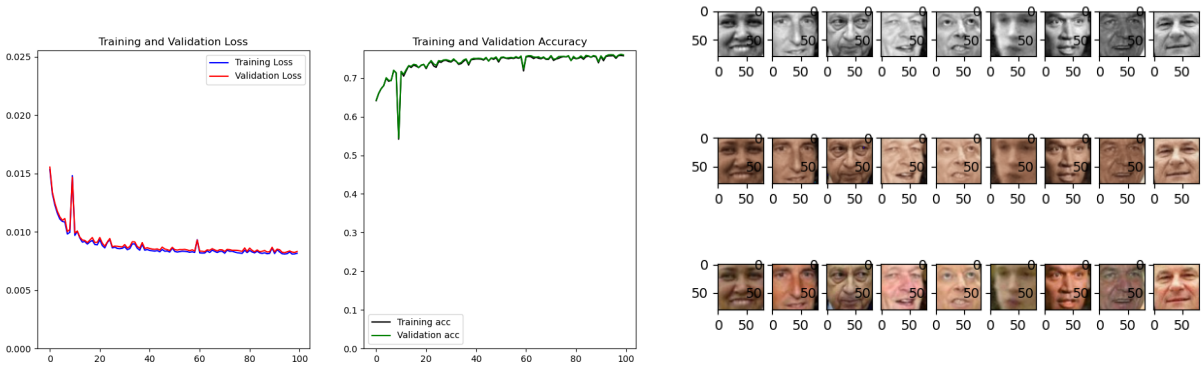


Figure 15: Weird-Net Architecture

Our results are as follows
Accuracy: 0.761421875000001
Losses: 0.008221826430410147
Finished Epoch: 99



(a) Loss and accuracy tracking on training and validation data

(b) Model input, prediction and ground

Figure 16: Using Weird-Net architecture.