CS 194-26: Image Manipulation and Computational Photography, Fall 2022

Project 5: Facial Keypoint Detection with Neural Networks

Ethan Gnibus

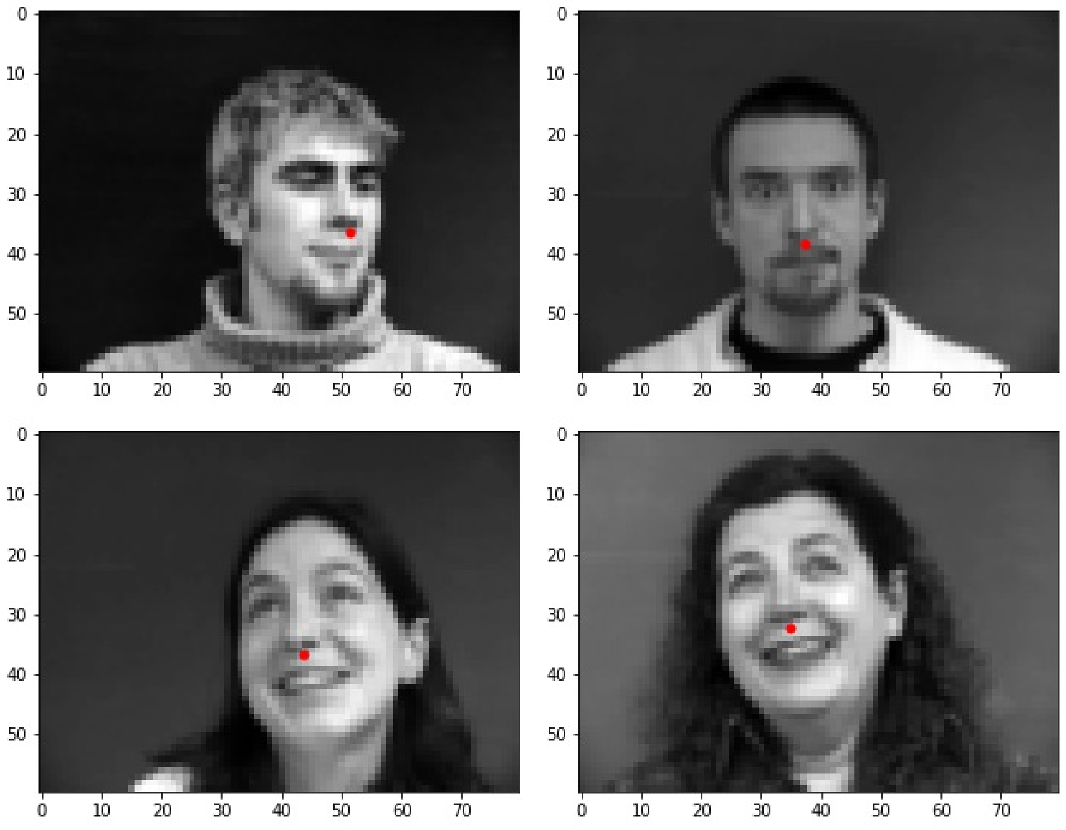
Overview

In this project I will detect facial keypoints on an image using many Deep Learning techniques for Computer Vision

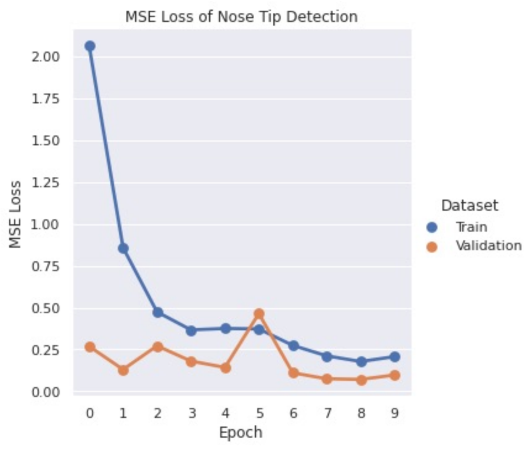
Part 1: Nose Tip Detection

Here I want to use a neural network to predict the tip of someone's nose. I will proceed by training with a small dataset, the IMM Face Database.

Here I show a few sampled images from my dataloader visualized with their ground-truth keypoints.

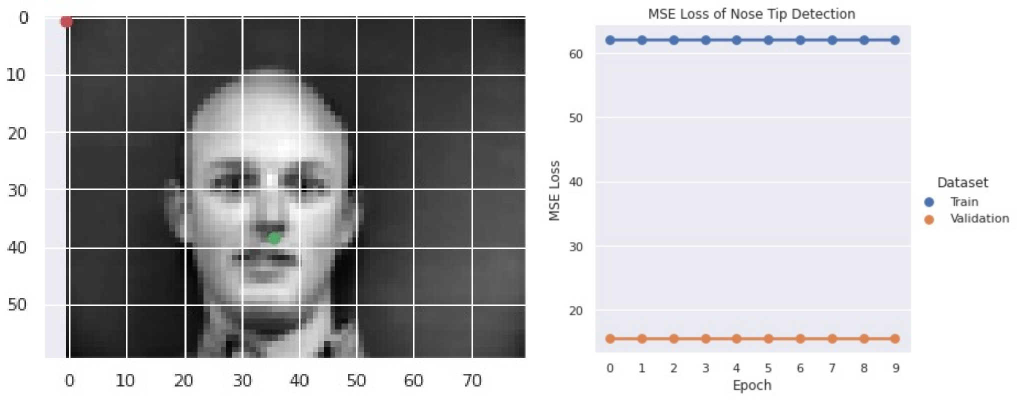


Here I plot the train and validation MSE loss during the training process.

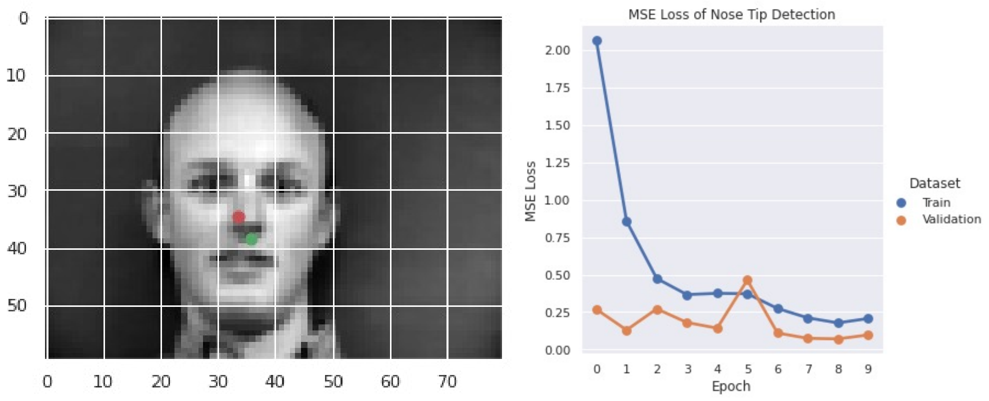


Here I show how hyper parameters effect results.

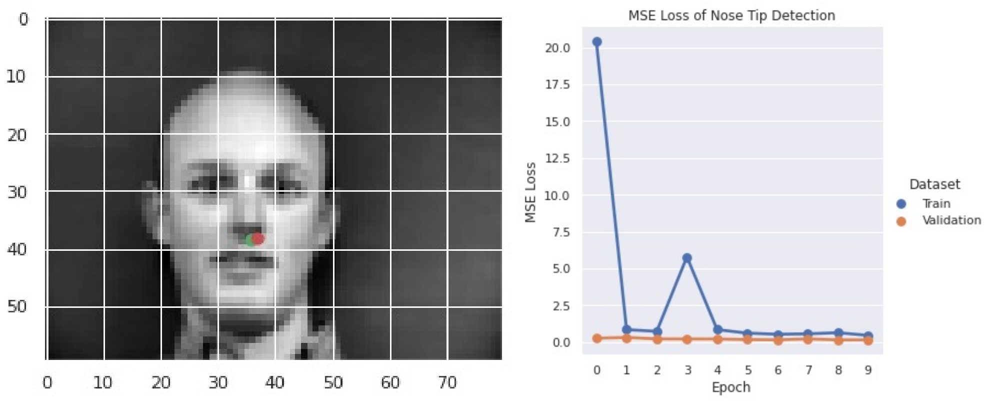
Here I change the learning rate



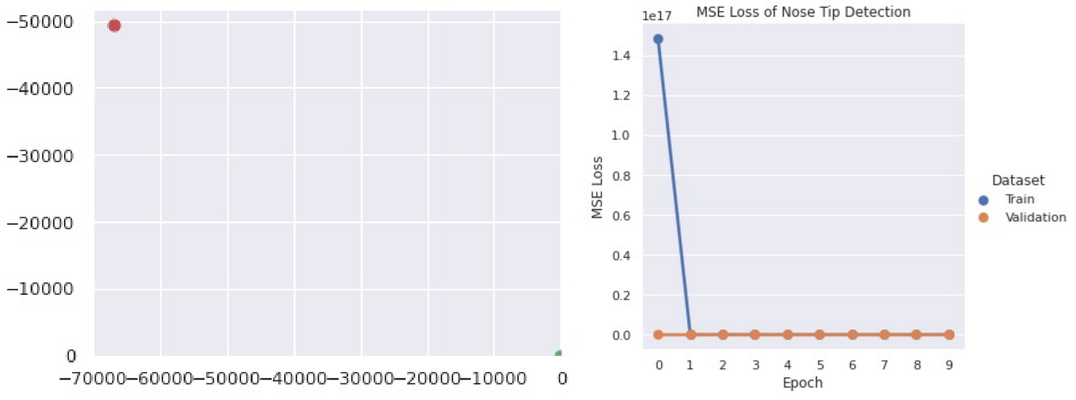
lr = 1e-100



lr = 1e-3



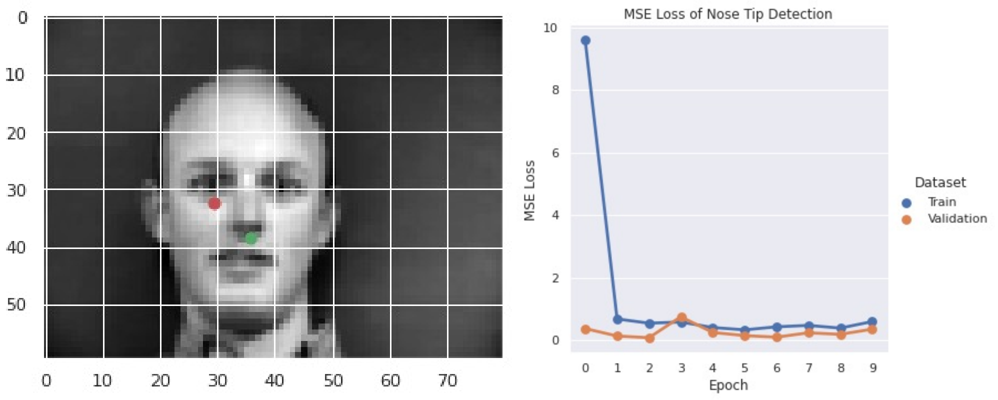
lr = 1e-2



lr = 1

As we can see, lower learning rates "slow down" learning so that the loss changes much less across epochs. Higher learning rates "explode" learning so that the loss changes drastically across epochs. I found lr = 1e-2 to be the best when training with 10 epochs for a model with 3 layers. The runtime stays roughly the same with different learning rates.

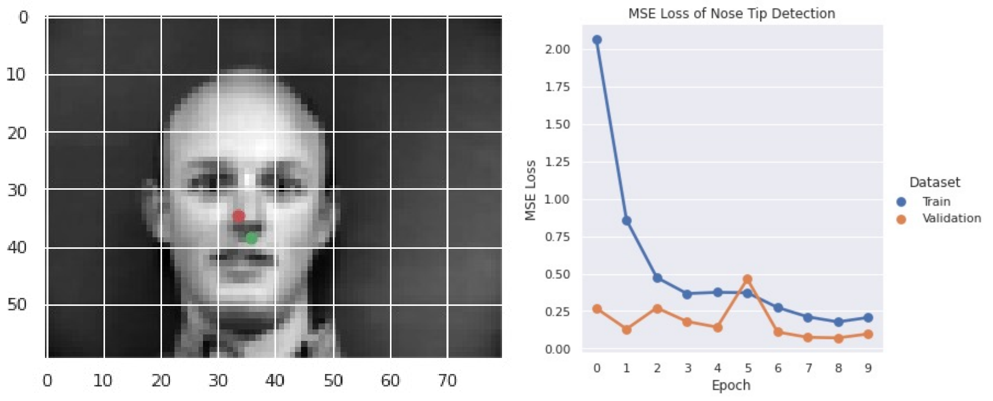
Here I change the number of layers with a learning rate of 1e-3



2 Convolutional Layers

Runtime = 1191.257 seconds

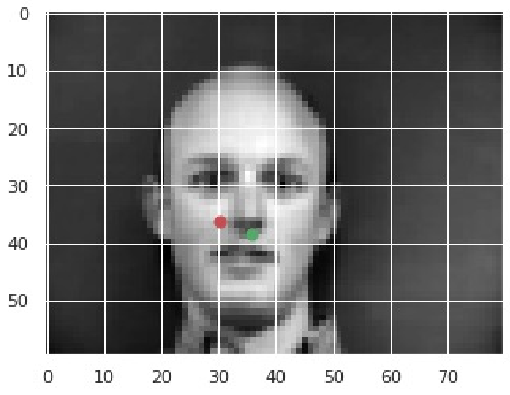
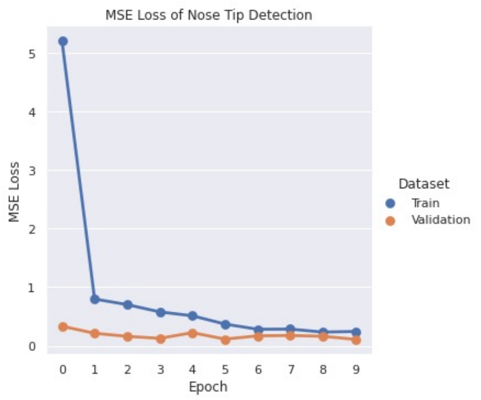
Number of Learnable Parameters = 31577378



3 Convolutional Layers

Runtime = 182.345 seconds

Number of Learnable Parameters = 940346

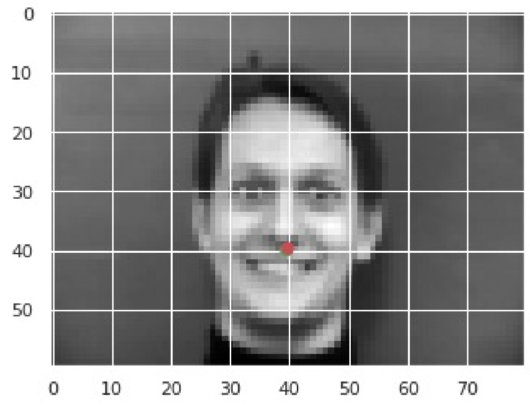
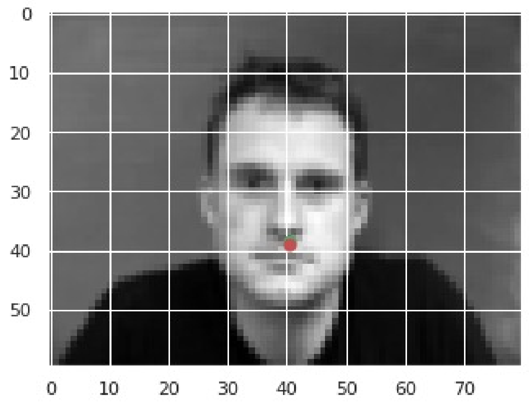
4 Convolutional Layers

Runtime = 134.378 seconds

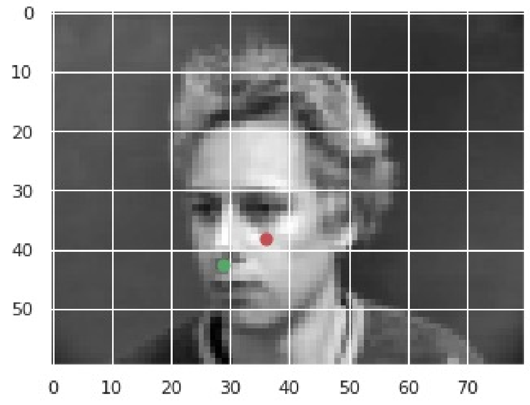
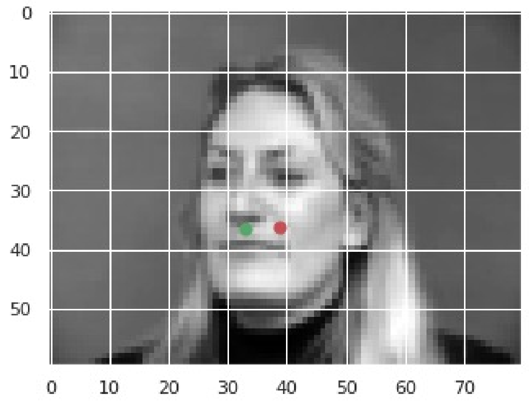
Number of Learnable Parameters = 26474

As we can see, as the number of convolutional layers increases, the number of learnable parameters in the network decreases and the runtime improves. This is becuase the less we convolve, the more features we have to train in the fully connected layers (MLP). I found 3 Layers to have the best result. I found 3 convolutional layers to get the best results.

Here I show two facial images where the network detects the nose correctly.

Here I show two facial images where the network detects the nose incorrectly.

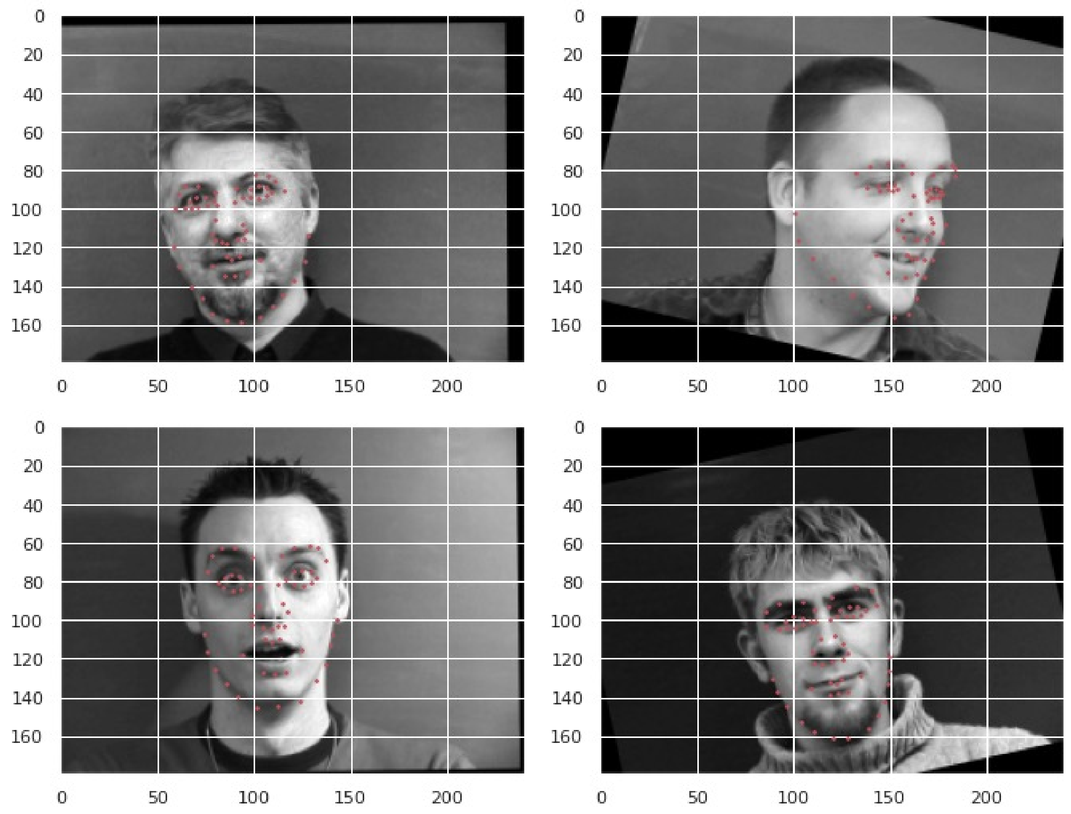



I think the neural network fails to detect keypoints because the model is overfit! I think the model has learned to output keypoints close to the center of the image because noses are generally photographed in the center of the image. Even though this behavior doesn't correctly solve the detection problem we wanted to complete, the model thinks it is preforming well. This is because learning to output points near the center of the image is similar to learning the average point, which has low loss. We would have to mitigate overfitting to change this behavior.

Part 2: Full Facial Keypoints Detection

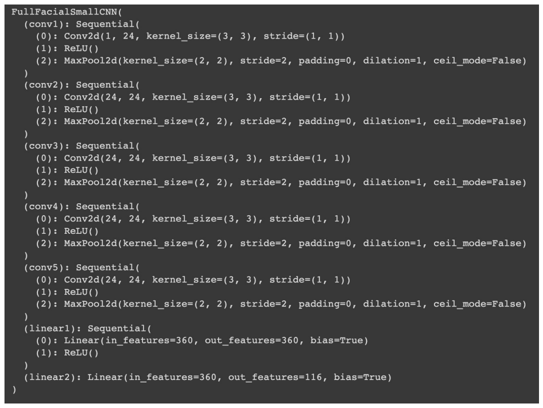
Here I want to use a neural network to predict all 58 facial keypoints from the IMM Face Database on new input images. I will proceed by training with a small dataset, the IMM Face Database.

Here I show a few sampled images from my dataloader visualized with their ground-truth keypoints.



My Model

My architecture:

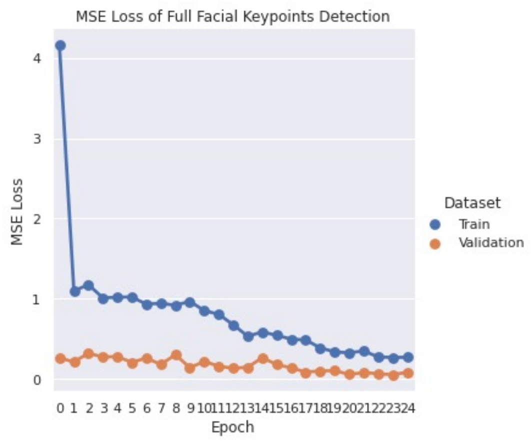


```
FullFacialSmallCNN(
  (conv1): Sequential(
    (0): Conv2d(1, 24, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(24, 24, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(24, 24, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(24, 24, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv5): Sequential(
    (0): Conv2d(24, 24, kernel_size=(3, 3), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (linear1): Sequential(
    (0): Linear(in_features=360, out_features=360, bias=True)
    (1): ReLU()
  )
  (linear2): Linear(in_features=360, out_features=116, bias=True)
)
```

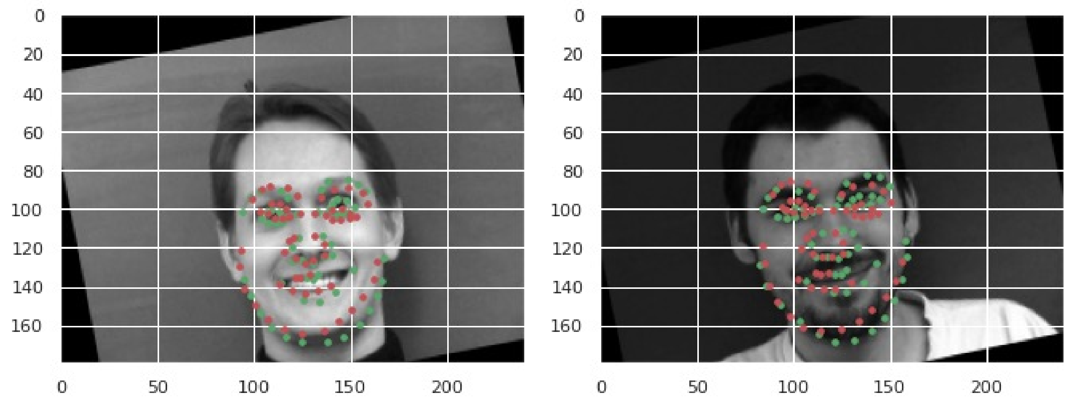
Hyperparameters:

Data Augmentation = {
imgaug.augmenters.Multiply,
imgaug.augmenters.GammaContrast,
Random Rotation,
Random Translation
}
Epochs = 25
Criterion = MSE Loss
Optimizer = Adam
Batch Size = 1
Convolutional Layers = 5
Trainable Parameters = 192908

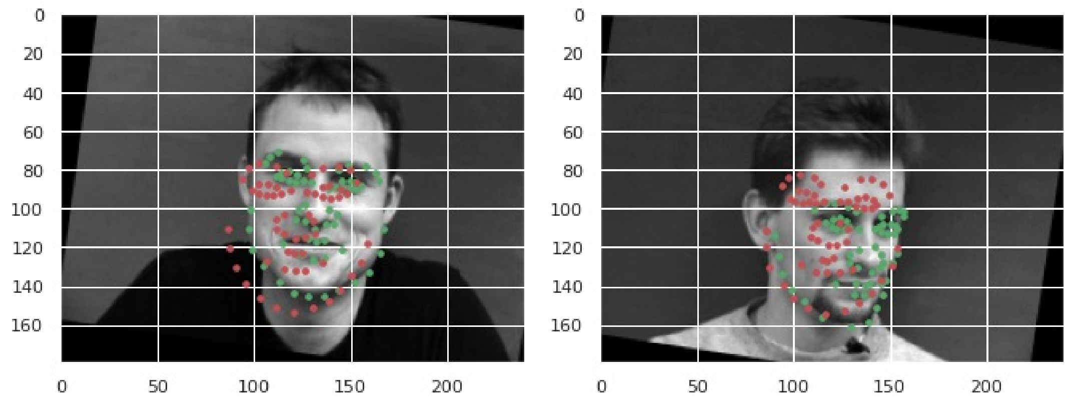
Here I plot the training and validation loss across iterations:



Here I show two facial images where the network detects the nose well.



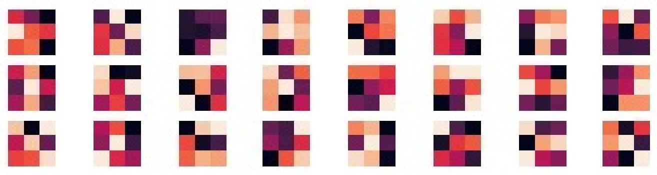
Here I show two facial images where the network detects the nose poorly.

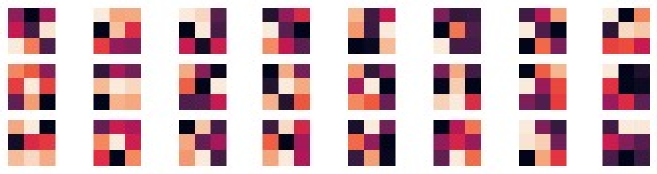


I think the neural network fails to detect keypoints because The dataset is still too small. Right now, it seems that the dataset is optimized to work for the small train set, but fails a lot on data not in that set. Even with data augmentation, it seems that the dataset fails to mitigate overfitting. To solve this, I can train on a bigger dataset.

During the learning process of a convolutional neural network, backpropagation updates the filters that are convolved with the input images. When trained correctly, these filters extract useful information that allows our network to predict keypoints. Filters in the first few layers of the network detect basic features such as lines and edges. As we go deeper into the network, these filters get more specialized and start learning more abstract features such as texture. Here I display some learned filters from the first 3 layers for simplicity:

Convolutional Layer 1


Convolutional Layer 2

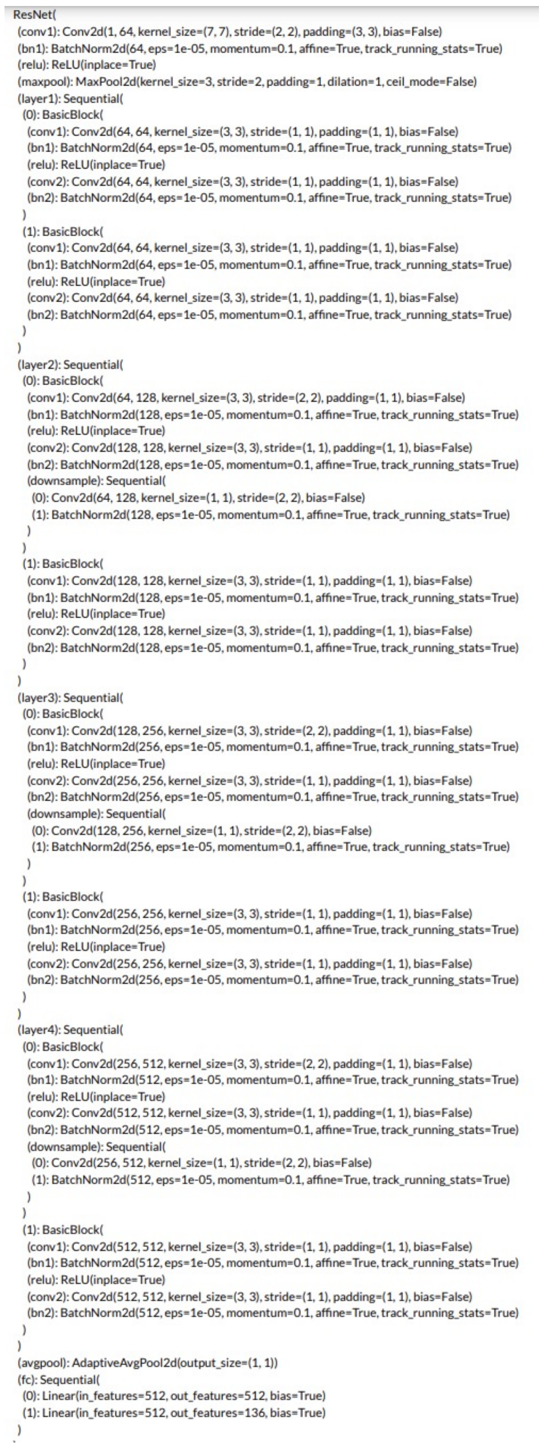

Convolutional Layer 3

Part 3: Train With Larger Dataset

Here I want to use a neural network to predict all 68 facial keypoints from the ibug_300W_large_face_landmark on new input images. I will proceed by training with the ibug_300W_large_face_landmark dataset, a dataset with 6666 images.

My Model

My architecture:



```
ResNet(
  (conv1): Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Sequential(
    (0): Linear(in_features=512, out_features=512, bias=True)
    (1): Linear(in_features=512, out_features=136, bias=True)
  )
)
```

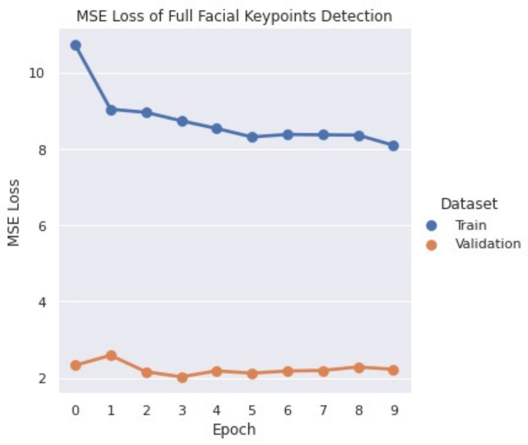
Hyperparameters:

Data Augmentation = {
  Multiply Brightness by (0.25, 2.0),
  Multiply Saturation by (0.25, 2.0),
  Multiply Contrast by (0.25, 2.0),
  Multiply Hue by (-0.5, 0.5),
  Affine Transforms: {
    Crop image by (0, 5) pixels,
    Scale x by (0.4, 1.5),
    Scale y by (0.4, 1.5),
    Rotate by (-25, 25) degrees,
    Translate x by (-20, 20) pixels,
    Translate y by (-20, 20) pixels,
    Shear image by (-20, 20)
  }
  iaa.Jigsaw with 1-3 rows and 1-3 columns,

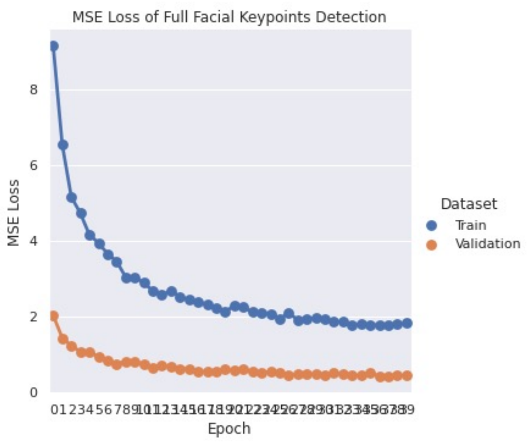
Resize image to 224x224
}
Normalize Color (Divide by 255 then subtract by 0.5)

Epochs
Training with old layers frozen and new layers unfrozen = 10
Training with all layers unfrozen = 40
Criterion = MSE Loss
Optimizer = Adam
Batch Size = 16
Number of Layers = 18
Transfer Learned From = ResNet-18
Trainable Parameters = 11502664

Here I plot the training and validation loss across iterations. I I train with all layers frozen except the first convolution and the fully connected layer at the end. This is because they are the only layers I update and I'm using transfer learning. Next, I unfreeze all layers then continue to train. Here are the results:

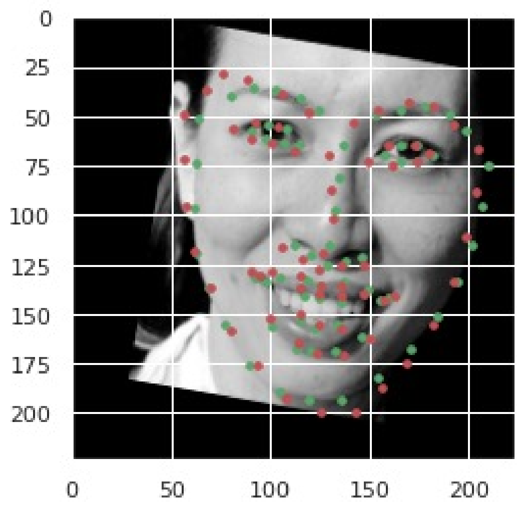


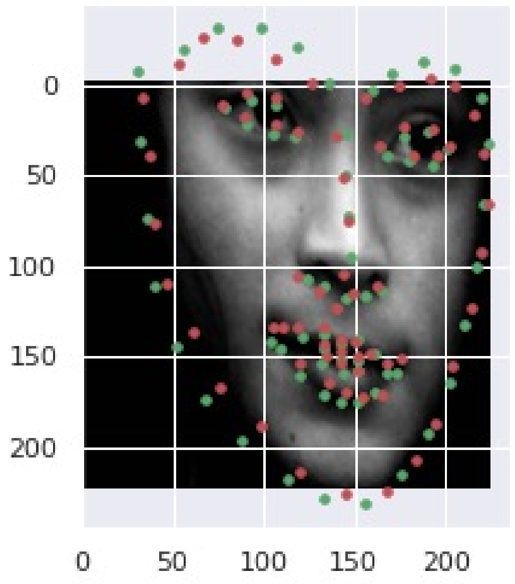
MSE Loss when only new layers are unfrozen

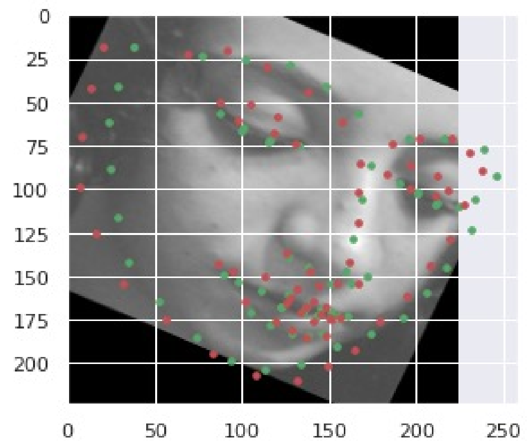


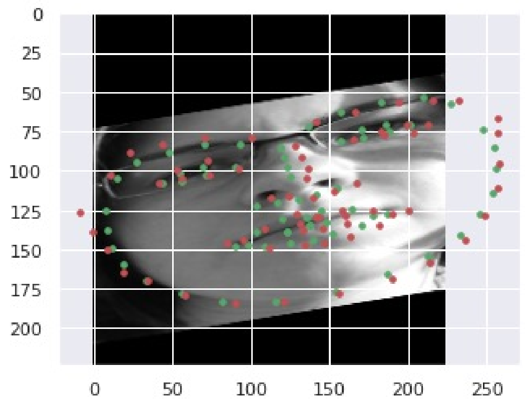
MSE Loss when all layers are unfrozen

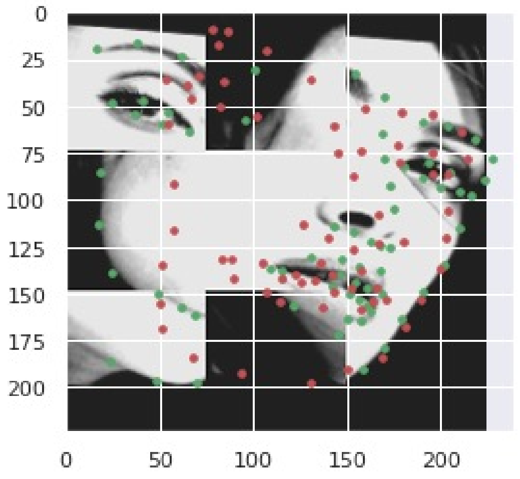
Here I visualize some images with the keypoints prediction in the validation set. I chose to jigsaw my training data so that my model doesn't assume that keypoints will always be in locations relative to each other. I ended training with a train mse loss of 1.812 and validation mse loss of 0.433.

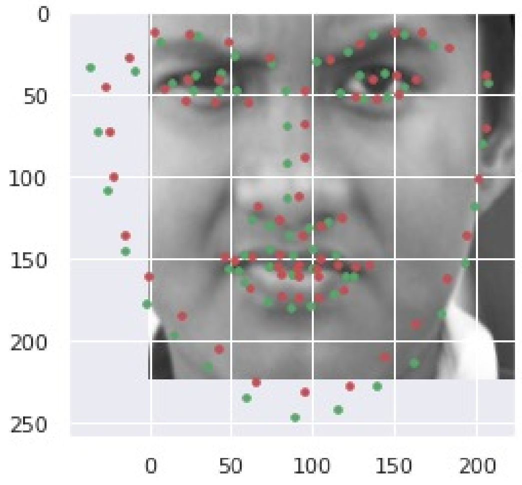

Fantastic results: slight mistake on eyebrow


Almost perfect even out of frame

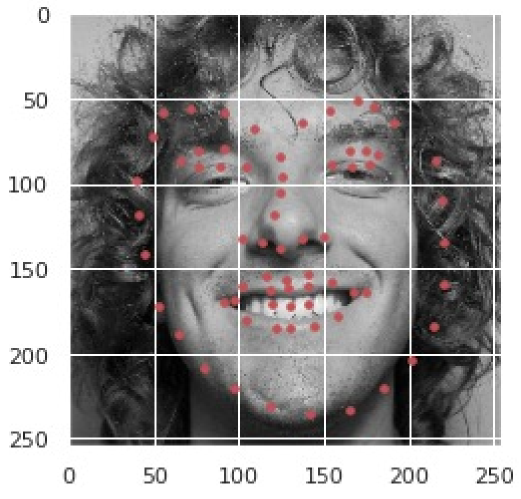

Great results but not super aligned with ground truth

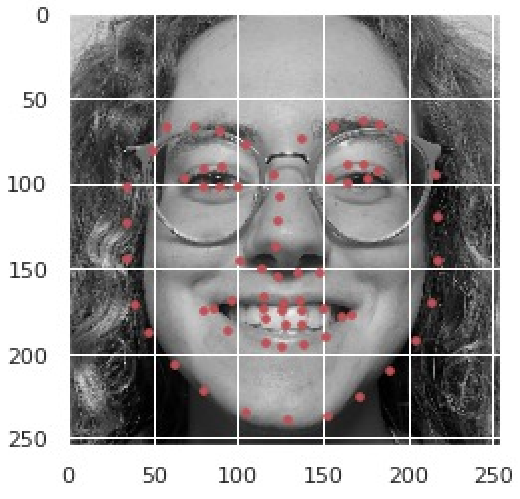

Amazing results with warp

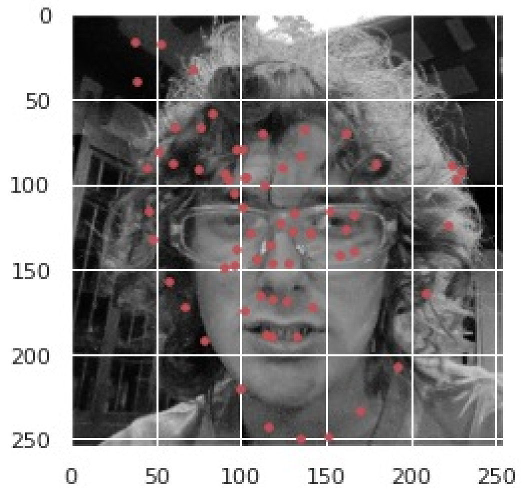

Poor results with Jigsaw (although some are right)

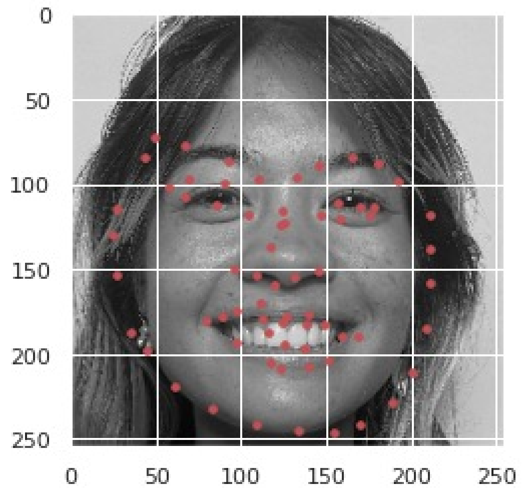

Nice results: messed up face shape

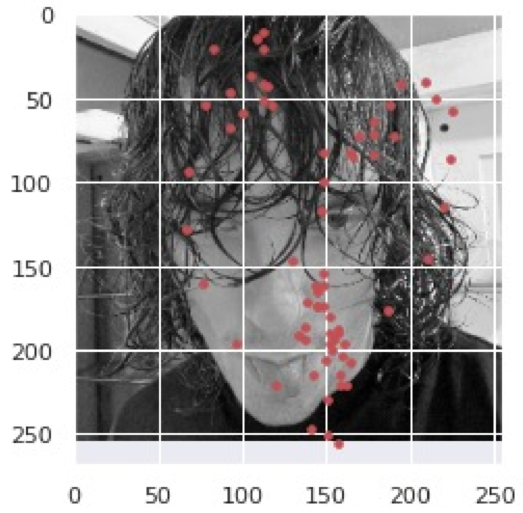
Try running the trained model on no less than 3 photos from your collection. Which ones does it get right? Which ones does it fail on? Here I run the model on 6 of my own images.

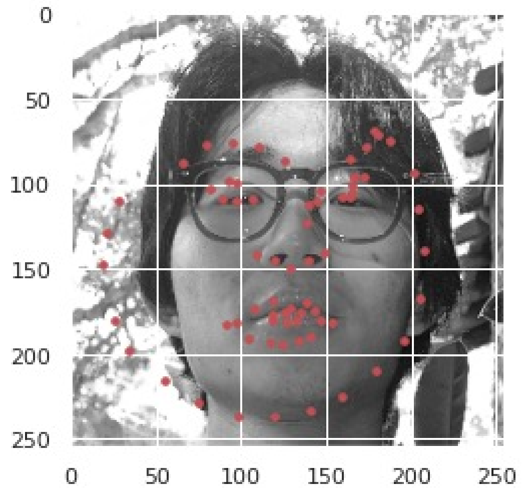

Nice: Messing up eyes


Perfect! (With glasses too!)


Bad: Noisy glasses and hair seem to mess it up


Great: Messing up face shape


Really Bad: Noisy hair making it miss almost all landmarks


Very Bad: Missing face shape, lips, and nose ridge. Probably because of glasses

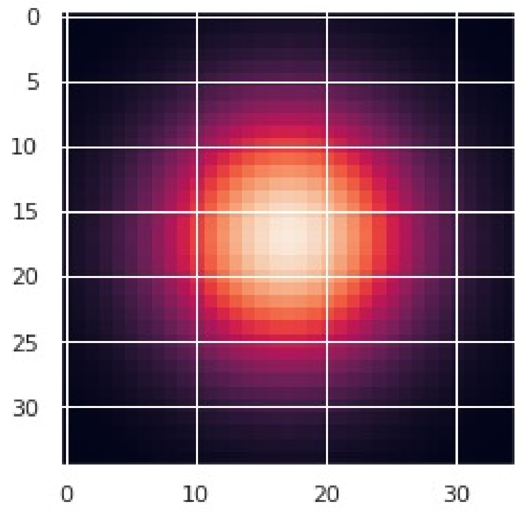
I submitted this score to the kaggle autograder with a score of 13.62223

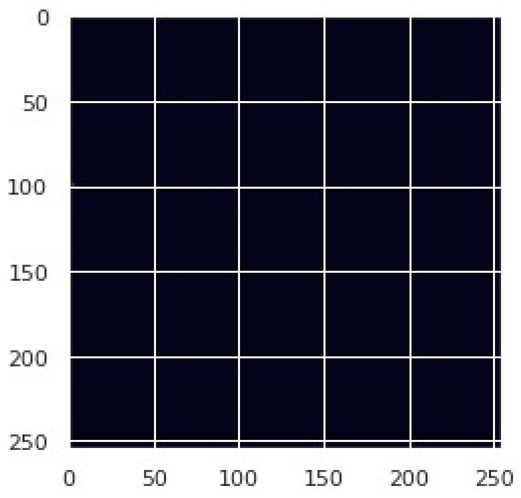
Part 4: Pixelwise Classification

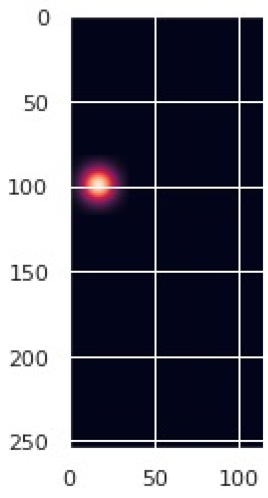
Report on the details of your implementation and your findings.

When making a Pixelwise Classification Model, I found that it might not be as useful as I thought without doing a lot of extra work. One problem that I noticed is that output keypoints are found by taking a weighted average of the heatmap for a given keypoint. This means that if the keypoint were to be outside of the image, it would be predicted wrong 100% of the time. This shows that not using pixelwise classification has a slight advantage in some cases (because it can predict outside of the image's coordinates among other things).

To generate heatmaps, I made a 2D Gaussian then shifted it to to the location of a keypoint (Gaussian Distribution). My process can be seen below. To make a 2D Gaussian, I took the outer product of a 1D Gaussian and its transpose. I made this 1D Gaussian with a kernel size of 35 and a sigma of 7.


2D Gaussian


Location of keypoint

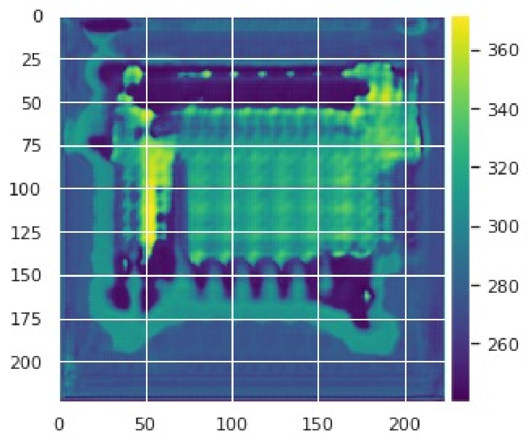

2D Gaussian shiftec

Here I show accumulated heatmaps of all landmarks of two images.

Image 1

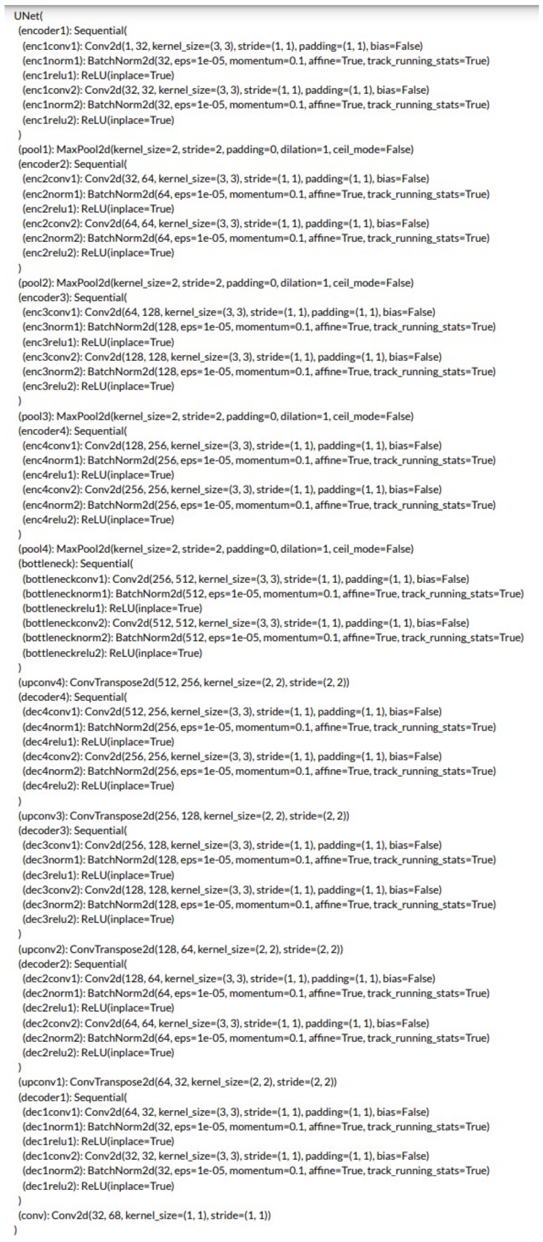
Accumulated heatmap of Image 1

Image 2


Accumulated heatmap of Image 2

My Model

My architecture:



```
UNet(
 (encoder1): Sequential(
  (enc1conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (enc1norm1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (enc1relu1): ReLU(inplace=True)
  (enc1conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (enc1norm2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (enc1relu2): ReLU(inplace=True)
 )
 (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
 (encoder2): Sequential(
  (enc2conv1): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (enc2norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (enc2relu1): ReLU(inplace=True)
  (enc2conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (enc2norm2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (enc2relu2): ReLU(inplace=True)
 )
 (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
 (encoder3): Sequential(
  (enc3conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (enc3norm1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (enc3relu1): ReLU(inplace=True)
  (enc3conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (enc3norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (enc3relu2): ReLU(inplace=True)
 )
 (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
 (encoder4): Sequential(
  (enc4conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (enc4norm1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (enc4relu1): ReLU(inplace=True)
  (enc4conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (enc4norm2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (enc4relu2): ReLU(inplace=True)
 )
 (pool4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
 (bottleneck): Sequential(
  (bottleneckconv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bottlenecknorm1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bottleneckrelu1): ReLU(inplace=True)
  (bottleneckconv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bottlenecknorm2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (bottleneckrelu2): ReLU(inplace=True)
 )
 (upconv4): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2))
 (decoder4): Sequential(
  (dec4conv1): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (dec4norm1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dec4relu1): ReLU(inplace=True)
  (dec4conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (dec4norm2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dec4relu2): ReLU(inplace=True)
 )
 (upconv3): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2))
 (decoder3): Sequential(
  (dec3conv1): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (dec3norm1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dec3relu1): ReLU(inplace=True)
  (dec3conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (dec3norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dec3relu2): ReLU(inplace=True)
 )
 (upconv2): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
 (decoder2): Sequential(
  (dec2conv1): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (dec2norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dec2relu1): ReLU(inplace=True)
  (dec2conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (dec2norm2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dec2relu2): ReLU(inplace=True)
 )
 (upconv1): ConvTranspose2d(64, 32, kernel_size=(2, 2), stride=(2, 2))
 (decoder1): Sequential(
  (dec1conv1): Conv2d(64, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (dec1norm1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dec1relu1): ReLU(inplace=True)
  (dec1conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (dec1norm2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (dec1relu2): ReLU(inplace=True)
 )
 (conv): Conv2d(32, 68, kernel_size=(1, 1), stride=(1, 1))
)
```

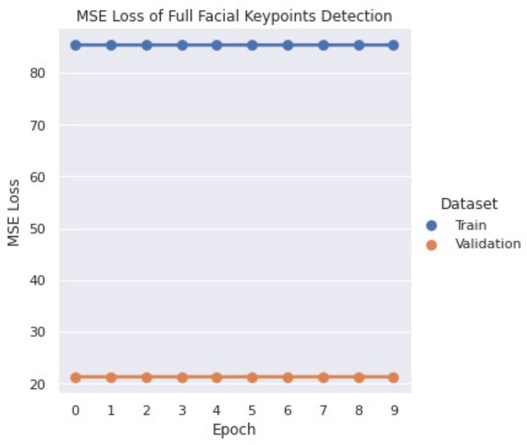
Hyperparameters:

```
Data Augmentation = {
   Multiply Brightness by (0.25, 2.0),
   Multiply Saturation by (0.25, 2.0),
   Multiply Contrast by (0.25, 2.0),
   Multiply Hue by (-0.5, 0.5),
   Affine Transforms: {
      Crop image by (0, 4) pixels,
      Scale x by (0.6, 1.4),
      Scale y by (0.6, 1.4),
      Rotate by (-20, 20) degrees,
      Translate x by (-15, 15) pixels,
      Translate y by (-15, 15) pixels,
      Shear image by (-15, 15)
   }
   Resize image to 224x224
}
Normalize Color (Divide by 255 then subtract by 0.5)
```

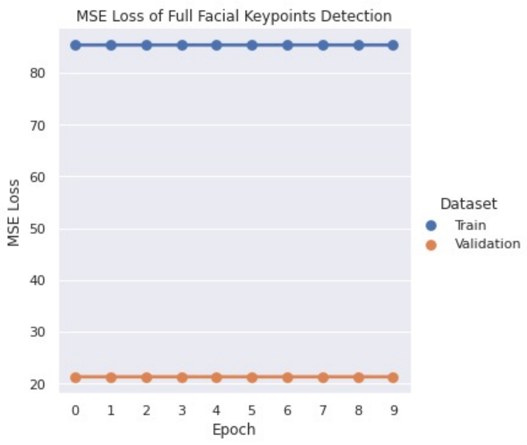
Epochs
   Training with old layers frozen and new layers unfrozen = 10
   Training with all layers unfrozen = 10
Criterion = MSE Loss

Optimizer = Adam
Batch Size = 16
Number of Layers = 18
Transfer Learned From = U-Net
Trainable Parameters = 7764676

Here I plot the training and validation loss across iterations. I I train with all layers frozen except the first convolution and the fully connected layer at the end. This is because they are the only layers I update and I'm using transfer learning. Next, I unfreeze all layers then continue to train. Here are the results:

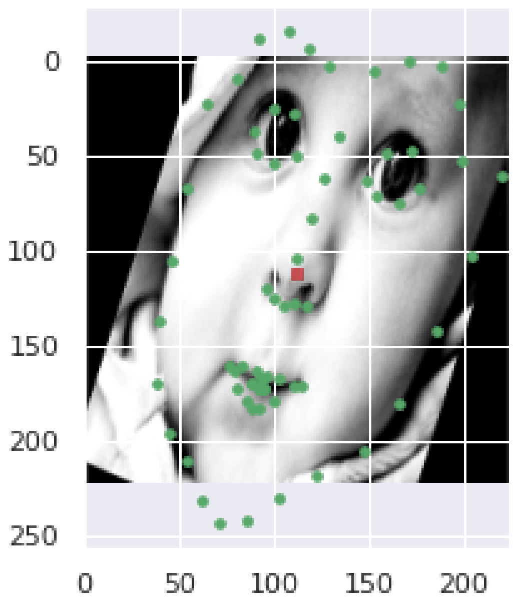


MSE Loss when only new layers are unfrozen

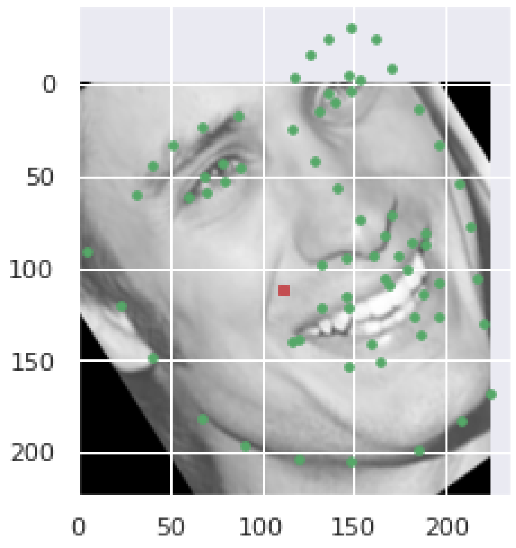


MSE Loss when all layers are unfrozen

Visualize some (two) images with the keypoints prediction in the testing set.



Test Prediction 1



Test Prediction 2

Try running the trained model on no less than 3 photos from your collection. Which ones does it get right? Which ones does it fail on?

FAIL

FAIL

FAIL

FAIL

Part 5: Kaggle

Report your best model (if it is different from part 3 or part 4, please describe the model architecture) and report the mean absolute error and Kaggle username on the website after uploading your predictions on the testing set to our class Kaggle competiton.

My best model was the one from part 3. My Mean absolute error was 13.62223 and my Kaggle username is Ethan Gnibus.

https://inst.eecs.berkeley.edu/~cs194-26/fa22/upload/files/proj5/cs194-26-ahn/