

Face in Point

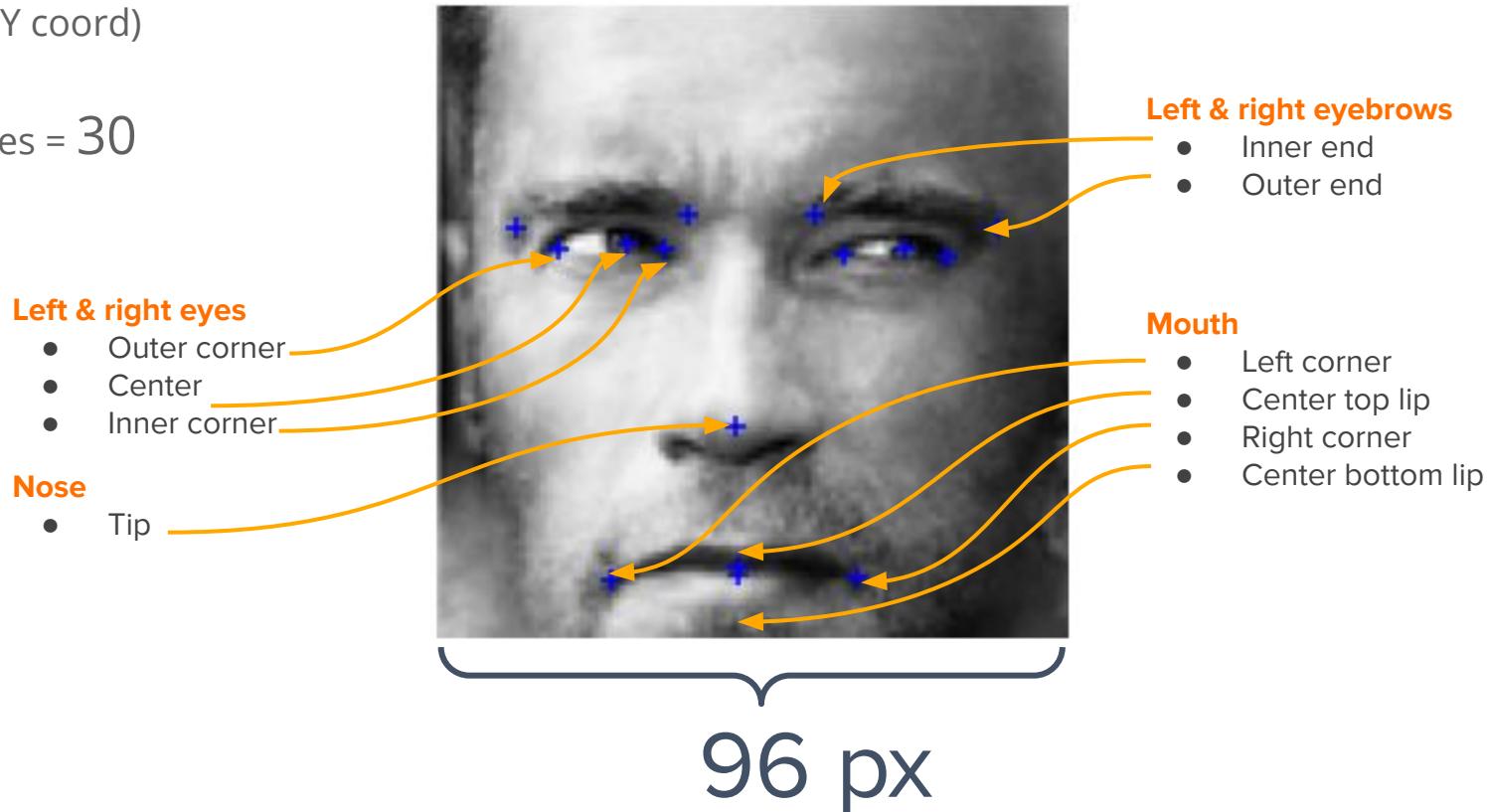
Facial Keypoint Detection, Kaggle Challenge

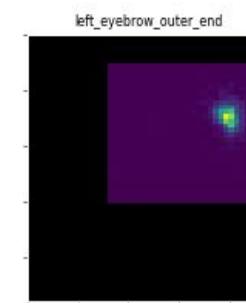
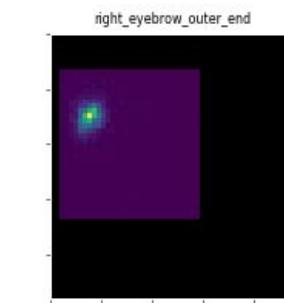
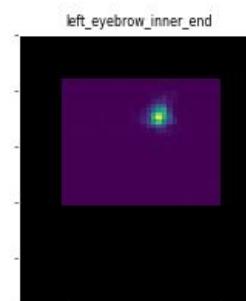
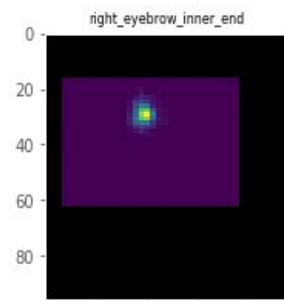
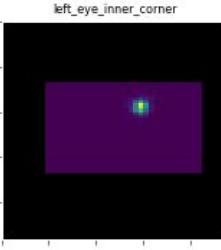
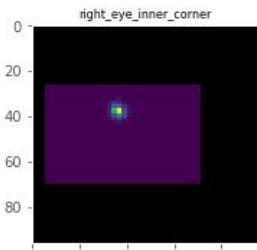
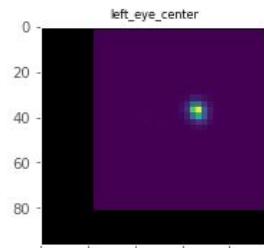
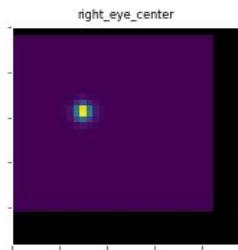
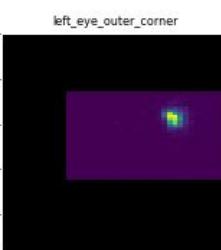
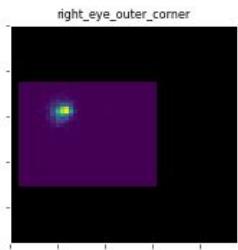
Julia Ying, Sang-hyeb Lee, Haerang Lee (*W207 Sp '20*)

Objective

Given a facial image, identify 15 facial keypoints' (X, Y coord)

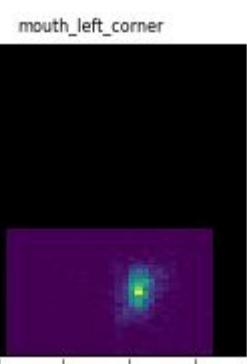
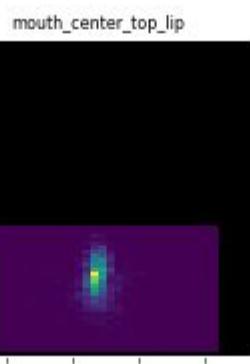
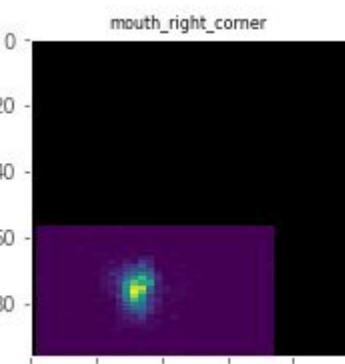
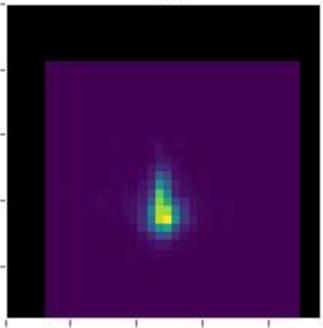
15 KP x 2 coordinates = 30



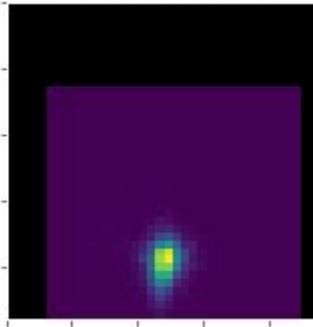




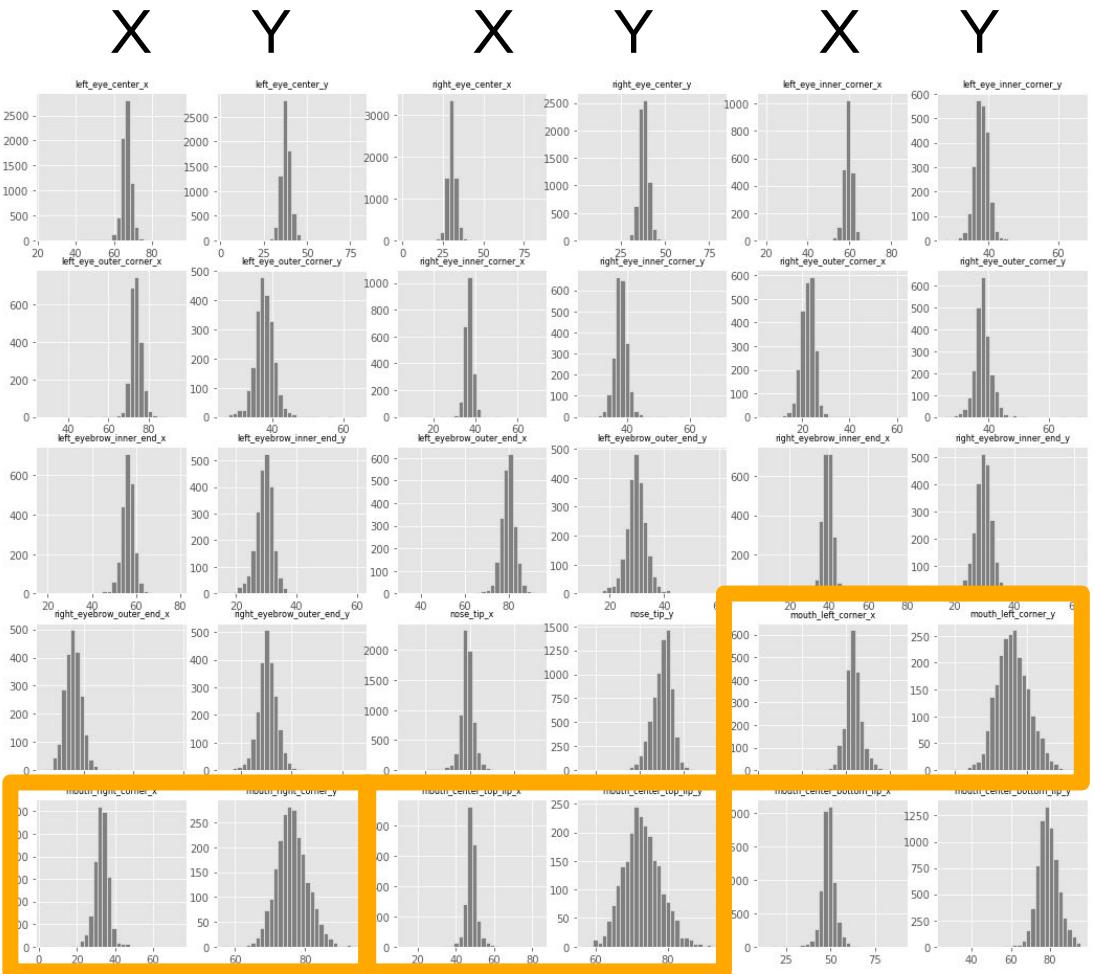
nose_tip



mouth_center_bottom_lip

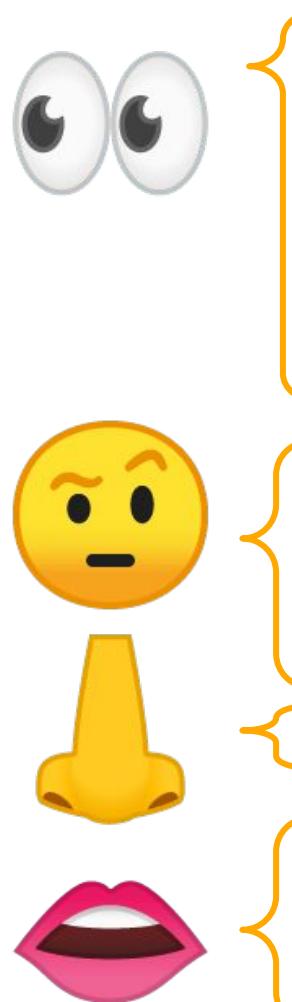


- Normal distributions
- Greater variance vertically (Y axis) than horizontally (X)
- Greatest variance near mouth
KPs: Smiling & opening mouths



But data's missing!

5K out of 7K training images
were missing KP data



Column	Non-Null Count
left_eye_center_x	7039
left_eye_center_y	7039
right_eye_center_x	7036
right_eye_center_y	7036
left_eye_inner_corner_x	2271
left_eye_inner_corner_y	2271
left_eye_outer_corner_x	2267
left_eye_outer_corner_y	2267
right_eye_inner_corner_x	2268
right_eye_inner_corner_y	2268
right_eye_outer_corner_x	2268
right_eye_outer_corner_y	2268
left_eyebrow_inner_end_x	2270
left_eyebrow_inner_end_y	2270
left_eyebrow_outer_end_x	2225
left_eyebrow_outer_end_y	2225
right_eyebrow_inner_end_x	2270
right_eyebrow_inner_end_y	2270
right_eyebrow_outer_end_x	2236
right_eyebrow_outer_end_y	2236
nose_tip_x	7049
nose_tip_y	7049
mouth_left_corner_x	2269
mouth_left_corner_y	2269
mouth_right_corner_x	2270
mouth_right_corner_y	2270
mouth_center_top_lip_x	2275
mouth_center_top_lip_y	2275
mouth_center_bottom_lip_x	7016
mouth_center_bottom_lip_y	7016

Only 2K for highlighted!

Dealing with Missing Data

Forward Fill (ffill)

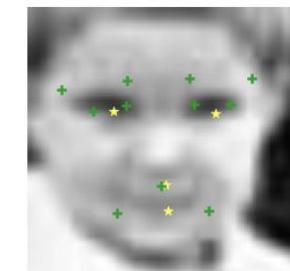
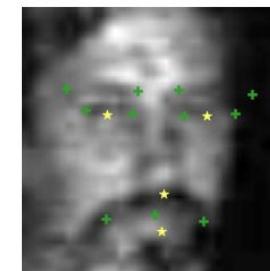
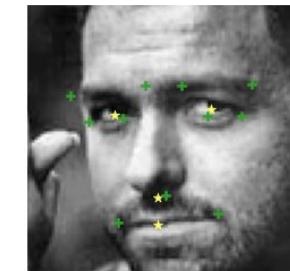
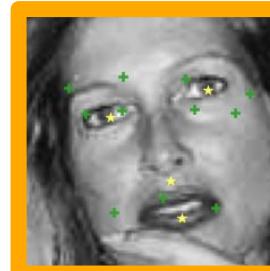
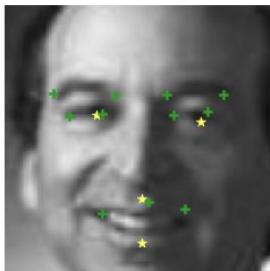
Use the last data available in previous non-empty row

Not ideal, as 70% images are missing at least one point



★ Original kp data from kaggle

✚ Extrapolated data to fill missing kps



Median

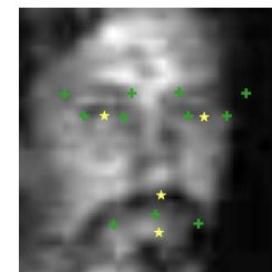
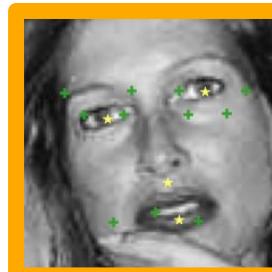
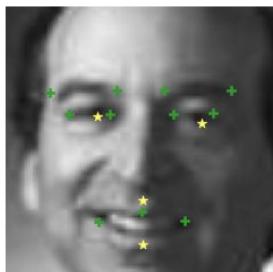
Use the median value of the respective point of interest

Works surprisingly well but not when faces are tilted



★ Original kp data from kaggle

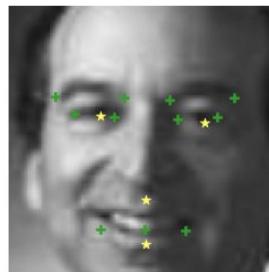
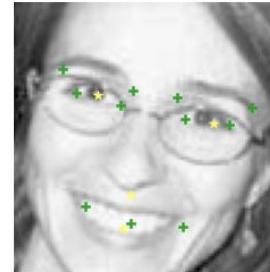
✚ Extrapolated data to fill missing kps



KNN

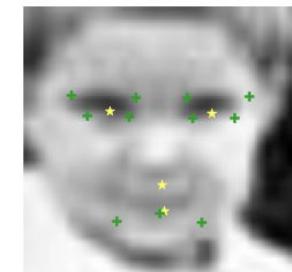
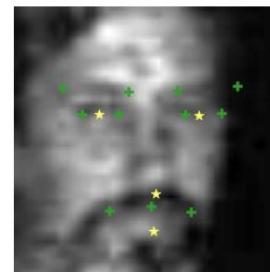
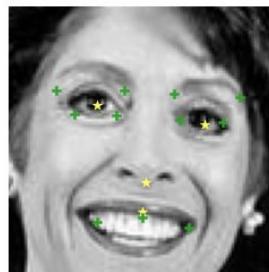
Find appropriate value to use using k-nearest neighbors, $k = 9$

Better performance than both previous attempt. Decent on tilted or turned faces



★ Original kp data from kaggle

✚ Extrapolated data to fill missing kps



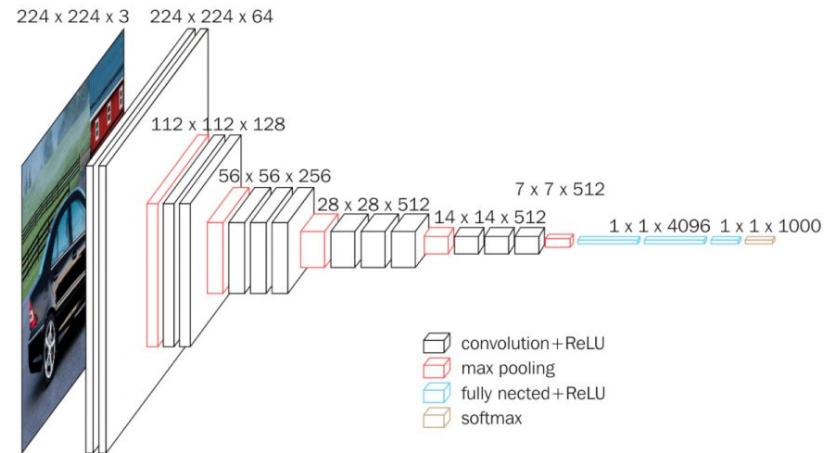
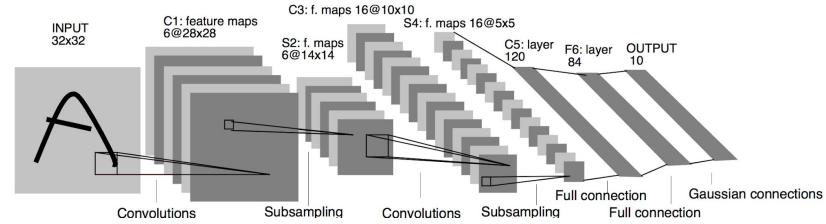
CNN Architectures

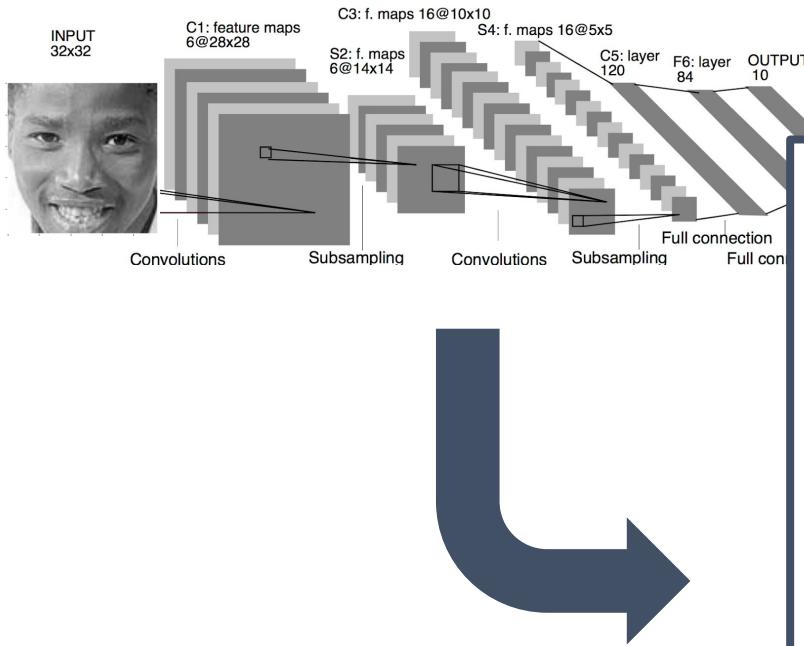
CNN

*"special kind of multi-layer neural networks,
designed to recognize visual patterns
directly from pixel images with minimal
preprocessing."*

Architectures we used

1. LeNet-5
2. VGG-16





LeNet-5

Layer shape changes from a full **96-by-96 image** to just **30 numbers** representing the KP

Input: (Num. Images, 96, 96, 1)

Model: "sequential_2"

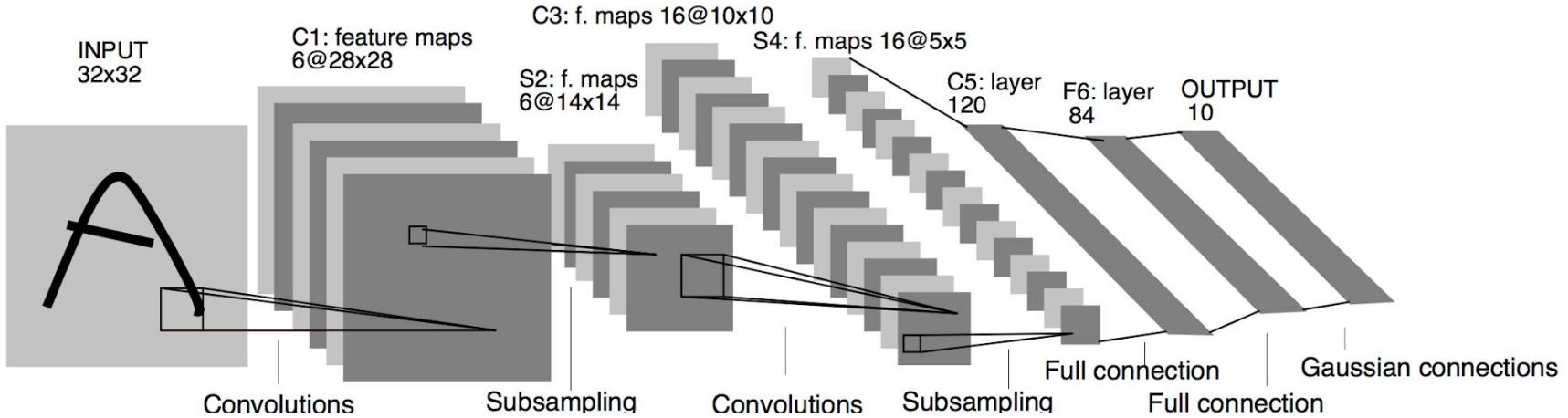
Layer (type)	Output Shape
<hr/>	
conv2d_4 (Conv2D)	(None, 94, 94, 6)
average_pooling2d_4 (Average)	(None, 47, 47, 6)
conv2d_5 (Conv2D)	(None, 45, 45, 16)
average_pooling2d_5 (Average)	(None, 22, 22, 16)
flatten_2 (Flatten)	(None, 7744)
dense_6 (Dense)	(None, 120)
dense_7 (Dense)	(None, 84)
dense_8 (Dense)	(None, 30)

Total params: 943,054

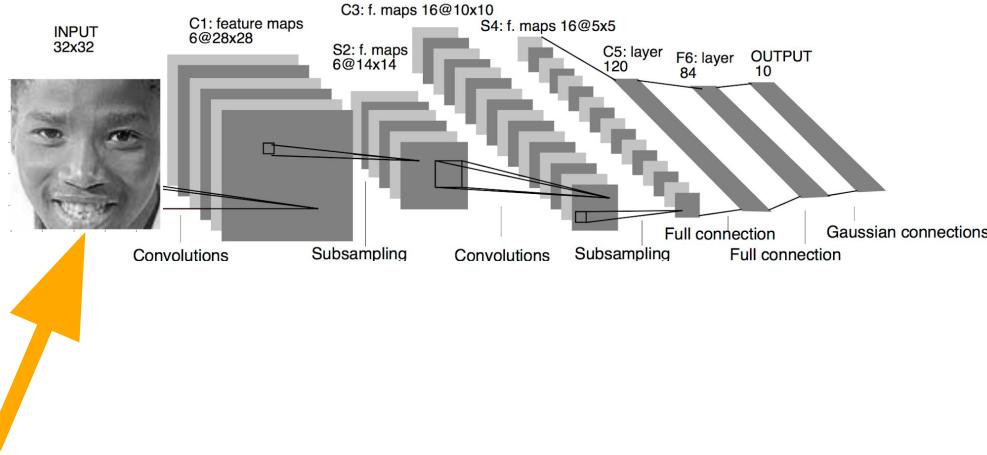
Trainable params: 943,054

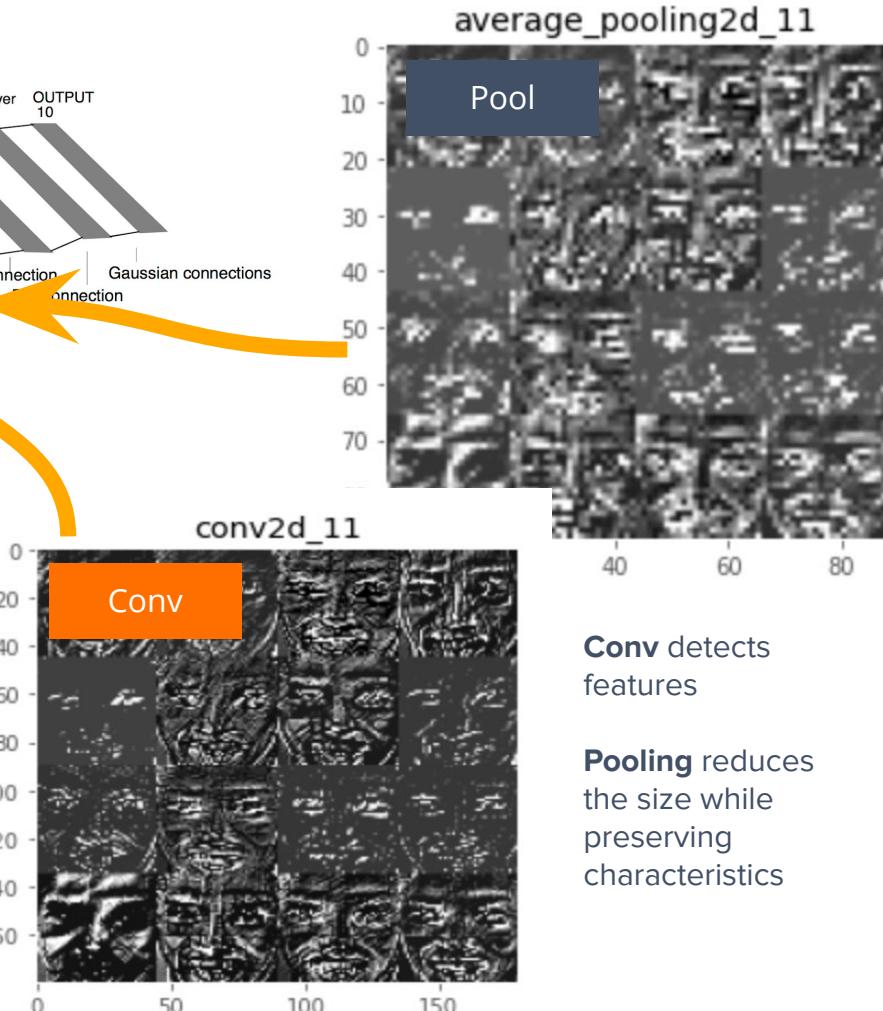
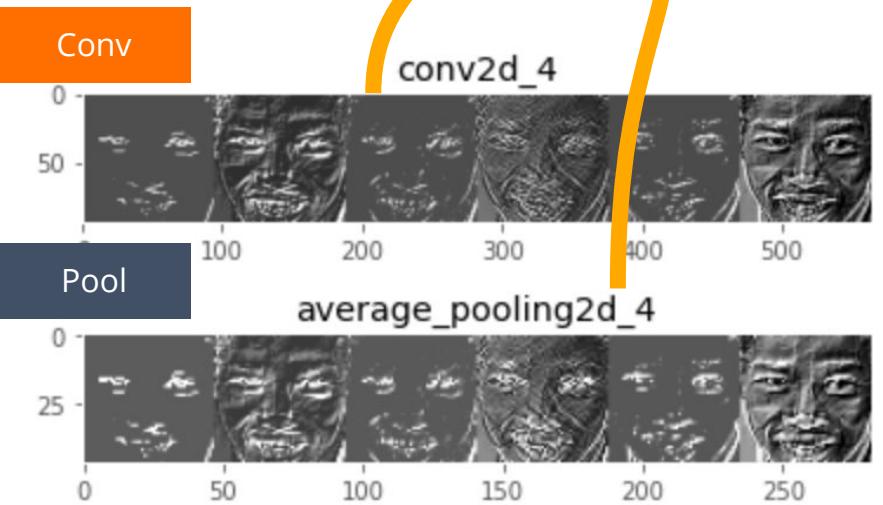
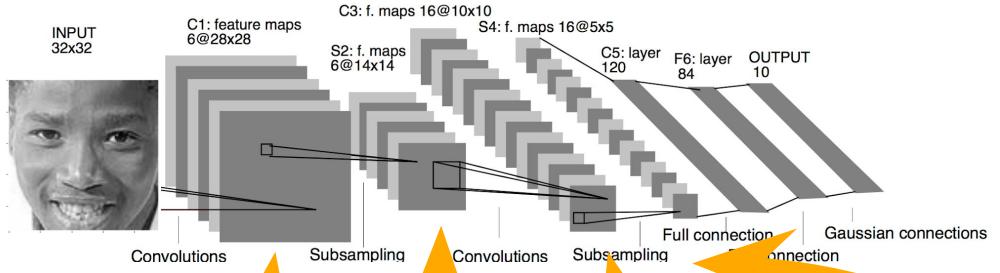
Non-trainable params: 0

None



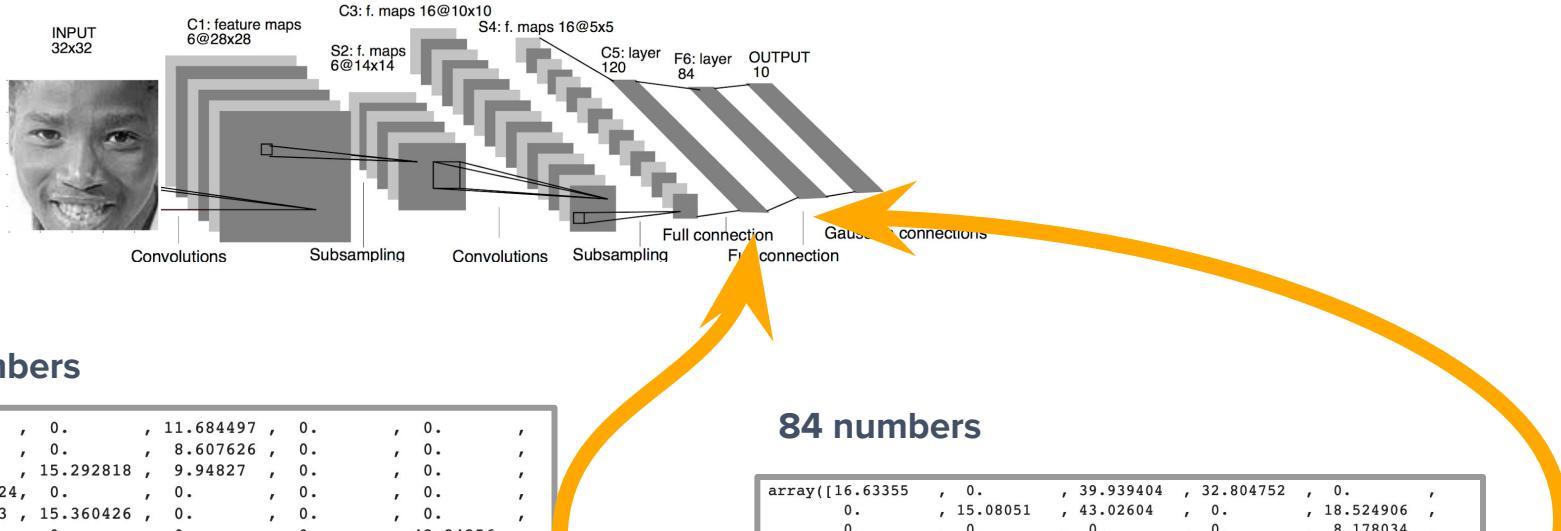
What does an ML model "see"?
LeNet5 activation layers





Conv detects features

Pooling reduces the size while preserving characteristics



120 numbers

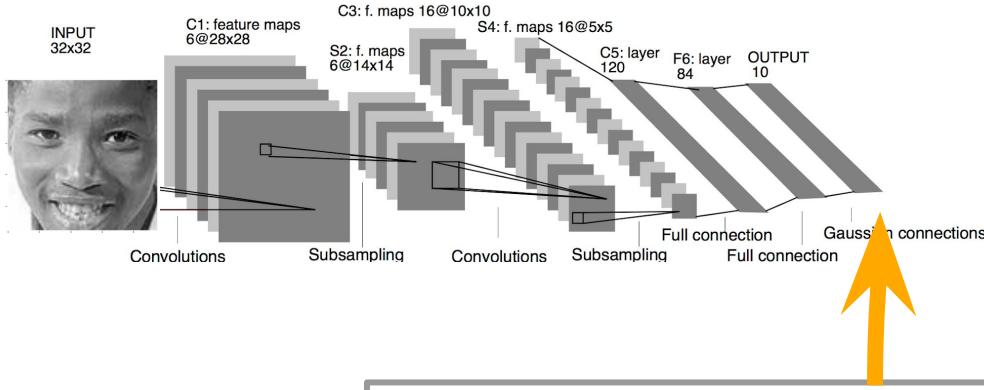
```
array([41.70368, 0., 11.684497, 0., 0., 0.,
       0., 0., 8.607626, 0., 0., 0.,
       0., 15.292818, 9.94827, 0., 0., 0.,
       6.8089824, 0., 0., 0., 0., 0.,
       16.973673, 15.360426, 0., 0., 0., 0.,
       9.47765, 0., 0., 0., 42.24256, 0.,
       12.297037, 0., 22.879719, 0., 0., 0.,
       6.487142, 0., 0., 10.376771, 0., 0.,
       0., 0., 17.08813, 0., 0., 0.,
       0., 0., 0., 11.450783, 12.42491, 0.,
       0., 8.61066, 17.49668, 0., 0., 0.,
       11.233514, 0., 0., 0., 0., 0.,
       12.334232, 0., 1.8129607, 0., 0., 0.,
       0., 9.862995, 10.680107, 0., 18.863989, 0.,
       0., 0., 13.35074, 0., 0., 0.,
       14.71458, 0., 0., 16.032688, 0., 0.,
       10.024964, 7.21192, 0., 0., 0., 0.,
       6.149487, 0., 0., 0., 0., 0.,
       10.679127, 0., 0., 0., 0., 0.,
       0., 11.183755, 0., 0., 4.282765, 0.,
       19.137138, 0., 0., 15.075655, 0., 0.,
       0., 0., 25.42046, 18.232054, 13.407069, 0.,
       0., 16.235718, 0., 0., 0., 0.,
       10.614934, 0., 0., 11.432421, 0., 0.],
      dtype=float32)
```

84 numbers

```
array([16.63355, 0., 39.939404, 32.804752, 0.,
       0., 15.08051, 43.02604, 0., 18.524906,
       0., 0., 0., 0., 0., 8.178034,
       0., 0., 0., 0., 2.4489286, 0.,
       54.874302, 0., 0., 0., 42.76598, 0.,
       42.408684, 48.62112, 0.4575051, 0., 0.,
       0.21308264, 0., 0., 0., 0., 0.,
       39.96471, 0., 22.610771, 46.044735, 0.2078133,
       32.51241, 33.131916, 0., 0., 33.57893, 0.,
       0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 46.528843, 0.,
       0., 26.983, 39.674263, 37.154236, 0.,
       34.24934, 0., 20.362234, 0., 30.49593, 0.,
       0., 0., 0., 0., 0., 0.,
       0., 1.4448502, 25.033195, 29.471706, 38.308865,
       0., 0., 2.6136963, 0., 0., 0.,
       0., 0., 0., 30.762205], dtype=float32)
```

All we know is that
number means ac

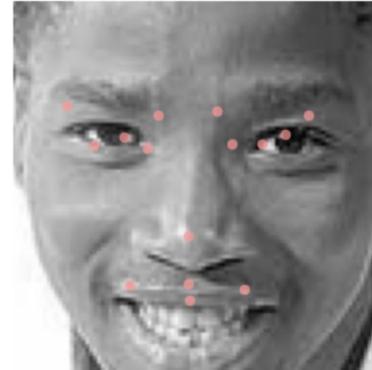
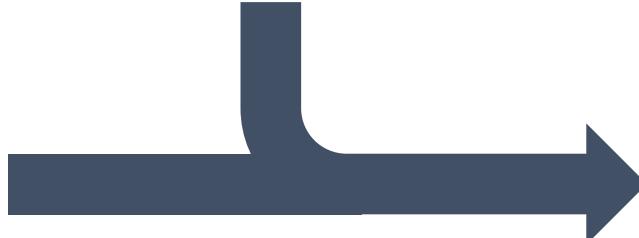


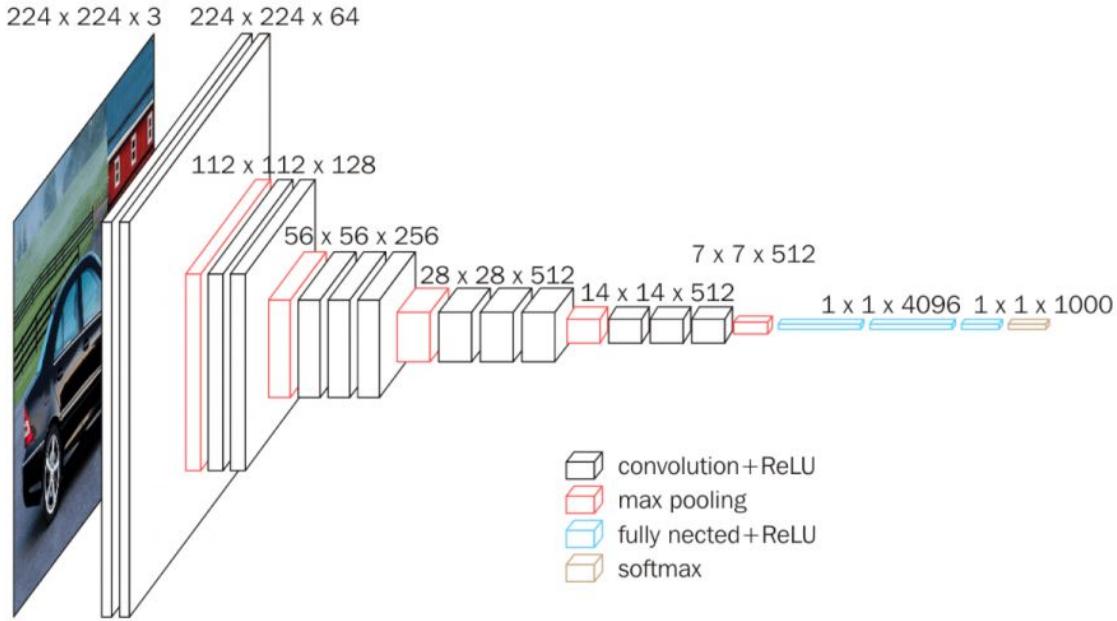


Output:

30 numbers for 15 KP

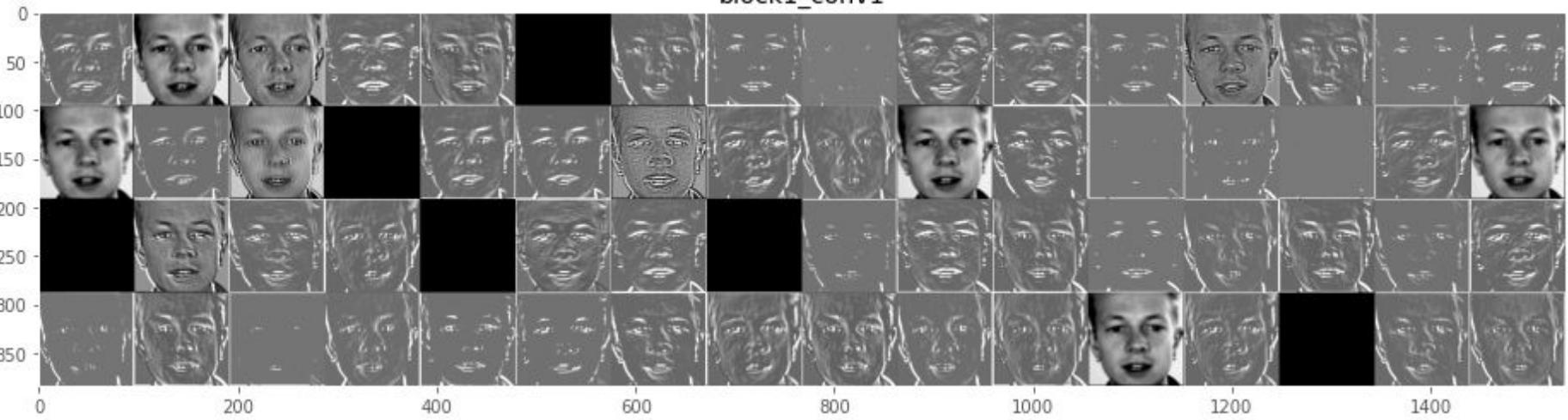
```
array([64.37013 , 36.601883, 28.56574 , 34.96842 , 56.406677, 36.769264,
       70.383644, 34.404564, 34.708054, 37.653595, 20.951923, 37.264774,
      52.6065 , 28.457956, 76.40988 , 29.269268, 37.388195, 29.48214 ,
     13.806782, 26.73549 , 45.301773, 60.582985, 59.48615 , 74.35807 ,
    30.009727, 73.12003 , 45.17569 , 72.86695 , 45.625507, 77.04493 ],
      dtype=float32)
```





More activation layers!
VGG-16

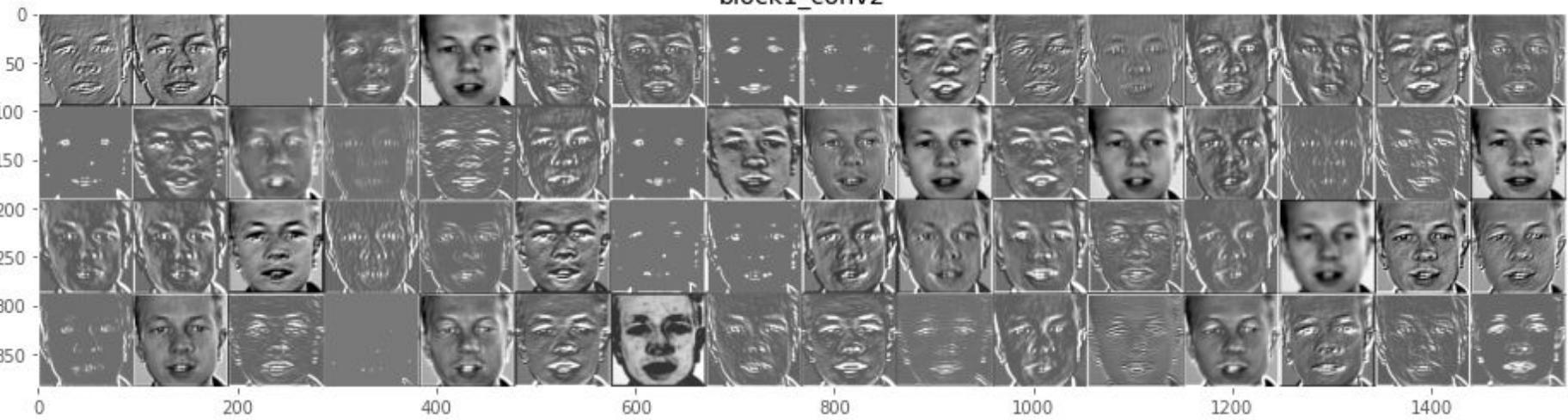
block1_conv1



Conv

(Identify important characteristics)

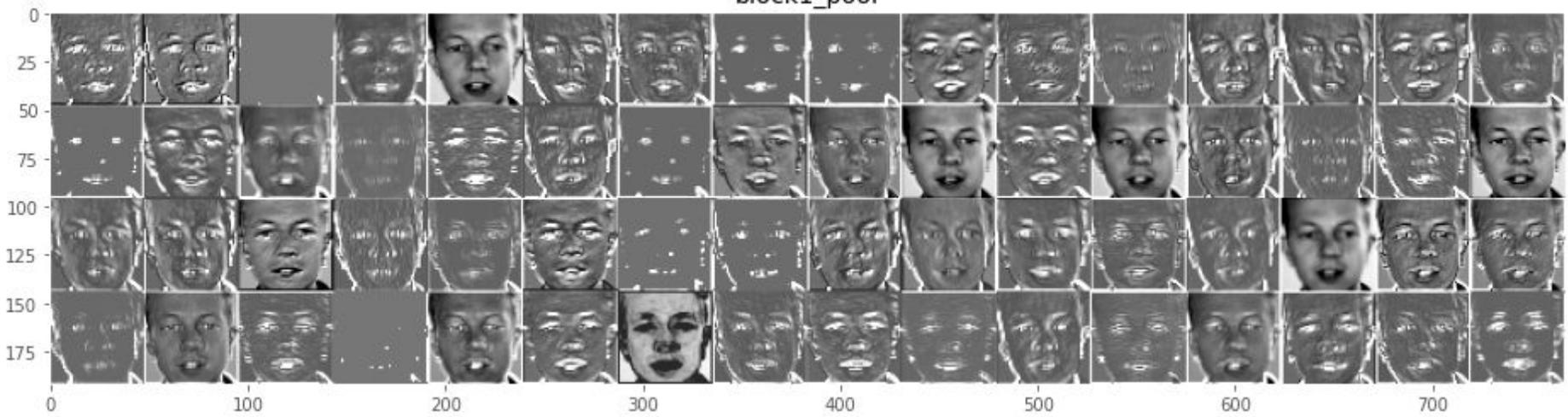
block1_conv2



Conv

(Identify important characteristics)

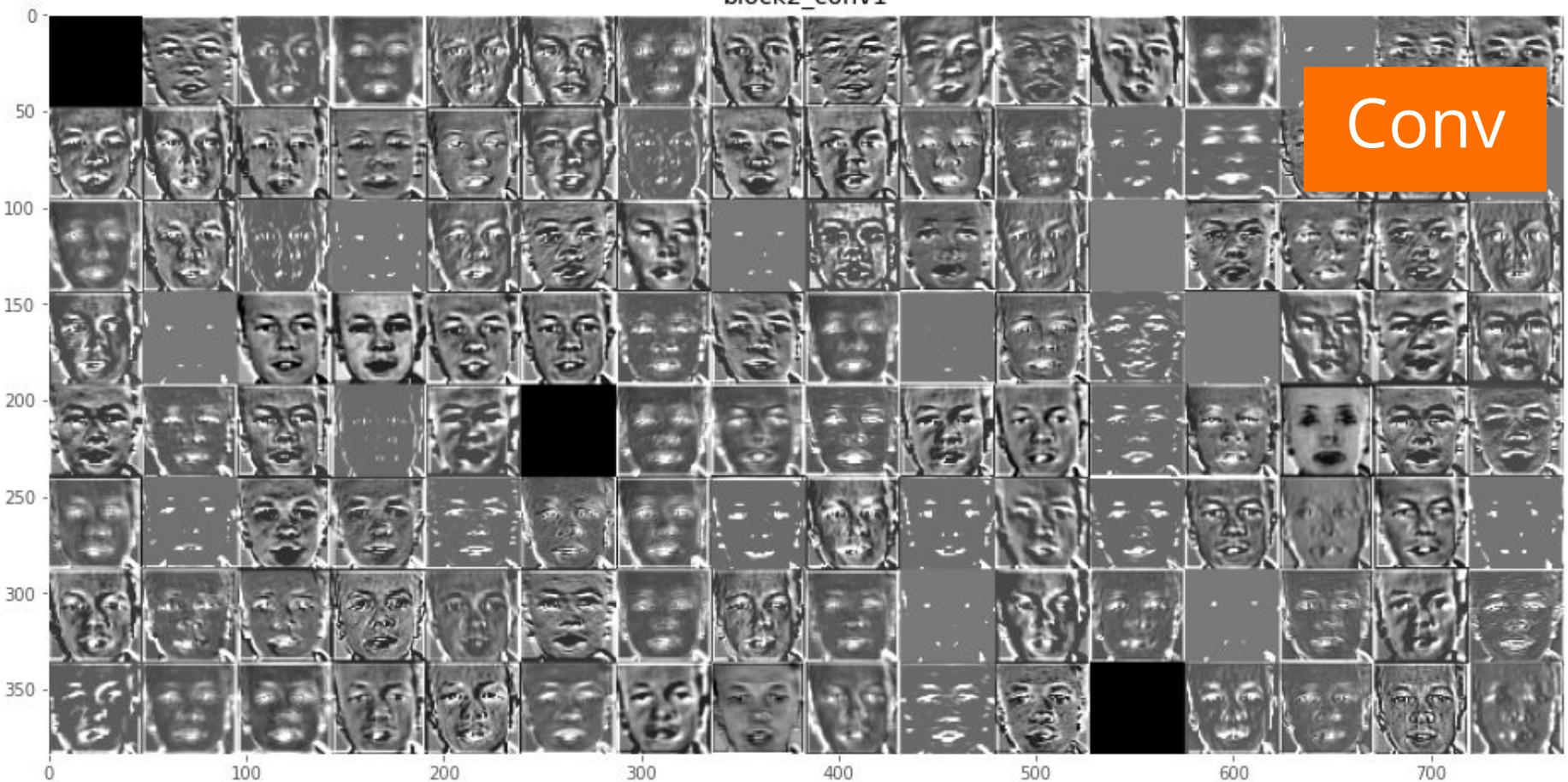
block1_pool



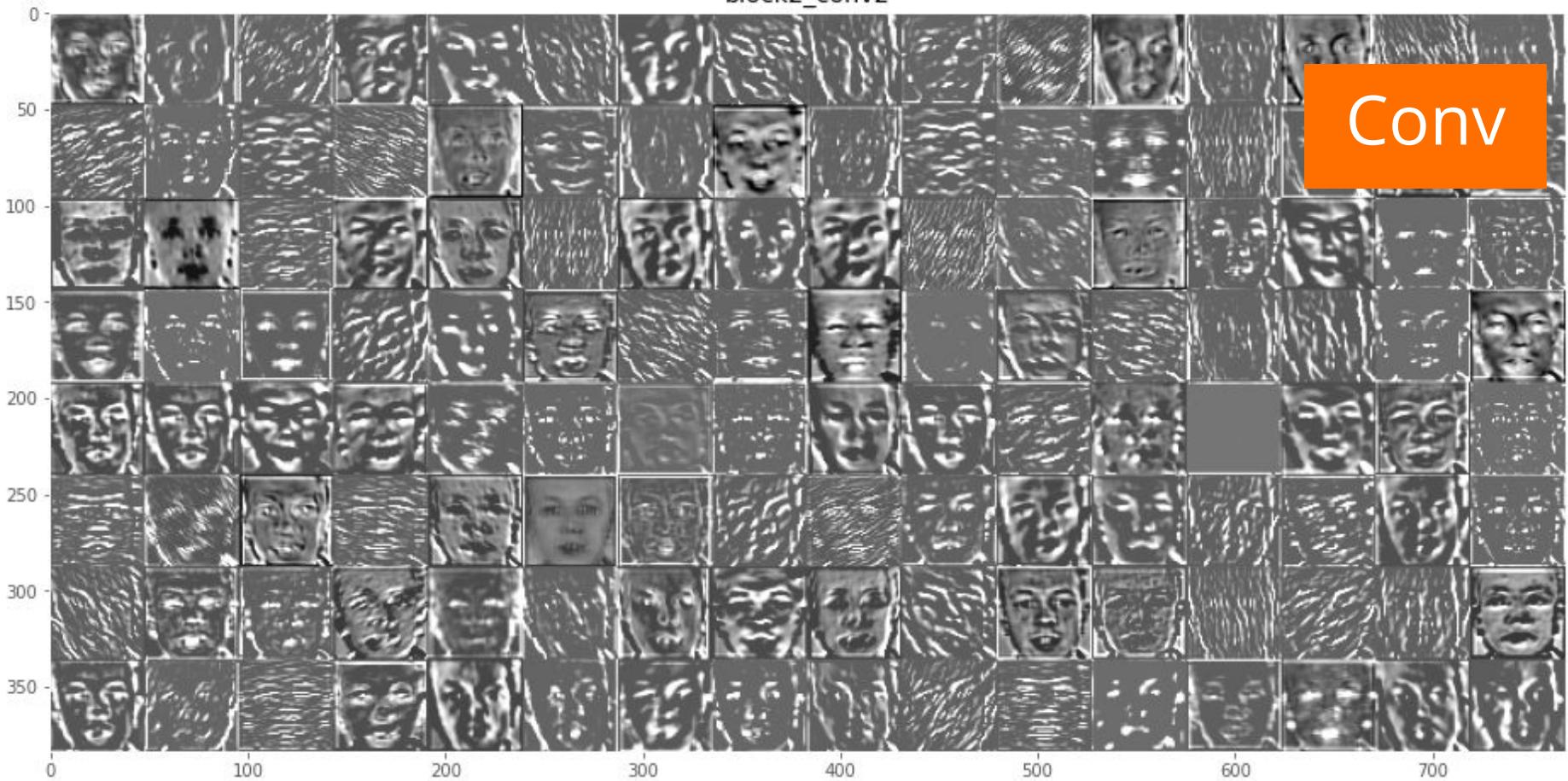
Pool

(Preserves characteristics, lower res)

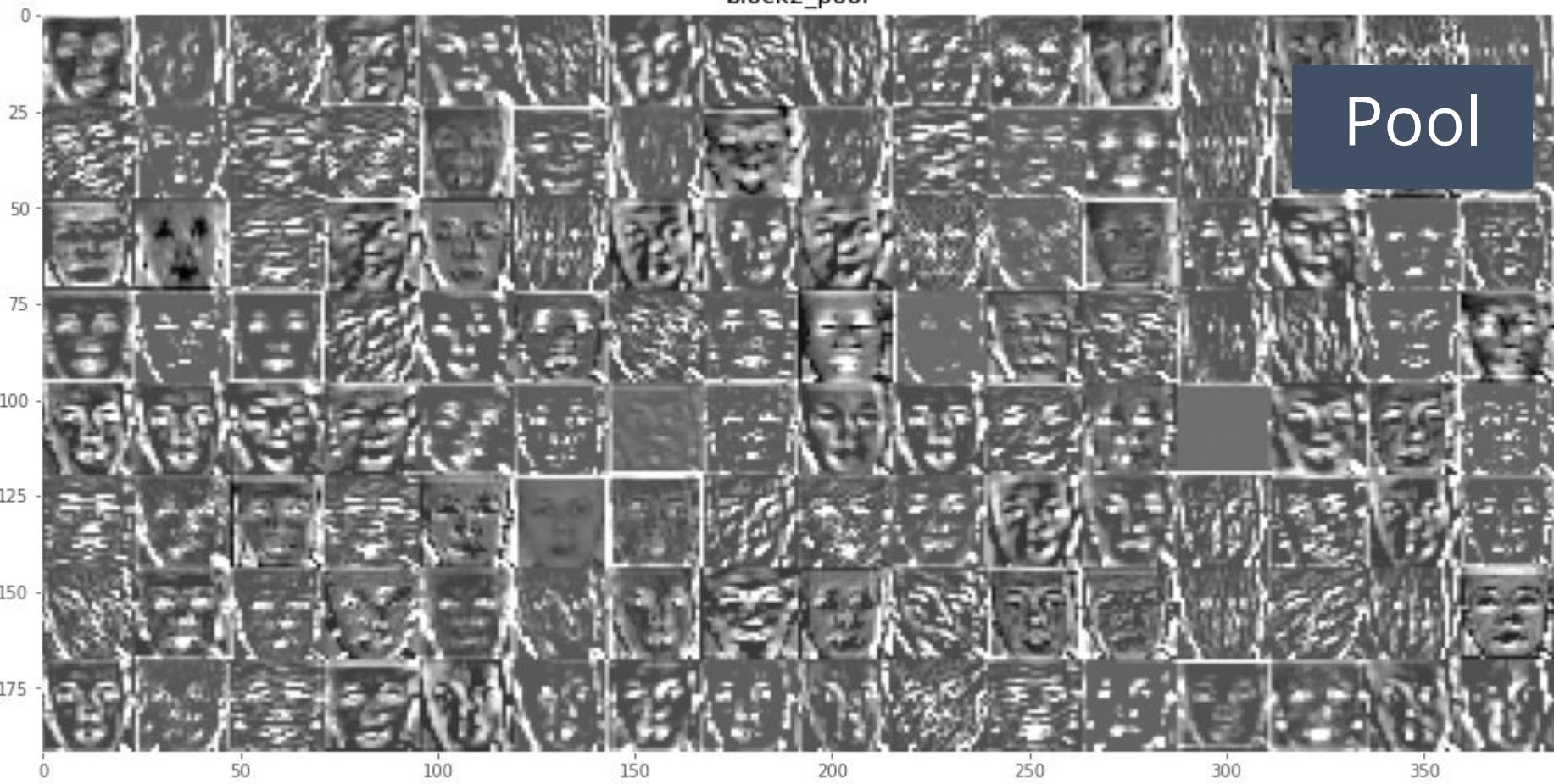
block2_conv1



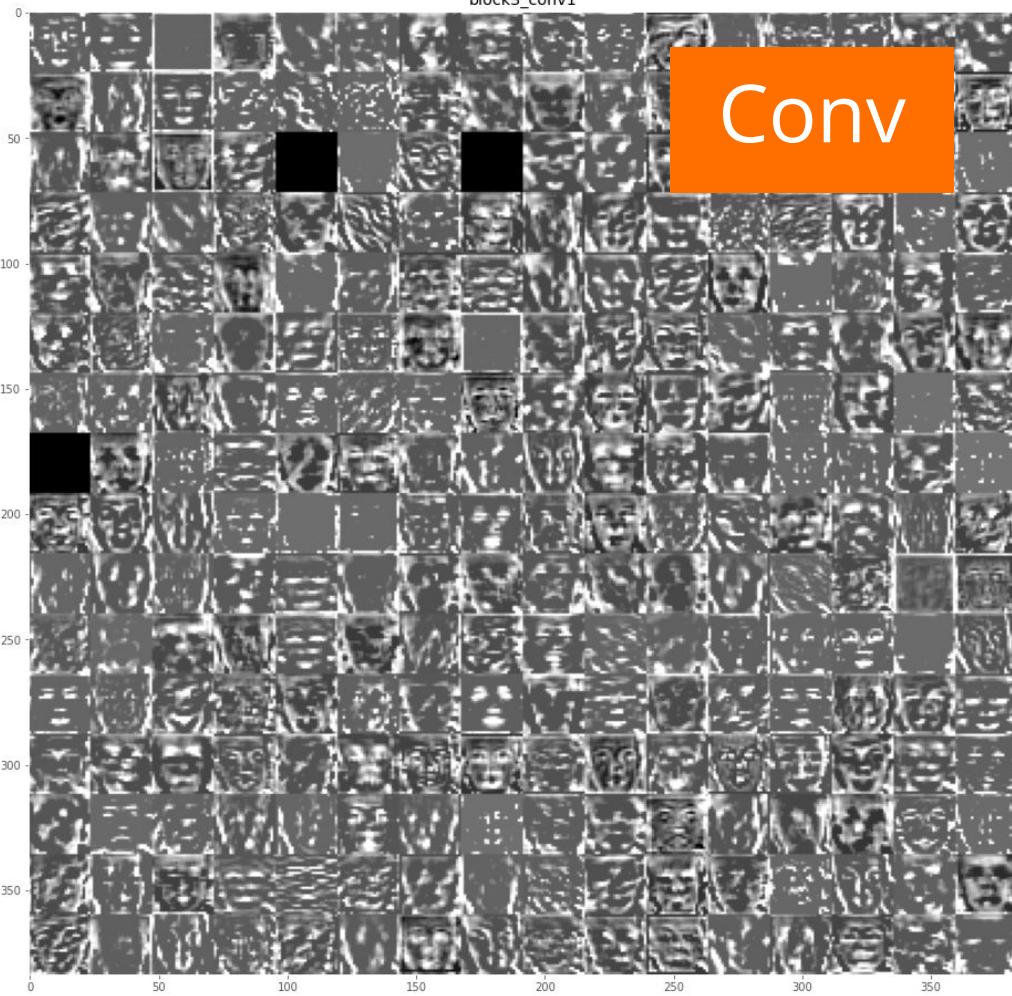
block2_conv2



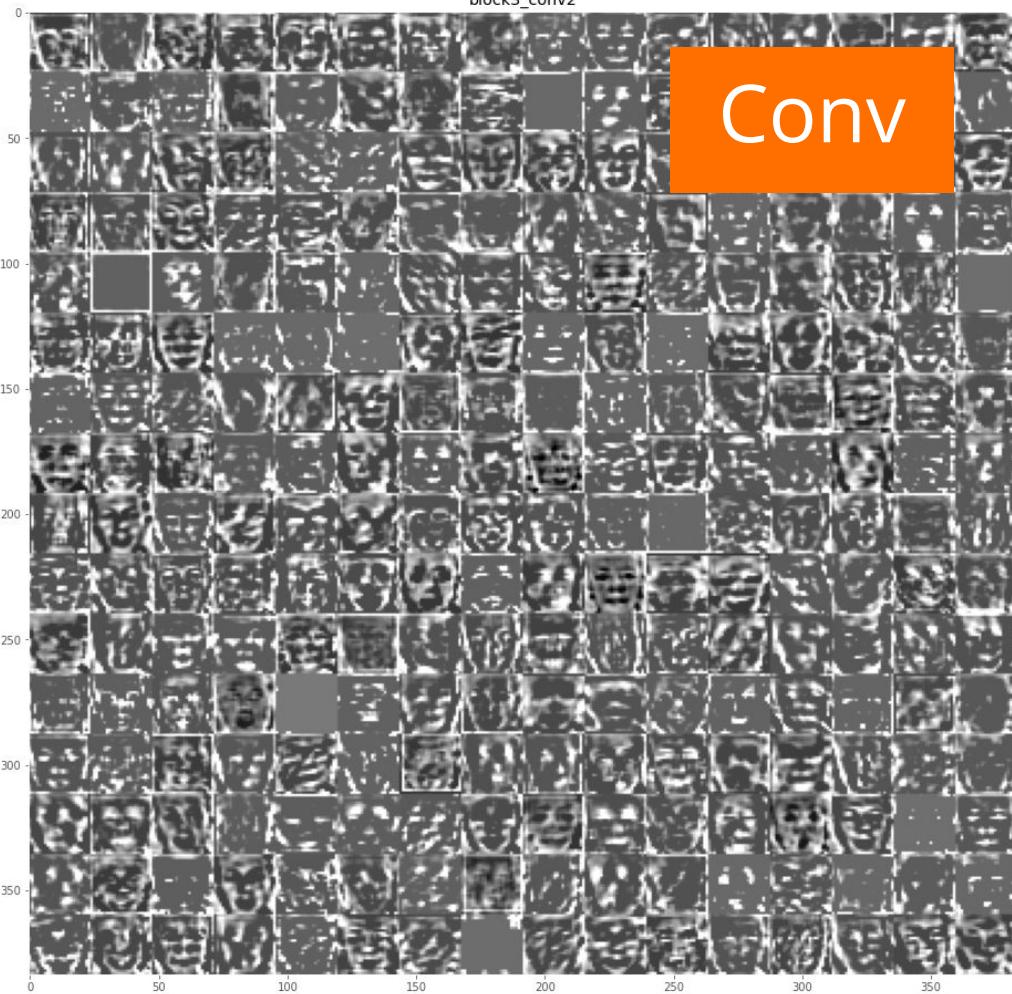
block2_pool



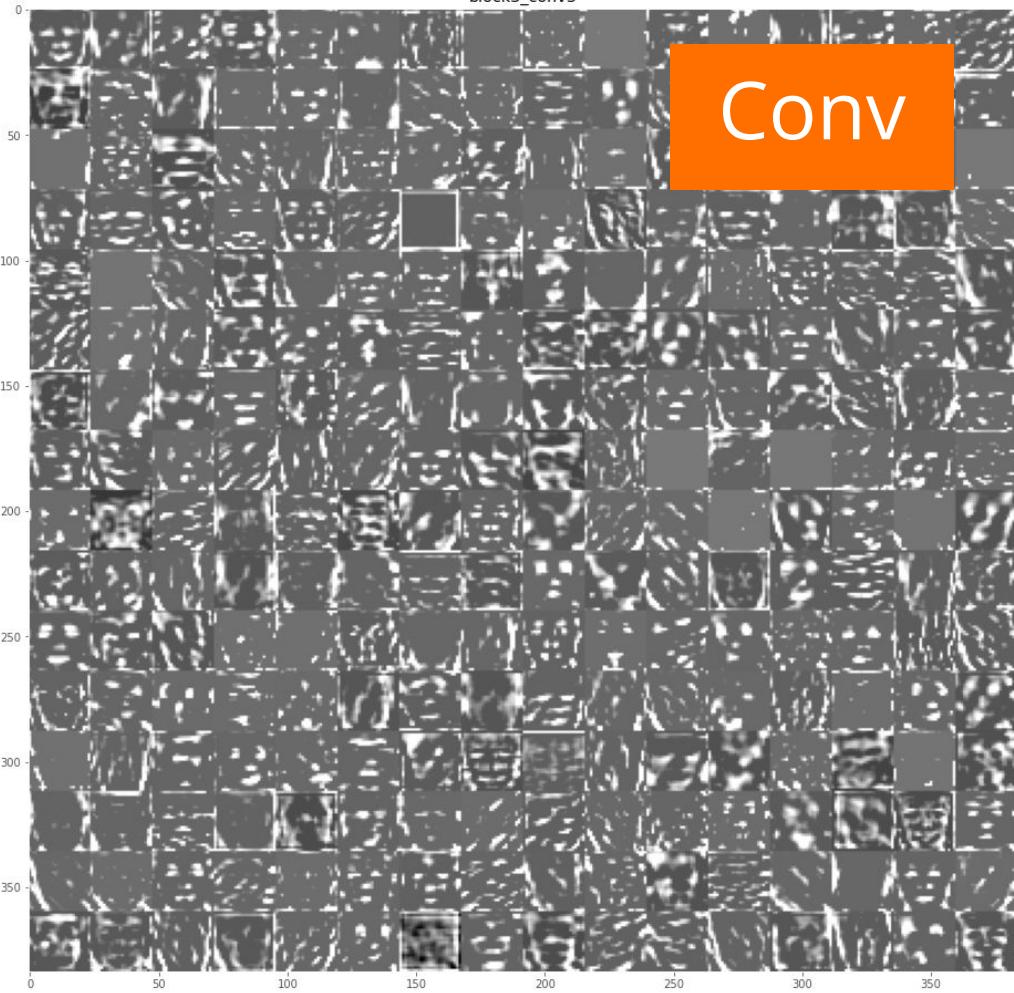
block3_conv1



block3_conv2



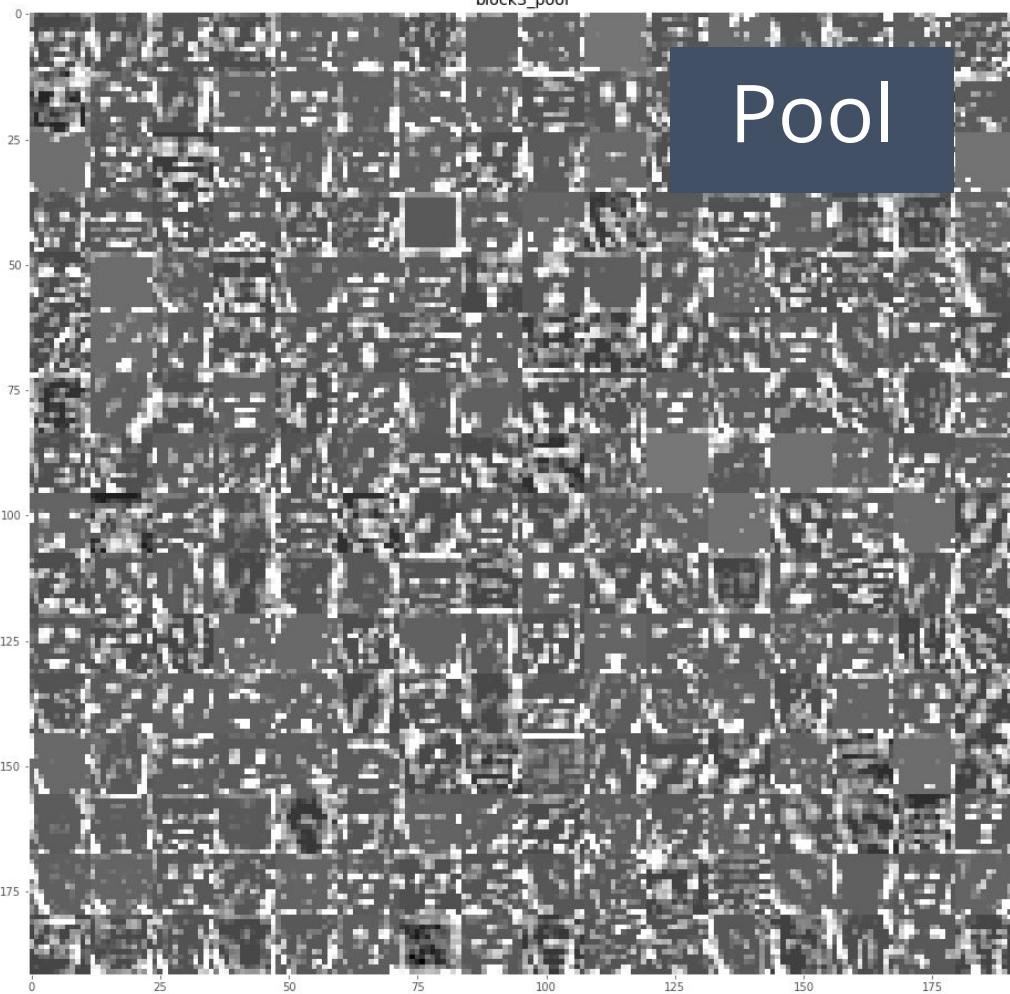
block3_conv3



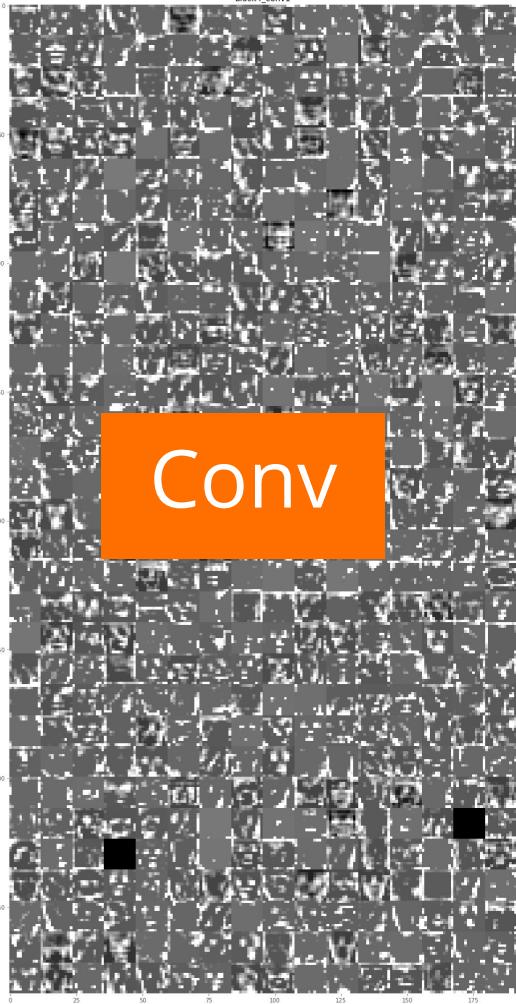
You can kinda see a face
or facial features.

block3_pool

Pool



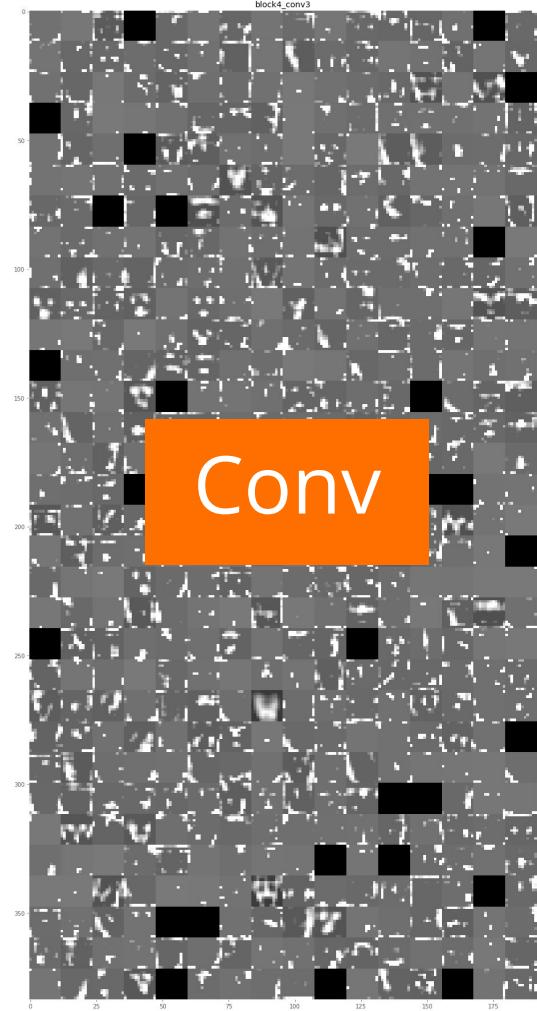
block4_conv1



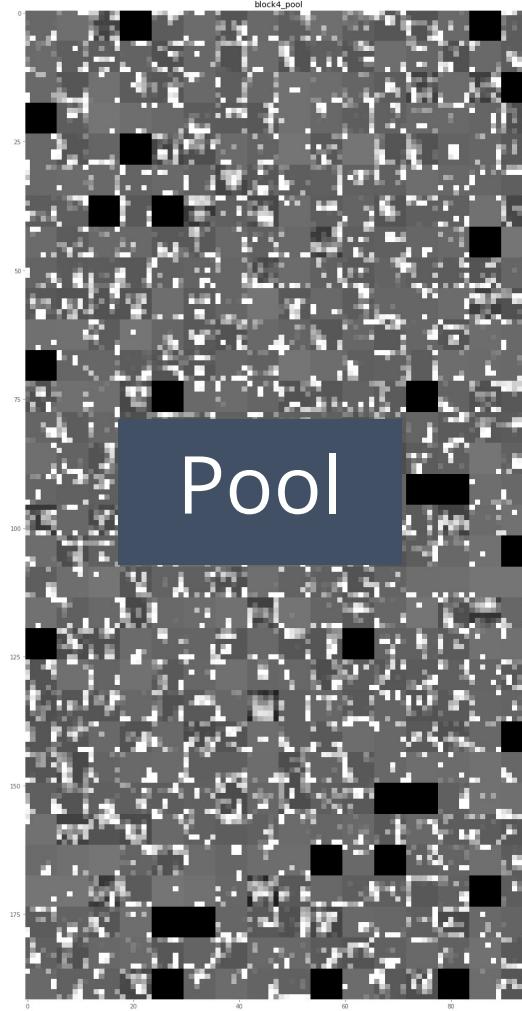
block4_conv2



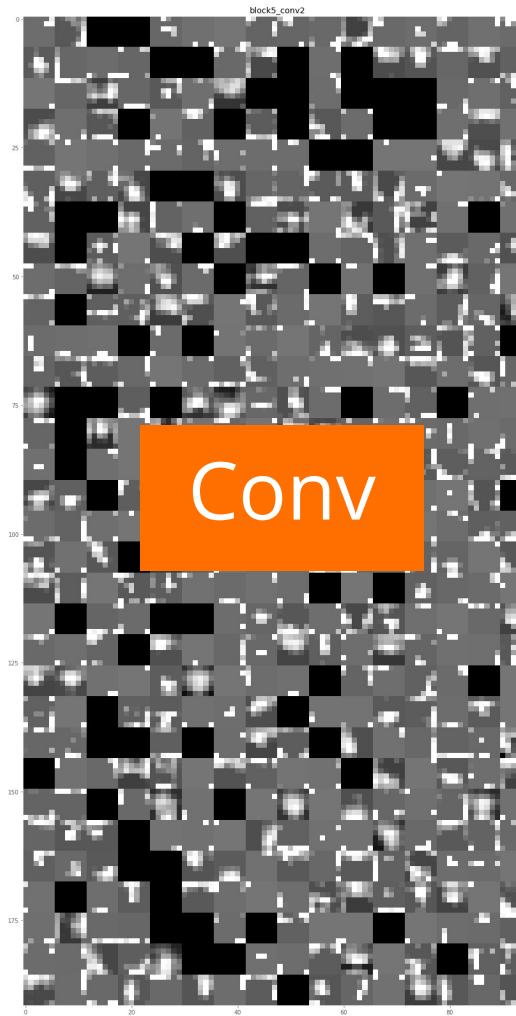
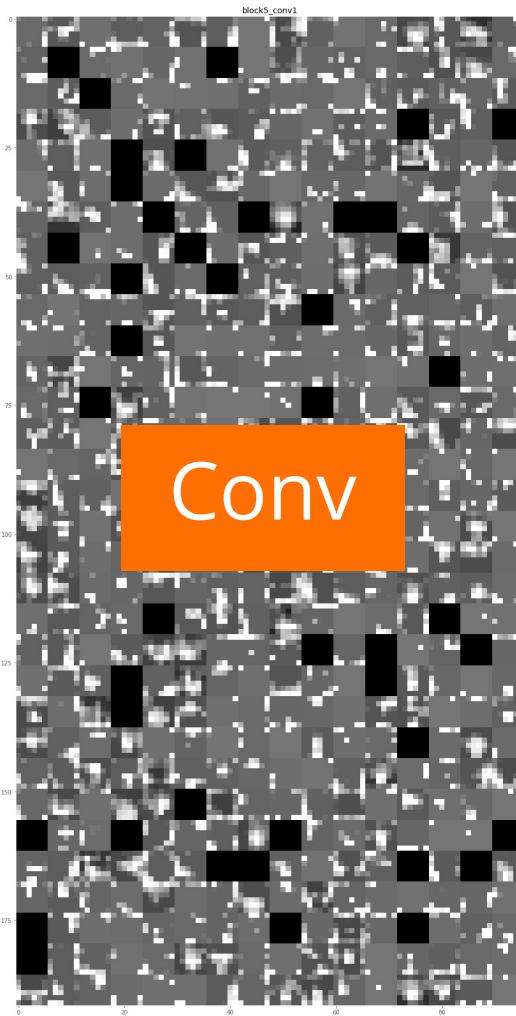
block4_conv3



block4_pool

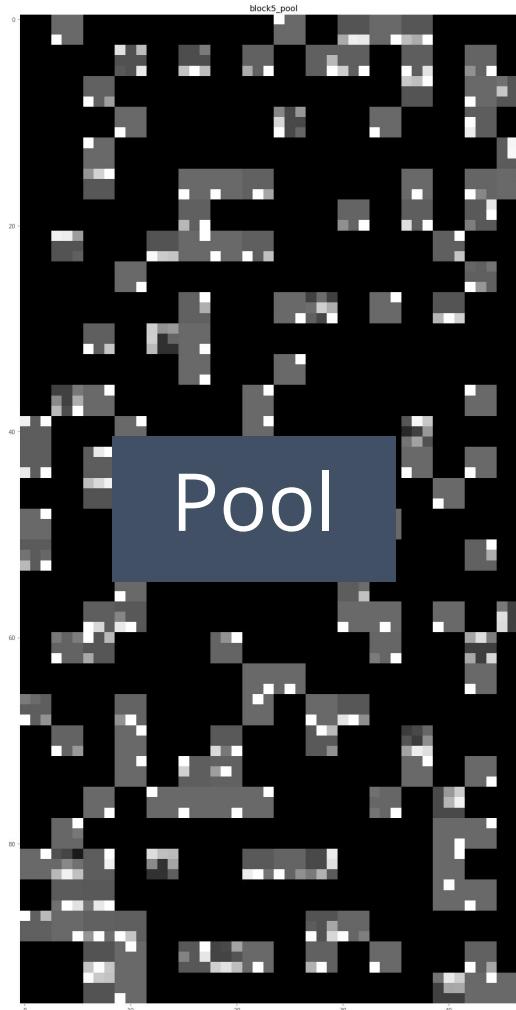
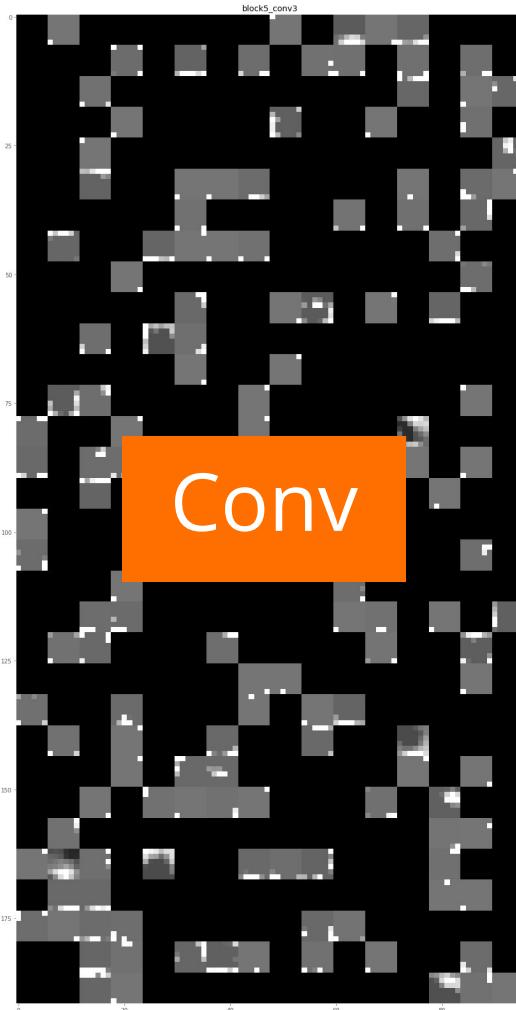


Now, each image is
hardly a face.



Each "image" less meaningful to humans.

Closer to vectors of numbers, not images.





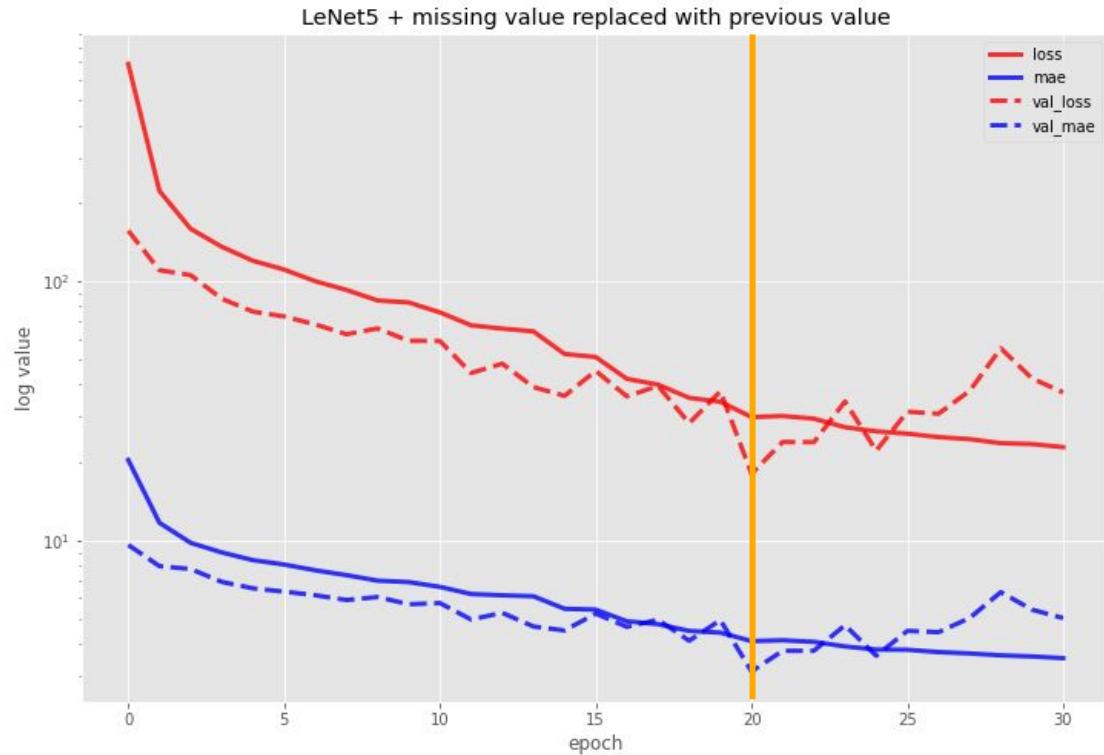
CNN Models

Procedure

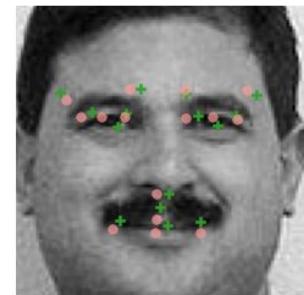
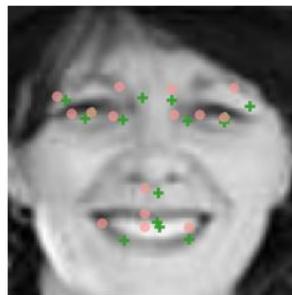
- Neural Network weights are randomized in the beginning, bad initial values may lead to high loss
- Created and trained 50 models for each variant
 - For each model, train up to 200 epochs
 - Use an callback to save check points at each epoch, and to track the lowest val_loss achieved for the variant
 - Use an EarlyStopping callback to stop training early and also prevent overfitting
 - Use regularization technique (drop-out) to prevent overfitting
 - Use Adam optimizer to speed up the convergence
- Used the weights at the epoch that had the minimal val_loss
- 6000 data points used from training set for training, remaining 1039 from the training set divided between training validation and final evaluation set.

LeNet-5 & ffill() Missing Data

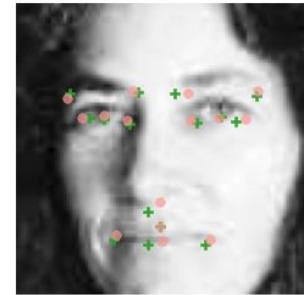
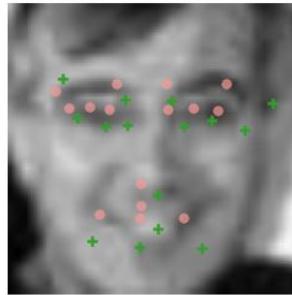
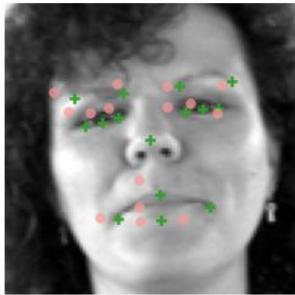
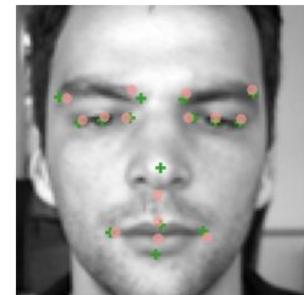
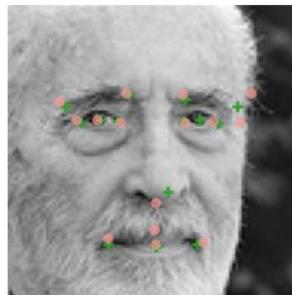
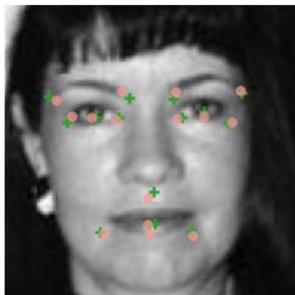
- Reached min val_loss at epoch 20
- Best performance during training:
 - loss ≈ 29.92
 - mae ≈ 4.08
 - val_loss ≈ 18.10
 - val_mae ≈ 3.12
- Evaluation of final model:
 - loss ≈ 20.01
 - MAE ≈ 3.29



LeNet-5 & ffill()

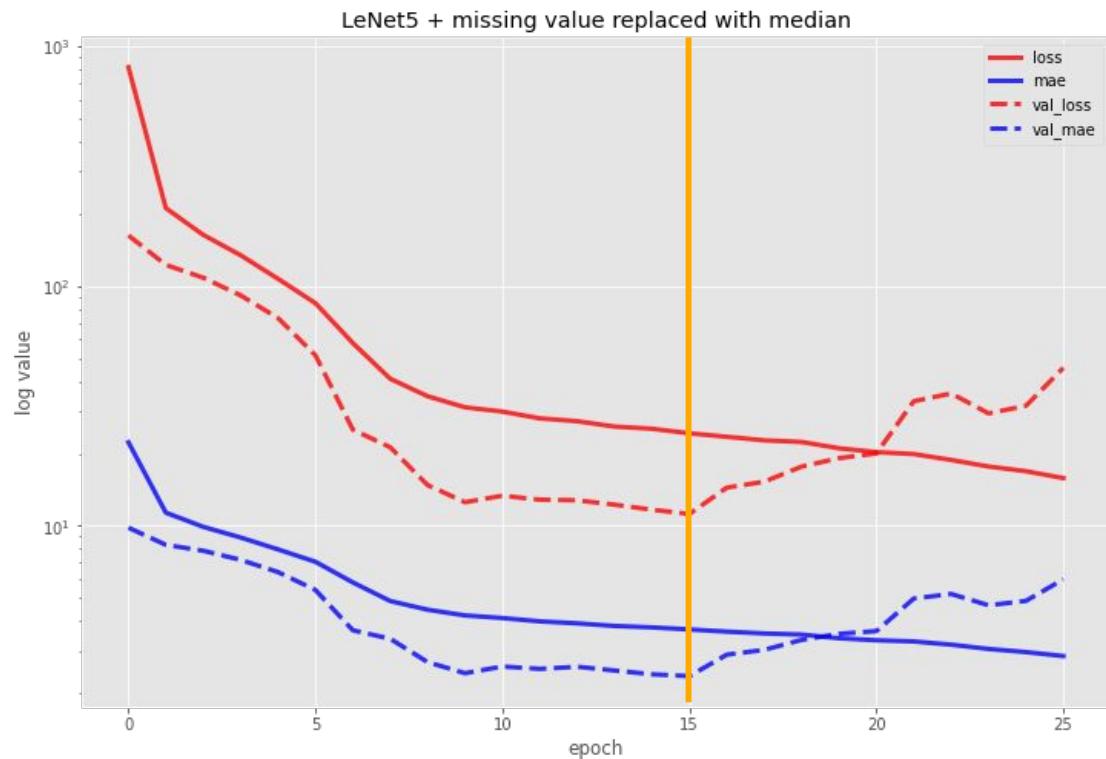


- + Dev output (contains filled missing values)
- Predicted output



LeNet-5 & Using Median for Missing Data

- Reached min val_loss at epoch 15
- Best performance during training:
 - loss ≈ 24.29
 - mae ≈ 3.69
 - val_loss ≈ 11.19
 - val_mae ≈ 2.36
- Evaluation of final model:
 - loss ≈ 13.61
 - MAE ≈ 2.52



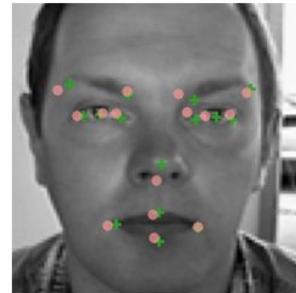
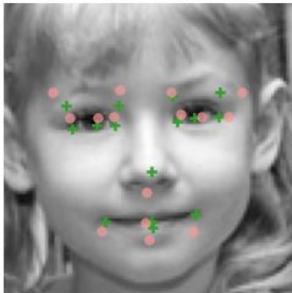
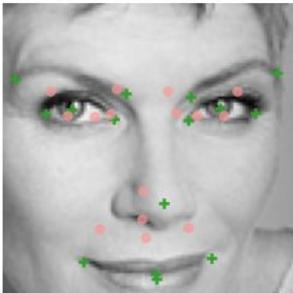
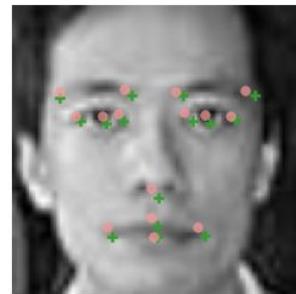
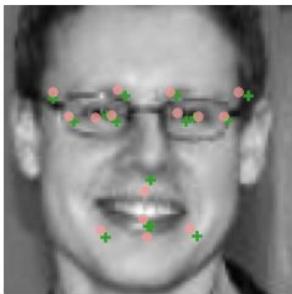
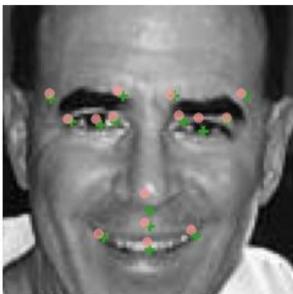
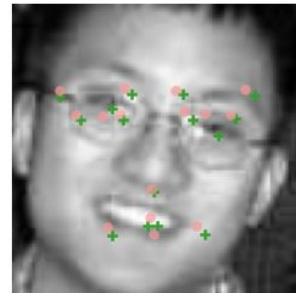
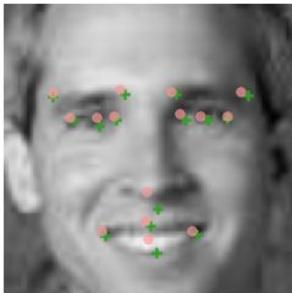
LeNet-5 & median



Dev output (contains filled
missing values)

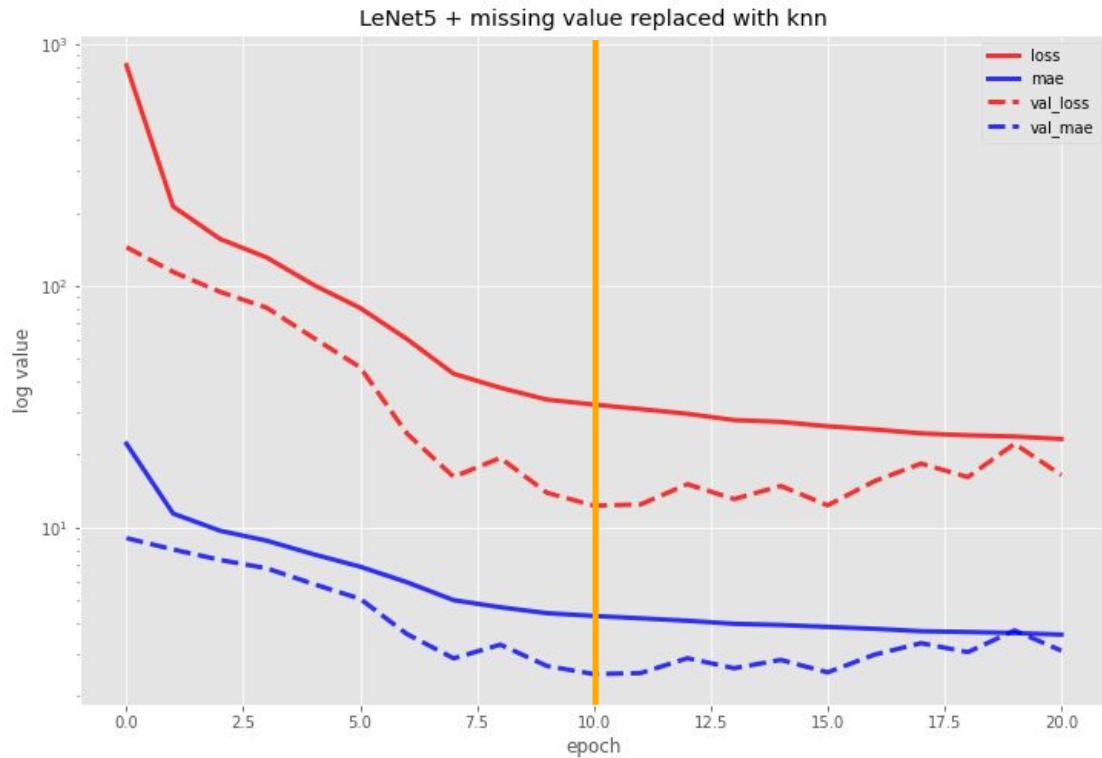


Predicted output

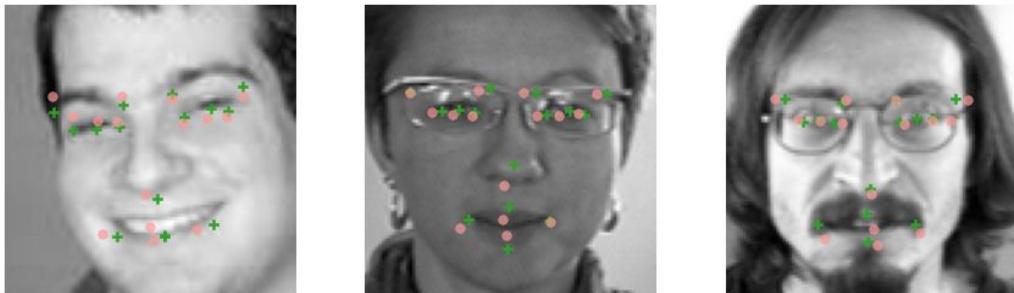


LeNet-5 & Using KNN to Fill Missing Data

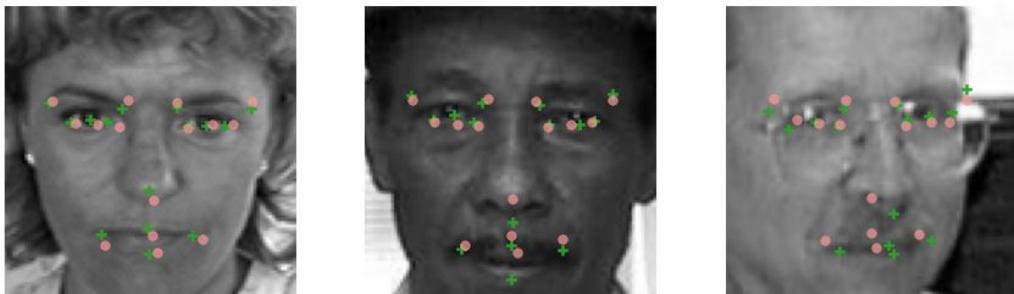
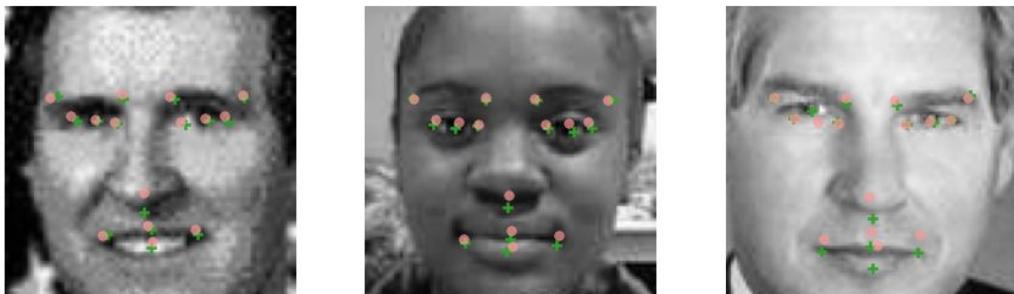
- Reached min val_loss at epoch 10
- Best performance during training:
 - loss ≈ 32.22
 - mae ≈ 4.30
 - val_loss ≈ 12.30
 - val_mae ≈ 2.47
- Evaluation of final model:
 - loss ≈ 19.95
 - MAE ≈ 2.83



LeNet-5 & KNN

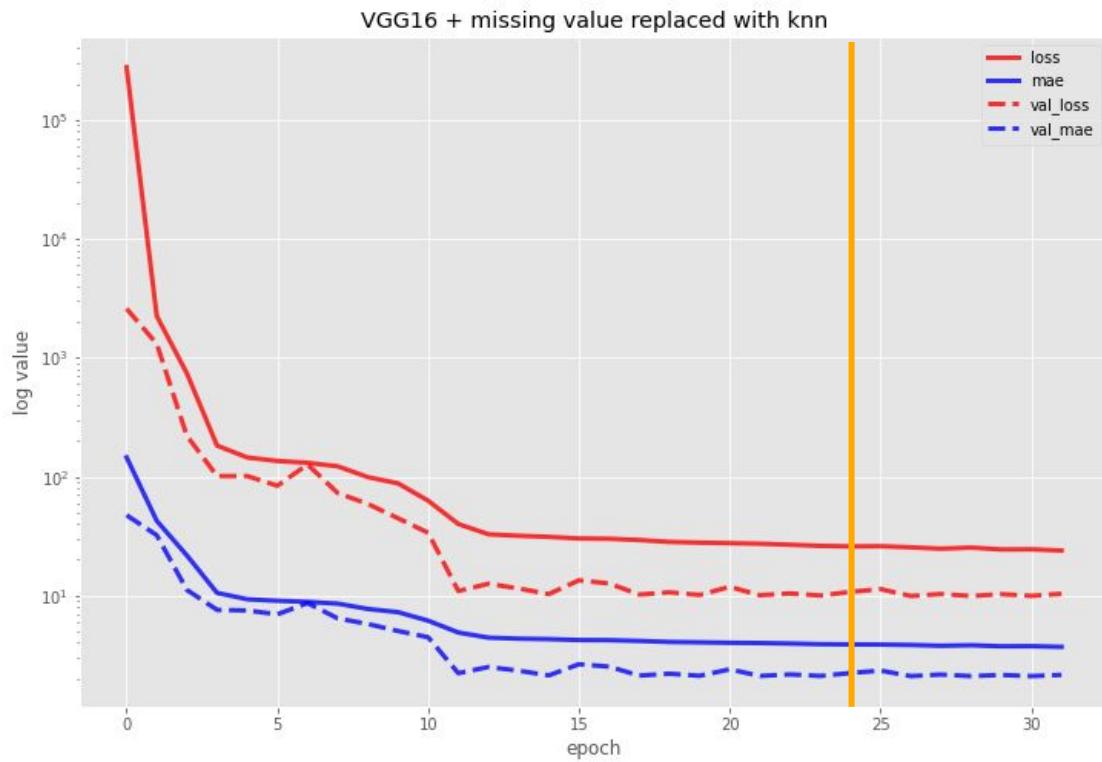


- ✚ Dev output (contains filled missing values)
- Predicted output



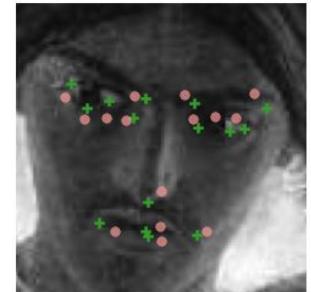
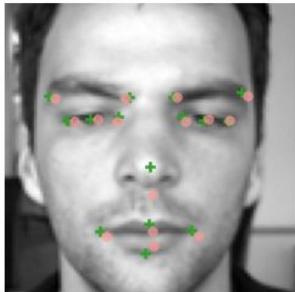
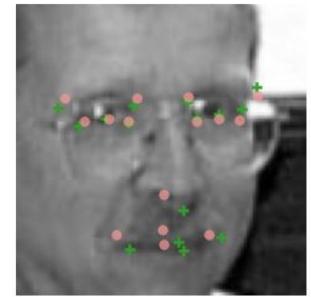
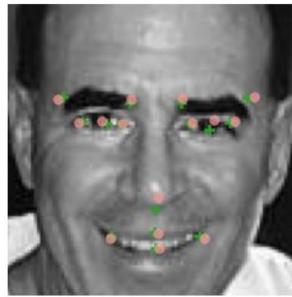
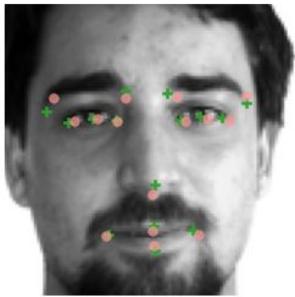
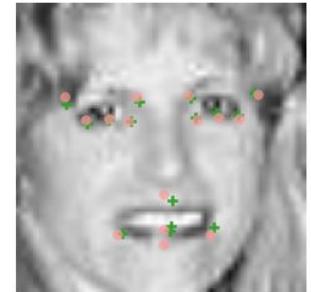
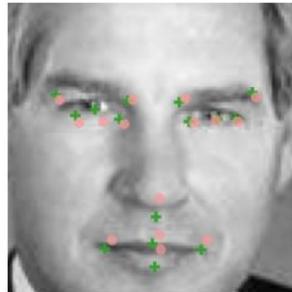
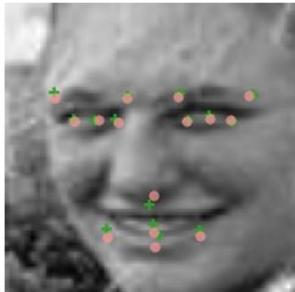
VGG-16 + KNN

- Reached min val_loss at epoch 24
- Best performance during training:
 - loss ≈ 28.19
 - mae ≈ 3.99
 - val_loss ≈ 9.79
 - val_mae ≈ 2.10
- Evaluation of final model:
 - loss ≈ 13.17
 - MAE ≈ 2.26



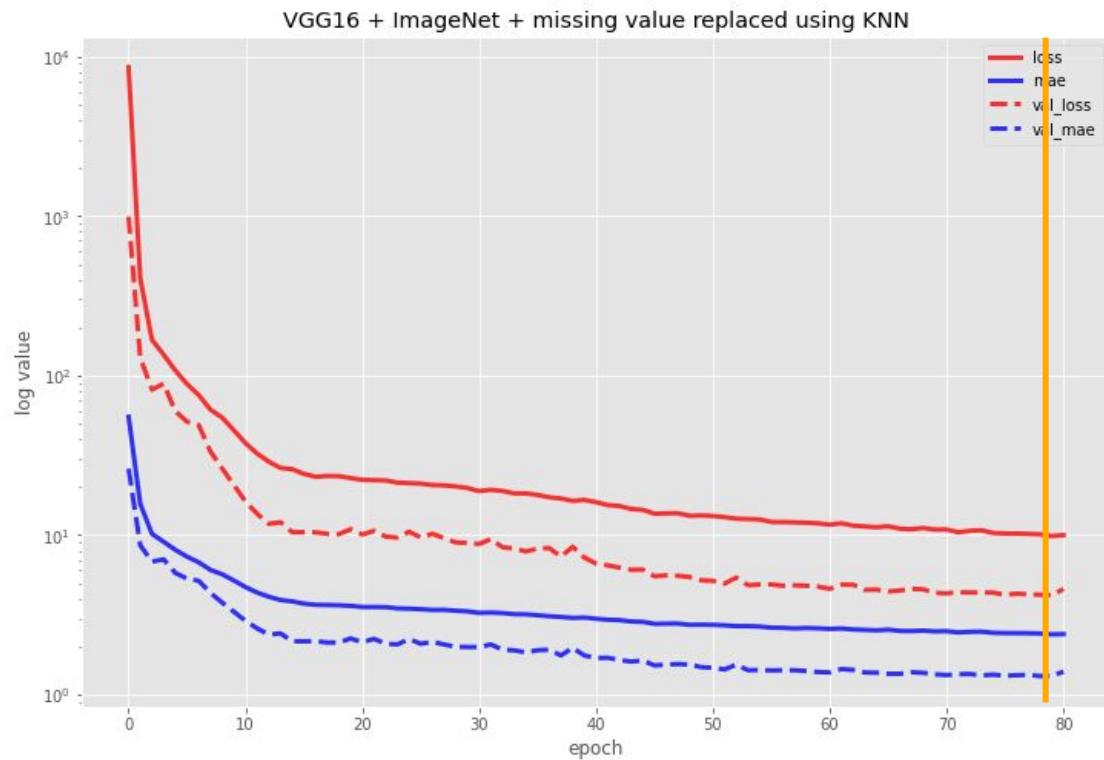
VGG16 & KNN

- ✚ Dev output (contains filled missing values)
- Predicted output

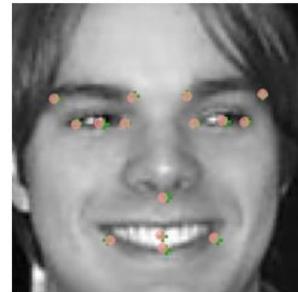
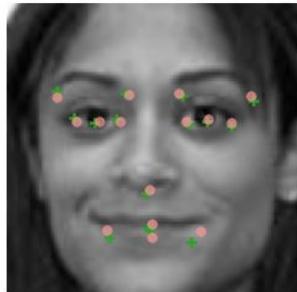
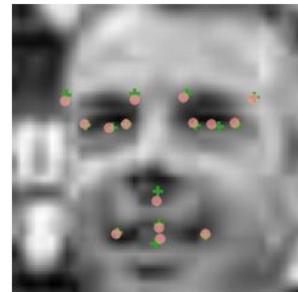
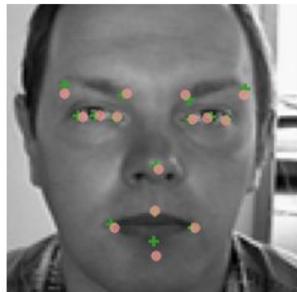
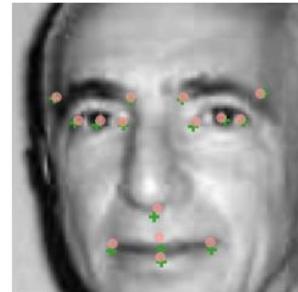
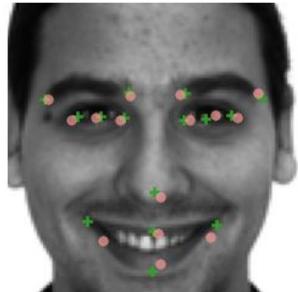
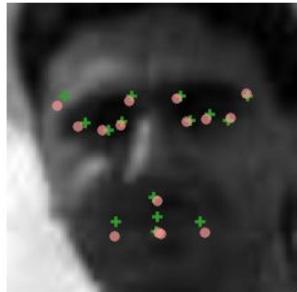


VGG-16 + Transfer Learning + KNN

- Reached min val_loss at epoch 79
- Best performance during training:
 - loss ≈ 9.86
 - mae ≈ 2.39
 - val_loss ≈ 4.20
 - val_mae ≈ 1.31
- Evaluation of final model:
 - loss ≈ 4.37
 - MAE ≈ 1.32



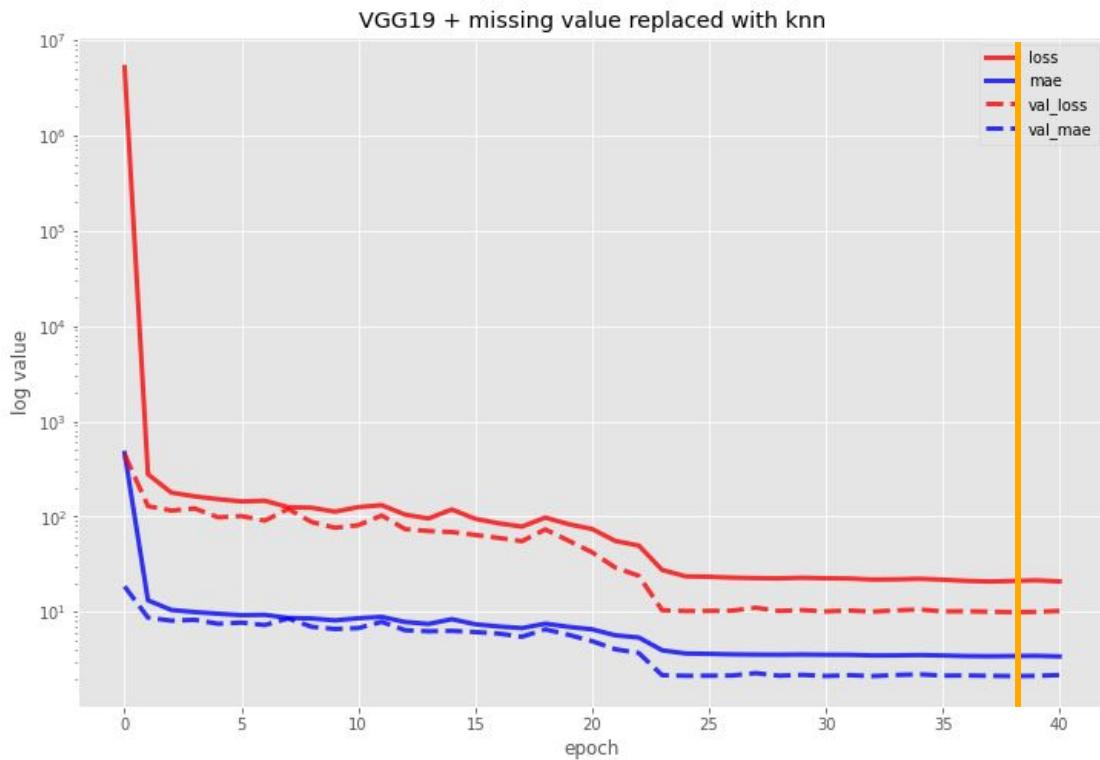
VGG16+Image Net & KNN



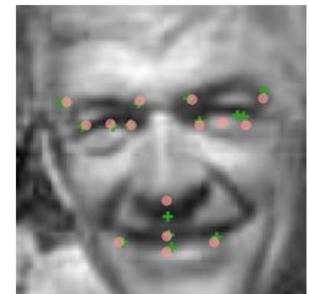
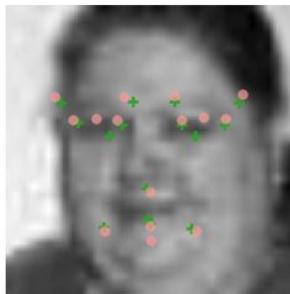
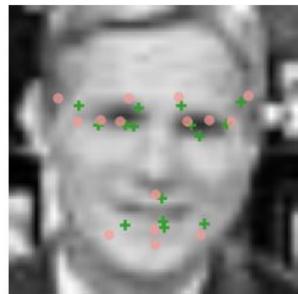
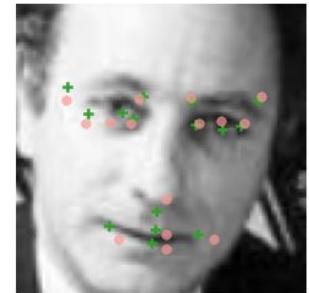
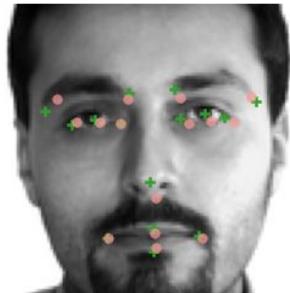
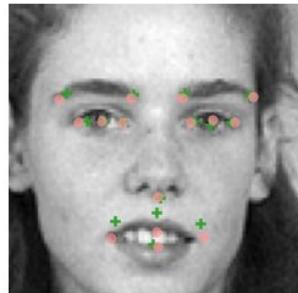
- ✚ Dev output (contains filled missing values)
- Predicted output

VGG-19 + KNN

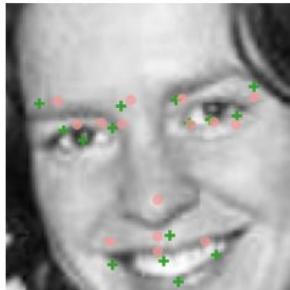
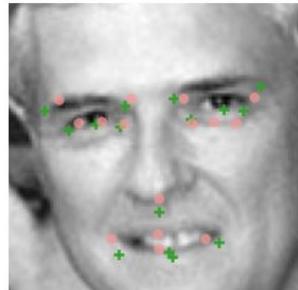
- Reached min val_loss at epoch 38
- Best performance during training:
 - loss ≈ 21.06
 - mae ≈ 3.43
 - val_loss ≈ 9.92
 - val_mae ≈ 2.11
- Evaluation of final model:
 - loss ≈ 13.18
 - MAE ≈ 2.28



VGG19+Image Net & KNN

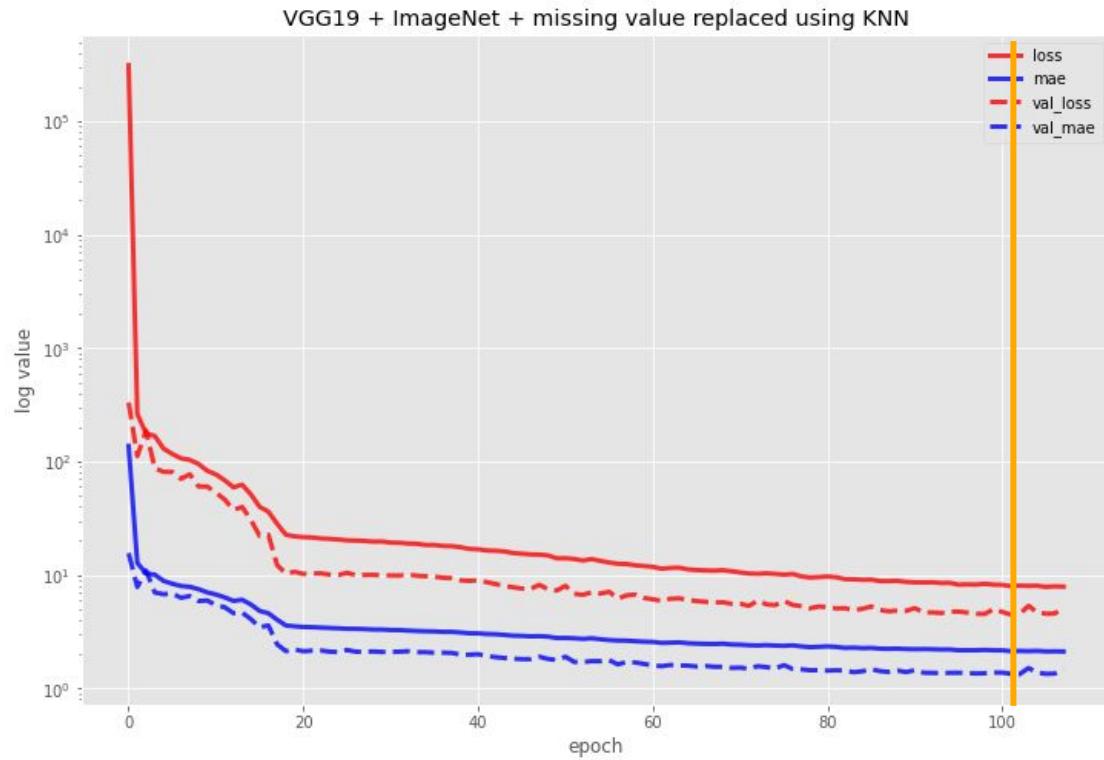


- ✚ Dev output (contains filled missing values)
- Predicted output

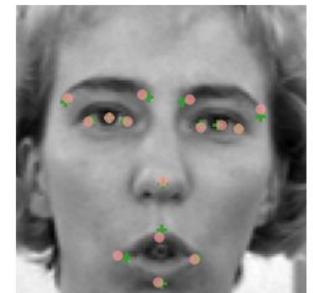
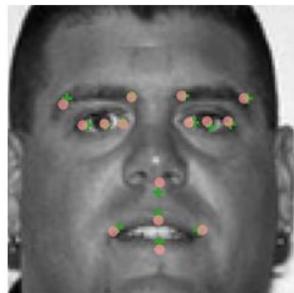
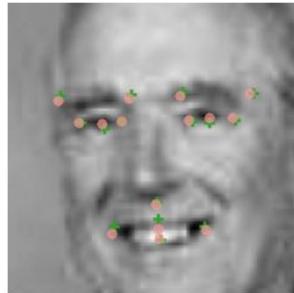
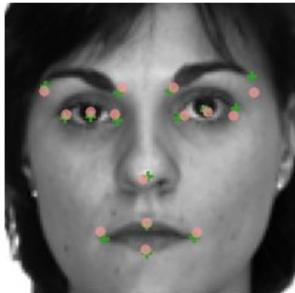


VGG-19 + Transfer Learning + KNN

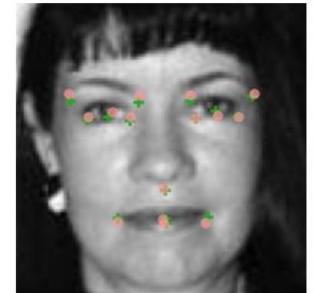
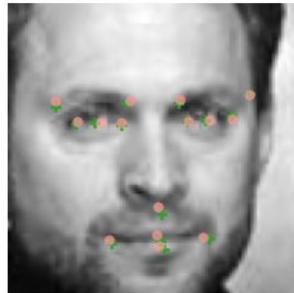
- Reached min val_loss at epoch 101
- Best performance during training:
 - loss ≈ 7.96
 - mae ≈ 2.13
 - val_loss ≈ 4.47
 - val_mae ≈ 1.34
- Evaluation of final model:
 - loss ≈ 4.29
 - MAE ≈ 1.34



VGG19+Image Net & KNN



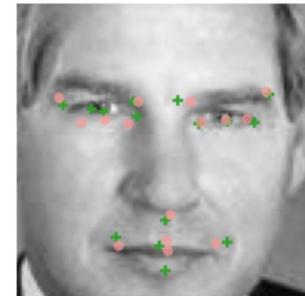
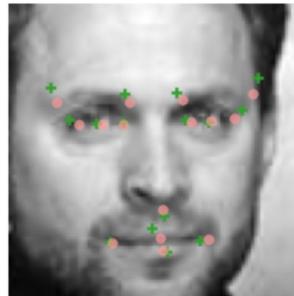
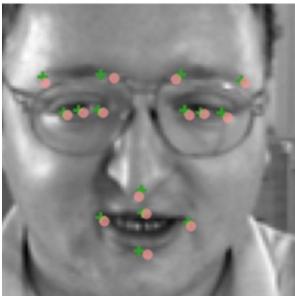
- ✚ Dev output (contains filled missing values)
- Predicted output



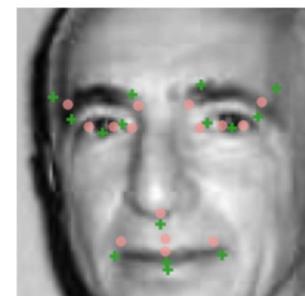
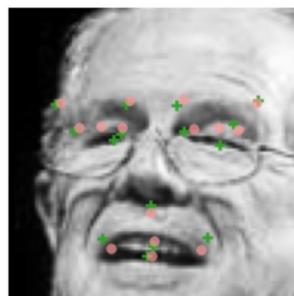
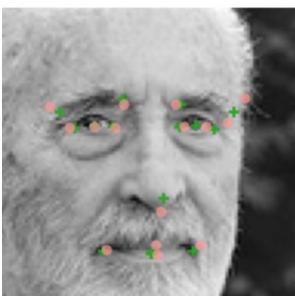
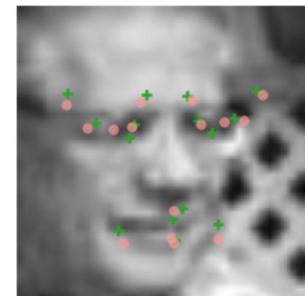
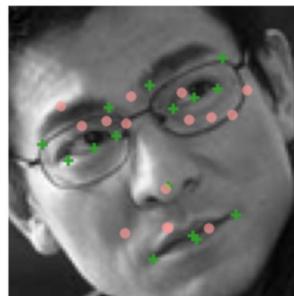
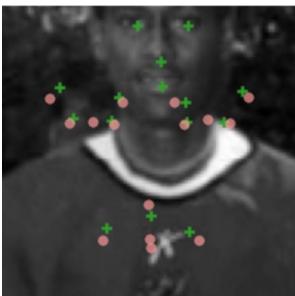
Baseline

Baseline

KNN regression ($k = 9$)



- Evaluated on entire dev output
 - MSE ≈ 11.32
 - MAE ≈ 1.97
- Evaluated on original data only
 - MSE ≈ 17.24
 - MAE ≈ 2.43



Performance Summary

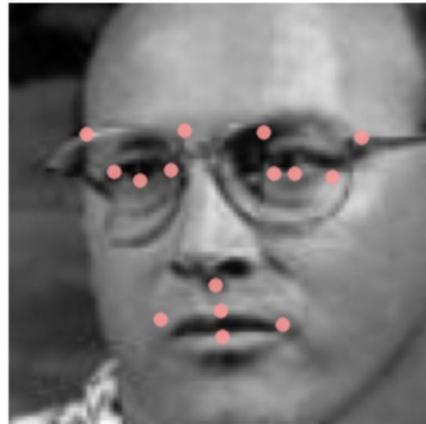
Model	KNN	LeNet5			VGG16		VGG19	
transfer learning	-	-	-	-	-	Yes	-	Yes
missing data	KNN	ffill	median	KNN	KNN	KNN	KNN	KNN
# of epochs	-	20	15	10	24	79	38	101
train loss (MSE)	-	29.92	24.29	32.22	28.19	9.86	21.06	7.96
train val_loss	-	18.10	11.19	12.30	9.79	4.20	9.92	4.47
dev val_loss	17.24	20.01	13.61	19.95	13.17	4.37	13.18	4.29

Data Augmentation

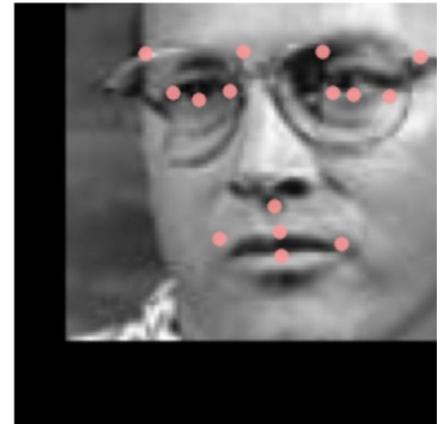
Data Augmentation

- Data Augmentation preprocesses images to create more unique training data
- Challenge: Keras's `ImageDataGenerator()` class does not transform Y
- Solution: custom generator class that also processes keypoint coordinates

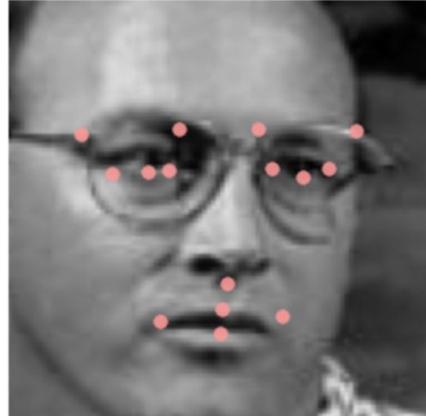
No change



Shifted



Flipped

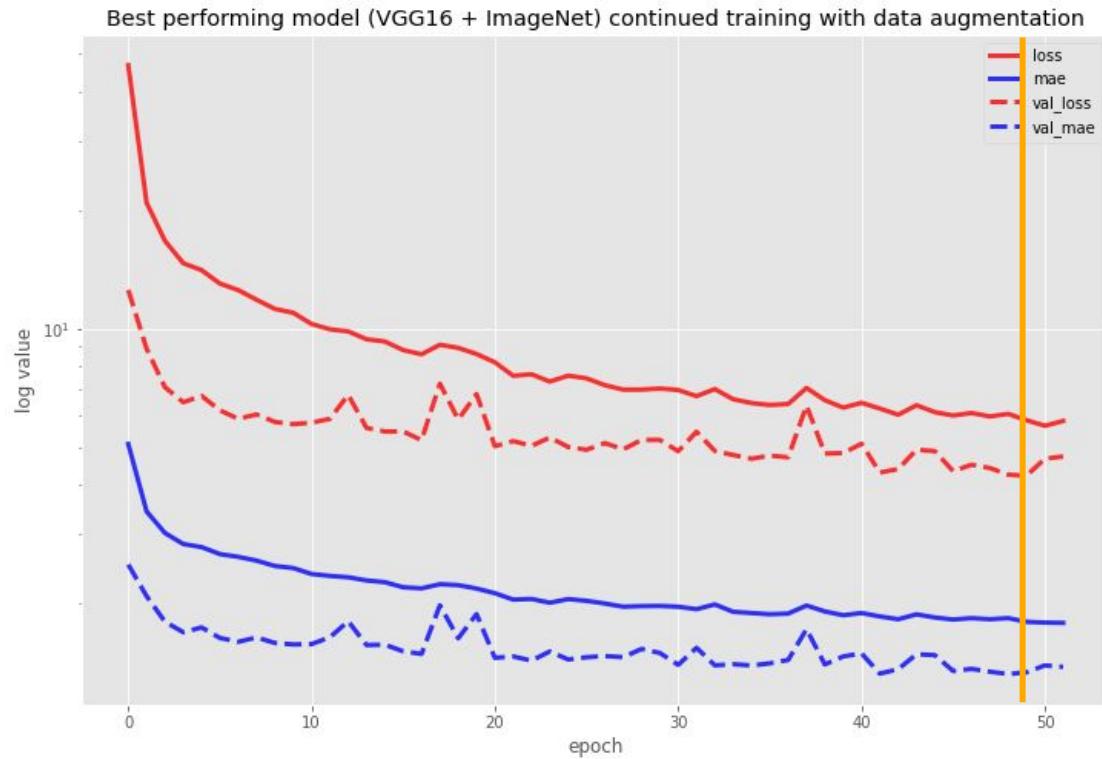


Rotated



VGG-16 + Transfer Learning + Data Augmentation

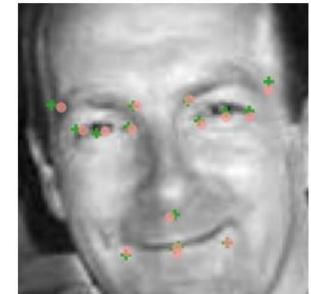
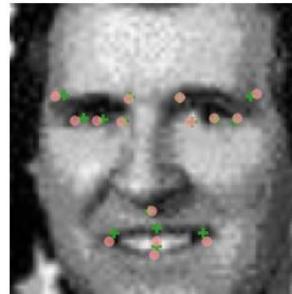
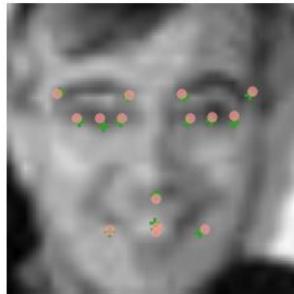
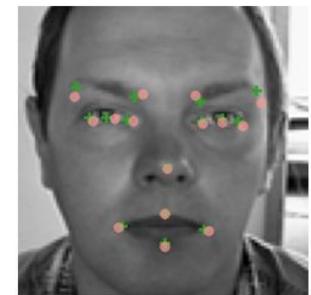
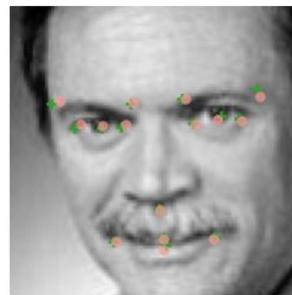
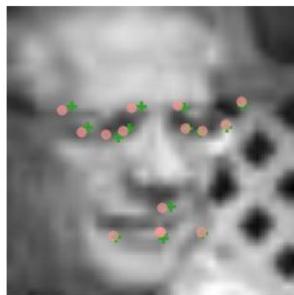
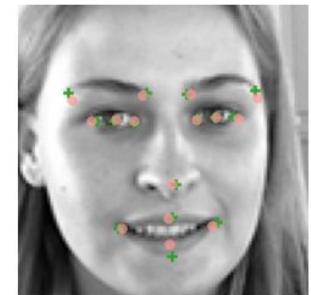
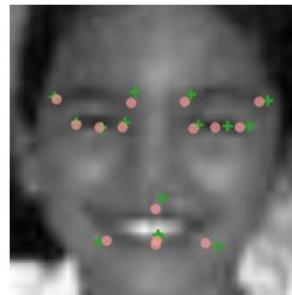
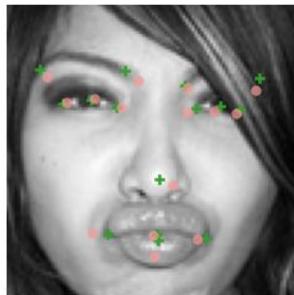
- Reached min loss_val at epoch 49
- Best performance during training:
 - loss \approx 5.85
 - mae \approx 1.80
 - val_loss \approx 4.23
 - val_mae \approx 1.34
- Dev set performance:
 - loss \approx 3.97
 - MAE \approx 1.33



VGG-16 + Transfer Learning + Data Augmentation

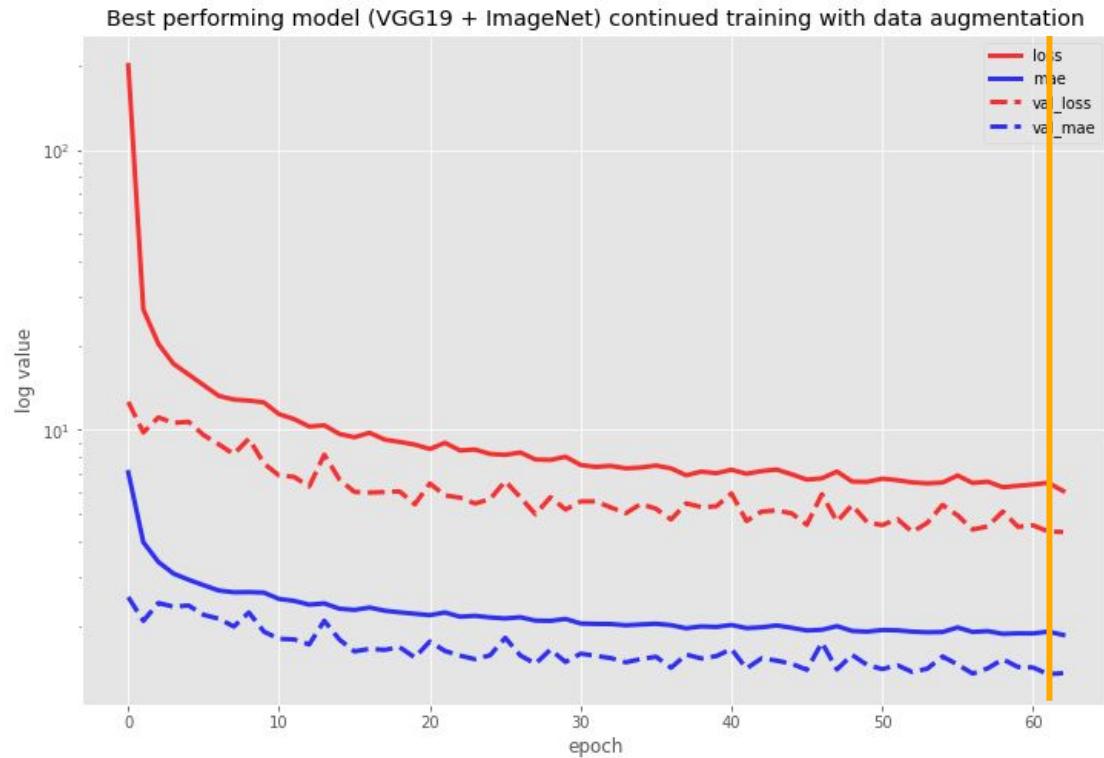
✚ Dev output (contains filled missing values)

● Predicted output



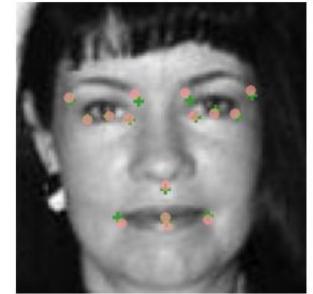
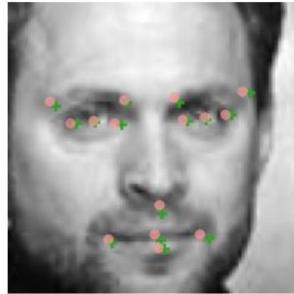
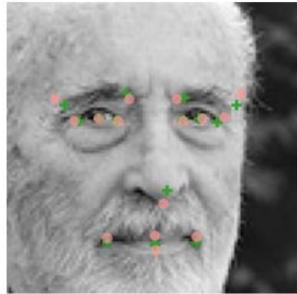
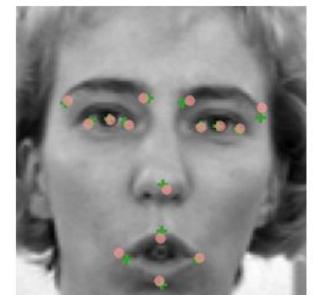
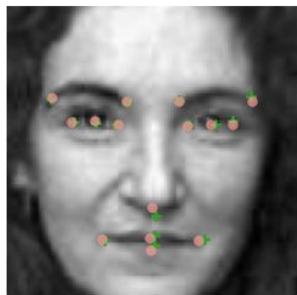
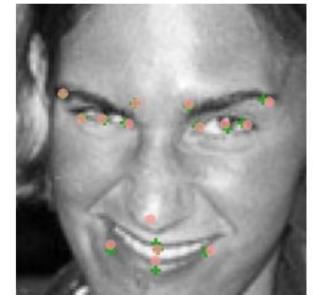
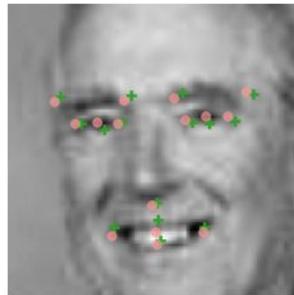
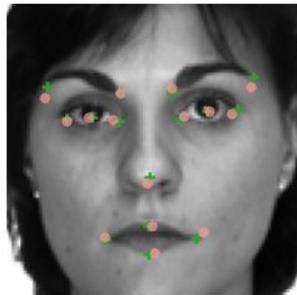
VGG-19 + Transfer Learning + Data Augmentation

- Reached min loss_val at epoch 62
- Best performance during training:
 - loss \approx 6.05
 - mae \approx 1.86
 - val_loss \approx 4.32
 - val_mae \approx 1.35
- Dev set performance:
 - loss \approx 4.31
 - MAE \approx 1.35



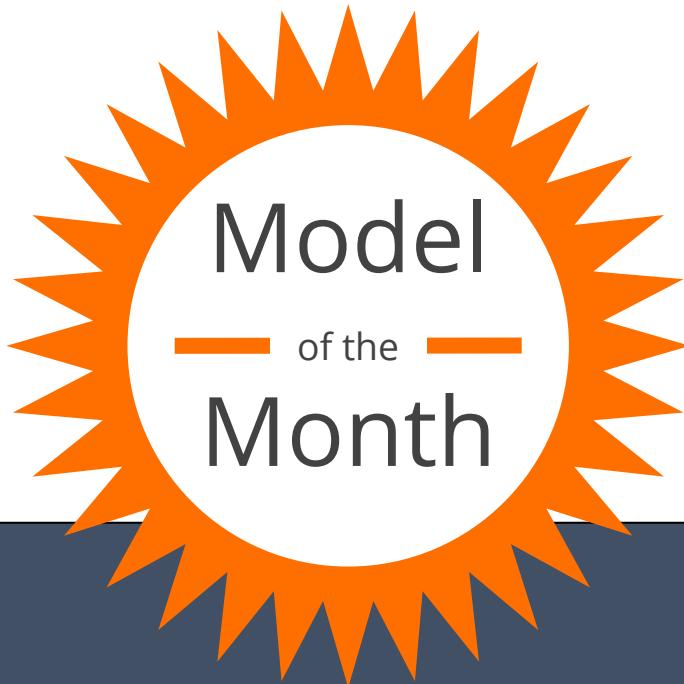
VGG-19 + Transfer Learning + Data Augmentation

- ✚ Dev output (contains filled missing values)
- Predicted output



Performance Summary

Model	VGG16		VGG19	
transfer learning	Yes	Yes	Yes	Yes
missing data	KNN	KNN	KNN	KNN
data augmentation	-	Yes	-	Yes
# of epochs	79	49	101	62
train loss (MSE)	9.86	5.85	7.96	6.05
train val_loss	4.20	4.23	4.47	4.32
dev val_loss	4.37	3.97	4.29	4.31



VGG 16

- Transfer learning
- Additional training with data augmentation

How Well Does It Work?

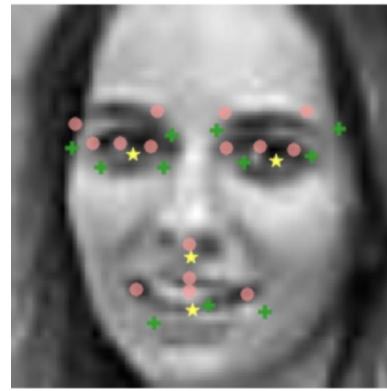
Error Analysis

- Challenge: some of the large errors is when the model gets it right, but the training key point coordinates were missing and filled erroneously

★ Original kp data from kaggle

+ Extrapolated data to fill missing kps

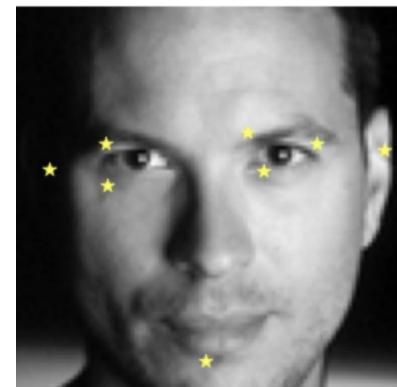
● CNN predicted output



Error Analysis

- Solution: manually calculate MSE for only the keypoint coordinates from original dataset, exclude any filled in coordinates
 - We discovered 2 images where the original data from kaggle was incorrect

★ Original kp data from kaggle



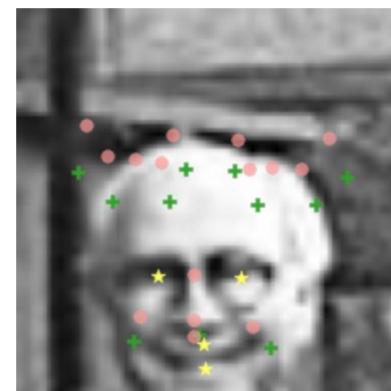
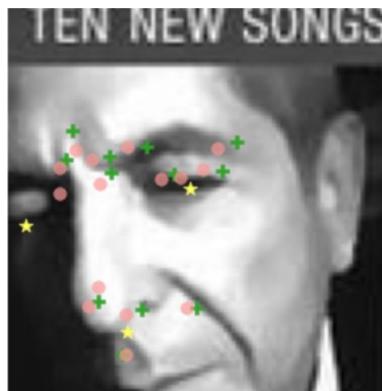
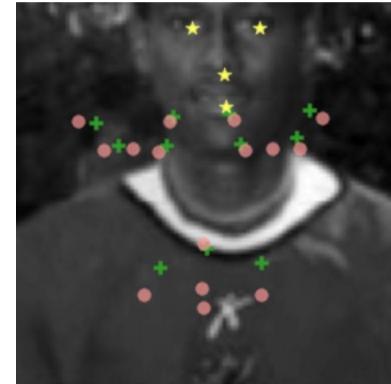
Error Analysis

- Many common images across all models, mostly faces far off center
- Extrapolating the missing points to complete training output also performed poorly

★ Original kp data from kaggle

✚ Extrapolated data to fill missing kps

● CNN predicted output



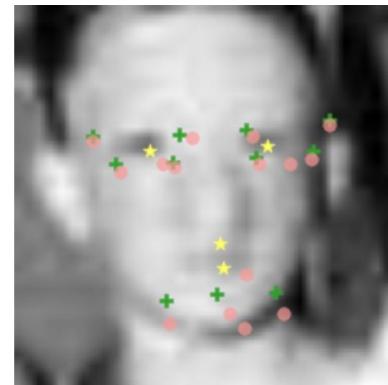
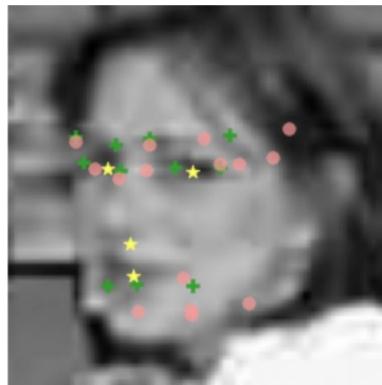
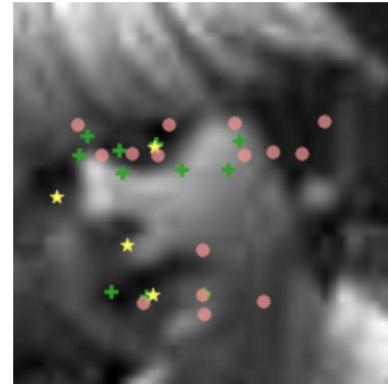
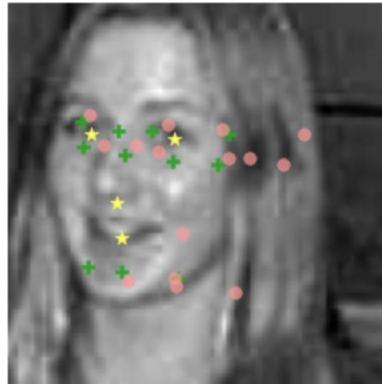
Error Analysis: VGG

- VGG16/VGG19 tend to have a hard time with highly pixelated images
- Extrapolated missing points were relatively closer to actual locations
- May be able to address this by adding in blurring to training data

★ Original kp data from kaggle

✚ Extrapolated data to fill missing kps

● CNN predicted output



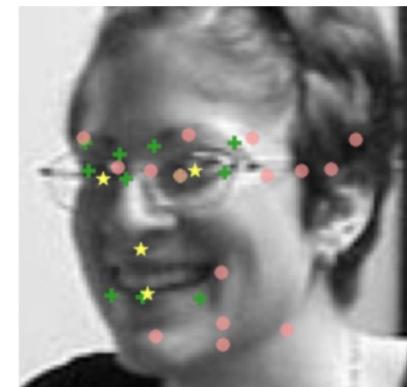
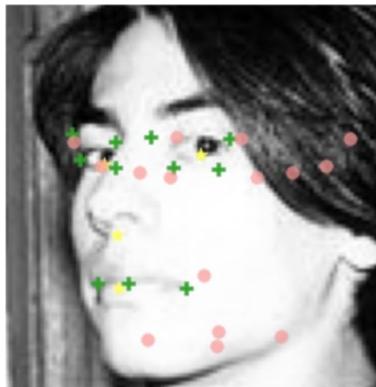
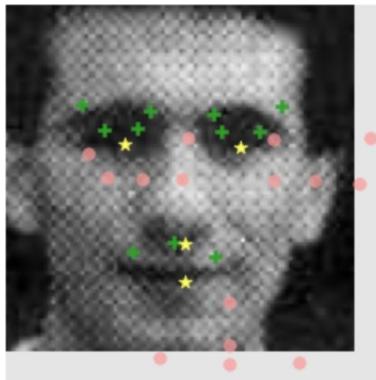
Error Analysis: LeNet5

- LeNet5 has difficulty with highly pixelated images as well
- Also have trouble with images with padded edges
- More problems with tilted faces comparing to VGG16/VGG19

★ Original kp data from kaggle

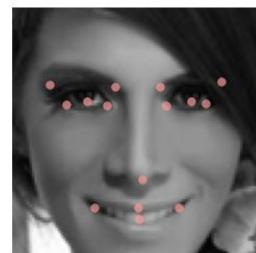
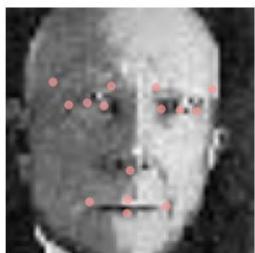
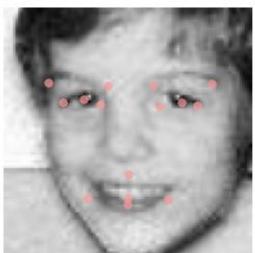
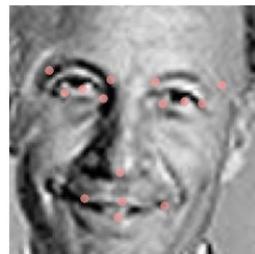
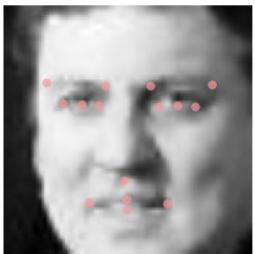
+ Extrapolated data to fill missing kps

● CNN predicted output



Test Set Results

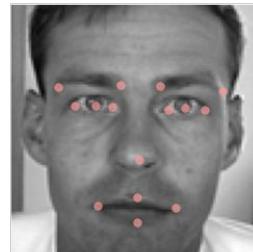
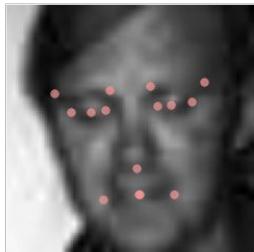
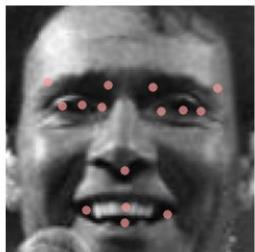
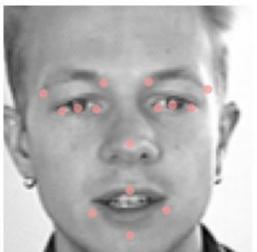
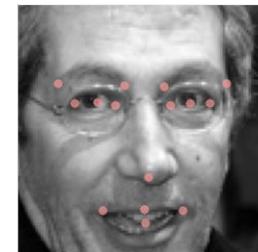
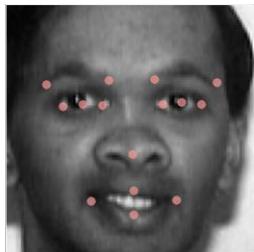
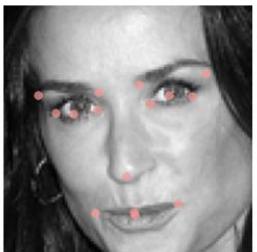
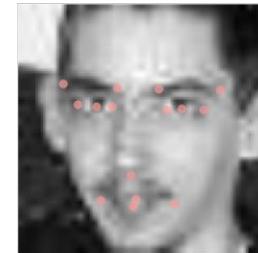
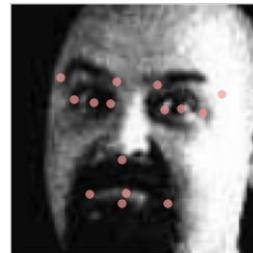
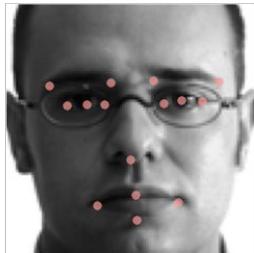
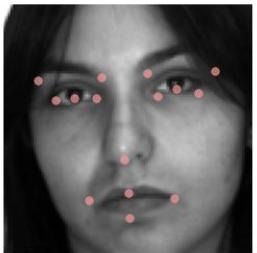
100

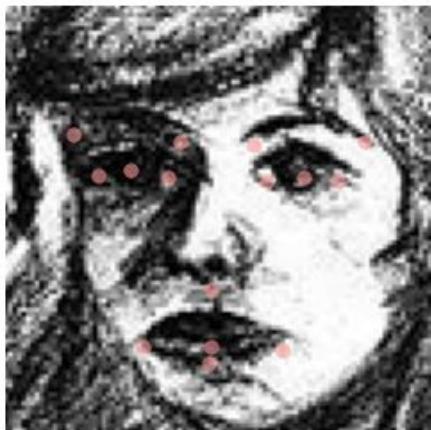


Test Set Results

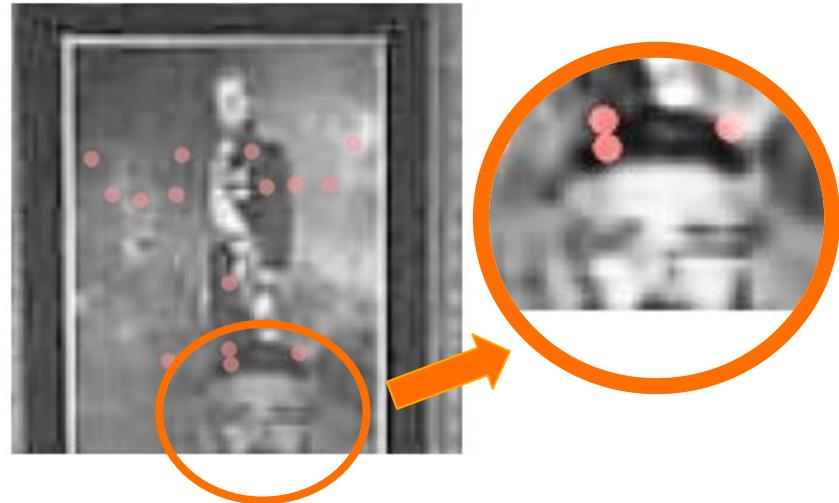


even on blurry or turned faces





OK on drawings



👎 on adversarial images

Show Time!



References

- Numerous Coursera classes by Andrew Ng
- <https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4>
- <https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5>
- <https://towardsdatascience.com/understand-the-architecture-of-cnn-90a25e244c7>
- Black box image source:
<https://singularityhub.com/2019/04/17/in-defense-of-black-box-ai/>
- Cherry on top image:
<https://www.flickr.com/photos/sharisberries/19440746753>