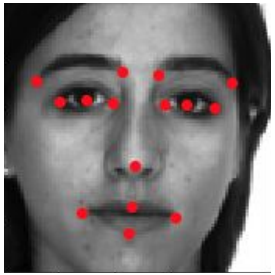
Facial Keypoints Detection
A Kaggle Challenge

Don Moon, Lester Yang, Shwetha Chitta Nagaraj

April 14th 2020

Challenge Overview



Objective: Train a regression model for detecting facial keypoints faces.

Kaggle Dataset details: 96x96 pixel images represented in (0,255) values,

7049 training images and 1783 testing images

15 keypoints: x, y values giving 30 columns

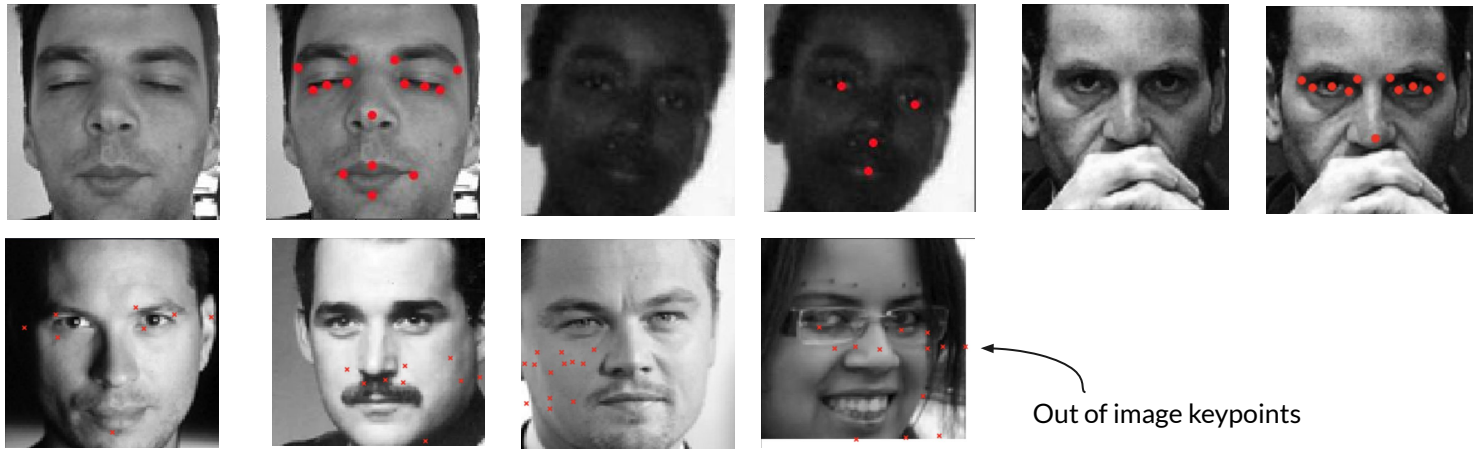
left_eye_center, right_eye_center, left_eye_inner_corner, left_eye_outer_corner, right_eye_inner_corner, right_eye_outer_corner, left_eyebrow_inner_end, left_eyebrow_outer_end, ….

Evaluation Criteria: Root Mean Square Error(RMSE)

$$RMSE = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

EDA Review:

Missing keypoints, varying facial expression, varying angles, brightness, clarity of images



Left_eye_center, right_eye_center, nose_tip, mouth_center_bottom_lip: in 7000+/7049 images

Remaining 11 keypoints have less than 2000 images to train with

⇨ build 2 separate models

Mouth and nose keypoints have particularly high variations due to facial expression

Some keypoints deviate significantly from ground truth -> bad training

For complete EDA please visit **FKP_Team_Baseline-Colab.ipynb** on our repo.

Convolutional Neural Networks:

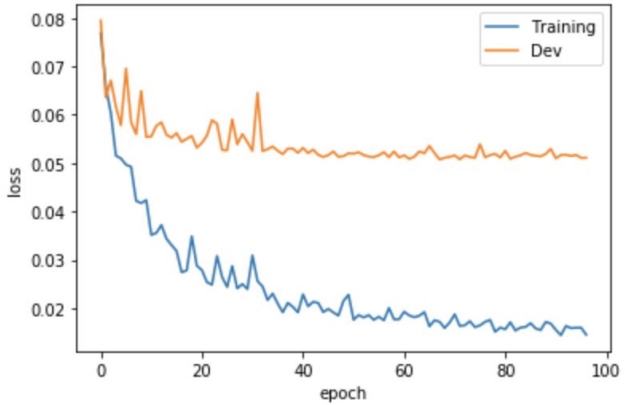
Image is the common input to any regression task, why CNN works so well?

CNNs use filters that scans through image to find the weights: essentially a dimensionality reduction that reduces the feature size, while taking neighboring pixel values into account.

Baseline model: CNN with 1 convoluted layer and 1 fully connected layer,

trained on trimmed dataset (less than 1692 images, with all keypoints)

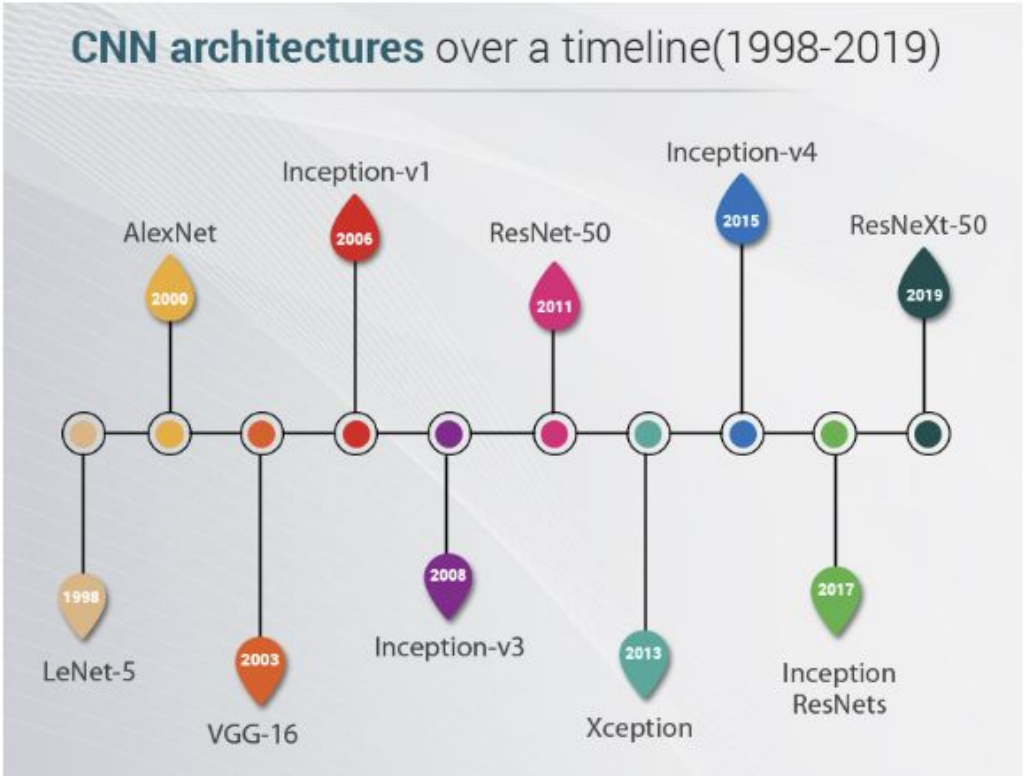
RMSE: 9.037



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 94, 94, 10)	100
flatten (Flatten)	(None, 88360)	0
dense (Dense)	(None, 30)	2650830

Total params: 2,650,930
Trainable params: 2,650,930
Non-trainable params: 0

CNN Architectures Explored



GPU Environment:

- AWS EC2 p2.xlarge instance
  - Ubuntu 18.04 AMI
  - Tensorflow 2.1.0
  - Keras 2.2.4-tf

- Google Colab
  - Tensorflow 2.1.0

- FloydHub
  - Tesla K80 12GB Memory
  - Tensorflow 2.1.0

https://www.aismartz.com/blog/cnn-architectures/

Using Pre-trained Models

- TensorFlow Hub
  - https://tfhub.dev/s?module-type=image-classification,image-feature-vector&tf-version=tf2
- Find the models pre-trained for image feature vectors and image classification
  - MobileNet
  - ResNet
  - Inception ResNet
- Infrastructure Limitation: Google Colab
  - Google Colab limits memories, GPUs, and training time
  - Identifying doable models on Google Colab matters
  - Be cautious of managing memory by freeing unused resources

Pre-trained MobileNet

- Small, low latency model for mobile and embedded vision applications
  - Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen: "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation", 2018
  - Pre-trained model name: mobilenet_v2_050_96
- Model complexity - reasonable for Google Colab simulation (batch size = 16)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_8 (KerasLayer)   (16, 1280)                706224
_________________________________________________________________
dense_8 (Dense)              (16, 30)                  38430
=================================================================
Total params: 744,654
Trainable params: 38,430
Non-trainable params: 706,224
```

Pre-trained ResNet

- Substantially deep Neural Network with residual learning
  - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: "Deep Residual Learning for Image Recognition", 2015.
  - Pre-trained model name: resnet_v2_50
- Model complexity - maybe too complex to run on Google Colab (batch size = 16)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_9 (KerasLayer)   (16, 2048)                23564800
_________________________________________________________________
dense_9 (Dense)              (16, 30)                  61470
=================================================================
Total params: 23,626,270
Trainable params: 61,470
Non-trainable params: 23,564,800
```

Pre-trained Inception ResNet

- ResNet with improved utilization of the computing resources inside the network
  - Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich: "Going deeper with convolutions", 2014.
  - Pre-trained model name: inception_v2
- Model complexity (batch size = 16)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_10 (KerasLayer)  (16, 1024)                10173112
_________________________________________________________________
dense_10 (Dense)             (16, 30)                  30750
=================================================================
Total params: 10,203,862
Trainable params: 30,750
Non-trainable params: 10,173,112
```

Adapting MobileNet/InceptionNet to FKD

- Split the training set into two, and train two MobileNets separately
  - 1st MobileNet to predict the 8 keypoints with more than 7000+ training images available
  - 2nd MobileNet to predict the 22 keypoints with less than 3000 training images available
- Add a fully-connected layer to the two pre-trained models
- Fix the all layers to untrainable other than the last fully-connected layer
- Try different hyper-parameters for further optimization
- Do the same things for InceptionNet
- Potential Improvements to try if time and more resources are available
  - Making more layers trainable in addition to the last fully-connected layer.

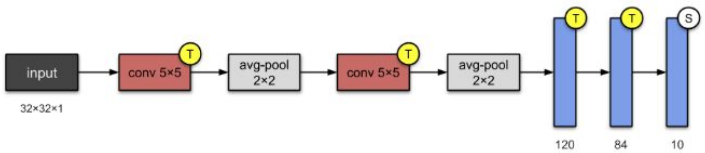
MobileNet/InceptionNet Results

- Notebook: FKP_leveraging_pretrained_models.ipynb
- Trained-models and Kaggle results: /pre-trained_models/

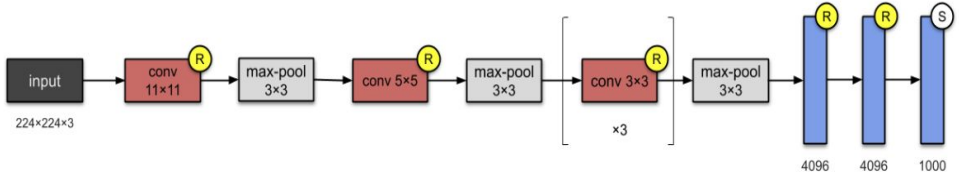
	Kaggle Score (RMSE)	# of Parameters	# of Trainable Parameters
MobileNet	3.86 (117th)	~750K	~40K
InceptionNet	4.06 (143th)	~10 M	~31K
ResNet	N/A	~21M	~61K

Modeling Inspirations

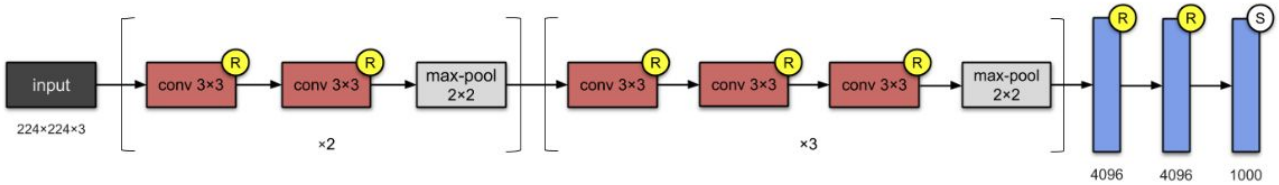
1. LeNet-5 (1998)



2. AlexNet (2012)



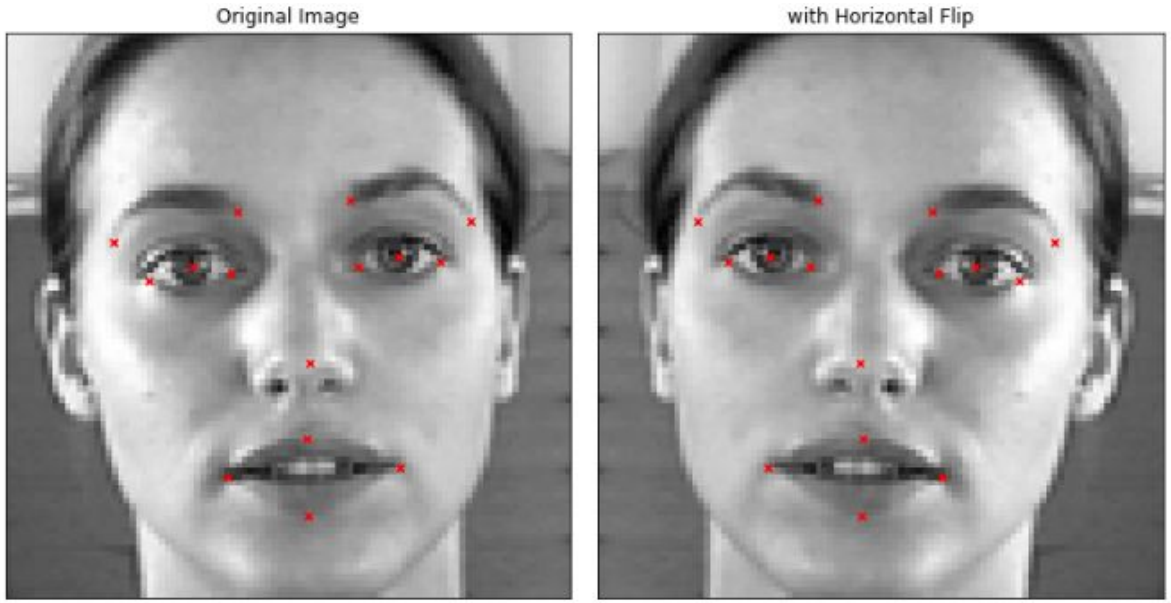
3. VGG-16 (2014)



Kaggle	Private	Public
Lenet-5	3.11456	3.21203
AlexNet	4.23367	4.28248
VGG-16 (Condensed)	2.30058	2.62112

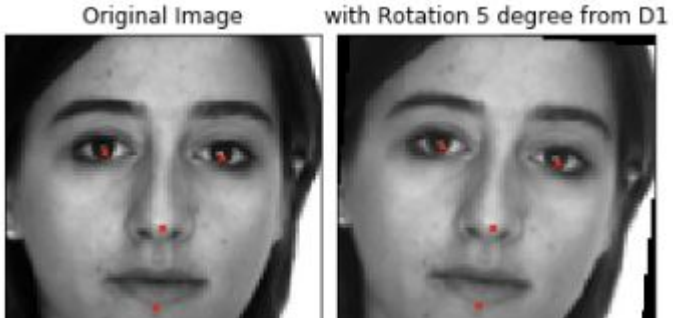
https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

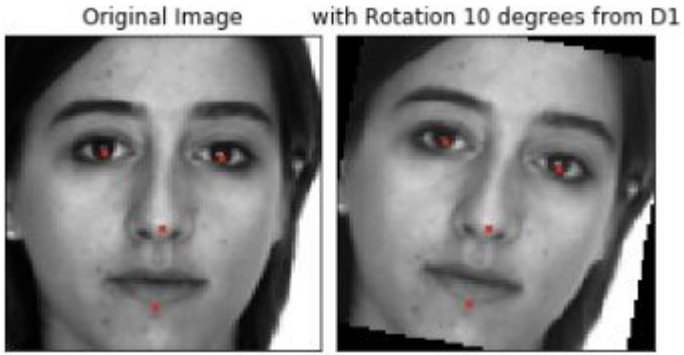
Data Transformation - Horizontal Flipping

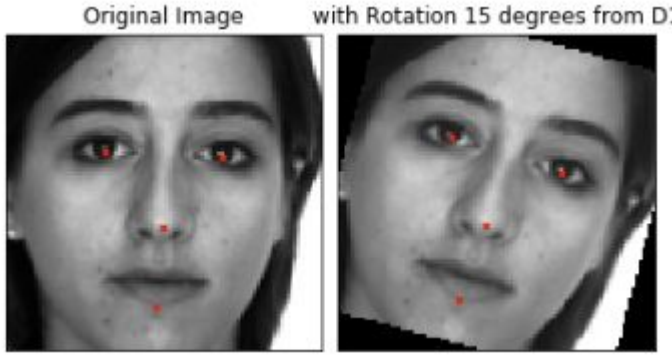


- Image Pixels
- Keypoint Labels

Data Transformation - Rotation at 5, 10, 15 degrees

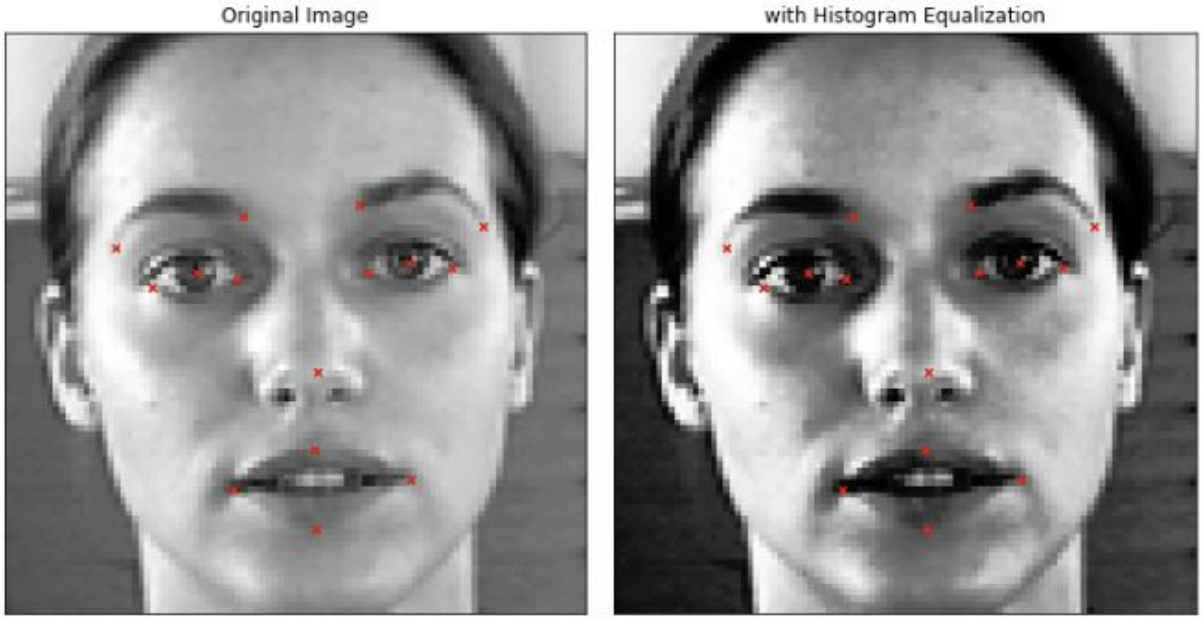






- Rotation of image pixels
- Keypoint co-ordinates

Data Transformation - Histogram Equalization

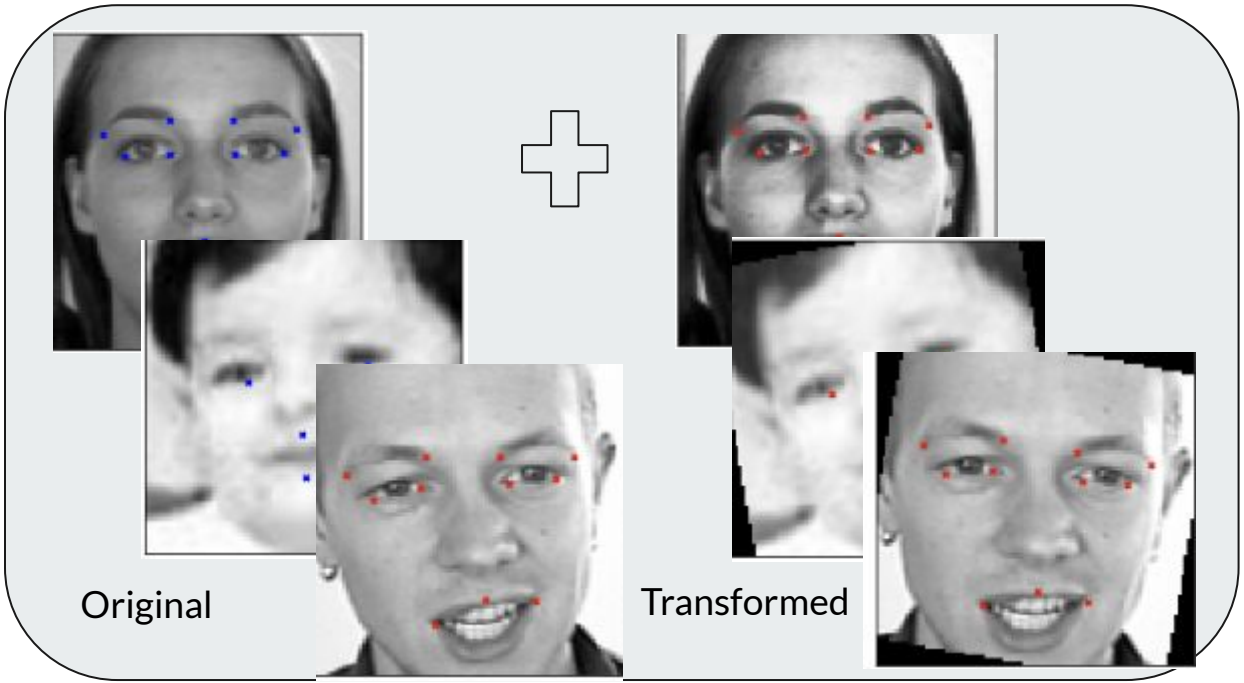


- Only image pixels are transformed

Data Transformation - Blurring

- Only image pixels are transformed

Data Augmentation and Testing



Augmented

Simple CNN

3 Conv. Layers

1 Hidden layer

ReLU

Adam

Batch norm.

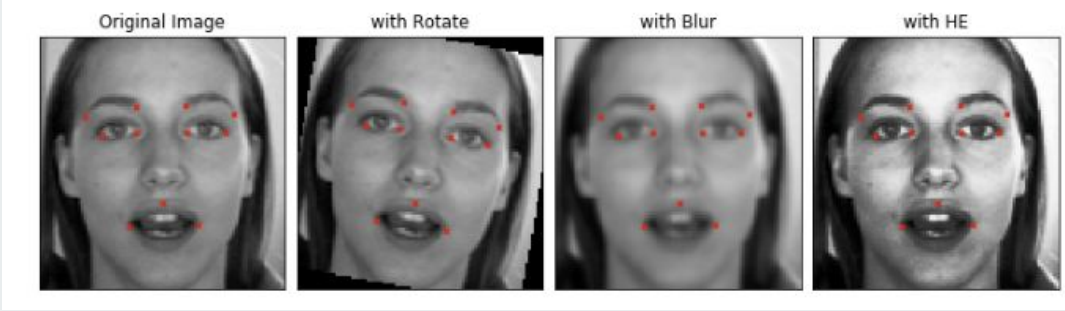
50 epochs

Batch_size = 50

Data Augmentation - Simple CNN: RESULTS

10 Models

- Flipping
- Rotation at 5 / 10 / 20 degrees
- Histogram Equalization(HE)
- Blurring
- Rotation at 10 + Blur
- Rotation at 10 + HE
- Blur + HE
- Rotation at 10 + HE + Blur



BEST Combination with lowest Val. RMSE

4 x Training Data!

Training on VGG-16

```
_________________________________________________________________
dense_2 (Dense)              (None, 8)
=================================================================
Total params: 24,822,464
Trainable params: 24,817,296
Non-trainable params: 5,168
_________________________________________________________________
```

Final Models

Model 1

5 CLs:
32-64-128-256-512

1FC:
512

BatchNormalization
MaxPooling
ReLU

Highlights:
Val. RMSE 2.12
Tr. RMSE 2.803
Consistent Accuracies

D1 underfitting before converging

Private: 1.80883
Public: 2.16564

Model 2

+ 1 CL after each CL:
32-32-64-64-128-128-256-256
-512-512
1FC:
512

BatchNormalization
MaxPooling
ReLU

Highlights:
Val. RMSE: 1.96
Tr. RMSE: 2.86
Consistent Accuracies

Exploding gradients

Private: 1.73005
Public: 2.12732

Model 3

+ 1 CL after each CL:
32-32-64-64-128-128-256-256
-512-512
1FC:
512

BatchNormalization
MaxPooling
Leaky ReLU alpha = 0.1
ReLU at FC

Highlights:
Val. RMSE: 1.96
Tr. RMSE: 1.73
Consistent Accuracies

Converges faster for D1
Underfitting for D2

Model 4

+ 1 CL after each CL:
32-32-64-64-128-128-256-256
-512-512
1FC:
512

BatchNormalization
MaxPooling
Leaky ReLU
Linear at FC

Highlights:
Val. RMSE: 2.03
Tr. RMSE: 1.21
Consistent Accuracies

D1 overfits
D2 better calibrated

Model 5

Same as Model 4

New mix of data transformations
HE(Blur(Rotate(data)+data)
Training set reduced by half

BatchNormalization
MaxPooling
Leaky ReLU alpha = 0.1
ReLU at FC

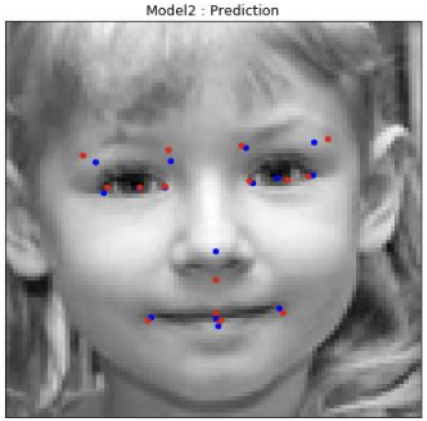
Highlights:
Val. RMSE: 2.17
Tr. RMSE: 1.17
Consistent Accuracies
Converges faster for D1 and D2
Well calibrated

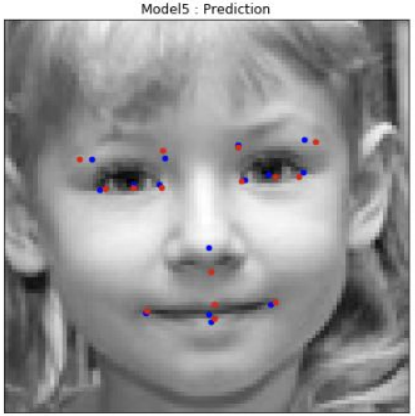
Private: 2.02078
Public: 2.30053

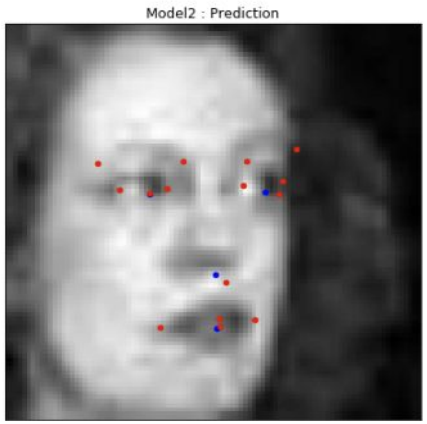
Epochs = 100, **Batch Size** = 100, **Optimizer** = adam

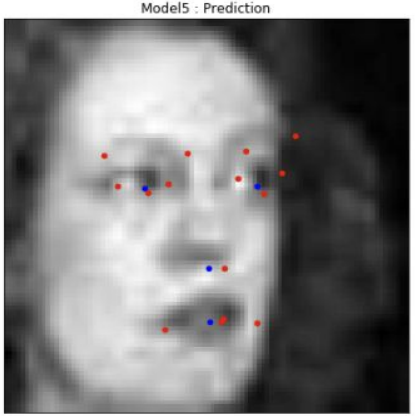
Callbacks - Early Stopping, Reduce LR on Plateau

Predictions - Models 2 and 5









BLUE - Given keypoint labels

RED - Predicted Keypoint labels

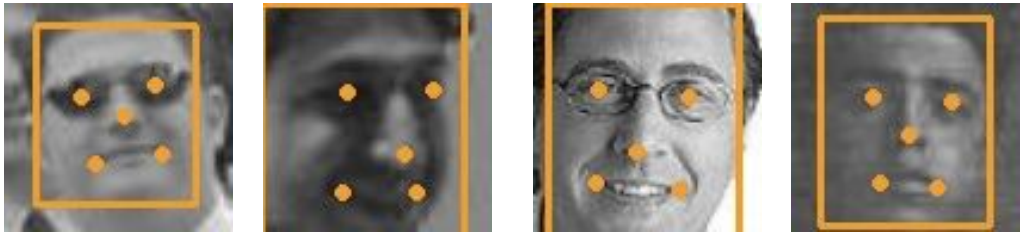
Further Improvements

Model pipeline

- Improve score by specialization of the models.
- Current top score achieved with a 4 step pipeline:

  face box detector → face recognizer → face aligner → predict keypoints

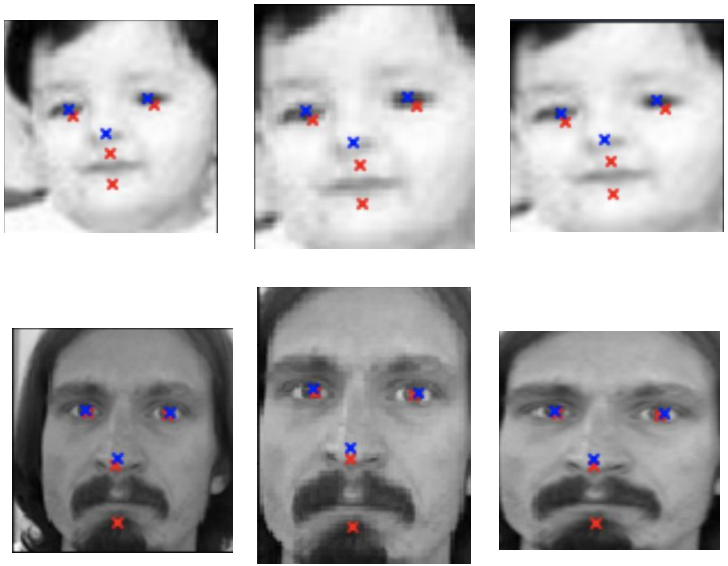
- Attempted face box detector with pre-trained cascaded convolutional networks, MTCNN.



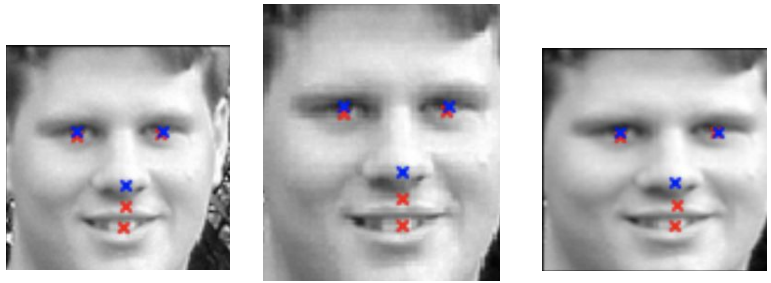
Further Improvements

Detect face box → crop → resize ⇒ keypoint detection

replace/ impute keypoints



Other data augmentation techniques:
Elastic Transform



Implementation found on **FKP_Facebox_detect.ipynb** on repo.

Thank you !