Facial Keypoint Detection

W207 Final Project

Jeremy Yeung, Simran Sachdev, Gabriel Louis-Kayen, Shanie Hsieh, Amy Jung

What is facial detection?

Our project focuses on **Facial Keypoints Detection**, detecting and predicting the location of keypoints on face images — the fundamental building block for various applications including:

- Tracking faces in images and videos
- Analyzing facial expressions
- Detecting dysmorphic facial signs for medical diagnosis
- Biometrics / face recognition.

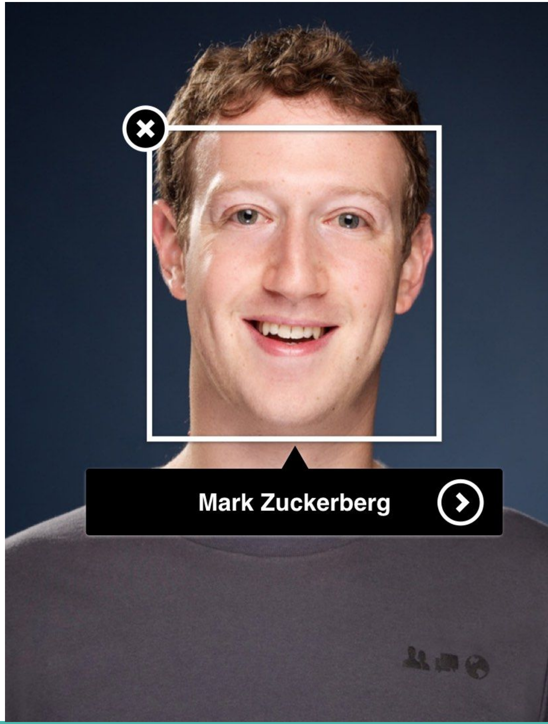
Facial Detection in the News

"Facebook Plans to Shut Down Its Facial Recognition System"

The New York Times

- Delete **face scan data** of 1 billion+ users
- Facebook's facial-recognition functions:
  - Automatically identify people
  - Flag accounts
  - Described photos to blind users

- NOT eliminate software (ie. DeepFace)
- NOT rule out incorporating facial recognition tech into future products



Data

Data source:
https://www.kaggle.com/c/facial-keypoints-detection/overview

The dataset is Facial Keypoints Detection data used to detect the location of keypoints on face images.

Size of dataset:

The dataset has 7049 images that each have 30 columns variables.

Main features used:

There are 30 columns associated with 15 features.

The 15 features:

- **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,
- right_eyebrow_inner_end,
- right_eyebrow_outer_end
- **nose_tip**
- mouth_left_corner
- mouth_right_corner
- mouth_center_top_lip,
- **mouth_center_bottom_lip**

Each of the 15 features has an x-axis column and a y-axis column corresponding to that feature's location on the image, leading to 30 columns overall.

EDA

Using the 2000+ images with 15 features

- Higher train accuracy
- Prone to overfitting

Using the 7000+ images with 4 features

- Easier to generalize
- More training data
- Less accuracy in identifying facial keypoints

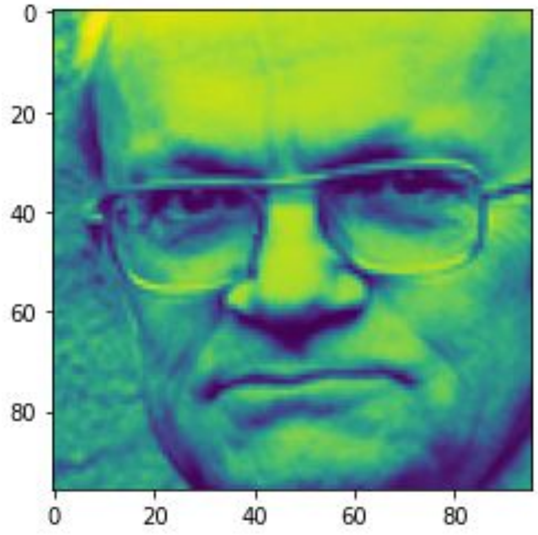
7000 images have data for 4 facial key points

	count	mean	std
left_eye_center_x	7000.0	66.34940047635854	3.377149279603647
left_eye_center_y	7000.0	37.61810350559417	3.0365916551708043
right_eye_center_x	7000.0	30.303406587113066	2.9489464986052036
right_eye_center_y	7000.0	37.94265611325309	2.884111354058055
nose_tip_x	7000.0	48.372452384140466	4.1715876082560435
nose_tip_y	7000.0	62.68202743453441	5.621674878670501
mouth_center_bottom_lip_x	7000.0	48.57167648140966	4.237941397514037
mouth_center_bottom_lip_y	7000.0	78.97570952261637	5.407682797359353

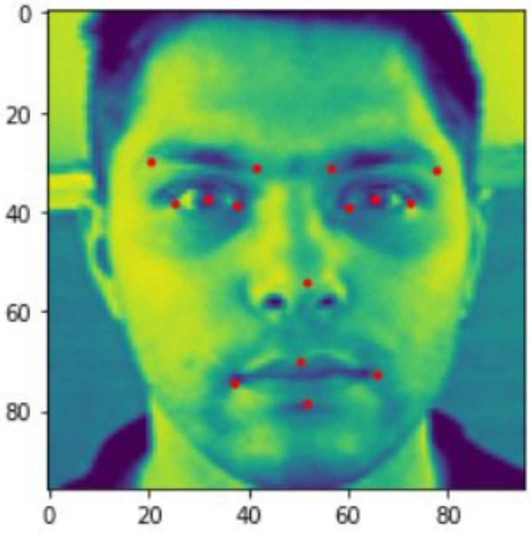
Of those 7000 images, only 2140 images of those have data for all 15 facial key points

	count	mean	std
left_eye_center_x	2140.0	66.22154868409592	2.087683355101556
left_eye_center_y	2140.0	36.842274165726266	2.294027490805707
right_eye_center_x	2140.0	29.64026856456148	2.051575209871264
right_eye_center_y	2140.0	37.06381489055456	2.2343335854467448
left_eye_inner_corner_x	2140.0	59.27212810062244	2.005630683413952
left_eye_inner_corner_y	2140.0	37.85601445389234	2.03450012751805
left_eye_outer_corner_x	2140.0	73.41247343419627	2.701639370765223
left_eye_outer_corner_y	2140.0	37.6401096830805	2.68416217097158
right_eye_inner_corner_x	2140.0	36.6031065182916	1.8227836818129908
	count	mean	std
right_eye_outer_corner_x	2140.0	22.36161709895906	2.7688040797668125
right_eye_outer_corner_y	2140.0	38.03457131359977	2.654902542892582
left_eyebrow_inner_end_x	2140.0	56.14799092743679	2.819913666924865
left_eyebrow_inner_end_y	2140.0	29.22230444909996	2.8671313510347325
left_eyebrow_outer_end_x	2140.0	79.61752316513792	3.3126467711070138
left_eyebrow_outer_end_y	2140.0	29.65657017639958	3.627186873003011
right_eyebrow_inner_end_x	2140.0	39.27208385866163	2.6096476570044818
right_eyebrow_inner_end_y	2140.0	29.41374657993314	2.8422186447220557
right_eyebrow_outer_end_x	2140.0	15.76170725407129	3.3379012928231457
right_eyebrow_outer_end_y	2140.0	30.452946698618238	3.6443422006653514
nose_tip_x	2140.0	47.95214068998041	3.276053208468195
nose_tip_y	2140.0	57.25392567086902	4.528635210886218
mouth_left_corner_x	2140.0	63.419076094887814	3.650131009318928
mouth_left_corner_y	2140.0	75.88765965132447	4.438565027075064
mouth_right_corner_x	2140.0	32.96736460044271	3.5951027258262207
mouth_right_corner_y	2140.0	76.13406536660167	4.259513821121693
mouth_center_top_lip_x	2140.0	48.081324634435525	2.7232735346715224
mouth_center_top_lip_y	2140.0	72.6811245530104	5.108675344728991
mouth_center_bottom_lip_x	2140.0	48.1496539871852	3.032388960435935
mouth_center_bottom_lip_y	2140.0	82.63041245065179	4.813557334126184

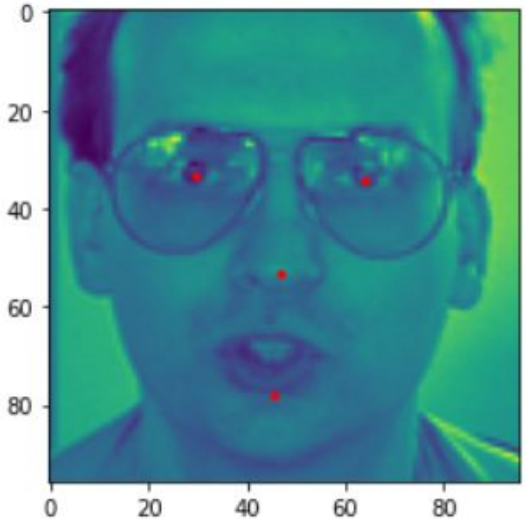
Example Images and Facial Key Points



Example Image



Example Facial Key Points (15 features)



Example Facial Key Points (4 features)

OLS: Approach

- OLS regressing each keypoint feature on the other keypoint features
- Attempted on 4 feature dataframe
  - Could produce predictions for the 15 feature dataframe
  - Could determine location of a facial keypoint based on known locations of other facial features
- Constraints:
  - Test data only has image pixels and does not have keypoints
  - We want predictions for a clean image without prior keypoint detection
  - Other image datasets will not typically have facial keypoints indicated

Preprocessing for OLS and CNN

- Example “Image” column = [238 236 237 238 240 240 239 241 241 243 240 239 231 212 190 173 148 122 104 92 79 73 74...]
    - Image size: 96 x 96 = 9216 columns
    - Extract “Image” into a 2D array with each column an integer from the pixels
    - Convert each pixel into ints
- 80/20 train test split

OLS: Approach

- We want predictions for keypoints based off of image pixels
- OLS regressing each keypoint on the pixels
- Each pixel is a feature
- N different multiple linear regressions, where N is the number of facial keypoints (8 total)

$$\hat{\mathbb{Y}} = \mathbb{X}\theta$$

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \\ \vdots \\ \hat{y}_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} & \dots & x_{1p} \\ 1 & x_{21} & x_{22} & x_{23} & \dots & x_{2p} \\ 1 & x_{31} & x_{32} & x_{33} & \dots & x_{3p} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & x_{n3} & \dots & x_{np} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \\ \theta_p \end{bmatrix}$$

OLS: Experiment

Train mae: 0.272

Test mae: 21.935

PCA idea: lessen the number of coefficients so model can’t overfit

Train

Test

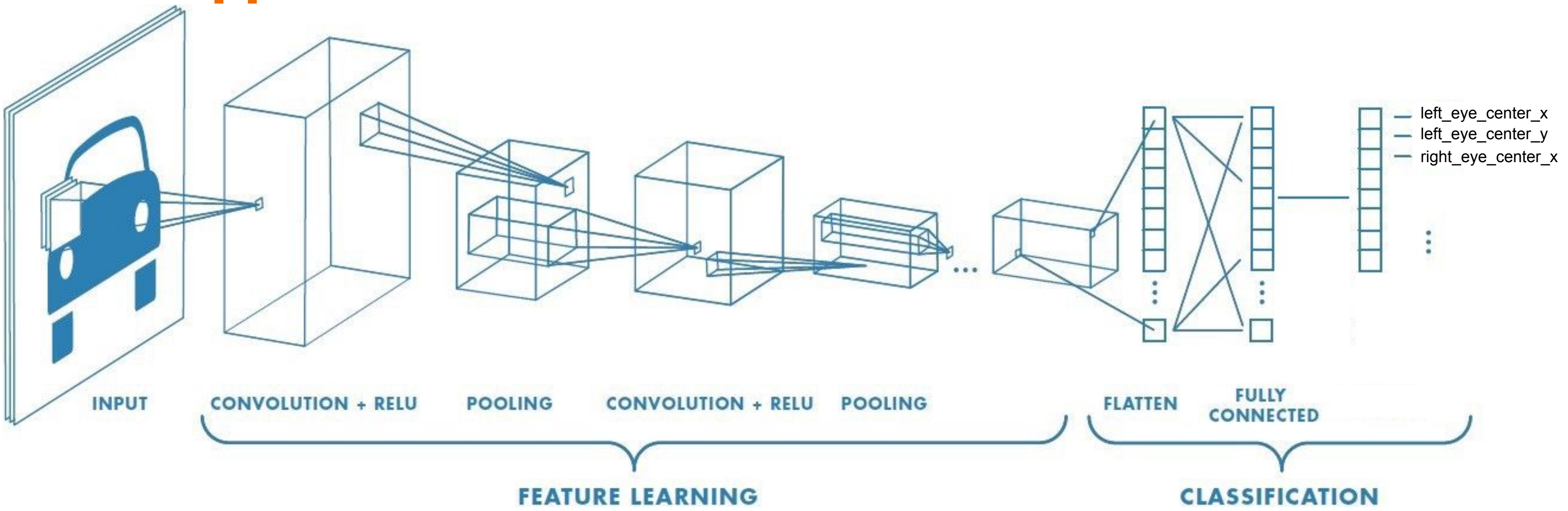
(All Pixels)

```
keypoint: left_eye_center_x
train mae: 0.2092160820024996
test mae: 26.86020130360977

keypoint: left_eye_center_y
train mae: 0.19532846710118398
test mae: 16.145948035968647
```

(PCA)

```
keypoint: left_eye_center_x
train mae: 5.022023809526552
test mae: 605.8114466879618

keypoint: left_eye_center_y
train mae: 4.933190476193213
test mae: 592.4514271495663
```

CNN: Approach



Transform each image to shape (96, 96, 1)

Convolution and pooling to extract features (e.g. eyes, nose, mouth)

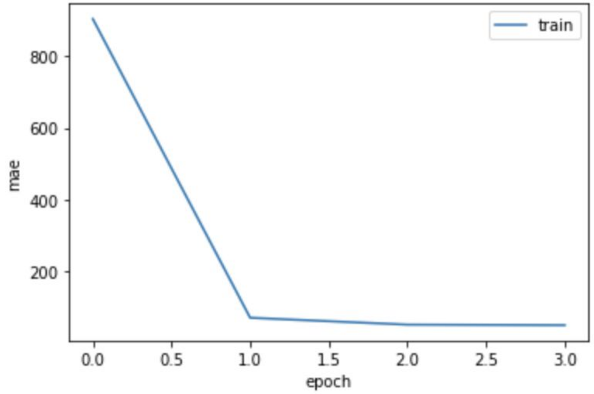
Smallest neural network to minimize weights

Metric: Mean Absolute Error

CNN - Model 1 & 2

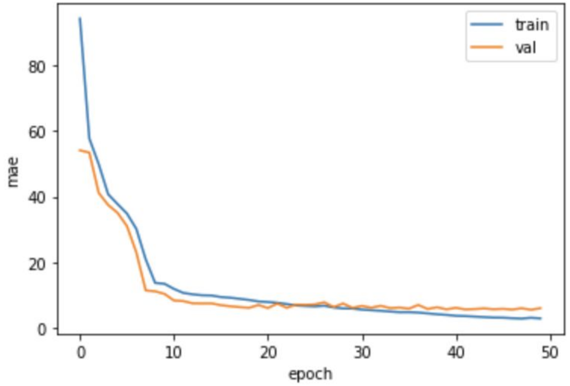
Model 1: Epoch 50/50 - mae: 19.113

```
#create model
model = Sequential()

#add model layers
model.add(Conv2D(256, kernel_size=3, activation='relu', input_shape=(96,96,1)))
model.add(Conv2D(64, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(8))

#compile model using accuracy to measure model performance
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
```



Model 2: Epoch 50/50 - mae: 2.980 - val_mae: 6.087

```
cnn = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(96,96,1)),
    MaxPooling2D((2, 2)),

    Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D((2, 2)),

    Flatten(),
    Dense(64, activation='relu'),
    Dense(8)
])
cnn.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
```

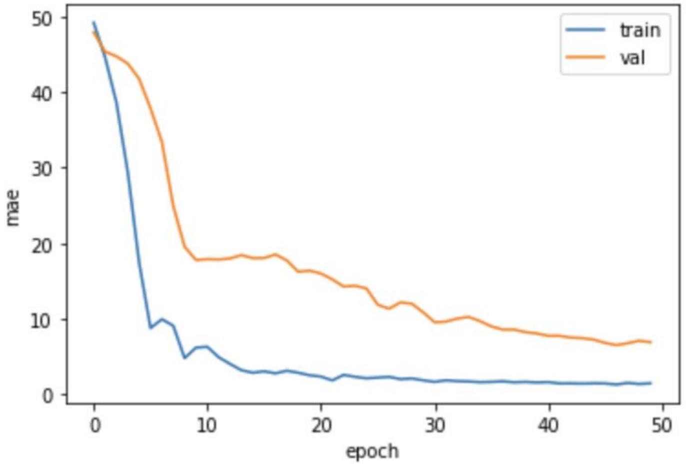


CNN - Model 3 & 4

Model 3

```
cnn = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(96,96,1)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
```

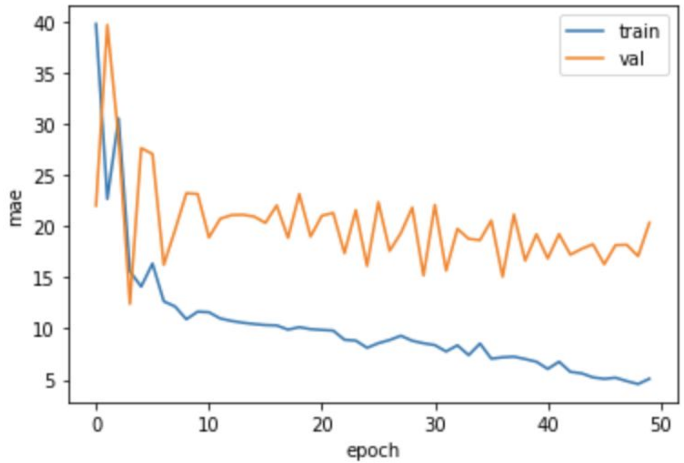
Epoch 50/50 - mae: 1.501 - val_mae: 6.948



Model 4

```
cnn = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(96,96,1)),
    Dropout(.1),
    MaxPooling2D((2, 2)),
```

Epoch 50/50 - mae: 5.088 - val_mae: 20.326



CNN - Model 5 & 6

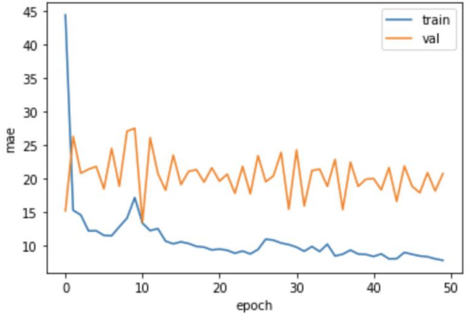
Model 5

Same as model 4 with Dense layers added to each layer

```
Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
Dropout(.1),
MaxPooling2D((2, 2)),
Dense(30),
```

⋮

Epoch 50/50 - mae: `7.855` - val_mae: `20.786`



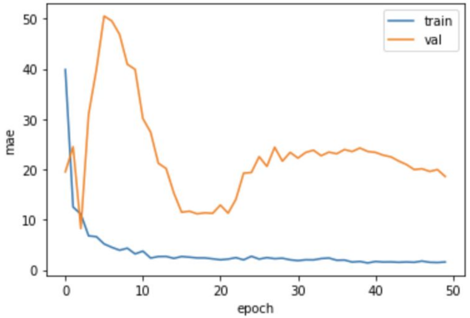
Model 6

Same as model 5 with BatchNormalization added to each layer

```
Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'),
BatchNormalization(),
Dropout(.1),
MaxPooling2D((2, 2)),
```

⋮

Epoch 50/50 - mae: `1.597` - val_mae: `18.612`



CNN - Model 7 & 8

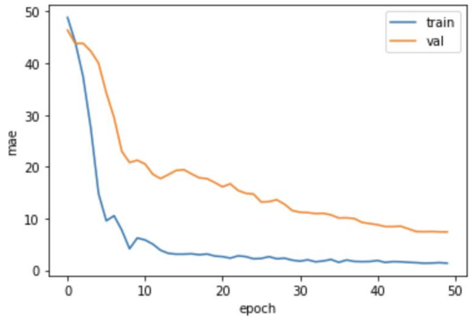
Model 7

Same as Model 3 but with LeakyReLU(alpha = 0.1) to each layer

```
cnn = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(96,96,1)),
    LeakyReLU(alpha = 0.1),
    BatchNormalization(),
    MaxPooling2D((2, 2)),
```

•
•
•

Epoch 50/50 - mae: 1.354 - val_mae: 7.400



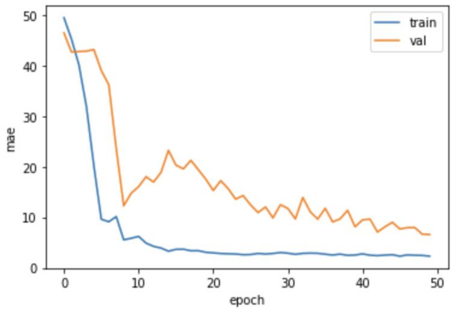
Model 8

Same as Model 7 but with Dropout(0.1) after dense layer

•
•
•

```
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.1),
    Dense(30)
```

Epoch 50/50 - mae: 2.298 - val_mae: 6.594



CNN: Best Model

Model 3 modified

Model #	Description.	train_acc	val_acc
1	first model	19.113	nan
2	2 Convolution + 1 Dense	2.980	6.078
3	5 Convolution + 1 Dense (with batch norm)	1.501	6.948
4	same as model 3 except using .1 dropout	5.088	20.326
5	same as model 4 with Dense layers	7.855	20.786
6	same as model 5 with batchnorm	1.597	18.612
7	same as model 3 with LeakyReLU	1.354	7.400
8	same as model 7 with Dropout(0.1) in Dense layer	2.298	6.594

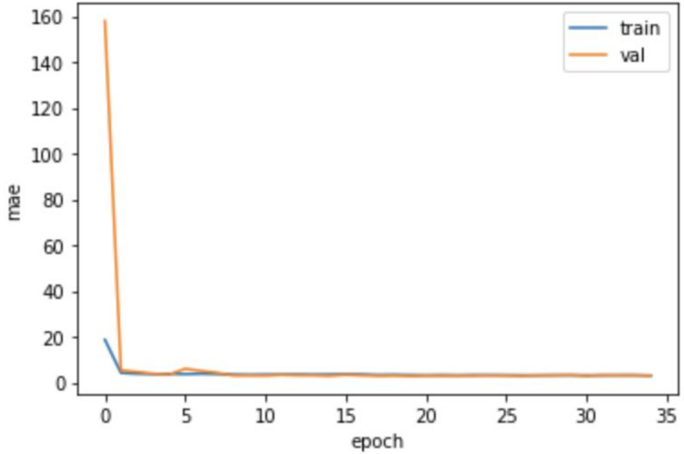
```
best_model = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(96,96,1)),
    MaxPooling2D((2, 2)),
    BatchNormalization(),

    Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    BatchNormalization(),

    Conv2D(filters=96, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    BatchNormalization(),

    Conv2D(filters=128, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    BatchNormalization(),

    Conv2D(filters=256, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    BatchNormalization(),

    Flatten(),
    Dense(512, activation='relu'),
    Dense(8)
])
```

CNN: Best Model

- Run on the full train set: Train MAE: `1.381` -- Val MAE: `1.942`
- Test MAE: `2.897`

Train

Test



Conclusions

Key Results:

- OLS: test mae of `21.935`
- CNN: test mae of `2.897`
- CNN had 19.056 improvement over the OLS baseline

Learned:

- how to build a convolutional neural network using Keras
- how to adjust the convolution and pooling blocks to improve performance

Avenues for Future Work

"Your Face Is, or Will Be, Your Boarding Pass"

The New York Times

- **Biometrics** (unique individual traits) can be used to automate + verify identity
- Facial recognition is **at least 99.5% accurate**
- e.g. Delta Air Lines, CLEAR Health Pass

References

https://www.kaggle.com/prateek146/facial-keypoints-detection-model-explained

https://www.nytimes.com/2021/12/07/travel/biometrics-airports-security.html?action=click&algo=bandit-all-surfaces-time-cutoff-10&block=more_in_recirc&fellback=false&imp_id=996217126&impression_id=38b8eef0-578a-11ec-afe3-6d88710900cc&index=0&pgtype=Article&pool=more_in_pools%2Ftechnology&region=footer&req_id=471771699&surface=eos-more-in&variant=2_bandit-all-surfaces-time-cutoff-10

https://www.nytimes.com/2021/11/02/technology/facebook-facial-recognition.html

https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5

Thank you!