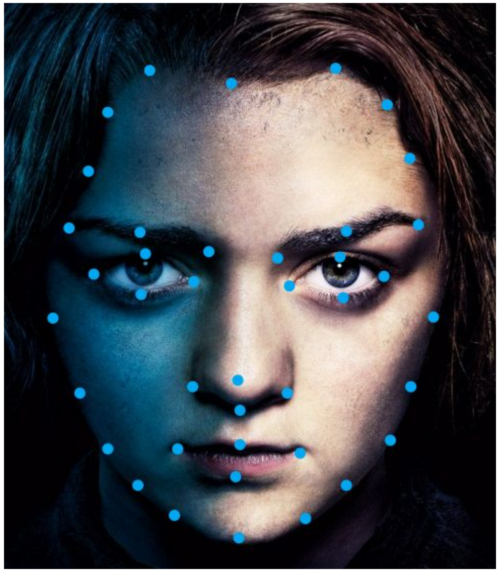
Facial Keypoints Detection

The Next Breakthrough in Computer Vision

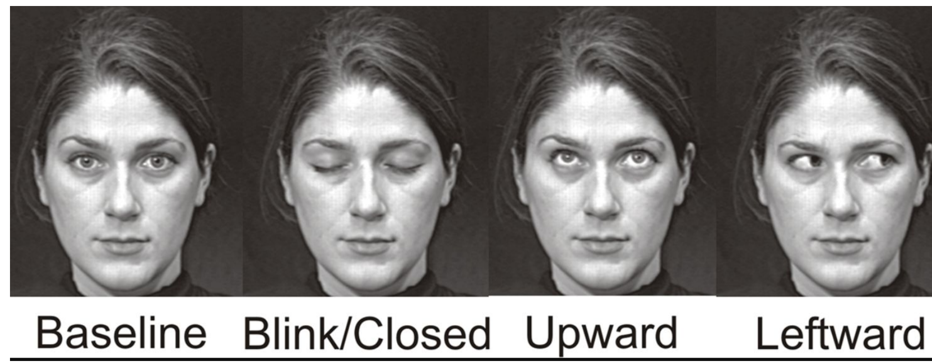
— Sirisha Bhupathi and Abhi Sharma

Faces Are Important



What are facial key points or landmarks?

- Eyes, nose, mouth - position and shape identifiers
- x & y coordinates



Why track these key points?

- Biometrics / face recognition
- Tracking faces in images and video
- Analysing facial expressions
- Detecting dysmorphic facial signs for medical diagnosis
- Instagram filters

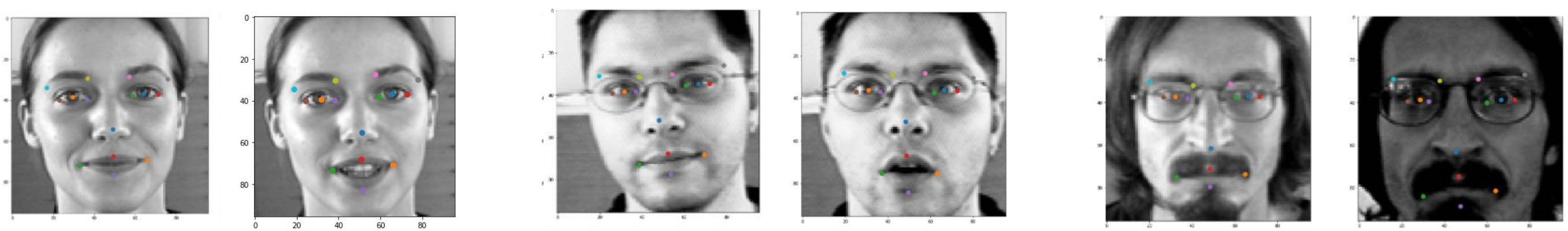
The Project

Objective: Predict locations of 15 key points

- **Source:** Kaggle [Facial Keypoints Detection]
- **Data**:
  - 7k training and 2k test, 96 x 96 images
  - 0-255 grayscale values for images
  - 0-96 x & y coordinates for keypoints
- **Score Metric**: RMSE of test predictions
- Leaderboard Scores: **#1**: 1.53; **#50**: 2.56; Highest: 50
- Our goal: Place top 50; RMSE < 2.56

Data

- Consists of multiple individuals
- Some people are repeated with different poses and different lighting conditions
- Mostly clean data (no cropped or blurry or distorted images)
- 30 key points (15 pairs) are given for each sample



```
COLUMNS = ['left_eye_center_x', 'left_eye_center_y', 'right_eye_center_x',
        'right_eye_center_y', 'left_eye_inner_corner_x',
        'left_eye_inner_corner_y', 'left_eye_outer_corner_x',
        'left_eye_outer_corner_y', 'right_eye_inner_corner_x',
        'right_eye_inner_corner_y', 'right_eye_outer_corner_x',
        'right_eye_outer_corner_y', 'left_eyebrow_inner_end_x',
        'left_eyebrow_inner_end_y', 'left_eyebrow_outer_end_x',
        'left_eyebrow_outer_end_y', 'right_eyebrow_inner_end_x',
        'right_eyebrow_inner_end_y', 'right_eyebrow_outer_end_x',
        'right_eyebrow_outer_end_y', 'nose_tip_x', 'nose_tip_y',
        'mouth_left_corner_x', 'mouth_left_corner_y', 'mouth_right_corner_x',
        'mouth_right_corner_y', 'mouth_center_top_lip_x',
        'mouth_center_top_lip_y', 'mouth_center_bottom_lip_x',
        'mouth_center_bottom_lip_y']
```

We Made A Plan

- **Preprocessing**

  1. Missing values

  2. Separate out labels (Y) from images (X)

- **Step1:** Base NN Model
- **Step2:** Base CNN Model and Tune
- **Step3: Data Augmentation**

  1. Apply different augmentations (increase samples 9x)

  2. Careful when picking transformations (cropping, zooming etc.)

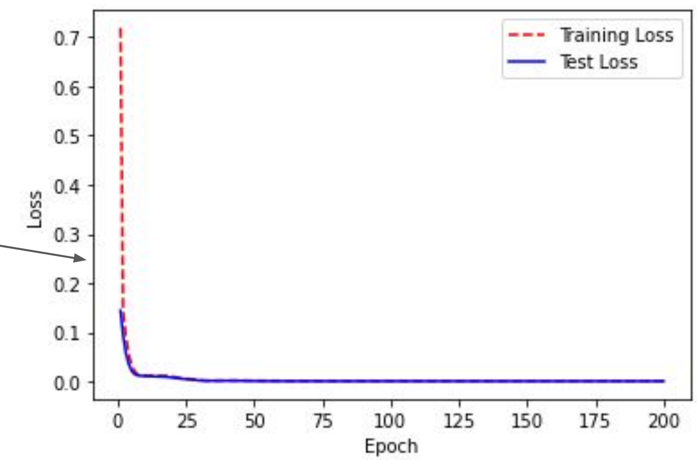
- **Step4:** Expand and tune CNN
- **Submission to Kaggle**
- **Productionizing model (Run on Containers + Cloud)**

Initial Model – Baseline NN

Goal:

- Simple Neural Network Model, no CNN - input 9126 & output 30
- Set up working pipeline for data processing and Kaggle submission

Missing Values	Data Values	NN Architecture	Model Fitting	RMSE
Previous value	0-255 images 0-96 key points	1: 128(ReLu)	Epochs: 200 Batch size: All	36.54
Previous value	**Scaled inputs 0-1**	1: 128(Sigmoid)	“	Errored out
Previous value	Scaled 0-1	**1: 500 (ReLu) 2: 100 (Sigmoid)**	“	4.13
Avg Values	Scaled 0-1	“	**Epochs: 100**	3.96
“	“	“	**Epoch: 50**	9.82
“	“	**+ Dropout(0.5)**	Epoch: 100	7.71
“	“	**No dropout + 1000(ReLu)**	“	3.97




```
model = Sequential()
model.add(Dense(500, input_dim=9216, activation= "relu"))
model.add(Dense(100, activation= "sigmoid"))
model.add(Dense(30))
model.compile(optimizer='adam',
              loss='mean_squared_error',metrics=['mae'])
history = model.fit(train_images,train_labels,epochs = 100,
                    batch_size = numTrainExamples,
                    validation_split = 0.2)
```

Fork of W207_Final_Project_BaseLine With CNN (version 1/16)	~Top 140	3.79534	3.96027

How good was it?

Our base NN model RMSE: 3.96

RMSE for simple average model: ~4.0!!!!

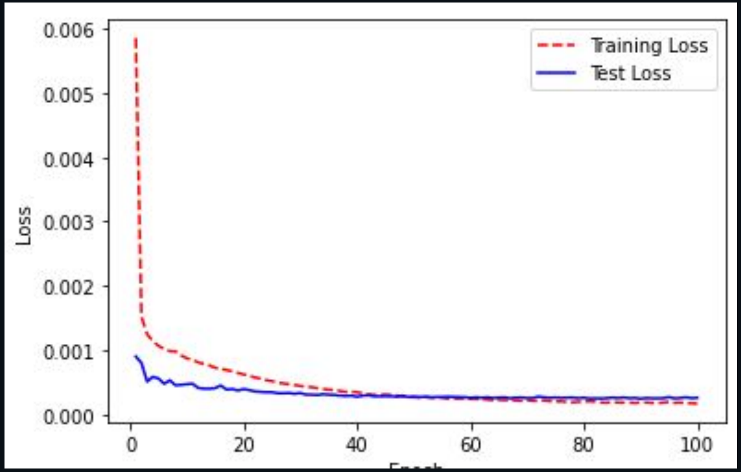
CNN Model Learnings

- **Scaling**: Biggest factor for accuracy improvement
- **Activation**: LeakyReLU better than Relu
- **Kernel size**: Speed vs. accuracy
- **Filters and layers:** Incremental addition & gradual increase
- **Epochs**: Speed vs. overfitting
- **Batch size**: Optimal batch size for speed of convergence

Choose Top 5 performing models on Dev set → Submit to Kaggle → **Kaggle final score: 2.316**

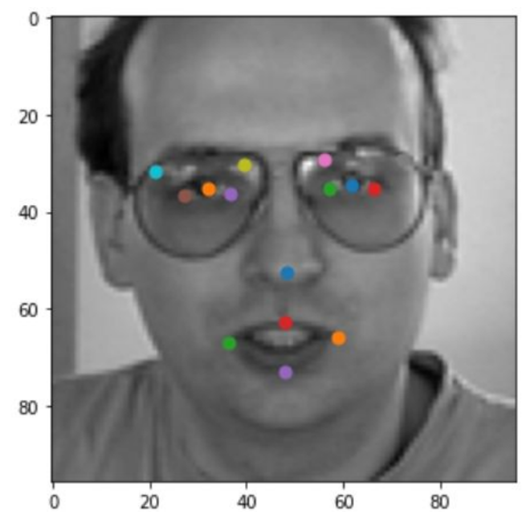
CNN Final Model

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(3,3),padding='same', use_bias=True ,input_shape=(96,96,1)))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(64, kernel_size=(3,3),padding='same', use_bias=True))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(128, kernel_size=(3,3),padding='same', use_bias=True))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(256, kernel_size=(3,3),padding='same', use_bias=True))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(2,2))

model.add(Conv2D(512, kernel_size=(3,3),padding='same', use_bias=True))
model.add(LeakyReLU(alpha=0.1))
model.add(MaxPooling2D(2,2))

model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(30, activation='sigmoid'))

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mse','mae'])
history = model.fit(train_images_2d,train_labels,epochs = 100,batch_size = 100,validation_split = 0.2)
```

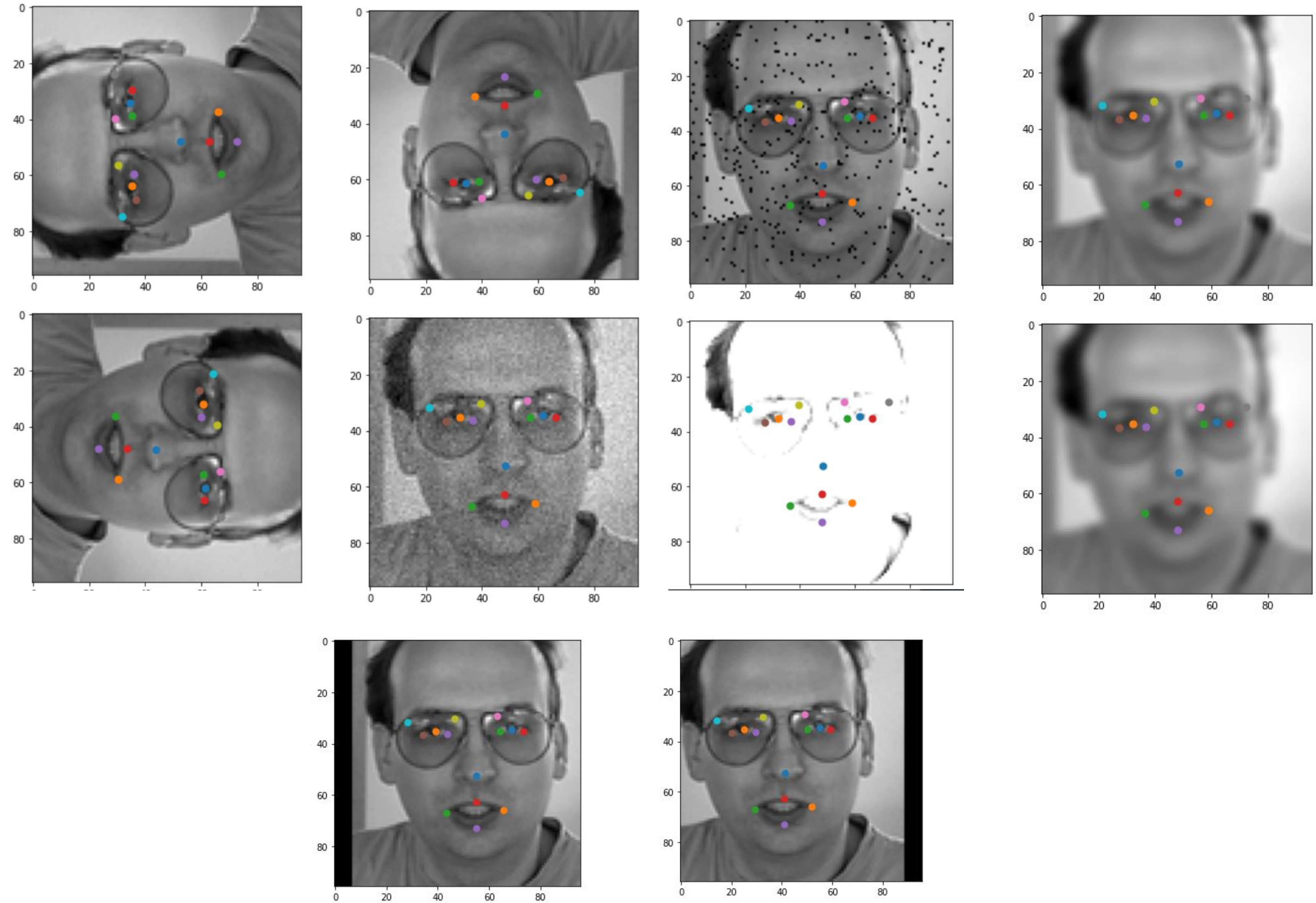


Augmentation

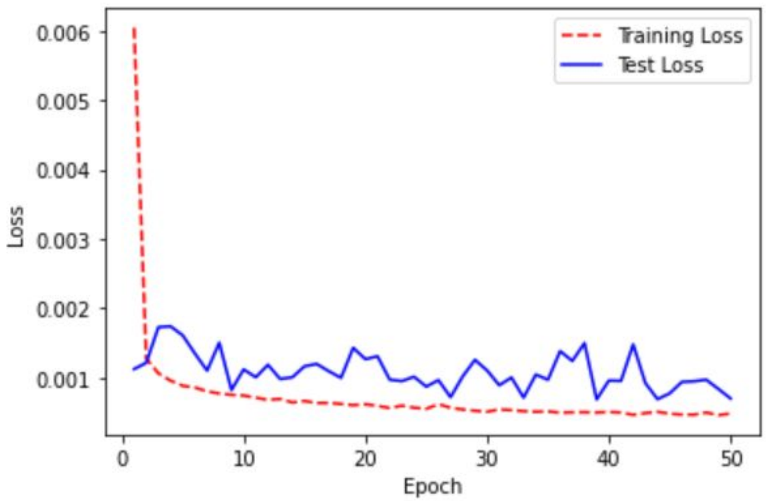
- **Tested several augmentation techniques:**
  1. Image Rotation (only multiples of 90)
  2. Noise Addition (Gaussian, Poisson, Salt & Pepper)
  3. Brightness / Darkness
  4. Translation (Left and Right)
  5. Blurring
- **Did not try other augmentations:**
  1. Partial Rotations (45 degrees)
  2. Image Zoom
  3. Horizontal Flip (Hard to place KeyPoints)
  4. Image Distortion
  5. Random Cropping



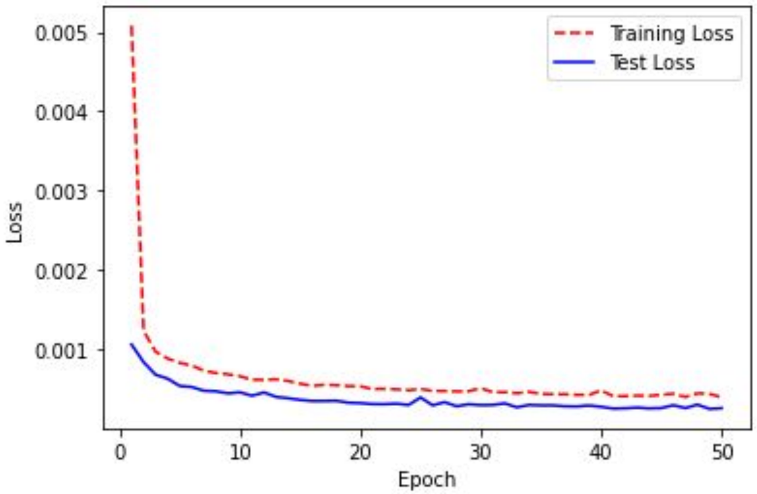
Augmentation



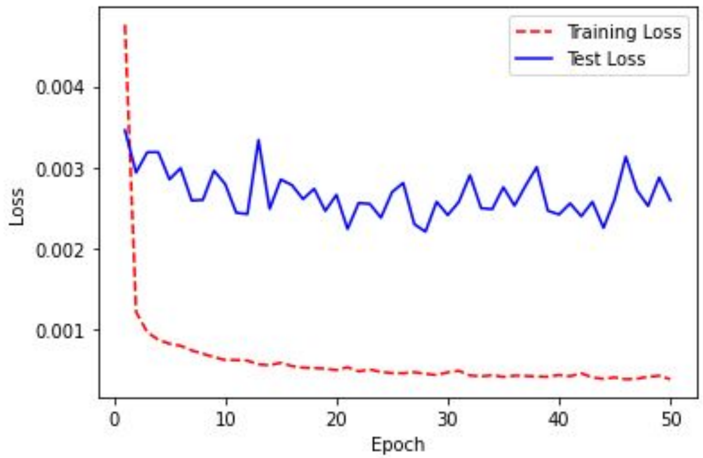
Training with Augmentation



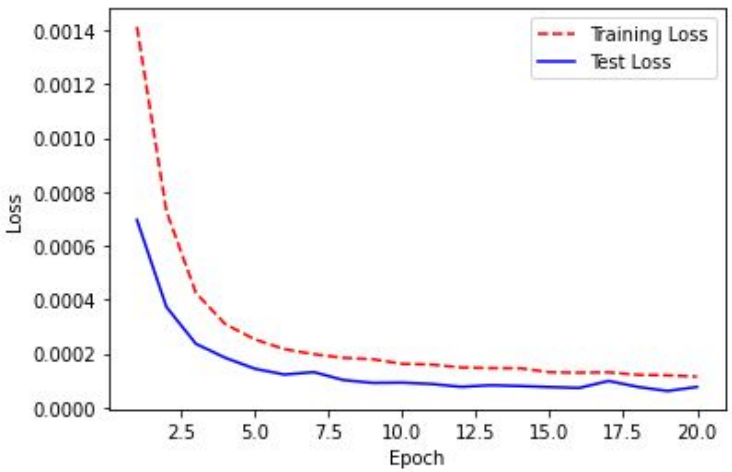
With Noise



With Blurring, Noise



With Blurring, Noise, Rotation, Translation



With Blurring, Noise, Bright / Dark

Productionize Training

- Motivation was to standardize and scale training for large datasets
- Thinking beyond 7000 images. Augmented size is 63k.
- Steps are given here
- IBM Cloud - Volta 100 GPU architecture
- Containerizing + cloud has its benefits:

  1. Better control over compute environment
  2. Can manage dependencies in a customized way (Pillow, OpenCv)
  3. Faster training time (dedicated compute + CUDA runtime + GPUs)
  4. Can be reproduced when dockerized

- Reduced training time from 4.8 min per epoch to 3 seconds per epoch

```
Total params: 7,268,670
Trainable params: 7,264,318
Non-trainable params: 4,352
------------------------------------------------------------
Epoch 1/50
81/81 [==============================] - 288s 4s/step
Epoch 2/50
81/81 [==============================] - 290s 4s/step
Epoch 3/50
11/81 [===>..........................] - ETA: 3:31 - l
```

```
Trainable params: 7,264,318
Non-trainable params: 4,352
________________________________________________________________
Epoch 1/50
81/81 [==============================] - 3s 34ms/step - loss: 0.3393
_mse: 0.1379 - val_mae: 0.3180
Epoch 2/50
81/81 [==============================] - 2s 30ms/step - loss: 0.0151
_mse: 0.0108 - val_mae: 0.0889
Epoch 3/50
```

Final Thoughts

- CNNs > Vanilla NN (for image) - **41.5% improvement from baseline**
- Augmentation didn't give the expected benefit we were hoping to see
- Rely on NN Architecture instead
- Careful on augmentations applied - Parity with Test Data
- Containerizing and productionizing training has many benefits
- Still room for improvement - **approx 44% behind the leaderboard**
- Can try other methods:
  - Pre-trained architectures (VGG, ResNet, GoogLeNet, Transfer Learning)
  - Auto-NN tuning (Hyperopt, Keras tuner)
  - NN Ensembles with cross validation
  - Go beyond Images as features

W207_Final_Project_Final_v4 origCNN, new NN, 3×3 (version 31/32)	2.00107	2.31658

#	Δpub	Team Name	Notebook	Team Members	Score	Entries	Last
1	▲1	olegra			1.28236	4	4y
50		Alex Staravoitau			2.56286	2	4y