W207 Final Project

Facial Keypoints Detection

Prepared by Jia Lu, Navya Sandadi, YuLing Chen

Project Summary

In this project, we first conducted EDA to examine the training dataset. We noticed a large number of missing data in the original dataset. Then we loaded the training data, removed all the missing values, and split into training and validation datasets. After that, we tried a few baseline models: KNN, two layer Neural Nets and CNN. Based on the RMSE in the baseline models, the CNN performs the best. We further improved the performance using data augmentation and hyper parameter tuning. We also tried shift and random brightness data augmentation, which did not provide significant improvement (please see appendix for details). We also tested the forward fill method to replace missing values and did not see much difference. Finally we visualized the performance of our models on the testing dataset by comparing the best one with our baseline model.

We evalute each of the model that we build in this project using RMSE. The best score we had was on modelh_12: **RMSE = 1.36**

In [ ]:

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt

from sklearn.utils import shuffle
from sklearn.datasets import fetch_openml
from sklearn.metrics import classification_report
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import KNeighborsRegressor
import time

from keras import optimizers
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers import Flatten
from keras.utils import np_utils
from keras.datasets import mnist
from keras import backend as K
from os import listdir

from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
```

```
np.random.seed(0)
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.

We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the `%tensorflow_version 1.x` magic: more info.

EDA - Some images have missing data points

In [ ]:

```
from google.colab import drive
from os import listdir
from os.path import isfile, join
drive.mount('/contents/')

FTRAIN = '/contents/My Drive/contents/training.csv'
FTEST = '/contents/My Drive/contents/test.csv'
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?cli
ent_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&
scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%
3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.c
om%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2faut
h%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /contents/
```

In [ ]:

```
fname = FTRAIN
df = pd.read_csv(os.path.expanduser(fname))  # load pandas dataframe
```

In [ ]:

```
def stringToImage(string):
    return np.array([int(item) for item in string.split()]).reshape((96, 9

def plot_faces(nrows=5, ncols=5):
    #Randomly displays some faces from the training data.
    selection = np.random.choice(df.index, size=(nrows*ncols), replace=Fal
    image_strings = df.loc[selection]['Image']
    fig, axes = plt.subplots(figsize=(10, 10), nrows=nrows, ncols=ncols)
    for string, ax in zip(image_strings, axes.ravel()):
        ax.imshow(stringToImage(string), cmap='gray')
        ax.axis('off')

def plot_faces_and_keypoints(nrows=5, ncols=5):
    #Randomly displays some faces from the training data with their keypoi
    selection = np.random.choice(df.index, size=(nrows*ncols), replace=Fal
    image_strings = df.loc[selection]['Image']
    keypoint_cols = list(df.columns)[:-1]
    keypoints = df.loc[selection][keypoint_cols]
    fig, axes = plt.subplots(figsize=(10, 10), nrows=nrows, ncols=ncols)
    for string, (iloc, keypoint), ax in zip(image_strings, keypoints.iterr
```

```
        xy = keypoint.values.reshape((15, 2))
        ax.imshow(stringToImage(string), cmap='gray')
        ax.plot(xy[:, 0], xy[:, 1], 'b.')
        ax.axis('off')
```

In [ ]:

```
plot_faces_and_keypoints()
```

Load Data

In [ ]:

```
def load(test=False, cols=None):
    """Loads data from FTEST if *test* is True, otherwise from FTRAIN.
    Pass a list of *cols* if you're only interested in a subset of the
    target columns.
    """
    fname = FTEST if test else FTRAIN
    df = pd.read_csv(os.path.expanduser(fname))  # load pandas dataframe
    #print (df.head())

    # The Image column has pixel values separated by space; convert
    # the values to numpy arrays:
    df['Image'] = df['Image'].apply(lambda im: np.fromstring(im, sep=' '))
```

```python
    if cols:  # get a subset of columns
        df = df[list(cols) + ['Image']]

    print(df.count())  # prints the number of values for each column
    df = df.dropna()  # drop all rows that have missing values in them
    #df.fillna(method = 'ffill', inplace = True)

    X = np.vstack(df['Image'].values) / 255.  # scale pixel values to [0,
    X = X.astype(np.float32)

    if not test:  # only FTRAIN has any target columns
        y = df[df.columns[:-1]].values
        y = (y - 48) / 48  # scale target coordinates to [-1, 1]
        X, y = shuffle(X, y, random_state=42)  # shuffle train data
        y = y.astype(np.float32)
    else:
        y = None

    return X, y
```

In [ ]:

```python
X, y = load()
print("X.shape == {}; X.min == {:.3f}; X.max == {:.3f}".format(
    X.shape, X.min(), X.max()))
print("y.shape == {}; y.min == {:.3f}; y.max == {:.3f}".format(
    y.shape, y.min(), y.max()))
```

```
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
Image                        7049
dtype: int64
X.shape == (2140, 9216); X.min == 0.000; X.max == 1.000
y.shape == (2140, 30); y.min == -0.920; y.max == 0.996
```

```
In [ ]: X
```

```
Out[ ]: array([[0.79607844, 0.7058824 , 0.59607846, ..., 0.11372549, 0.14901961,
         0.17254902],
        [0.12941177, 0.21960784, 0.34509805, ..., 0.21176471, 0.22352941,
         0.23137255],
        [0.34509805, 0.12156863, 0.10196079, ..., 0.10980392, 0.11764706,
         0.12156863],
        ...,
        [0.18039216, 0.18431373, 0.20392157, ..., 0.73333335, 0.73333335,
         0.7254902 ],
        [0.42352942, 0.18039216, 0.11764706, ..., 1.        , 0.99215686,
         1.        ],
        [0.11372549, 0.08235294, 0.09803922, ..., 0.37254903, 0.3882353 ,
         0.38431373]], dtype=float32)
```

```
In [ ]: y
```

```
Out[ ]: array([[ 0.3816111 , -0.21757638, -0.40208334, ...,  0.4403889 ,
          0.03376389,  0.8259514 ],
        [ 0.4330242 , -0.21624877, -0.3466828 , ...,  0.52398473,
         -0.08612007,  0.5925943 ],
        [ 0.3582826 , -0.26738405, -0.388     , ...,  0.41946375,
         -0.01155797,  0.67042756],
        ...,
        [ 0.40102914, -0.25295144, -0.3799806 , ...,  0.38052428,
         -0.01551456,  0.7536699 ],
        [ 0.45343795, -0.1929708 , -0.4018394 , ...,  0.7215474 ,
         -0.00937226,  0.8918613 ],
        [ 0.45054716, -0.32877925, -0.4011132 , ...,  0.4048302 ,
          0.06266037,  0.7168113 ]], dtype=float32)
```

Split data set into training and testing (dev) data set

```
In [ ]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
        print(X_train.shape)
```

```
(1712, 9216)
```

```
In [ ]: print(y_train.shape)
```

```
(1712, 30)
```

```
In [ ]: len(X_train)
```

```
Out[ ]: 1712
```

```
In [ ]: y_train
```

```
Out[ ]: array([[ 0.30770212, -0.26010212, -0.3642553 , ...,  0.4181844 ,
          0.00341844,  0.6463972 ],
        [ 0.36245108, -0.22979438, -0.3536102 , ...,  0.50230634,
          0.07019986,  0.6201859 ],
```

```
[ 0.34921804, -0.27222106, -0.33625564, ...,  0.44685715,
  0.08712782,  0.7358346 ],
...,
[ 0.38566402, -0.35536698, -0.38166365, ...,  0.57420635,
  0.30079365,  0.7191043 ],
[ 0.40919355, -0.22560807, -0.34046775, ...,  0.50241774,
  0.04158064,  0.75470966],
[ 0.34777445, -0.26929024, -0.4049173 , ...,  0.32210526,
  0.01846617,  0.5707669 ]], dtype=float32)
```

Baseline Models

- Model 1 - KNN

In [ ]:

```
for i in range(1, 50, 5):
    knn = KNeighborsRegressor(i)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print ('K = %d, MSE = %.4f' %(i, mse))
```

```
K = 1, MSE = 0.0031
K = 6, MSE = 0.0024
K = 11, MSE = 0.0025
K = 16, MSE = 0.0026
K = 21, MSE = 0.0026
K = 26, MSE = 0.0027
K = 31, MSE = 0.0028
K = 36, MSE = 0.0029
K = 41, MSE = 0.0029
K = 46, MSE = 0.0030
```

In [ ]:

```
for i in range(1, 10, 1):
    knn = KNeighborsRegressor(i)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print ('K = %d, MSE = %.4f' %(i, mse))
```

```
K = 1, MSE = 0.0031
K = 2, MSE = 0.0026
K = 3, MSE = 0.0025
K = 4, MSE = 0.0025
K = 5, MSE = 0.0024
K = 6, MSE = 0.0024
K = 7, MSE = 0.0024
K = 8, MSE = 0.0024
K = 9, MSE = 0.0024
```

In [ ]:

```
#root-mean-square error
knn = KNeighborsRegressor(5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(0.0026) * 48
print ('K = 5, MSE = %.4f' % (mse))
print('root-mean-square error is %.5f' % (rmse))
```

```
K = 5, MSE = 0.0024
root-mean-square error is 2.44753
```

- Model 2 - Two layer Neural Net

In [ ]:

```
model2 = Sequential()
model2.add(Dense(units=100, input_dim=9216, activation='relu'))
model2.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

model2.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

print(model2.summary())

history = model2.fit(X_train, y_train, shuffle=False,verbose=0, epochs=100

hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
print(hist)

loss, mae, mse = model2.evaluate(X_test, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:66: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:541: The name tf.placeholder is deprecated. Pleas
e use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:4432: The name tf.random_uniform is deprecated. P
lease use tf.random.uniform instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optim
izers.py:793: The name tf.train.Optimizer is deprecated. Please use tf.com
pat.v1.train.Optimizer instead.

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 100)               921700
_________________________________________________________________
dense_2 (Dense)              (None, 30)                3030
=================================================================
Total params: 924,730
Trainable params: 924,730
Non-trainable params: 0
_________________________________________________________________
None
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
```

```
nd/tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Pleas
e use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:1020: The name tf.assign is deprecated. Please us
e tf.compat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:3005: The name tf.Session is deprecated. Please u
se tf.compat.v1.Session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:190: The name tf.get_default_session is deprecate
d. Please use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:197: The name tf.ConfigProto is deprecated. Pleas
e use tf.compat.v1.ConfigProto instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:207: The name tf.global_variables is deprecated.
Please use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:216: The name tf.is_variable_initialized is depre
cated. Please use tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:223: The name tf.variables_initializer is depreca
ted. Please use tf.compat.v1.variables_initializer instead.

        loss  mean_absolute_error  mean_squared_error  epoch
0   3.586261             0.686706            3.586261      0
1   0.132683             0.309734            0.132683      1
2   0.102068             0.264956            0.102068      2
3   0.077267             0.220043            0.077267      3
4   0.057476             0.179192            0.057476      4
..       ...                  ...                 ...    ...
95  0.004444             0.048480            0.004444     95
96  0.004444             0.048480            0.004444     96
97  0.004444             0.048480            0.004444     97
98  0.004444             0.048480            0.004444     98
99  0.004444             0.048480            0.004444     99

[100 rows x 4 columns]
Testing set RMSE:  3.14
```

- Model 3 - CNN

In [ ]:

```
X_train_b = X_train.reshape(X_train.shape[0], 96, 96, 1)
X_test_b = X_test.reshape(X_test.shape[0], 96, 96, 1)
```

In [ ]:

```
X_train_b.shape
```

Out[ ]:

```
(1712, 96, 96, 1)
```

In [ ]:

```
X_train
```

Out[ ]:
```
array([[0.9607843 , 0.9607843 , 0.9607843 , ..., 0.3372549 , 0.3254902 ,
        0.3529412 ],
       [0.38039216, 0.3529412 , 0.25882354, ..., 0.08235294, 0.07450981,
        0.09411765],
       [0.9607843 , 0.9607843 , 0.9607843 , ..., 0.53333336, 0.5882353 ,
        0.58431375],
       ...,
       [0.10980392, 0.01960784, 0.05882353, ..., 0.5921569 , 0.53333336,
        0.4862745 ],
       [0.98039216, 0.9764706 , 0.9647059 , ..., 0.17254902, 0.25882354,
        0.38039216],
       [0.30980393, 0.30980393, 0.2901961 , ..., 0.03137255, 0.02745098,
        0.04313726]], dtype=float32)
```

In [ ]:
```
model3 = Sequential()
model3.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(96

model3.add(Conv2D(64, (3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.2))

model3.add(Flatten())

model3.add(Dense(units=50, input_dim=128, activation='relu'))
model3.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

model3.compile(loss='mse',
              optimizer=optimizer,
              metrics=['mae', 'mse'])

history = model3.fit(X_train_b, y_train, shuffle=False,verbose=0, epochs=1

hist = pd.DataFrame(history.history)
hist['epoch'] = history.epoch
print(hist)

loss, mae, mse = model3.evaluate(X_test_b, y_test, verbose=2)
rmse = np.sqrt(mse) * 48
print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
        loss  mean_absolute_error  mean_squared_error  epoch
0   2.790509             0.489261            2.790509      0
1   0.044726             0.169311            0.044726      1
2   0.017062             0.097593            0.017062      2
3   0.010859             0.079607            0.010859      3
4   0.008755             0.070995            0.008755      4
..       ...                  ...                 ...    ...
95  0.000493             0.016741            0.000493     95
96  0.000477             0.016417            0.000477     96
97  0.000484             0.016501            0.000484     97
98  0.000480             0.016489            0.000480     98
99  0.000481             0.016535            0.000481     99

[100 rows x 4 columns]
Testing set RMSE:  2.09
```

Data Augmentation

Vertical Flip

In [ ]:

```
def plot_sample(x, y, axis):
    img = x.reshape(96, 96)
    axis.imshow(img, cmap='gray')
    axis.scatter(y[0::2] * 48 + 48, y[1::2] * 48 + 48, marker='x', s=10)
```

In [ ]:

```
class DataModifier(object):
    def fit(self,X_,y_):
        return(NotImplementedError)

class FlipPic(DataModifier):
    def __init__(self,flip_indices=None):
        if flip_indices is None:
            flip_indices = [
                (0, 2), (1, 3),
                (4, 8), (5, 9), (6, 10), (7, 11),
                (12, 16), (13, 17), (14, 18), (15, 19),
                (22, 24), (23, 25)
                ]

        self.flip_indices = flip_indices

    def fit(self,X_batch,y_batch):
        batch_size = X_batch.shape[0]
        indices = np.random.choice(batch_size, batch_size//2, replace=Fals
        X_batch[indices] = X_batch[indices, :, ::-1,:]
        y_batch[indices, ::2] = y_batch[indices, ::2] * -1

        # flip left eye to right eye, left mouth to right mouth, etc.
        for a, b in self.flip_indices:
            y_batch[indices, a], y_batch[indices, b] = (y_batch[indices, b
        return X_batch, y_batch
```

In [ ]:

```
from keras.preprocessing.image import ImageDataGenerator

generator = ImageDataGenerator()
modifier = FlipPic()

fig = plt.figure(figsize=(7,7))

count = 1
for batch in generator.flow(X_train_b[:2],y_train[:2]):
    X_batch, y_batch = modifier.fit(*batch)

    ax = fig.add_subplot(3,3, count,xticks=[],yticks=[])
    plot_sample(X_batch[0],y_batch[0],ax)
    count += 1
    if count == 10:
        break
plt.show()
```

In [ ]:

```
def fit(model,modifier,train,validation,batch_size=32,epochs=2000,print_ev

    X_train_b, y_train = train
    X_test_b, y_test    = validation

    generator = ImageDataGenerator()

    history = {"loss":[],"val_loss":[]}
    for e in range(epochs):
        if e % print_every == 0:
            print('Epoch {:4}:'.format(e)),

        batches = 0
        loss_epoch = []
        for X_batch, y_batch in generator.flow(X_train_b, y_train, batch_s
            X_batch, y_batch = modifier.fit(X_batch, y_batch)
            hist = model.fit(X_batch, y_batch,verbose=False,epochs=1)
            loss_epoch.extend(hist.history["loss"])
            batches += 1
            if batches >= len(X_train_b) / batch_size:
                # we need to break the loop by hand because
                # the generator loops indefinitely
                break
        loss = np.mean(loss_epoch)
        history["loss"].append(loss)

        y_pred = model.predict(X_test_b)
        val_loss = np.mean((y_pred - y_test)**2)
        history["val_loss"].append(val_loss)
        if e % print_every == 0:
            print("loss - {:6.5f}, val_loss - {:6.5f}".format(loss,val_los
        min_val_loss = np.min(history["val_loss"])
```

In [ ]:

```
model4 = Sequential()
model4.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(96
model4.add(Conv2D(64, (3, 3), activation='relu'))
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.2))
model4.add(Flatten())
model4.add(Dense(units=50, input_dim=128, activation='relu'))
model4.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

model4.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(model4, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=100)

loss, mae, mse = model4.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:4267: The name tf.nn.max_pool is deprecated. Plea
se use tf.nn.max_pool2d instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:148: The name tf.placeholder_with_default is depr
ecated. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backe
nd/tensorflow_backend.py:3733: calling dropout (from tensorflow.python.op
s.nn_ops) with keep_prob is deprecated and will be removed in a future ver
sion.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1
- keep_prob`.
Epoch    0:
loss - 6.65896, val_loss - 0.11203
Epoch   10:
loss - 0.00654, val_loss - 0.00378
Epoch   20:
loss - 0.00281, val_loss - 0.00235
Epoch   30:
loss - 0.00165, val_loss - 0.00189
Epoch   40:
loss - 0.00103, val_loss - 0.00163
Epoch   50:
loss - 0.00073, val_loss - 0.00146
Epoch   60:
loss - 0.00058, val_loss - 0.00128
Epoch   70:
loss - 0.00047, val_loss - 0.00125
Epoch   80:
loss - 0.00042, val_loss - 0.00113
Epoch   90:
loss - 0.00040, val_loss - 0.00112
Testing set RMSE:  1.60
```

CNN HyperParameter Tuning

We obtained the optimal CNN model with the following combination of hyperparameters:

- Kernel Size = 3
- Dropout rate = 0.2
- Number of Convolutional layers = 5
- Need MaxPooling between each Convolutional layers
- Epoch = 300

Our best score after HyperParameter tuning is 1.38

1. Kernel Size Tunning

- We tried the kernel sizes smaller or bigger than 3. Both of them had worse or similar performances compared with kernel_size 3. We understood that usually bigger kernel_size works with larger data sets; smaller kernel_size works well with more complex architecture of the CNN.

In [ ]:

```
### Kernel size = 2
modelh_1 = Sequential()
modelh_1.add(Conv2D(32, kernel_size=(2, 2),activation='relu',input_shape=(
modelh_1.add(Conv2D(64, (3, 3), activation='relu'))
modelh_1.add(MaxPooling2D(pool_size=(2, 2)))
modelh_1.add(Dropout(0.2))
modelh_1.add(Flatten())
modelh_1.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_1.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_1.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_1, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=100)

loss, mae, mse = modelh_1.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 3.78959, val_loss - 0.08097
Epoch   10:
loss - 0.00505, val_loss - 0.00382
Epoch   20:
loss - 0.00342, val_loss - 0.00410
Epoch   30:
loss - 0.00242, val_loss - 0.00258
Epoch   40:
loss - 0.00193, val_loss - 0.00244
```

```
Epoch   50:
loss - 0.00177, val_loss - 0.00205
Epoch   60:
loss - 0.00162, val_loss - 0.00201
Epoch   70:
loss - 0.00155, val_loss - 0.00190
Epoch   80:
loss - 0.00146, val_loss - 0.00186
Epoch   90:
loss - 0.00135, val_loss - 0.00182
Testing set RMSE:  1.99
```

In [ ]:

```python
### Kernel size = 4
modelh_2 = Sequential()
modelh_2.add(Conv2D(32, kernel_size=(4, 4),activation='relu',input_shape=(
modelh_2.add(Conv2D(64, (3, 3), activation='relu'))
modelh_2.add(MaxPooling2D(pool_size=(2, 2)))
modelh_2.add(Dropout(0.2))
modelh_2.add(Flatten())
modelh_2.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_2.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_2.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_2, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=100)

loss, mae, mse = modelh_2.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 3.10455, val_loss - 0.06266
Epoch   10:
loss - 0.00438, val_loss - 0.00439
Epoch   20:
loss - 0.00310, val_loss - 0.00251
Epoch   30:
loss - 0.00234, val_loss - 0.00206
Epoch   40:
loss - 0.00166, val_loss - 0.00192
Epoch   50:
loss - 0.00149, val_loss - 0.00185
Epoch   60:
loss - 0.00119, val_loss - 0.00223
Epoch   70:
loss - 0.00104, val_loss - 0.00145
Epoch   80:
loss - 0.00096, val_loss - 0.00172
Epoch   90:
loss - 0.00085, val_loss - 0.00183
Testing set RMSE:  2.47
```

2. Dropout rate tuning

- Dropout=0.2 was used for the following models
- Smaller (0.1) and larger(0.5) dropout rates generated similar or larger RMSE
- Please refer to the following code

In [ ]:

```
### Dropout = 0.5
modelh_3 = Sequential()
modelh_3.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(
modelh_3.add(Conv2D(64, (3, 3), activation='relu'))
modelh_3.add(MaxPooling2D(pool_size=(2, 2)))
modelh_3.add(Dropout(0.5))
modelh_3.add(Flatten())
modelh_3.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_3.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_3.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_3, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=100)

loss, mae, mse = modelh_3.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 0.89752, val_loss - 0.05016
Epoch   10:
loss - 0.00439, val_loss - 0.00325
Epoch   20:
loss - 0.00239, val_loss - 0.00317
Epoch   30:
loss - 0.00181, val_loss - 0.00231
Epoch   40:
loss - 0.00158, val_loss - 0.00182
Epoch   50:
loss - 0.00138, val_loss - 0.00256
Epoch   60:
loss - 0.00118, val_loss - 0.00158
Epoch   70:
loss - 0.00115, val_loss - 0.00182
Epoch   80:
loss - 0.00102, val_loss - 0.00136
Epoch   90:
loss - 0.00090, val_loss - 0.00174
Testing set RMSE:  1.73
```

In [ ]:

```
### Dropout = 0.1
modelh_4 = Sequential()
modelh_4.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(
modelh_4.add(Conv2D(64, (3, 3), activation='relu'))
modelh_4.add(MaxPooling2D(pool_size=(2, 2)))
modelh_4.add(Dropout(0.1))
modelh_4.add(Flatten())
```

```
modelh_4.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_4.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_4.compile(loss='mse',
                 optimizer=optimizer,
                 metrics=['mae', 'mse'])

history = fit(modelh_4, modifier, train=(X_train_b,y_train),
              validation=(X_test_b,y_test),
              batch_size=32,epochs=100)

loss, mae, mse = modelh_4.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 1.86719, val_loss - 0.04390
Epoch   10:
loss - 0.00401, val_loss - 0.00501
Epoch   20:
loss - 0.00241, val_loss - 0.00248
Epoch   30:
loss - 0.00169, val_loss - 0.00242
Epoch   40:
loss - 0.00136, val_loss - 0.00210
Epoch   50:
loss - 0.00114, val_loss - 0.00172
Epoch   60:
loss - 0.00103, val_loss - 0.00330
Epoch   70:
loss - 0.00088, val_loss - 0.00145
Epoch   80:
loss - 0.00081, val_loss - 0.00258
Epoch   90:
loss - 0.00069, val_loss - 0.00177
Testing set RMSE:  1.66
```

3. Convolutional layers and Maxpooling tuning

- Five convolutional layers with Maxpooling in between is optimal
- Four or six layers generated higher or similar RMSE
- We need to have MaxPooling between convolutional layers.
- When removing one MaxPooling between the first and second convolutional layers, we observed much longer training time.

In [ ]:

```
### four convolutional layers with MaxPooling in between
modelh_5 = Sequential()
modelh_5.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(
modelh_5.add(MaxPooling2D(pool_size=(2, 2)))
modelh_5.add(Conv2D(64, (3, 3), activation='relu'))
modelh_5.add(MaxPooling2D(pool_size=(2, 2)))
modelh_5.add(Conv2D(128, (3, 3), activation='relu'))
modelh_5.add(MaxPooling2D(pool_size=(2, 2)))
modelh_5.add(Conv2D(256, (3, 3), activation='relu'))
modelh_5.add(MaxPooling2D(pool_size=(2, 2)))
```

```
modelh_5.add(Dropout(0.2))
modelh_5.add(Flatten())
modelh_5.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_5.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_5.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_5, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=100)

loss, mae, mse = modelh_5.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 0.03946, val_loss - 0.00915
Epoch   10:
loss - 0.00485, val_loss - 0.00443
Epoch   20:
loss - 0.00269, val_loss - 0.00277
Epoch   30:
loss - 0.00184, val_loss - 0.00202
Epoch   40:
loss - 0.00151, val_loss - 0.00144
Epoch   50:
loss - 0.00120, val_loss - 0.00144
Epoch   60:
loss - 0.00101, val_loss - 0.00147
Epoch   70:
loss - 0.00087, val_loss - 0.00114
Epoch   80:
loss - 0.00078, val_loss - 0.00105
Epoch   90:
loss - 0.00067, val_loss - 0.00118
Testing set RMSE:  1.63
```

In [ ]:

```
### five convolutional layers with MaxPooling in between
modelh_6 = Sequential()
modelh_6.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(
modelh_6.add(MaxPooling2D(pool_size=(2, 2)))
modelh_6.add(Conv2D(64, (3, 3), activation='relu'))
modelh_6.add(MaxPooling2D(pool_size=(2, 2)))
modelh_6.add(Conv2D(128, (3, 3), activation='relu'))
modelh_6.add(MaxPooling2D(pool_size=(2, 2)))
modelh_6.add(Conv2D(256, (3, 3), activation='relu'))
modelh_6.add(MaxPooling2D(pool_size=(2, 2)))
modelh_6.add(Conv2D(512, (3, 3), activation='relu'))
modelh_6.add(MaxPooling2D(pool_size=(2, 2)))
modelh_6.add(Dropout(0.2))
modelh_6.add(Flatten())
modelh_6.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_6.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)
```

```
modelh_6.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_6, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=100)

loss, mae, mse = modelh_6.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 0.04272, val_loss - 0.00709
Epoch   10:
loss - 0.00510, val_loss - 0.00439
Epoch   20:
loss - 0.00348, val_loss - 0.00321
Epoch   30:
loss - 0.00151, val_loss - 0.00179
Epoch   40:
loss - 0.00094, val_loss - 0.00105
Epoch   50:
loss - 0.00071, val_loss - 0.00100
Epoch   60:
loss - 0.00060, val_loss - 0.00092
Epoch   70:
loss - 0.00052, val_loss - 0.00098
Epoch   80:
loss - 0.00049, val_loss - 0.00094
Epoch   90:
loss - 0.00045, val_loss - 0.00090
Testing set RMSE:  1.44
```

In [ ]:

```
### six convolutional layers
modelh_66 = Sequential()
modelh_66.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=
modelh_66.add(MaxPooling2D(pool_size=(2, 2)))
modelh_66.add(Conv2D(64, (3, 3), activation='relu'))
modelh_66.add(MaxPooling2D(pool_size=(2, 2)))
modelh_66.add(Conv2D(128, (3, 3), activation='relu'))
modelh_66.add(MaxPooling2D(pool_size=(2, 2)))
modelh_66.add(Conv2D(256, (3, 3), activation='relu'))
modelh_66.add(Conv2D(512, (3, 3), activation='relu'))
modelh_66.add(Conv2D(512, (3, 3), activation='relu'))
modelh_66.add(Dropout(0.2))
modelh_66.add(Flatten())
modelh_66.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_66.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_66.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_66, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
```

```
            batch_size=32,epochs=100)

loss, mae, mse = modelh_66.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 5.31502, val_loss - 0.03192
Epoch   10:
loss - 0.00494, val_loss - 0.00501
Epoch   20:
loss - 0.00236, val_loss - 0.00293
Epoch   30:
loss - 0.00147, val_loss - 0.00154
Epoch   40:
loss - 0.00108, val_loss - 0.00137
Epoch   50:
loss - 0.00088, val_loss - 0.00129
Epoch   60:
loss - 0.00072, val_loss - 0.00090
Epoch   70:
loss - 0.00060, val_loss - 0.00095
Epoch   80:
loss - 0.00050, val_loss - 0.00087
Epoch   90:
loss - 0.00043, val_loss - 0.00091
Testing set RMSE:  1.45
```

In [ ]:

```
### Removing one MaxPooling between convolutional layers
### Resulted in much longer training time and higher RMSE
modelh_6_1 = Sequential()
modelh_6_1.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape

modelh_6_1.add(Conv2D(64, (3, 3), activation='relu'))
modelh_6_1.add(MaxPooling2D(pool_size=(2, 2)))
modelh_6_1.add(Conv2D(128, (3, 3), activation='relu'))
modelh_6_1.add(MaxPooling2D(pool_size=(2, 2)))
modelh_6_1.add(Conv2D(256, (3, 3), activation='relu'))
modelh_6_1.add(MaxPooling2D(pool_size=(2, 2)))
modelh_6_1.add(Conv2D(512, (3, 3), activation='relu'))
modelh_6_1.add(MaxPooling2D(pool_size=(2, 2)))
modelh_6_1.add(Dropout(0.2))
modelh_6_1.add(Flatten())
modelh_6_1.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_6_1.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_6_1.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_6_1, modifier, train=(X_train_b,y_train),
            validation=(X_test_b,y_test),
            batch_size=32,epochs=100)

loss, mae, mse = modelh_6_1.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48
```

```
print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 0.15158, val_loss - 0.00766
Epoch   10:
loss - 0.00500, val_loss - 0.00457
Epoch   20:
loss - 0.00301, val_loss - 0.00255
Epoch   30:
loss - 0.00194, val_loss - 0.00173
Epoch   40:
loss - 0.00146, val_loss - 0.00139
Epoch   50:
loss - 0.00114, val_loss - 0.00120
Epoch   60:
loss - 0.00100, val_loss - 0.00146
Epoch   70:
loss - 0.00088, val_loss - 0.00099
Epoch   80:
loss - 0.00076, val_loss - 0.00104
Epoch   90:
loss - 0.00064, val_loss - 0.00115
Testing set RMSE:  1.45
```

4. Epoch tuning

- Epoch = 300 is optimal
- Smaller or larger Epoch generated higher RMSE
- Please refer to the following code

In [ ]:

```
### Epoch = 40
modelh_8 = Sequential()
modelh_8.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(
modelh_8.add(MaxPooling2D(pool_size=(2, 2)))
modelh_8.add(Conv2D(64, (3, 3), activation='relu'))
modelh_8.add(MaxPooling2D(pool_size=(2, 2)))
modelh_8.add(Conv2D(128, (3, 3), activation='relu'))
modelh_8.add(MaxPooling2D(pool_size=(2, 2)))
modelh_8.add(Conv2D(256, (3, 3), activation='relu'))
modelh_8.add(MaxPooling2D(pool_size=(2, 2)))
modelh_8.add(Conv2D(512, (3, 3), activation='relu'))
modelh_8.add(MaxPooling2D(pool_size=(2, 2)))
modelh_8.add(Dropout(0.2))
modelh_8.add(Flatten())
modelh_8.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_8.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_8.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_8, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=40)

loss, mae, mse = modelh_8.evaluate(X_test_b, y_test, verbose=2)
```

```
rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 0.03578, val_loss - 0.00799
Epoch   10:
loss - 0.00505, val_loss - 0.00538
Epoch   20:
loss - 0.00276, val_loss - 0.00232
Epoch   30:
loss - 0.00136, val_loss - 0.00199
Testing set RMSE:  1.58
```

In [ ]:

```
### Epoch = 100
modelh_9 = Sequential()
modelh_9.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(
modelh_9.add(MaxPooling2D(pool_size=(2, 2)))
modelh_9.add(Conv2D(64, (3, 3), activation='relu'))
modelh_9.add(MaxPooling2D(pool_size=(2, 2)))
modelh_9.add(Conv2D(128, (3, 3), activation='relu'))
modelh_9.add(MaxPooling2D(pool_size=(2, 2)))
modelh_9.add(Conv2D(256, (3, 3), activation='relu'))
modelh_9.add(MaxPooling2D(pool_size=(2, 2)))
modelh_9.add(Conv2D(512, (3, 3), activation='relu'))
modelh_9.add(MaxPooling2D(pool_size=(2, 2)))
modelh_9.add(Dropout(0.2))
modelh_9.add(Flatten())
modelh_9.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_9.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_9.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_9, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=100)

loss, mae, mse = modelh_9.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 0.05028, val_loss - 0.00832
Epoch   10:
loss - 0.00515, val_loss - 0.00433
Epoch   20:
loss - 0.00347, val_loss - 0.00329
Epoch   30:
loss - 0.00159, val_loss - 0.00198
Epoch   40:
loss - 0.00104, val_loss - 0.00178
Epoch   50:
loss - 0.00076, val_loss - 0.00097
Epoch   60:
loss - 0.00061, val_loss - 0.00101
```

```
Epoch   70:
loss - 0.00054, val_loss - 0.00092
Epoch   80:
loss - 0.00049, val_loss - 0.00087
Epoch   90:
loss - 0.00043, val_loss - 0.00087
Testing set RMSE:  1.43
```

In [ ]:

```
### Epoch = 200
modelh_10 = Sequential()
modelh_10.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=
modelh_10.add(MaxPooling2D(pool_size=(2, 2)))
modelh_10.add(Conv2D(64, (3, 3), activation='relu'))
modelh_10.add(MaxPooling2D(pool_size=(2, 2)))
modelh_10.add(Conv2D(128, (3, 3), activation='relu'))
modelh_10.add(MaxPooling2D(pool_size=(2, 2)))
modelh_10.add(Conv2D(256, (3, 3), activation='relu'))
modelh_10.add(MaxPooling2D(pool_size=(2, 2)))
modelh_10.add(Conv2D(512, (3, 3), activation='relu'))
modelh_10.add(MaxPooling2D(pool_size=(2, 2)))
modelh_10.add(Dropout(0.2))
modelh_10.add(Flatten())
modelh_10.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_10.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_10.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_10, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=200)

loss, mae, mse = modelh_10.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 0.04466, val_loss - 0.00719
Epoch   10:
loss - 0.00512, val_loss - 0.00446
Epoch   20:
loss - 0.00297, val_loss - 0.00229
Epoch   30:
loss - 0.00140, val_loss - 0.00139
Epoch   40:
loss - 0.00093, val_loss - 0.00135
Epoch   50:
loss - 0.00069, val_loss - 0.00093
Epoch   60:
loss - 0.00056, val_loss - 0.00096
Epoch   70:
loss - 0.00051, val_loss - 0.00082
Epoch   80:
loss - 0.00046, val_loss - 0.00085
Epoch   90:
loss - 0.00043, val_loss - 0.00084
Epoch  100:
```

```
loss - 0.00040, val_loss - 0.00082
Epoch  110:
loss - 0.00038, val_loss - 0.00095
Epoch  120:
loss - 0.00037, val_loss - 0.00086
Epoch  130:
loss - 0.00035, val_loss - 0.00084
Epoch  140:
loss - 0.00034, val_loss - 0.00085
Epoch  150:
loss - 0.00032, val_loss - 0.00081
Epoch  160:
loss - 0.00032, val_loss - 0.00084
Epoch  170:
loss - 0.00032, val_loss - 0.00079
Epoch  180:
loss - 0.00031, val_loss - 0.00095
Epoch  190:
loss - 0.00030, val_loss - 0.00084
Testing set RMSE:  1.43
```

In [ ]:

```
### Epoch = 300
modelh_11 = Sequential()
modelh_11.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=
modelh_11.add(MaxPooling2D(pool_size=(2, 2)))
modelh_11.add(Conv2D(64, (3, 3), activation='relu'))
modelh_11.add(MaxPooling2D(pool_size=(2, 2)))
modelh_11.add(Conv2D(128, (3, 3), activation='relu'))
modelh_11.add(MaxPooling2D(pool_size=(2, 2)))
modelh_11.add(Conv2D(256, (3, 3), activation='relu'))
modelh_11.add(MaxPooling2D(pool_size=(2, 2)))
modelh_11.add(Conv2D(512, (3, 3), activation='relu'))
modelh_11.add(MaxPooling2D(pool_size=(2, 2)))
modelh_11.add(Dropout(0.2))
modelh_11.add(Flatten())
modelh_11.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_11.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_11.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_11, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=300)

loss, mae, mse = modelh_11.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 0.03814, val_loss - 0.00634
Epoch   10:
loss - 0.00502, val_loss - 0.00462
Epoch   20:
loss - 0.00381, val_loss - 0.00323
Epoch   30:
loss - 0.00168, val_loss - 0.00163
```

```
Epoch  40:
loss - 0.00102, val_loss - 0.00156
Epoch  50:
loss - 0.00072, val_loss - 0.00105
Epoch  60:
loss - 0.00061, val_loss - 0.00100
Epoch  70:
loss - 0.00053, val_loss - 0.00086
Epoch  80:
loss - 0.00048, val_loss - 0.00093
Epoch  90:
loss - 0.00043, val_loss - 0.00092
Epoch 100:
loss - 0.00043, val_loss - 0.00122
Epoch 110:
loss - 0.00039, val_loss - 0.00086
Epoch 120:
loss - 0.00036, val_loss - 0.00092
Epoch 130:
loss - 0.00036, val_loss - 0.00083
Epoch 140:
loss - 0.00035, val_loss - 0.00084
Epoch 150:
loss - 0.00033, val_loss - 0.00095
Epoch 160:
loss - 0.00032, val_loss - 0.00084
Epoch 170:
loss - 0.00031, val_loss - 0.00081
Epoch 180:
loss - 0.00031, val_loss - 0.00084
Epoch 190:
loss - 0.00030, val_loss - 0.00087
Epoch 200:
loss - 0.00029, val_loss - 0.00083
Epoch 210:
loss - 0.00029, val_loss - 0.00082
Epoch 220:
loss - 0.00029, val_loss - 0.00093
Epoch 230:
loss - 0.00028, val_loss - 0.00088
Epoch 240:
loss - 0.00028, val_loss - 0.00092
Epoch 250:
loss - 0.00027, val_loss - 0.00085
Epoch 260:
loss - 0.00027, val_loss - 0.00084
Epoch 270:
loss - 0.00027, val_loss - 0.00081
Epoch 280:
loss - 0.00026, val_loss - 0.00087
Epoch 290:
loss - 0.00026, val_loss - 0.00094
Testing set RMSE:  1.38
```

In [ ]:

```
### Epoch = 400
modelh_11_1 = Sequential()
modelh_11_1.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shap
modelh_11_1.add(MaxPooling2D(pool_size=(2, 2)))
modelh_11_1.add(Conv2D(64, (3, 3), activation='relu'))
modelh_11_1.add(MaxPooling2D(pool_size=(2, 2)))
modelh_11_1.add(Conv2D(128, (3, 3), activation='relu'))
modelh_11_1.add(MaxPooling2D(pool_size=(2, 2)))
modelh_11_1.add(Conv2D(256, (3, 3), activation='relu'))
```

```
modelh_11_1.add(MaxPooling2D(pool_size=(2, 2)))
modelh_11_1.add(Conv2D(512, (3, 3), activation='relu'))
modelh_11_1.add(MaxPooling2D(pool_size=(2, 2)))
modelh_11_1.add(Dropout(0.2))
modelh_11_1.add(Flatten())
modelh_11_1.add(Dense(units=50, input_dim=128, activation='relu'))
modelh_11_1.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_11_1.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_11_1, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=400)

loss, mae, mse = modelh_11_1.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 0.04863, val_loss - 0.01396
Epoch   10:
loss - 0.00500, val_loss - 0.00429
Epoch   20:
loss - 0.00271, val_loss - 0.00198
Epoch   30:
loss - 0.00137, val_loss - 0.00126
Epoch   40:
loss - 0.00088, val_loss - 0.00156
Epoch   50:
loss - 0.00067, val_loss - 0.00095
Epoch   60:
loss - 0.00057, val_loss - 0.00099
Epoch   70:
loss - 0.00051, val_loss - 0.00099
Epoch   80:
loss - 0.00047, val_loss - 0.00106
Epoch   90:
loss - 0.00043, val_loss - 0.00090
Epoch  100:
loss - 0.00041, val_loss - 0.00107
Epoch  110:
loss - 0.00037, val_loss - 0.00084
Epoch  120:
loss - 0.00037, val_loss - 0.00085
Epoch  130:
loss - 0.00036, val_loss - 0.00108
Epoch  140:
loss - 0.00034, val_loss - 0.00089
Epoch  150:
loss - 0.00032, val_loss - 0.00089
Epoch  160:
loss - 0.00032, val_loss - 0.00092
Epoch  170:
loss - 0.00030, val_loss - 0.00091
Epoch  180:
loss - 0.00030, val_loss - 0.00084
Epoch  190:
```

```
loss - 0.00029, val_loss - 0.00091
Epoch  200:
loss - 0.00029, val_loss - 0.00087
Epoch  210:
loss - 0.00029, val_loss - 0.00083
Epoch  220:
loss - 0.00028, val_loss - 0.00082
Epoch  230:
loss - 0.00027, val_loss - 0.00091
Epoch  240:
loss - 0.00026, val_loss - 0.00088
Epoch  250:
loss - 0.00027, val_loss - 0.00090
Epoch  260:
loss - 0.00026, val_loss - 0.00088
Epoch  270:
loss - 0.00026, val_loss - 0.00092
Epoch  280:
loss - 0.00026, val_loss - 0.00086
Epoch  290:
loss - 0.00026, val_loss - 0.00087
Epoch  300:
loss - 0.00025, val_loss - 0.00091
Epoch  310:
loss - 0.00025, val_loss - 0.00083
Epoch  320:
loss - 0.00024, val_loss - 0.00085
Epoch  330:
loss - 0.00024, val_loss - 0.00081
Epoch  340:
loss - 0.00024, val_loss - 0.00090
Epoch  350:
loss - 0.00024, val_loss - 0.00084
Epoch  360:
loss - 0.00023, val_loss - 0.00086
Epoch  370:
loss - 0.00024, val_loss - 0.00088
Epoch  380:
loss - 0.00024, val_loss - 0.00090
Epoch  390:
loss - 0.00023, val_loss - 0.00085
Testing set RMSE:  1.40
```

5. Padding

- The same padding on the first layer improved the performance slightly.

In [ ]:

```
### Epoch = 300
modelh_12 = Sequential()
modelh_12.add(Conv2D(32, kernel_size=(3, 3), padding="same", activation='r
modelh_12.add(MaxPooling2D(pool_size=(2, 2)))
modelh_12.add(Conv2D(64, (3, 3), activation='relu'))
modelh_12.add(MaxPooling2D(pool_size=(2, 2)))
modelh_12.add(Conv2D(128, (3, 3), activation='relu'))
modelh_12.add(MaxPooling2D(pool_size=(2, 2)))
modelh_12.add(Conv2D(256, (3, 3), activation='relu'))
modelh_12.add(MaxPooling2D(pool_size=(2, 2)))
modelh_12.add(Conv2D(512, (3, 3), activation='relu'))
modelh_12.add(MaxPooling2D(pool_size=(2, 2)))
modelh_12.add(Dropout(0.2))
modelh_12.add(Flatten())
modelh_12.add(Dense(units=50, input_dim=128, activation='relu'))
```

```python
modelh_12.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

modelh_12.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(modelh_12, modifier, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=300)

loss, mae, mse = modelh_12.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 0.04082, val_loss - 0.01589
Epoch   10:
loss - 0.00506, val_loss - 0.00443
Epoch   20:
loss - 0.00325, val_loss - 0.00261
Epoch   30:
loss - 0.00157, val_loss - 0.00150
Epoch   40:
loss - 0.00097, val_loss - 0.00118
Epoch   50:
loss - 0.00073, val_loss - 0.00101
Epoch   60:
loss - 0.00060, val_loss - 0.00098
Epoch   70:
loss - 0.00052, val_loss - 0.00085
Epoch   80:
loss - 0.00048, val_loss - 0.00089
Epoch   90:
loss - 0.00044, val_loss - 0.00088
Epoch  100:
loss - 0.00042, val_loss - 0.00093
Epoch  110:
loss - 0.00040, val_loss - 0.00081
Epoch  120:
loss - 0.00037, val_loss - 0.00081
Epoch  130:
loss - 0.00037, val_loss - 0.00085
Epoch  140:
loss - 0.00035, val_loss - 0.00080
Epoch  150:
loss - 0.00033, val_loss - 0.00082
Epoch  160:
loss - 0.00032, val_loss - 0.00092
Epoch  170:
loss - 0.00032, val_loss - 0.00085
Epoch  180:
loss - 0.00031, val_loss - 0.00081
Epoch  190:
loss - 0.00030, val_loss - 0.00079
Epoch  200:
loss - 0.00029, val_loss - 0.00082
Epoch  210:
loss - 0.00029, val_loss - 0.00081
Epoch  220:
```

```
loss - 0.00029, val_loss - 0.00086
Epoch  230:
loss - 0.00028, val_loss - 0.00082
Epoch  240:
loss - 0.00027, val_loss - 0.00080
Epoch  250:
loss - 0.00027, val_loss - 0.00088
Epoch  260:
loss - 0.00027, val_loss - 0.00079
Epoch  270:
loss - 0.00026, val_loss - 0.00082
Epoch  280:
loss - 0.00027, val_loss - 0.00084
Epoch  290:
loss - 0.00026, val_loss - 0.00085
Testing set RMSE:  1.38
```

Conclusions

In this project, we explored various machine learning models for facial keypoints detection. We tried a series of models including KNN, two layer Neural Network, and CNN. With Data Augmentation and Hyper Parameter Tuning, we found out that CNN model performs the best. Comparing our best model using CNN with the worse model in our baseline, we do see the obvious accuracy improvement between the two models in the visualization.

In [ ]:

```
!pip install tabletext
```

```
Requirement already satisfied: tabletext in /usr/local/lib/python3.6/dist-packages (0.1)
```

In [ ]:

```
from IPython.display import HTML, display
import tabulate
import tabletext
data = [["Model Names", "KMSE"],
         ["KNN",2.45 ],
          ["2 Layer Neural Net",3.14],
          ["CNN Baseline",2.09],
          ["CNN Baseline + Data Augmentation",1.67],
          ["CNN Data Augmentation + HyperParameter Tunning", 1.36]]
#display(HTML(tabulate.tabulate(table, tablefmt='html')))

print (tabletext.to_text(data))
```

Model Names	KMSE
KNN	2.45
2 Layer Neural Net	3.14
CNN Baseline	2.09
CNN Baseline + Data Augmentation	1.67
CNN Data Augmentation + HyperParameter Tunning	1.36

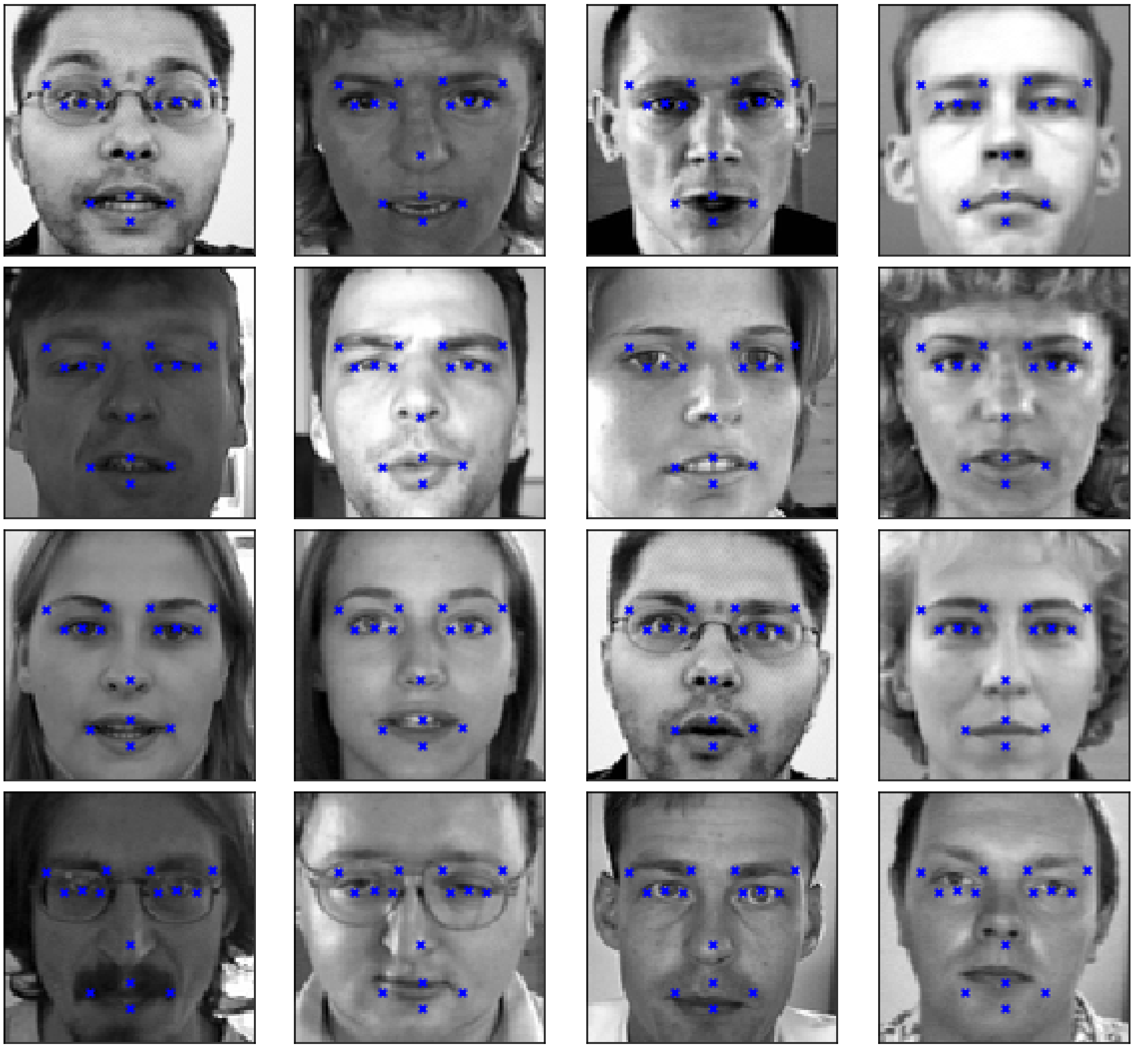
Two Layer Neural Net Visualization

In [ ]:

```
 def plot_sample(x, y, axis):
     img = x.reshape(96, 96)
     axis.imshow(img, cmap='gray')
     axis.scatter(y[0::2] * 48 + 48, y[1::2] * 48 + 48, marker='x', s=14,

X_train, _ = load(test=True)
y_pred = model2.predict(X_train)


fig = plt.figure(figsize=(8, 8))
fig.subplots_adjust(
    left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)

for i in range(16):
    ax = fig.add_subplot(4, 4, i + 1, xticks=[], yticks=[])
    plot_sample(X_train[i], y_pred[i], ax)
fig.subplots_adjust(top=0.90)
fig.suptitle('Baseline Model (Two Layer Neural Net)', fontsize=18)
plt.show()
```

```
ImageId    1783
Image      1783
dtype: int64
```

Baseline Model (Two Layer Neural Net)



Best CNN Model with Data Augmentation and Hyper Parameter Tuning Visualization

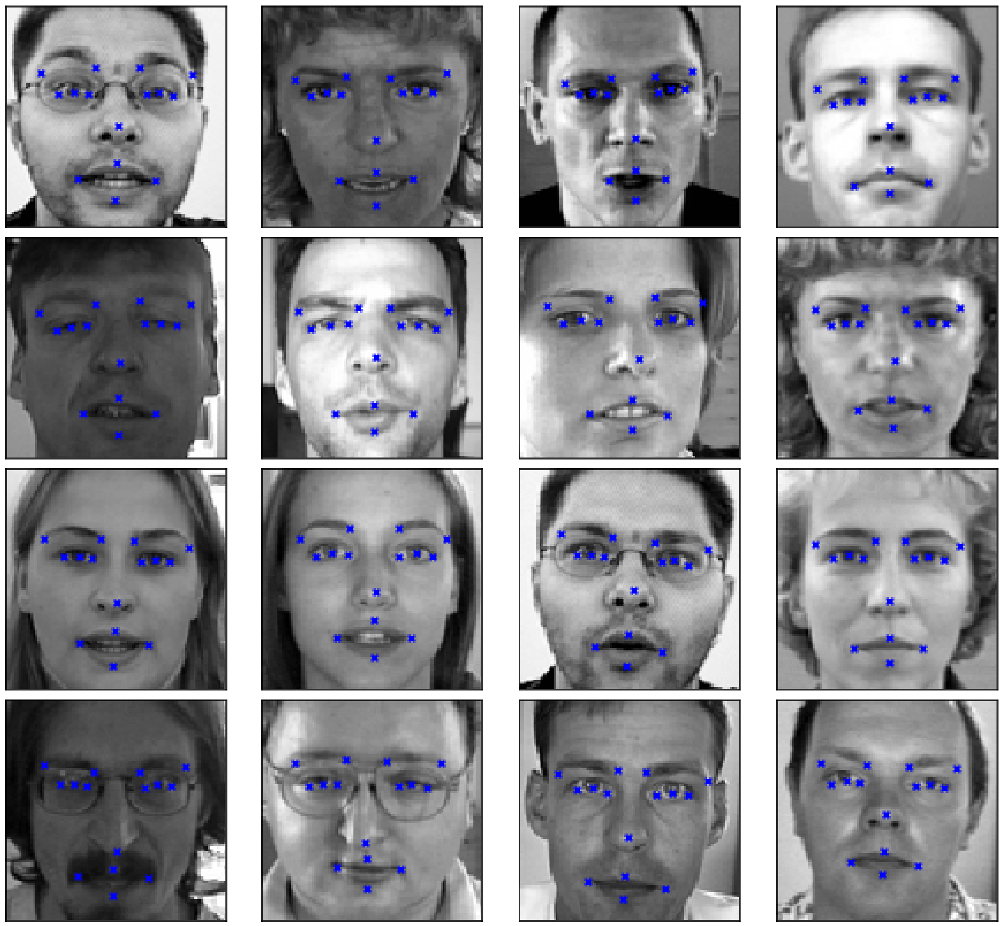
In [ ]:

```
 def load2d(test=False,cols=None):
     re = load(test, cols)
     X = re[0].reshape(-1,96,96,1)
     y = re[1]
     return X, y

X, _ = load2d(test=True)
y_pred = modelh_12.predict(X)

fig = plt.figure(figsize=(8, 8))
fig.subplots_adjust(
    left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)

for i in range(16):
    ax = fig.add_subplot(4, 4, i + 1, xticks=[], yticks=[])
    plot_sample(X[i], y_pred[i], ax)
fig.subplots_adjust(top=0.90)
```

```
fig.suptitle('Best Model (CNN with Data Augmentation and Hyper Parameter T
plt.show()
```

```
ImageId    1783
Image      1783
dtype: int64
```

Best Model (CNN with Data Augmentation and Hyper Parameter Tuning)



Appendix

This section includes approaches that we tried but didn't get major RMSE changes.

Other Data Augmentation

Shift images

In [ ]:

```
class ShiftFlipPic(FlipPic):
    def __init__(self,flip_indices=None,prop=0.1):
        super(ShiftFlipPic,self).__init__(flip_indices)
        self.prop = prop

    def fit(self,X,y):
        X, y = super(ShiftFlipPic,self).fit(X,y)
        X, y = self.shift_image(X,y,prop=self.prop)
        return(X,y)
```

```python
def random_shift(self,shift_range,n=96):
    '''
    :param shift_range:
    The maximum number of columns/rows to shift
    :return:
    keep(0):   minimum row/column index to keep
    keep(1):   maximum row/column index to keep
    assign(0): minimum row/column index to assign
    assign(1): maximum row/column index to assign
    shift:     amount to shift the landmark

    assign(1) - assign(0) == keep(1) - keep(0)
    '''
    shift = np.random.randint(-shift_range,
                              shift_range)
    def shift_left(n,shift):
        shift = np.abs(shift)
        return(0,n - shift)
    def shift_right(n,shift):
        shift = np.abs(shift)
        return(shift,n)

    if shift < 0:
        keep = shift_left(n,shift)
        assign = shift_right(n,shift)
    else:
        assign = shift_left(n,shift) ## less than 96
        keep = shift_right(n,shift)

    return((keep,  assign, shift))

def shift_single_image(self,x_,y_,prop=0.1):
    '''
    :param x_: a single picture array (96, 96, 1)
    :param y_: 15 landmark locations
               [0::2] contains x axis values
               [1::2] contains y axis values
    :param prop: proportion of random horizontal and vertical shift
                 relative to the number of columns
                 e.g. prop = 0.1 then the picture is moved at least by
                 0.1*96 = 8 columns/rows
    :return:
    x_, y_
    '''
    w_shift_max = int(x_.shape[0] * prop)
    h_shift_max = int(x_.shape[1] * prop)

    w_keep,w_assign,w_shift = self.random_shift(w_shift_max)
    h_keep,h_assign,h_shift = self.random_shift(h_shift_max)

    x_[w_assign[0]:w_assign[1],
       h_assign[0]:h_assign[1],:] = x_[w_keep[0]:w_keep[1],
                                       h_keep[0]:h_keep[1],:]

    y_[0::2] = y_[0::2] - h_shift/float(x_.shape[0]/2.)
    y_[1::2] = y_[1::2] - w_shift/float(x_.shape[1]/2.)
    return(x_,y_)

def shift_image(self,X,y,prop=0.1):
        ## This function may be modified to be more efficient e.g. get
        for irow in range(X.shape[0]):
```

```
                x_ = X[irow]
                y_ = y[irow]
                X[irow],y[irow] = self.shift_single_image(x_,y_,prop=prop)
            return(X,y)
```

In [ ]:

```
from keras.preprocessing.image import ImageDataGenerator
generator = ImageDataGenerator()
shiftFlipPic = ShiftFlipPic(prop=0.1)

fig = plt.figure(figsize=(7,7))

count = 1
for batch in generator.flow(X_train_b[:2],y_train[:2]):
    X_batch, y_batch = shiftFlipPic.fit(*batch)

    ax = fig.add_subplot(3,3, count,xticks=[],yticks=[])
    plot_sample(X_batch[0],y_batch[0],ax)
    count += 1
    if count == 10:
        break
plt.show()
```

In [ ]:

```
print(X_train_b.shape)
```

(1712, 96, 96, 1)

In [ ]:

```
model5 = Sequential()
model5.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(96
model5.add(Conv2D(64, (3, 3), activation='relu'))
model5.add(MaxPooling2D(pool_size=(2, 2)))
model5.add(Dropout(0.2))
model5.add(Flatten())
```

```
model5.add(Dense(units=50, input_dim=128, activation='relu'))
model5.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)

model5.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

history = fit(model5, shiftFlipPic, train=(X_train_b,y_train),
           validation=(X_test_b,y_test),
           batch_size=32,epochs=100)

loss, mae, mse = model5.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Epoch    0:
loss - 5.35730, val_loss - 0.10955
Epoch   10:
loss - 0.00788, val_loss - 0.00545
Epoch   20:
loss - 0.00414, val_loss - 0.00342
Epoch   30:
loss - 0.00326, val_loss - 0.00248
Epoch   40:
loss - 0.00282, val_loss - 0.00215
Epoch   50:
loss - 0.00263, val_loss - 0.00210
Epoch   60:
loss - 0.00252, val_loss - 0.00221
Epoch   70:
loss - 0.00233, val_loss - 0.00212
Epoch   80:
loss - 0.00219, val_loss - 0.00222
Epoch   90:
loss - 0.00211, val_loss - 0.00186
Testing set RMSE:  2.25
```

Random Brightness

In [ ]:

```
model6 = Sequential()
model6.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(96
model6.add(Conv2D(64, (3, 3), activation='relu'))
model6.add(MaxPooling2D(pool_size=(2, 2)))
model6.add(Dropout(0.2))
model6.add(Flatten())
model6.add(Dense(units=50, input_dim=128, activation='relu'))
model6.add(Dense(30))

optimizer = optimizers.RMSprop(0.001)
#optimizer = 'adam'

model6.compile(loss='mse',
            optimizer=optimizer,
            metrics=['mae', 'mse'])

batch_size = 64
```

```
# brightness_range=[0.2,1]
datagen = ImageDataGenerator(brightness_range=[0.5,1.5])

history = model6.fit_generator(datagen.flow(X_train_b, y_train, batch_size
                        steps_per_epoch=X_train_b.shape[0] // batch_size,
                        epochs=50,verbose=0,
                        validation_data=(X_test_b, y_test))

#hist = pd.DataFrame(history.history)
#hist['epoch'] = history.epoch
#print(hist)

loss, mae, mse = model6.evaluate(X_test_b, y_test, verbose=2)

rmse = np.sqrt(mse) * 48

print("Testing set RMSE: {:5.2f}".format(rmse))
```

```
Testing set RMSE:  3.14
```

In [ ]:

```
!wget https://worksheets.codalab.org/rest/bundles/0x6b567e1cf2e041ec80d709
```

```
--2020-02-09 05:15:09--  https://worksheets.codalab.org/rest/bundles/0x6b5
67e1cf2e041ec80d7098f031c5c9e/contents/blob/
Resolving worksheets.codalab.org (worksheets.codalab.org)... 40.71.231.153
Connecting to worksheets.codalab.org (worksheets.codalab.org)|40.71.231.15
3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Syntax error in Set-Cookie: codalab_session=""; expires=Thu, 01 Jan 1970 0
0:00:00 GMT; Max-Age=-1; Path=/ at position 70.
Length: unspecified [text/x-python]
Saving to: ‘index.html’

index.html              [ <=>                ]  10.30K  --.-KB/s    in 0s

2020-02-09 05:15:09 (229 MB/s) - ‘index.html’ saved [10547]
```

In [ ]:

```
cat index.html
```

```
"""Official evaluation script for SQuAD version 2.0.

In addition to basic functionality, we also compute additional statistics
and
plot precision-recall curves if an additional na_prob.json file is provide
d.
This file is expected to map question ID's to the model's predicted probab
ility
that a question is unanswerable.
"""
import argparse
import collections
import json
import numpy as np
import os
import re
import string
import sys

OPTS = None
```

```
def parse_args():
  parser = argparse.ArgumentParser('Official evaluation script for SQuAD v
ersion 2.0.')
  parser.add_argument('data_file', metavar='data.json', help='Input data J
SON file.')
  parser.add_argument('pred_file', metavar='pred.json', help='Model predic
tions.')
  parser.add_argument('--out-file', '-o', metavar='eval.json',
                      help='Write accuracy metrics to file (default is std
out).')
  parser.add_argument('--na-prob-file', '-n', metavar='na_prob.json',
                      help='Model estimates of probability of no answer.')
  parser.add_argument('--na-prob-thresh', '-t', type=float, default=1.0,
                      help='Predict "" if no-answer probability exceeds th
is (default = 1.0).')
  parser.add_argument('--out-image-dir', '-p', metavar='out_images', defau
lt=None,
                      help='Save precision-recall curves to directory.')
  parser.add_argument('--verbose', '-v', action='store_true')
  if len(sys.argv) == 1:
    parser.print_help()
    sys.exit(1)
  return parser.parse_args()

def make_qid_to_has_ans(dataset):
  qid_to_has_ans = {}
  for article in dataset:
    for p in article['paragraphs']:
      for qa in p['qas']:
        qid_to_has_ans[qa['id']] = bool(qa['answers'])
  return qid_to_has_ans

def normalize_answer(s):
  """Lower text and remove punctuation, articles and extra whitespace."""
  def remove_articles(text):
    regex = re.compile(r'\b(a|an|the)\b', re.UNICODE)
    return re.sub(regex, ' ', text)
  def white_space_fix(text):
    return ' '.join(text.split())
  def remove_punc(text):
    exclude = set(string.punctuation)
    return ''.join(ch for ch in text if ch not in exclude)
  def lower(text):
    return text.lower()
  return white_space_fix(remove_articles(remove_punc(lower(s))))

def get_tokens(s):
  if not s: return []
  return normalize_answer(s).split()

def compute_exact(a_gold, a_pred):
  return int(normalize_answer(a_gold) == normalize_answer(a_pred))

def compute_f1(a_gold, a_pred):
  gold_toks = get_tokens(a_gold)
  pred_toks = get_tokens(a_pred)
  common = collections.Counter(gold_toks) & collections.Counter(pred_toks)
  num_same = sum(common.values())
  if len(gold_toks) == 0 or len(pred_toks) == 0:
    # If either is no-answer, then F1 is 1 if they agree, 0 otherwise
    return int(gold_toks == pred_toks)
  if num_same == 0:
    return 0
  precision = 1.0 * num_same / len(pred_toks)
  recall = 1.0 * num_same / len(gold_toks)
```

```
  f1 = (2 * precision * recall) / (precision + recall)
  return f1

def get_raw_scores(dataset, preds):
  exact_scores = {}
  f1_scores = {}
  for article in dataset:
    for p in article['paragraphs']:
      for qa in p['qas']:
        qid = qa['id']
        gold_answers = [a['text'] for a in qa['answers']
                        if normalize_answer(a['text'])]
        if not gold_answers:
          # For unanswerable questions, only correct answer is empty strin
g
          gold_answers = ['']
        if qid not in preds:
          print('Missing prediction for %s' % qid)
          continue
        a_pred = preds[qid]
        # Take max over all gold answers
        exact_scores[qid] = max(compute_exact(a, a_pred) for a in gold_ans
wers)
        f1_scores[qid] = max(compute_f1(a, a_pred) for a in gold_answers)
  return exact_scores, f1_scores

def apply_no_ans_threshold(scores, na_probs, qid_to_has_ans, na_prob_thres
h):
  new_scores = {}
  for qid, s in scores.items():
    pred_na = na_probs[qid] > na_prob_thresh
    if pred_na:
      new_scores[qid] = float(not qid_to_has_ans[qid])
    else:
      new_scores[qid] = s
  return new_scores

def make_eval_dict(exact_scores, f1_scores, qid_list=None):
  if not qid_list:
    total = len(exact_scores)
    return collections.OrderedDict([
        ('exact', 100.0 * sum(exact_scores.values()) / total),
        ('f1', 100.0 * sum(f1_scores.values()) / total),
        ('total', total),
    ])
  else:
    total = len(qid_list)
    return collections.OrderedDict([
        ('exact', 100.0 * sum(exact_scores[k] for k in qid_list) / total),
        ('f1', 100.0 * sum(f1_scores[k] for k in qid_list) / total),
        ('total', total),
    ])

def merge_eval(main_eval, new_eval, prefix):
  for k in new_eval:
    main_eval['%s_%s' % (prefix, k)] = new_eval[k]

def plot_pr_curve(precisions, recalls, out_image, title):
  plt.step(recalls, precisions, color='b', alpha=0.2, where='post')
  plt.fill_between(recalls, precisions, step='post', alpha=0.2, color='b')
  plt.xlabel('Recall')
  plt.ylabel('Precision')
  plt.xlim([0.0, 1.05])
  plt.ylim([0.0, 1.05])
  plt.title(title)
```

```
  plt.savefig(out_image)
  plt.clf()

def make_precision_recall_eval(scores, na_probs, num_true_pos, qid_to_has_
ans,
                               out_image=None, title=None):
  qid_list = sorted(na_probs, key=lambda k: na_probs[k])
  true_pos = 0.0
  cur_p = 1.0
  cur_r = 0.0
  precisions = [1.0]
  recalls = [0.0]
  avg_prec = 0.0
  for i, qid in enumerate(qid_list):
    if qid_to_has_ans[qid]:
      true_pos += scores[qid]
    cur_p = true_pos / float(i+1)
    cur_r = true_pos / float(num_true_pos)
    if i == len(qid_list) - 1 or na_probs[qid] != na_probs[qid_list[i+1]]:
      # i.e., if we can put a threshold after this point
      avg_prec += cur_p * (cur_r - recalls[-1])
      precisions.append(cur_p)
      recalls.append(cur_r)
  if out_image:
    plot_pr_curve(precisions, recalls, out_image, title)
  return {'ap': 100.0 * avg_prec}

def run_precision_recall_analysis(main_eval, exact_raw, f1_raw, na_probs,
                                  qid_to_has_ans, out_image_dir):
  if out_image_dir and not os.path.exists(out_image_dir):
    os.makedirs(out_image_dir)
  num_true_pos = sum(1 for v in qid_to_has_ans.values() if v)
  if num_true_pos == 0:
    return
  pr_exact = make_precision_recall_eval(
      exact_raw, na_probs, num_true_pos, qid_to_has_ans,
      out_image=os.path.join(out_image_dir, 'pr_exact.png'),
      title='Precision-Recall curve for Exact Match score')
  pr_f1 = make_precision_recall_eval(
      f1_raw, na_probs, num_true_pos, qid_to_has_ans,
      out_image=os.path.join(out_image_dir, 'pr_f1.png'),
      title='Precision-Recall curve for F1 score')
  oracle_scores = {k: float(v) for k, v in qid_to_has_ans.items()}
  pr_oracle = make_precision_recall_eval(
      oracle_scores, na_probs, num_true_pos, qid_to_has_ans,
      out_image=os.path.join(out_image_dir, 'pr_oracle.png'),
      title='Oracle Precision-Recall curve (binary task of HasAns vs. NoAn
s)')
  merge_eval(main_eval, pr_exact, 'pr_exact')
  merge_eval(main_eval, pr_f1, 'pr_f1')
  merge_eval(main_eval, pr_oracle, 'pr_oracle')

def histogram_na_prob(na_probs, qid_list, image_dir, name):
  if not qid_list:
    return
  x = [na_probs[k] for k in qid_list]
  weights = np.ones_like(x) / float(len(x))
  plt.hist(x, weights=weights, bins=20, range=(0.0, 1.0))
  plt.xlabel('Model probability of no-answer')
  plt.ylabel('Proportion of dataset')
  plt.title('Histogram of no-answer probability: %s' % name)
  plt.savefig(os.path.join(image_dir, 'na_prob_hist_%s.png' % name))
  plt.clf()

def find_best_thresh(preds, scores, na_probs, qid_to_has_ans):
```

```
  num_no_ans = sum(1 for k in qid_to_has_ans if not qid_to_has_ans[k])
  cur_score = num_no_ans
  best_score = cur_score
  best_thresh = 0.0
  qid_list = sorted(na_probs, key=lambda k: na_probs[k])
  for i, qid in enumerate(qid_list):
    if qid not in scores: continue
    if qid_to_has_ans[qid]:
      diff = scores[qid]
    else:
      if preds[qid]:
        diff = -1
      else:
        diff = 0
    cur_score += diff
    if cur_score > best_score:
      best_score = cur_score
      best_thresh = na_probs[qid]
  return 100.0 * best_score / len(scores), best_thresh

def find_all_best_thresh(main_eval, preds, exact_raw, f1_raw, na_probs, qi
d_to_has_ans):
  best_exact, exact_thresh = find_best_thresh(preds, exact_raw, na_probs,
qid_to_has_ans)
  best_f1, f1_thresh = find_best_thresh(preds, f1_raw, na_probs, qid_to_ha
s_ans)
  main_eval['best_exact'] = best_exact
  main_eval['best_exact_thresh'] = exact_thresh
  main_eval['best_f1'] = best_f1
  main_eval['best_f1_thresh'] = f1_thresh

def main():
  with open(OPTS.data_file) as f:
    dataset_json = json.load(f)
    dataset = dataset_json['data']
  with open(OPTS.pred_file) as f:
    preds = json.load(f)
  if OPTS.na_prob_file:
    with open(OPTS.na_prob_file) as f:
      na_probs = json.load(f)
  else:
    na_probs = {k: 0.0 for k in preds}
  qid_to_has_ans = make_qid_to_has_ans(dataset)  # maps qid to True/False
  has_ans_qids = [k for k, v in qid_to_has_ans.items() if v]
  no_ans_qids = [k for k, v in qid_to_has_ans.items() if not v]
  exact_raw, f1_raw = get_raw_scores(dataset, preds)
  exact_thresh = apply_no_ans_threshold(exact_raw, na_probs, qid_to_has_an
s,
                                        OPTS.na_prob_thresh)
  f1_thresh = apply_no_ans_threshold(f1_raw, na_probs, qid_to_has_ans,
                                     OPTS.na_prob_thresh)
  out_eval = make_eval_dict(exact_thresh, f1_thresh)
  if has_ans_qids:
    has_ans_eval = make_eval_dict(exact_thresh, f1_thresh, qid_list=has_an
s_qids)
    merge_eval(out_eval, has_ans_eval, 'HasAns')
  if no_ans_qids:
    no_ans_eval = make_eval_dict(exact_thresh, f1_thresh, qid_list=no_ans_
qids)
    merge_eval(out_eval, no_ans_eval, 'NoAns')
  if OPTS.na_prob_file:
    find_all_best_thresh(out_eval, preds, exact_raw, f1_raw, na_probs, qid
_to_has_ans)
  if OPTS.na_prob_file and OPTS.out_image_dir:
    run_precision_recall_analysis(out_eval, exact_raw, f1_raw, na_probs,
```

```
                                  qid_to_has_ans, OPTS.out_image_dir)
    histogram_na_prob(na_probs, has_ans_qids, OPTS.out_image_dir, 'hasAn
s')
    histogram_na_prob(na_probs, no_ans_qids, OPTS.out_image_dir, 'noAns')
  if OPTS.out_file:
    with open(OPTS.out_file, 'w') as f:
      json.dump(out_eval, f)
  else:
    print(json.dumps(out_eval, indent=2))

if __name__ == '__main__':
  OPTS = parse_args()
  if OPTS.out_image_dir:
    import matplotlib
    matplotlib.use('Agg')
    import matplotlib.pyplot as plt
  main()
```