Futurice Academy 2021 - Homework Assignment

This is my solution to the Homework assignment belongs to Futurice Academy application on September 2021.

Learning track: Data Specialist in cloud native environment

Homework: Machine Learning - Use GitHub's REST API to extract information of a repo (issues, stars, watchers, etc.) and predict the amount of contribution (number of contributors) of the project. The model will be trained on approximately 5,800 datapoints.

List of content:

1. Retrieve Data (approx. 7,100 datapoints)
2. Clean Data (approx. 5,800 datapoints left)
3. Visualize Data
4. Preprocess Data
5. Build Model
6. Train/Validate Model
7. Test Model
8. Conclusion

Additional comments: This project could be useful in practice where a user wants their project to achieve a certain amount of attentions, the machine would tell the user approximately how many co-creators needed for a project of that scale. Due to the limited amount of times GitHub API allow me to fetch information, the model in this homework will focus mainly on the aforementioned problem, but given more resources, the model could be scaled to solve much bigger problem such as suggesting collaborators for the user's project.

Last modified: Oct 1st, 2021

In [1]:
```
# Useful libraries
import math
import requests
import pprint as pp
import numpy as np
from scipy.stats import pearsonr
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras import regularizers
from tensorflow.keras.layers import Concatenate, Input, Dropout, Dense, LayerNormalization, BatchNormalization,
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.initializers import Constant, RandomNormal, Zeros
from tensorflow.keras.models import load_model, Model
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping ,LearningRateScheduler
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.metrics import RootMeanSquaredError
tf.executing_eagerly()
```

Out[1]: True

1. Retrieve Data

Fetch data from the API and save them to **scraped_data** as type *dict*. After that, concatenate the final column indicating the number of contributors (labels for the model) to the dict and turn it to *Pandas DataFrame* **scraped_df**. Data in the dataframe will be in their original format and saved to CSV file name 'github_data.csv' in the same folder for later use.

In [2]:
```
# The model take the following information about a repo as the input
data = {'language': [],
        'size': [],
        'has_issues': [],
        'has_projects': [],
        'has_downloads': [],
        'has_wiki': [],
        'has_pages': [],
        'archived': [],
        'disabled': [],
        'open_issues_count': [],
        'allow_forking': [],
        'open_issues': [],
        'network_count': [],
        'subscribers_count': [],
        'stargazers_count': [],
        'watchers_count': [],
        'forks_count': []}

def store_data(nof_repo, since_repo):
    # STEP 1: Extracting information of 5,000 repos from API
    # Query string values
    since = since_repo
    # Count the number of repos extracted so far
    cnt_repo = 0
    miss_repo = 0
    # Contributor counts list
    contributors = []
    # Each iteration extracts 100 repos
    while cnt_repo < nof_repo:
        # Get the response (100 repos) from the API
        token = 'token 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'
        repos = requests.get(f'https://api.github.com/repositories?since={since}',
                             headers={'Authorization': token}
                            ).json()
        for repo in repos:
            try:
                # Single repo
                repo_info = requests.get(repo['url'],
                                         headers={'Authorization': token}
                                        ).json()
                # Check if repo access fail before adding to actual data variable
                temp_repo = {}
                for key in data.keys():
                    temp_repo[key] = repo_info[key]
                # Retrieve labels
                cnt_ctrbt = len(requests.get(repo['contributors_url'],
                                             headers={'Authorization': token}
                                            ).json())
                contributors.append(float(cnt_ctrbt))
                # Retrieve features
                for key in data.keys():
                    data[key].append(temp_repo[key])
                # Update on how many repos have been added so far
                print(f"Repo with ID {repo['id']} added.", end='\r')
            except Exception as e:
                miss_repo += 1
            finally:
                # Accumulate cnt_repo
                cnt_repo += 1
                since += 1
    print(f'\nAdded {len(contributors)} repos.')
    print(f'Access to {miss_repo} repos were blocked.')
    # Add labels column to the scraped data
    data['contributors_count'] = contributors

    # STEP 2: Store data in CSV file for later use, this 'dframe' is not global
    dframe = pd.DataFrame(data)
    return dframe
```

Note

The following cell is used to scrape data from the API and merge it to the file 'github_df.csv'. Current file already contains approx. 7,000 datapoints (uncleaned), hence, only run the following cell if more data is needed.

In [ ]:
```
# Fetch data
fetched_df = store_data(100, 300000)

# Merge data to specific file
def merge_data(file_name, new_df):
    dframe = pd.read_csv(file_name, index_col=0)
    dframe = dframe.append(new_df, ignore_index=True)
    dframe.to_csv(file_name, index=False)
    return dframe

# Merge data
dframe = merge_data('github_df.csv', fetched_df)
```

Note

The following cell is used to read stored datapoints from 'github_df.csv' file. Always run it to get the data.

In [3]:
```
# Run this if no new data needs to be merged
dframe = pd.read_csv('github_df.csv')
```

2. Clean Data

This part choose important features for training and cleans out all data points that are associated with a **0 labels** (unfound contributor) and data points with **negative** feature values (errors). Due to the scale of this project and the API rate limitation, the chosen features are only basic information about a repo.

In [4]:
```
# Choose important features and have a look at the retrieved dataframe
important_keys = [
    'language',
    #'has_issues', 'has_projects', 'has_downloads', 'has_wiki', 'has_pages', 'archived', 'dsiabled',
    'size', 'open_issues', 'subscribers_count', 'stargazers_count', 'network_count', 'forks_count', 'contributo
]

# Filtered out unimportant features and have a look at 'dframe'
dframe = dframe[important_keys]
dframe
```

Out[4]:

	language	size	open_issues	subscribers_count	stargazers_count	network_count	forks_count	contributors_count
0	JavaScript	192.0	0.0	3.0	3.0	1.0	1.0	1.0
1	C++	2777.0	0.0	2.0	2.0	9.0	0.0	1.0
2	Java	1913.0	0.0	3.0	3.0	29.0	0.0	2.0
3	Ruby	96.0	0.0	3.0	2.0	0.0	0.0	1.0
4	Ruby	99.0	4.0	4.0	64.0	23.0	23.0	1.0
...	...	...	...	...	...	...	...	...
7178	Shell	256.0	0.0	2.0	0.0	0.0	0.0	1.0
7179	Objective-C	532.0	0.0	2.0	3.0	0.0	0.0	1.0
7180	Ruby	47.0	0.0	1.0	0.0	0.0	0.0	1.0
7181	Ruby	1360.0	0.0	1.0	1.0	0.0	0.0	1.0
7182	PHP	1355.0	1.0	3.0	3.0	0.0	1.0	1.0

7183 rows × 8 columns

In [5]:
```
# Filter based on the aforementioned conditions
filtered_df = dframe[dframe['contributors_count'] > 0]
for key in important_keys[1:-1]:
    filtered_df = filtered_df[filtered_df[key] >= 0]

# Reset index of the filtered dataframe
filtered_df = filtered_df.reset_index(drop=True)
```

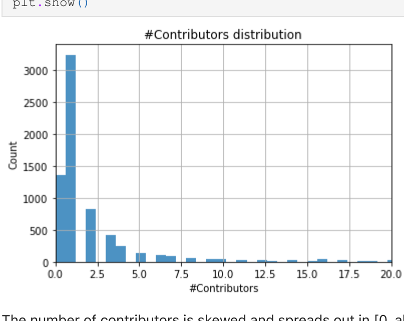
In [6]:
```
print(f'Filtered out: {len(dframe) - len(filtered_df)}/{len(dframe)} rows.')
```

```
Filtered out: 1365/7183 rows.
```

3. Visualize Data

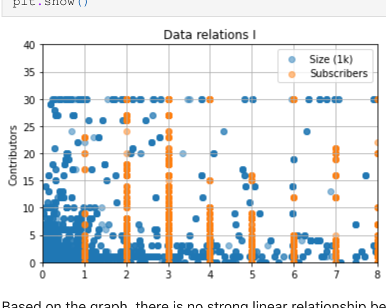
Have some visualization over the data in order to choose how to **preporcess** the data and which **training method** to use on it.

In [7]:
```
plt.hist(dframe['contributors_count'], bins=50, alpha=0.8)
plt.title('#Contributors distribution')
plt.xlabel('#Contributors')
plt.ylabel('Count')
plt.grid(True)
plt.xlim([0, 20])
#plt.ylim([0, 40])
plt.show()
```



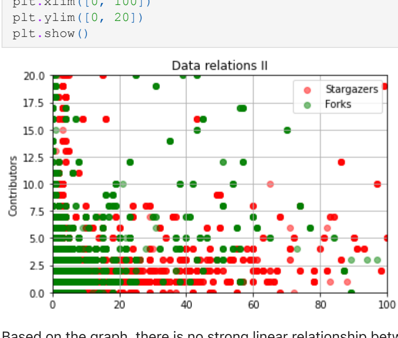
The number of contributors is skewed and spreads out in [0, above 20]. The other features in the dataset such as 'network_count', 'forks_count', etc. also have the similar distribution, but to not make the notebook unecessarily long, the graphs for them are omitted.

In [8]:
```
plt.scatter(dframe['size']/1000, dframe['contributors_count'], alpha=0.5, label='Size (1k)')
plt.scatter(dframe['subscribers_count'], dframe['contributors_count'], alpha=0.5, label='Subscribers')
plt.title('Data relations I')
plt.legend(loc='upper right')
plt.ylabel('Contributors')
plt.grid(True)
plt.xlim([0, 8])
plt.ylim([0, 40])
plt.show()
```



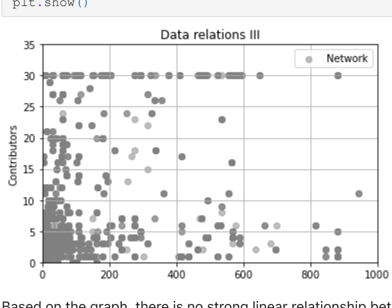
Based on the graph, there is no strong linear relationship between 'size', 'subscribers' and the label 'contributors_count'.

In [9]:
```
plt.scatter(dframe['stargazers_count'], dframe['contributors_count'], alpha=0.5, color='red', label='Stargazers
plt.scatter(dframe['forks_count'], dframe['contributors_count'], alpha=0.5, color='green', label='Forks')
plt.title('Data relations II')
plt.legend(loc='upper right')
plt.ylabel('Contributors')
plt.grid(True)
plt.xlim([0, 100])
plt.ylim([0, 20])
plt.show()
```



Based on the graph, there is no strong linear relationship between 'stargazers', 'forks' and the label 'contributors_count'.

In [10]:
```
plt.scatter(dframe['network_count'], dframe['contributors_count'], alpha=0.5, color='gray', label='Network')
plt.title('Data relations III')
plt.legend(loc='upper right')
plt.ylabel('Contributors')
plt.grid(True)
plt.xlim([0, 1000])
plt.ylim([0, 35])
plt.show()
```



Based on the graph, there is no strong linear relationship between 'network_count' and the label 'contributors_count'.

Conclusion:

Data has skewed distribution and falls into many different ranges and also does not have strong linear relationships.

Hence:

- I will use logarithm function (log base e) on all of the features to normalize features distribution to a certain extent.
- After being applied to log function, the data will be normalized to shrink them to the shame range so that all features have equal impacts on the prediction. Without normalization, the 'size' feature with the largest value range ([0, above 200000]) will outweigh other features' effects.
- The model would use neural networks for learning to learn more complex (non-linear) relationship, since the data does not have strong linear relationship.

4. Preprocess Data

- Use one-hot encoding for programming languages of the repo
- Other original numerical values is processed as previously stated
- Shuffle data and split it to train/test sets

In [11]:
```
# This is according to the GitHub docs version when this homework is made.
# There might be inconsistency later, but that won't affect much.
supported_languages = ['c', 'c++', 'c#', 'go', 'java', 'javascript', 'php', 'python', 'ruby', 'scala', 'typescr
nof_langs = len(supported_languages)
print(f'There are {nof_langs} supported languages.')

# Simple one-hot encode. The 2nd last 1 indicates unsupported language; the last 1 indicates no language at al
def encode_language(language):
    result = np.zeros(nof_langs + 2)
    if isinstance(language, str):
        detected = False
        for i in range(nof_langs):
            if language.lower() == supported_languages[i]:
                result[i] = 1
                detected = True
        if not detected:
            result[nof_langs] = 1
    else:
        result[nof_langs + 1] = 1
    return result
```

```
There are 11 supported languages.
```

In [57]:
```
# Encode language data and other features as stated
def encode_data(inp_df, scale_coeff):
    # Make a copy so the original df is not mutated
    df = inp_df

    # Full data length (including both train and test sets)
    data_len = len(df)

    # Separate inputs
    languages = np.zeros((data_len, nof_langs + 2))
    other_fts = np.zeros((data_len, len(important_keys) - 2)) # (minus the language and contributors column)

    # Labels
    labels = np.zeros(data_len)
    initial_labels = np.zeros(data_len)

    # Encode and store data for learning to numpy arrays
    for i in range(data_len):
        # Encode language and store its values
        languages[i] = encode_language(df['language'][i])
        # Numerize other features and store their values
        other_fts_el = np.zeros(len(important_keys) - 2)
        for key in enumerate(important_keys[1:-1]):
            other_fts_el[key[0]] = math.log(max(df[key[1]][i], 1))/scale_coeff
        other_fts[i] = other_fts_el
        # Adding labels
        labels[i] = math.log(df['contributors_count'][i])/scale_coeff
        initial_labels[i] = df['contributors_count'][i]

    return languages, other_fts, labels, initial_labels
```

In [58]:
```
# Split data into Train/Test set. Validation set is also included in Train set.
def split_data(test_proportion, input_df, scale_coeff):
    # Encode
    languages, other_fts, labels, initial_labels = encode_data(input_df, scale_coeff)
    # sklearn
    train_lg, test_lg, train_ft, test_ft, train_lb, test_lb, train_ilabels, test_ilabels = train_test_split(
        languages, other_fts, labels, initial_labels, test_size=test_proportion, random_state=88
    )

    # Train data
    train_set = (train_lg, train_ft, train_lb, train_ilabels)

    # Test data
    test_set = (test_lg, test_ft, test_lb, test_ilabels)

    return train_set, test_set
```

5. Build Model

Based on the previous discussion about data visualization and preprocessing, the model will have the following architect.

Model Architect

In [14]:
```
def create_model(language_dim, other_dim, label_dim, learning_rate):
    # Input languages and other features separately
    language_input = Input(shape=(language_dim,))
    other_input = Input(shape=(other_dim,))

    # Feed language in to get language score
    language_score = Dense(1, activation='linear')(language_input)

    # Concatenate the score with other inputs
    combined = Concatenate(axis=-1)([language_score, other_input])


    # Some more preprocess before enter main neural network
    combined = BatchNormalization()(combined)
    combined = Dropout(0.3)(combined)

    # Main neural network
    dense = Dense(15, activation='linear')(combined)
    dense = Dropout(0.3)(dense)
    dense = Dense(15, activation='linear')(dense)
    dense = Dropout(0.3)(dense)

    # Output - predicted number of contributors
    pred = Dense(1, activation='linear')(dense)

    # Model
    model = Model(inputs = [language_input, other_input], outputs = pred)
    model.compile(optimizer = Adam(learning_rate=learning_rate), loss = [MeanSquaredError()], metrics = [RootMe

    return model
```

6. Train/Validate Model

I will use 5-fold Cross Validation in this problem. During training, there are callbacks Early Stoping and Model Checkpoint to optimize training process and save the best model based on cross validation.

In [15]:
```
# Function to seed everything
def seed_everything(seed):
    np.random.seed(seed)
    tf.random.set_seed(seed)
```

In [16]:
```
# Get model name for each fold in k-fold cv
def get_model_name(k):
    return 'model_' + str(k) + '.h5'
```

In [19]:
```
def train_model(input_df, scale_coeff, test_size, language_dim, other_dim, label_dim, learning_rate, epochs, ba
    # Randomize
    seed_everything(seed)

    # Get datasets
    train_set, test_set = split_data(test_size, input_df, scale_coeff)
    train_len = len(train_set[0])
    test_len = len(test_set[0])
    print(f'Train set length: {train_len}\nTest set length: {test_len}')

    # Create model
    model = create_model(language_dim, other_dim, label_dim, learning_rate)
    model.summary()

    # k-fold cv
    kfold = KFold(n_splits = folds, shuffle = True, random_state = seed)
    out_of_fold_pred = np.zeros(train_len)
    for fold, (train_ind, val_ind) in enumerate(kfold.split(train_set[0])):

        # Verbosity
        print('\nFold', fold+1, '*'*50)

        # Callbacks
        checkpoint = ModelCheckpoint(cp_path + get_model_name(fold), monitor='val_root_mean_squared_error', ver

        reduce_lr = ReduceLROnPlateau(monitor="val_root_mean_squared_error", factor=0.5, patience=5, min_lr=1e-

        early_stopping = EarlyStopping(monitor="val_root_mean_squared_error", min_delta=0, patience=5, verbose=


        # Training
        model.fit(x = [train_set[0][train_ind], train_set[1][train_ind]],
                  y = train_set[2][train_ind],
                  batch_size = batch_size,
                  epochs = epochs,
                  validation_data = ([train_set[0][val_ind], train_set[1][val_ind]], train_set[2][val_ind]),
                  callbacks = [checkpoint, reduce_lr, early_stopping])

        model.load_weights(cp_path + get_model_name(fold))
        cv_pred = model.predict([train_set[0][val_ind], train_set[1][val_ind]])
        out_of_fold_pred[val_ind] = cv_pred.reshape(-1)

    cv_rmse = np.sqrt(mean_squared_error(train_set[2], out_of_fold_pred))
    print('\nOut-of-fold RMSE is:', cv_rmse)

    # Predict on test data
    test_pred = model.predict([test_set[0], test_set[1]])
    return test_pred, test_set[2], test_set[3]
```

Note

The following cell determines some crucial parameters and start **training / validating** the model. At the end of the output text, there are result for out-of-fold root mean squared error.

In [20]:
```
# Declare all training info for the model
SCALE_COEFF = 20
TEST_SIZE = 0.3
LANGUAGE_DIM = nof_langs + 2
OTHER_DIM = len(important_keys) - 2
LABEL_DIM = 1
LEARNING_RATE = 1e-5
EPOCHS = 50
BATCH_SIZE = 2
FOLDS = 5
SEED = 1024
CP_PATH = ''

# Start training and validating the model
test_pred, test_observed, test_observed_initial = train_model(input_df      = filtered_df,
                                                              scale_coeff   = SCALE_COEFF,
                                                              test_size     = TEST_SIZE,
                                                              language_dim  = LANGUAGE_DIM,
                                                              other_dim     = OTHER_DIM,
                                                              label_dim     = LABEL_DIM,
                                                              learning_rate = LEARNING_RATE,
                                                              epochs        = EPOCHS,
                                                              batch_size    = BATCH_SIZE,
                                                              folds         = FOLDS,
                                                              seed          = SEED,
                                                              cp_path       = CP_PATH)
```

```
Train set length: 4072
Test set length: 1746
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 13)]         0
__________________________________________________________________________________________________
dense (Dense)                   (None, 1)            14          input_1[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 6)]          0
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 7)            0           dense[0][0]
                                                                 input_2[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 7)            28          concatenate[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 7)            0           batch_normalization[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 15)           120         dropout[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 15)           0           dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 15)           240         dropout_1[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 15)           0           dense_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 1)            16          dropout_2[0][0]
==================================================================================================
Total params: 418
Trainable params: 404
Non-trainable params: 14
__________________________________________________________________________________________________

Fold 1 **************************************************
Epoch 1/50
1629/1629 [==============================] - 3s 1ms/step - loss: 0.9346 - root_mean_squared_error: 0.9668 - val
_loss: 0.6546 - val_root_mean_squared_error: 0.8090

Epoch 00001: val_root_mean_squared_error improved from inf to 0.80904, saving model to model_0.h5
Epoch 2/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.7764 - root_mean_squared_error: 0.8812 - val
_loss: 0.4947 - val_root_mean_squared_error: 0.7033

Epoch 00002: val_root_mean_squared_error improved from 0.80904 to 0.70334, saving model to model_0.h5
Epoch 3/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.7190 - root_mean_squared_error: 0.8480 - val
_loss: 0.3404 - val_root_mean_squared_error: 0.5835

Epoch 00003: val_root_mean_squared_error improved from 0.70334 to 0.58346, saving model to model_0.h5
Epoch 4/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.6162 - root_mean_squared_error: 0.7850 - val
_loss: 0.2667 - val_root_mean_squared_error: 0.5164

Epoch 00004: val_root_mean_squared_error improved from 0.58346 to 0.51641, saving model to model_0.h5
Epoch 5/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.5049 - root_mean_squared_error: 0.7106 - val
_loss: 0.2094 - val_root_mean_squared_error: 0.4576

Epoch 00005: val_root_mean_squared_error improved from 0.51641 to 0.45757, saving model to model_0.h5
Epoch 6/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.4800 - root_mean_squared_error: 0.6928 - val
_loss: 0.1484 - val_root_mean_squared_error: 0.3853

Epoch 00006: val_root_mean_squared_error improved from 0.45757 to 0.38529, saving model to model_0.h5
Epoch 7/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.4207 - root_mean_squared_error: 0.6486 - val
_loss: 0.1230 - val_root_mean_squared_error: 0.3507- loss: 0.4413 - root_mea

Epoch 00007: val_root_mean_squared_error improved from 0.38529 to 0.35070, saving model to model_0.h5
Epoch 8/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.3986 - root_mean_squared_error: 0.6313 - val
_loss: 0.1000 - val_root_mean_squared_error: 0.3163

Epoch 00008: val_root_mean_squared_error improved from 0.35070 to 0.31627, saving model to model_0.h5
Epoch 9/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.3503 - root_mean_squared_error: 0.5918 - val
_loss: 0.0788 - val_root_mean_squared_error: 0.2808

Epoch 00009: val_root_mean_squared_error improved from 0.31627 to 0.28075, saving model to model_0.h5
Epoch 10/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.2931 - root_mean_squared_error: 0.5414 - val
_loss: 0.0653 - val_root_mean_squared_error: 0.2556

Epoch 00010: val_root_mean_squared_error improved from 0.28075 to 0.25557, saving model to model_0.h5
Epoch 11/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.2914 - root_mean_squared_error: 0.5398 - val
_loss: 0.0521 - val_root_mean_squared_error: 0.2283

Epoch 00011: val_root_mean_squared_error improved from 0.25557 to 0.22829, saving model to model_0.h5
Epoch 12/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.2482 - root_mean_squared_error: 0.4982 - val
_loss: 0.0403 - val_root_mean_squared_error: 0.2008

Epoch 00012: val_root_mean_squared_error improved from 0.22829 to 0.20080, saving model to model_0.h5
Epoch 13/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.2354 - root_mean_squared_error: 0.4852 - val
_loss: 0.0331 - val_root_mean_squared_error: 0.1818

Epoch 00013: val_root_mean_squared_error improved from 0.20080 to 0.18182, saving model to model_0.h5
Epoch 14/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.2248 - root_mean_squared_error: 0.4742 - val
_loss: 0.0281 - val_root_mean_squared_error: 0.1675

Epoch 00014: val_root_mean_squared_error improved from 0.18182 to 0.16752, saving model to model_0.h5
Epoch 15/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.2033 - root_mean_squared_error: 0.4509 - val
_loss: 0.0214 - val_root_mean_squared_error: 0.1463

Epoch 00015: val_root_mean_squared_error improved from 0.16752 to 0.14628, saving model to model_0.h5
Epoch 16/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.1848 - root_mean_squared_error: 0.4299 - val
_loss: 0.0164 - val_root_mean_squared_error: 0.1280

Epoch 00016: val_root_mean_squared_error improved from 0.14628 to 0.12805, saving model to model_0.h5
Epoch 17/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.1638 - root_mean_squared_error: 0.4047 - val
_loss: 0.0147 - val_root_mean_squared_error: 0.1211

Epoch 00017: val_root_mean_squared_error improved from 0.12805 to 0.12114, saving model to model_0.h5
Epoch 18/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.1540 - root_mean_squared_error: 0.3924 - val
_loss: 0.0121 - val_root_mean_squared_error: 0.1099

Epoch 00018: val_root_mean_squared_error improved from 0.12114 to 0.10987, saving model to model_0.h5
Epoch 19/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.1461 - root_mean_squared_error: 0.3822 - val
_loss: 0.0105 - val_root_mean_squared_error: 0.1025

Epoch 00019: val_root_mean_squared_error improved from 0.10987 to 0.10254, saving model to model_0.h5
Epoch 20/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.1249 - root_mean_squared_error: 0.3535 - val
_loss: 0.0083 - val_root_mean_squared_error: 0.0909

Epoch 00020: val_root_mean_squared_error improved from 0.10254 to 0.09094, saving model to model_0.h5
Epoch 21/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.1184 - root_mean_squared_error: 0.3441 - val
_loss: 0.0077 - val_root_mean_squared_error: 0.0877

Epoch 00021: val_root_mean_squared_error improved from 0.09094 to 0.08770, saving model to model_0.h5
Epoch 22/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.1065 - root_mean_squared_error: 0.3264 - val
_loss: 0.0059 - val_root_mean_squared_error: 0.0770

Epoch 00022: val_root_mean_squared_error improved from 0.08770 to 0.07695, saving model to model_0.h5
Epoch 23/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0993 - root_mean_squared_error: 0.3151 - val
_loss: 0.0049 - val_root_mean_squared_error: 0.0702

Epoch 00023: val_root_mean_squared_error improved from 0.07695 to 0.07021, saving model to model_0.h5
Epoch 24/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0954 - root_mean_squared_error: 0.3088 - val
_loss: 0.0044 - val_root_mean_squared_error: 0.0663

Epoch 00024: val_root_mean_squared_error improved from 0.07021 to 0.06628, saving model to model_0.h5
Epoch 25/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0874 - root_mean_squared_error: 0.2956 - val
_loss: 0.0038 - val_root_mean_squared_error: 0.0613

Epoch 00025: val_root_mean_squared_error improved from 0.06628 to 0.06134, saving model to model_0.h5
Epoch 26/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0821 - root_mean_squared_error: 0.2866 - val
_loss: 0.0034 - val_root_mean_squared_error: 0.0582

Epoch 00026: val_root_mean_squared_error improved from 0.06134 to 0.05824, saving model to model_0.h5
Epoch 27/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0792 - root_mean_squared_error: 0.2815 - val
_loss: 0.0030 - val_root_mean_squared_error: 0.0551

Epoch 00027: val_root_mean_squared_error improved from 0.05824 to 0.05508, saving model to model_0.h5
Epoch 28/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0694 - root_mean_squared_error: 0.2634 - val
_loss: 0.0030 - val_root_mean_squared_error: 0.0544

Epoch 00028: val_root_mean_squared_error improved from 0.05508 to 0.05442, saving model to model_0.h5
Epoch 29/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0653 - root_mean_squared_error: 0.2555 - val
_loss: 0.0025 - val_root_mean_squared_error: 0.0500

Epoch 00029: val_root_mean_squared_error improved from 0.05442 to 0.04999, saving model to model_0.h5
Epoch 30/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0598 - root_mean_squared_error: 0.2446 - val
_loss: 0.0021 - val_root_mean_squared_error: 0.0463

Epoch 00030: val_root_mean_squared_error improved from 0.04999 to 0.04628, saving model to model_0.h5
Epoch 31/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0566 - root_mean_squared_error: 0.2380 - val
_loss: 0.0021 - val_root_mean_squared_error: 0.0457

Epoch 00031: val_root_mean_squared_error improved from 0.04628 to 0.04566, saving model to model_0.h5
Epoch 32/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0477 - root_mean_squared_error: 0.2185 - val
_loss: 0.0020 - val_root_mean_squared_error: 0.0450

Epoch 00032: val_root_mean_squared_error improved from 0.04566 to 0.04497, saving model to model_0.h5
Epoch 33/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0505 - root_mean_squared_error: 0.2247 - val
_loss: 0.0020 - val_root_mean_squared_error: 0.0448

Epoch 00033: val_root_mean_squared_error improved from 0.04497 to 0.04483, saving model to model_0.h5
Epoch 34/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0478 - root_mean_squared_error: 0.2187 - val
_loss: 0.0018 - val_root_mean_squared_error: 0.0425

Epoch 00034: val_root_mean_squared_error improved from 0.04483 to 0.04251, saving model to model_0.h5
Epoch 35/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0424 - root_mean_squared_error: 0.2058 - val
_loss: 0.0018 - val_root_mean_squared_error: 0.0426

Epoch 00035: val_root_mean_squared_error did not improve from 0.04251
Epoch 36/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0404 - root_mean_squared_error: 0.2010 - val
_loss: 0.0018 - val_root_mean_squared_error: 0.0426

Epoch 00036: val_root_mean_squared_error did not improve from 0.04251
Epoch 37/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0364 - root_mean_squared_error: 0.1908 - val
_loss: 0.0017 - val_root_mean_squared_error: 0.0416

Epoch 00037: val_root_mean_squared_error improved from 0.04251 to 0.04162, saving model to model_0.h5
Epoch 38/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0343 - root_mean_squared_error: 0.1853 - val
_loss: 0.0017 - val_root_mean_squared_error: 0.0414

Epoch 00038: val_root_mean_squared_error improved from 0.04162 to 0.04142, saving model to model_0.h5
Epoch 39/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0326 - root_mean_squared_error: 0.1806 - val
_loss: 0.0017 - val_root_mean_squared_error: 0.0410

Epoch 00039: val_root_mean_squared_error improved from 0.04142 to 0.04095, saving model to model_0.h5
Epoch 40/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0311 - root_mean_squared_error: 0.1763 - val
_loss: 0.0016 - val_root_mean_squared_error: 0.0396

Epoch 00040: val_root_mean_squared_error improved from 0.04095 to 0.03956, saving model to model_0.h5
Epoch 41/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0279 - root_mean_squared_error: 0.1672 - val
_loss: 0.0017 - val_root_mean_squared_error: 0.0408

Epoch 00041: val_root_mean_squared_error did not improve from 0.03956
Epoch 42/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0257 - root_mean_squared_error: 0.1602 - val
_loss: 0.0016 - val_root_mean_squared_error: 0.0398

Epoch 00042: val_root_mean_squared_error did not improve from 0.03956
Epoch 43/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0260 - root_mean_squared_error: 0.1612 - val
_loss: 0.0016 - val_root_mean_squared_error: 0.0402

Epoch 00043: val_root_mean_squared_error did not improve from 0.03956
Epoch 44/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0215 - root_mean_squared_error: 0.1466 - val
_loss: 0.0016 - val_root_mean_squared_error: 0.0399

Epoch 00044: val_root_mean_squared_error did not improve from 0.03956
Epoch 45/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0218 - root_mean_squared_error: 0.1475 - val
_loss: 0.0016 - val_root_mean_squared_error: 0.0395

Epoch 00045: val_root_mean_squared_error improved from 0.03956 to 0.03947, saving model to model_0.h5
Epoch 46/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0201 - root_mean_squared_error: 0.1419 - val
_loss: 0.0015 - val_root_mean_squared_error: 0.0394

Epoch 00046: val_root_mean_squared_error improved from 0.03947 to 0.03935, saving model to model_0.h5
Epoch 47/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0198 - root_mean_squared_error: 0.1407 - val
_loss: 0.0016 - val_root_mean_squared_error: 0.0401

Epoch 00047: val_root_mean_squared_error did not improve from 0.03935
Epoch 48/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0188 - root_mean_squared_error: 0.1370 - val
_loss: 0.0016 - val_root_mean_squared_error: 0.0394

Epoch 00048: val_root_mean_squared_error did not improve from 0.03935
Epoch 49/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0180 - root_mean_squared_error: 0.1342 - val
_loss: 0.0015 - val_root_mean_squared_error: 0.0384

Epoch 00049: val_root_mean_squared_error improved from 0.03935 to 0.03844, saving model to model_0.h5
Epoch 50/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0180 - root_mean_squared_error: 0.1341 - val
_loss: 0.0015 - val_root_mean_squared_error: 0.0385

Epoch 00050: val_root_mean_squared_error did not improve from 0.03844

Fold 2 **************************************************
Epoch 1/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0182 - root_mean_squared_error: 0.1351 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0349

Epoch 00001: val_root_mean_squared_error improved from inf to 0.03488, saving model to model_1.h5
Epoch 2/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0182 - root_mean_squared_error: 0.1349 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0348

Epoch 00002: val_root_mean_squared_error improved from 0.03488 to 0.03483, saving model to model_1.h5
Epoch 3/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0168 - root_mean_squared_error: 0.1294 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0347

Epoch 00003: val_root_mean_squared_error improved from 0.03483 to 0.03469, saving model to model_1.h5
Epoch 4/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0168 - root_mean_squared_error: 0.1295 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0351

Epoch 00004: val_root_mean_squared_error did not improve from 0.03469
Epoch 5/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0160 - root_mean_squared_error: 0.1265 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0347

Epoch 00005: val_root_mean_squared_error improved from 0.03469 to 0.03465, saving model to model_1.h5
Epoch 6/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0150 - root_mean_squared_error: 0.1225 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0352

Epoch 00006: val_root_mean_squared_error did not improve from 0.03465
Epoch 7/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0155 - root_mean_squared_error: 0.1246 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0351

Epoch 00007: val_root_mean_squared_error did not improve from 0.03465
Epoch 8/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0140 - root_mean_squared_error: 0.1183 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0347

Epoch 00008: val_root_mean_squared_error did not improve from 0.03465
Epoch 9/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0143 - root_mean_squared_error: 0.1196 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0348

Epoch 00009: val_root_mean_squared_error did not improve from 0.03465
Epoch 10/50
1629/1629 [==============================] - 2s 1ms/step - loss: 0.0136 - root_mean_squared_error: 0.1168 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0353

Epoch 00010: val_root_mean_squared_error did not improve from 0.03465

Fold 3 **************************************************
Epoch 1/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0156 - root_mean_squared_error: 0.1251 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0347

Epoch 00001: val_root_mean_squared_error improved from inf to 0.03470, saving model to model_2.h5
Epoch 2/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0153 - root_mean_squared_error: 0.1239 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0343

Epoch 00002: val_root_mean_squared_error improved from 0.03470 to 0.03430, saving model to model_2.h5
Epoch 3/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0149 - root_mean_squared_error: 0.1220 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0342

Epoch 00003: val_root_mean_squared_error improved from 0.03430 to 0.03422, saving model to model_2.h5
Epoch 4/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0153 - root_mean_squared_error: 0.1238 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0338

Epoch 00004: val_root_mean_squared_error improved from 0.03422 to 0.03375, saving model to model_2.h5
Epoch 5/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0146 - root_mean_squared_error: 0.1210 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0339

Epoch 00005: val_root_mean_squared_error did not improve from 0.03375
Epoch 6/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0139 - root_mean_squared_error: 0.1180 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0337

Epoch 00006: val_root_mean_squared_error improved from 0.03375 to 0.03375, saving model to model_2.h5
Epoch 7/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0142 - root_mean_squared_error: 0.1192 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0335

Epoch 00007: val_root_mean_squared_error improved from 0.03375 to 0.03354, saving model to model_2.h5
Epoch 8/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0138 - root_mean_squared_error: 0.1174 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0336

Epoch 00008: val_root_mean_squared_error did not improve from 0.03354
Epoch 9/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0141 - root_mean_squared_error: 0.1187 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0340

Epoch 00009: val_root_mean_squared_error did not improve from 0.03354
Epoch 10/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0140 - root_mean_squared_error: 0.1183 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0340

Epoch 00010: val_root_mean_squared_error did not improve from 0.03354
Epoch 11/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0133 - root_mean_squared_error: 0.1154 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0337

Epoch 00011: val_root_mean_squared_error did not improve from 0.03354
Epoch 12/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0131 - root_mean_squared_error: 0.1145 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0341

Epoch 00012: val_root_mean_squared_error did not improve from 0.03354

Fold 4 **************************************************
Epoch 1/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0142 - root_mean_squared_error: 0.1193 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0337

Epoch 00001: val_root_mean_squared_error improved from inf to 0.03366, saving model to model_3.h5
Epoch 2/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0136 - root_mean_squared_error: 0.1167 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0338

Epoch 00002: val_root_mean_squared_error did not improve from 0.03366
Epoch 3/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0135 - root_mean_squared_error: 0.1163 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0335

Epoch 00003: val_root_mean_squared_error improved from 0.03366 to 0.03347, saving model to model_3.h5
Epoch 4/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0132 - root_mean_squared_error: 0.1150 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0338

Epoch 00004: val_root_mean_squared_error did not improve from 0.03347
Epoch 5/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0135 - root_mean_squared_error: 0.1163 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0342

Epoch 00005: val_root_mean_squared_error did not improve from 0.03347
Epoch 6/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0138 - root_mean_squared_error: 0.1174 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0338

Epoch 00006: val_root_mean_squared_error did not improve from 0.03347
Epoch 7/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0133 - root_mean_squared_error: 0.1151 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0334

Epoch 00007: val_root_mean_squared_error improved from 0.03347 to 0.03343, saving model to model_3.h5
Epoch 8/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0130 - root_mean_squared_error: 0.1141 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0338

Epoch 00008: val_root_mean_squared_error did not improve from 0.03343
Epoch 9/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0135 - root_mean_squared_error: 0.1161 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0338

Epoch 00009: val_root_mean_squared_error did not improve from 0.03343
Epoch 10/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0134 - root_mean_squared_error: 0.1156 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0336

Epoch 00010: val_root_mean_squared_error did not improve from 0.03343
Epoch 11/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0131 - root_mean_squared_error: 0.1146 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0336

Epoch 00011: val_root_mean_squared_error did not improve from 0.03343
Epoch 12/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0128 - root_mean_squared_error: 0.1132 - val
_loss: 0.0011 - val_root_mean_squared_error: 0.0335

Epoch 00012: val_root_mean_squared_error did not improve from 0.03343

Fold 5 **************************************************
Epoch 1/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0131 - root_mean_squared_error: 0.1144 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0347

Epoch 00001: val_root_mean_squared_error improved from inf to 0.03473, saving model to model_4.h5
Epoch 2/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0139 - root_mean_squared_error: 0.1178 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0343

Epoch 00002: val_root_mean_squared_error improved from 0.03473 to 0.03426, saving model to model_4.h5
Epoch 3/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0137 - root_mean_squared_error: 0.1169 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0346

Epoch 00003: val_root_mean_squared_error did not improve from 0.03426
Epoch 4/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0129 - root_mean_squared_error: 0.1134 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0345

Epoch 00004: val_root_mean_squared_error did not improve from 0.03426
Epoch 5/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0131 - root_mean_squared_error: 0.1146 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0352

Epoch 00005: val_root_mean_squared_error did not improve from 0.03426
Epoch 6/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0131 - root_mean_squared_error: 0.1142 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0346

Epoch 00006: val_root_mean_squared_error did not improve from 0.03426
Epoch 7/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0130 - root_mean_squared_error: 0.1140 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0343

Epoch 00007: val_root_mean_squared_error improved from 0.03426 to 0.03425, saving model to model_4.h5
Epoch 8/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0129 - root_mean_squared_error: 0.1138 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0346

Epoch 00008: val_root_mean_squared_error did not improve from 0.03425
Epoch 9/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0126 - root_mean_squared_error: 0.1121 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0344

Epoch 00009: val_root_mean_squared_error did not improve from 0.03425
Epoch 10/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0133 - root_mean_squared_error: 0.1155 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0347

Epoch 00010: val_root_mean_squared_error did not improve from 0.03425
Epoch 11/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0128 - root_mean_squared_error: 0.1131 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0347

Epoch 00011: val_root_mean_squared_error did not improve from 0.03425
Epoch 12/50
1629/1629 [==============================] - 3s 2ms/step - loss: 0.0127 - root_mean_squared_error: 0.1126 - val
_loss: 0.0012 - val_root_mean_squared_error: 0.0350

Epoch 00012: val_root_mean_squared_error did not improve from 0.03425

Out-of-fold RMSE is: 0.03491139867002192
```

7. Test Model

Below I calculated root mean squared error on the test set.

In [21]:
```python
# Extract all test results

# This is the raw output of the model
test_pred = np.array(test_pred.reshape(-1))

# This is the result processed back to the number of contributors
test_pred_initial = np.round(math.e ** (test_pred * SCALE_COEFF))
```

In [22]:
```python
rmse = np.sqrt(mean_squared_error(test_observed, test_pred))
print(f'Testing root mean squared error: {rmse}')
```

```
Testing root mean squared error: 0.03451733300038445
```

Note

Since the label has been preprocessed, the root mean square of raw outputs does not tell us much about how the model is performing in real case situations, hence, I try to insert some other ways to help us visualize the performance better.

In the follwing cells, I will only work with post-processed predictions (predictions that has been transformed back to the number of contributors) a.k.a. the 'test_pred_initial' array.

In [23]:
```python
# This function tell us the percentage of predictions the model get right within a specified range
def test_accuracy(allowed_error, prediction, actual, verbose=True):
    corrects_percentage = np.mean(np.absolute(prediction - actual) <= allowed_error)
    if verbose:
        print(f'Around {round(corrects_percentage * 100)}% of predicted results are within {allowed_error} unit
    return corrects_percentage
```

In [61]:
```python
# Some demonstration for the previous function
_ = test_accuracy(0, test_pred_initial, test_observed_initial)
_ = test_accuracy(1, test_pred_initial, test_observed_initial)
```

```
Around 43% of predicted results are within 0 units different from actual results.
Around 76% of predicted results are within 1 units different from actual results.
```

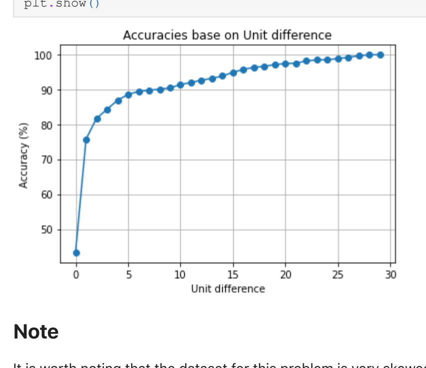
Conclude

So based on the previous demonstration, arounf 43% of the predictions are exactly the same as the actual observations; around 76% of the predictions are within 1 unit difference from the actual results. So given other features, the model can predict correctly the number of contributors for 76% of the cases, give or take 1 contributors.

Below is the graph plotting the relation between the **allowed difference between predictions and observations** and the **percentage of the predictions are within that range**.

In [25]:
```python
start = 0
stop = 30
step = 1
testing_range = np.arange(start, stop, step)
testing_accuracies = []
for e in testing_range:
    testing_accuracies.append(test_accuracy(e, test_pred_initial, test_observed_initial, verbose=False) * 100)

plt.plot(testing_range, testing_accuracies, marker='o')
plt.title('Accuracies base on Unit difference')
plt.grid(True)
plt.xlabel('Unit difference')
plt.ylabel('Accuracy (%)')
plt.show()
```



Note

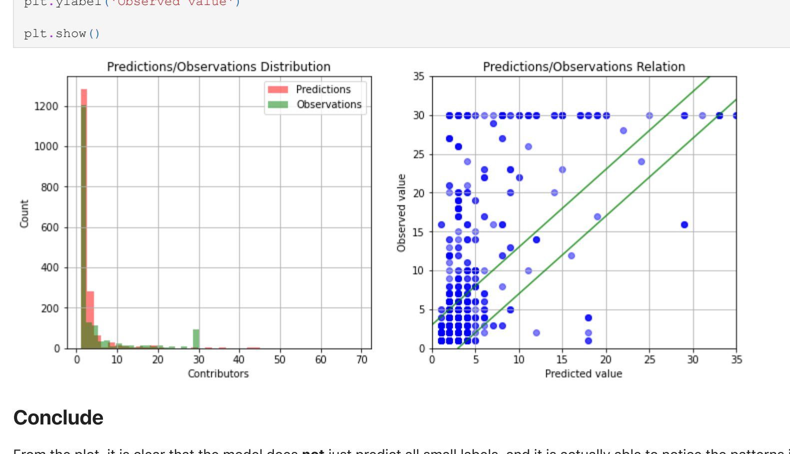
It is worth noting that the dataset for this problem is very skewed, the majority of datapoints has very low value label, the appearance of large label values are so insignificant that might cause the model to ignore the all the large label observations or in other words, the model does not make any high prediction at all.

Hence, below I plot the predictions distribution and the relation between predictions and actual results to check for that. Just a smaller note, the points in the area between the 2 green lines in plot 2 is all the predictions that are within 3 unit difference from actual results.

In [26]:
```python
plt.subplots(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.hist(test_pred_initial, alpha=0.5, bins=40, color='red', label='Predictions')
plt.hist(test_observed_initial, alpha=0.5, bins=20, color='green', label='Observations')
plt.title('Predictions/Observations Distribution')
plt.grid(True)
plt.legend(loc='upper right')
#plt.xlim([0, 20])
plt.xlabel('Contributors')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
plt.scatter(test_pred_initial, test_observed_initial, alpha=0.5, color='blue')
unit_diff = 3
plt.plot([0, 35-unit_diff], [unit_diff, 35], alpha=0.8, color='green')
plt.plot([unit_diff, 35], [0, 35-unit_diff], alpha=0.8, color='green')
plt.title('Predictions/Observations Relation')
plt.grid(True)
plt.xlim([0, 35])
plt.ylim([0, 35])
plt.xlabel('Predicted value')
plt.ylabel('Observed value')

plt.show()
```



Conclude

From the plot, it is clear that the model does **not** just predict all small labels, and it is actually able to notice the patterns in some of the the large labels (Yay!).

8. Conclusion

With only 5,800 datapoints and a shallow net structure, the model has been able to notice some patterns in large/small labels. Hence, to improve the model and solve this problem more thoroughly, here is my thoughts on what to do next:

- Fetch more data from the API, preferably somewhere around 1 million datapoints so the model can have around 1~10 thousand exceptional (large)-label data to learn from.
- Extend API rate limitation to fetch more features from the API beside the basic info that I used.
- Try out more complex functions than Logarithm to normalize the data distributions and values.
- Make the neural net deeper and larger so the model can recognize more complex pattern.
- Try more complex structure for the model, such as residual nets, or convolution nets, etc.
- Try pretrained model.

Thank you for reading.