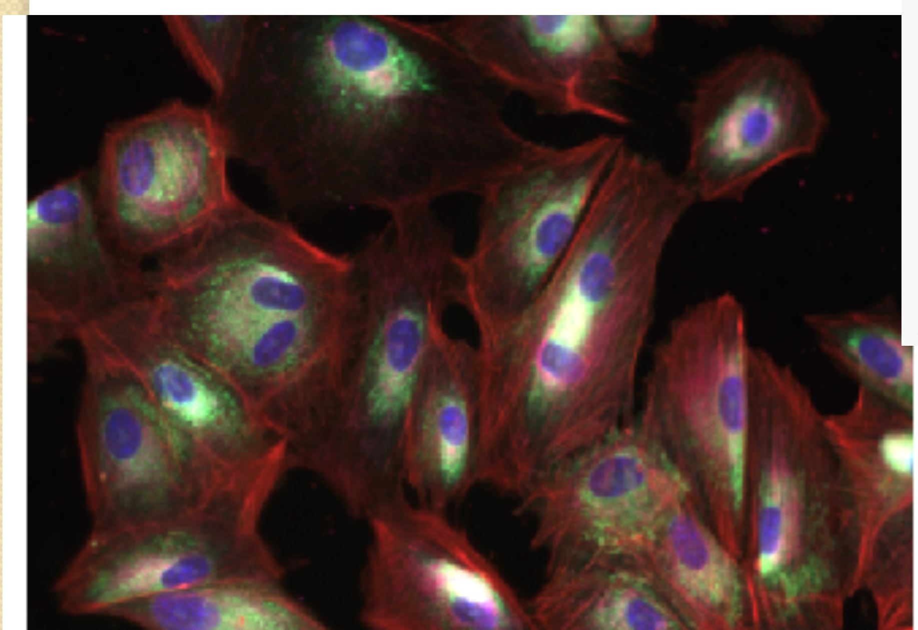
Kaggle: Fluorescent Microscopy Deep Learning with Keras

Capstone Final Project
Summer 2019
By MR. Fugu



```
t = rio.load_site('train', 'RPE-05', 3, 'D19', 2)
dir(rio)

x = rio.convert_tensor_to_rgb(t)
x.shape

plt.figure(figsize=(6, 6))
plt.axis('off')
_ = plt.imshow(x)
```

Research Question:

- Create a model in two steps:

1.) Classify (1108) types of siRNA

2.) After building and deploying the model: take care of noise to improve classification.

- Ideally, this project would have great significance in the medical community enhancing pharmacological studies and advancements as well as aiding in mitigating batch effects. Naturally, batch effects cause a great deal of difficulty in classifying images.

Data Extraction:

!git clone https://github.com/recursionpharma/rxrx1-utils

sys.path.append('/home/*Your_path*/rxrx1-utils')

** Insert your home directory path of local machine or Google storage bucket

"You red steps if you are using Google Colab":

from google.colab import auth

auth.authenticate_user()

from rxrx.main import main

Metadata:

File Description:

[train/test].zip: the image data. The image paths, such as `U2OS-01/Plate1/B02_s2_w3.png` can be read as:

`Cell line and batch number`: (*U2OS batch 1*)

`Plate number`: (*01*)

`Well location on plate`: (*column B, row 02*)

`Site`: (*2*)

`Microscope channel`: (*3*)

Biological Experiment Overview:

- 1108 different siRNA's knocking down 1108 different genes
- 51 instances of the same experiment using different batches.
- 384 well-plates

Cells are isolated and placed in the wells, each well has 1 of the 1108 siRNA's, this creates distinct biological 'genetic' conditions. Due to environmental effects the outer perimeter of the wells are not used and only 308 of the 384 wells are used for each plate.

- There are 4 plates in this experiment
  - each plate has 30 control siRNA's and 277 different non-control siRNA's
  - as well as 1 untreated well
- each well is 3.3 (mm^2)

location of the 1108 siRNA's were randomized

- Each `Well` in each plate contain two (512,512,6) images
  - each image was obtained from nonoverlapping regions within a well

- Each `Batch` is a single cell type: `24 in HUVEC, 11 in RPE, 11 in HepG2, and 5 in U2OS`
  - A `Batch` is a set of experiment plates, executed at the same time with similar reagents.
- **Batch to Batch** variations occur, creating experimental or enviromental variaiton which is unavoidable.

Experimental Section:

The raw images were preprocessed and converted to *tf_records* for easy of processing by *Recurssion Pharma*

- **Filters** : each layer (64)
- **Activation**: Layers (1,2) used 'Relu', layer (3) was 'Softmax'
- **Epochs**: 17
    - I used 'Accuracy' for this model for two reasons:
    - *First* : data from the company was expressed as 'balanced'
    - *Second*: reason was the way *Kaggle* described the submission requests.

__________________________ **There Were Three Experiements** ______________________________

1.) Utilize `Google TPU via Colab`

2.) Single *free k80 GPU with 12GB* within `Google-colab`

3.) Multpile * [2]- P100 GPU's* with 32GB *RAM*

3A.) Single T4 GPU with ~50GB *RAM*

Building Data Stream:

```
]: import rxrx.input as input
   # import rxrx.input_test as input_test

   DEFAULT_INPUT_FN_PARAMS = {
       'tfrecord_dataset_buffer_size': 256,
       'tfrecord_dataset_num_parallel_reads': None,
       'parallel_interleave_cycle_length': 32,
       'parallel_interleave_block_length': 2,
       'parallel_interleave_buffer_output_elements': None,
       'parallel_interleave_prefetch_input_elements': None,
       'map_and_batch_num_parallel_calls': 128,
       'transpose_num_parallel_calls': 128,
       'prefetch_buffer_size': tf.contrib.data.AUTOTUNE,
   }

   GLOBAL_PIXEL_STATS = (np.array([6.74696984, 14.74640167, 10.51260864,
                                   10.45369445,  5.49959796, 9.81545561]),
                          np.array([7.95876312, 12.17305868, 5.86172946,
                                   7.83451711, 4.701167, 5.43130431]))

   params={'batch_size':128}
```

6 channel Summary Stats

```
]: number_examples=70000
   step_size=int(number_examples/params['batch_size'])
   # step_size=6
```

CNN: Using Keras

```python
num_classes = 1108 #SiRNA are the classes here

def cnn_layers():
    Inp = tf.keras.Input(name='cell_img_input',shape=(512,512,6),)

    tf_records_glob=os.path.join( 'gs://rxrx1-us-central1/tfrecords/random-42', 'train', '*.tfrecord')

    filenames_dataset = tf.data.Dataset.list_files(tf_records_glob)

    dataset=input.input_fn(tf_records_glob,input_fn_params=DEFAULT_INPUT_FN_PARAMS,
                           pixel_stats=GLOBAL_PIXEL_STATS,params=params,transpose_input=False)
    print(type(dataset))
    print('+++++++++++++++++++++++++++++++++++++++++')

# 1st Convolutional Layer:
    x = layers.Conv2D(64,kernel_size=(5, 5),strides=(2,2),
                  activation='relu', padding='valid')(Inp)
    x = layers.MaxPooling2D(pool_size=(4, 4))(x)
# # 2nd Convolutional Layer:
    x = layers.Conv2D(64, (2, 2), activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
# # 3rd Convolutional Layer:
# # 4rd Convolutional Layer:
    x = layers.Conv2D(96, (2, 2), activation='relu')(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
# # First Fully Connected Layer:
    x = layers.Flatten()(x)
    x = layers.Dense(1108, activation='relu')(x)
    x = layers.Dropout(0.25)(x)
# Final Fully Connected Layer:
    predictions = layers.Dense(num_classes,
                           activation='softmax',
                           name='x_train_out')(x)

    model = tensorflow.keras.models.Model(Inp,outputs=predictions)
    return model,dataset
```

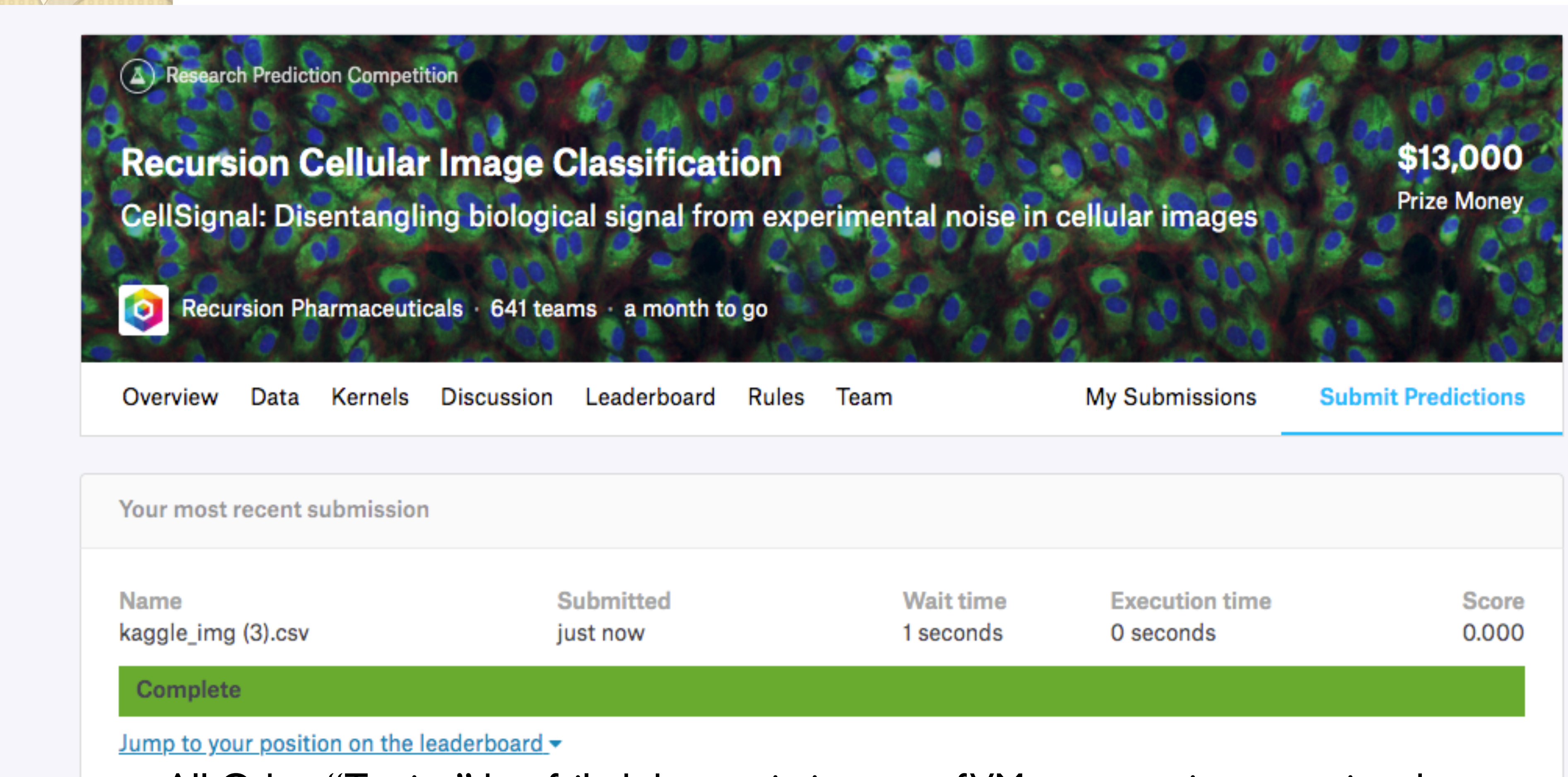
Training:

```
with tf.device('/cpu:0'):
        model,datasets = cnn_layers()

optimizer=tf.train.AdamOptimizer(learning_rate=1e-3 )

# model = multi_gpu_model(model, gpus=2)

model.compile(metrics=['acc'],
              optimizer=optimizer,
      loss='sparse_categorical_crossentropy')

filepath = './my_model.h5'

# checkpoint = ModelCheckpoint(filepath, monitor = 'loss', verbose = 1, save_best_only = True, mode = 'min')
checkpoint = ModelCheckpoint(filepath, monitor = 'acc', save_best_only = True, mode = 'max')


history=model.fit(datasets,
        epochs=8,
        steps_per_epoch=step_size,callbacks=[checkpoint])

model.save(filepath)

new_model = tensorflow.keras.models.load_model('my_model.h5')

model.summary()
```

```python
import rxrx.input_test as input_test
# Calling Test Data from Google Storage Bucket:
tf_records_glob_test=os.path.join( 'gs://rxrx1-us-central1/tfrecords/random-42', 'test', '*.tfrecord')


filenames_dataset_test = tf.data.Dataset.list_files(tf_records_glob_test)

# Create Test Dataset:
dataset_test=input_test.input_fn(tf_records_glob_test,input_fn_params=DEFAULT_INPUT_FN_PARAMS,
                                 pixel_stats=GLOBAL_PIXEL_STATS,params=params,transpose_input=False)
score=model.predict(dataset_test)

print(score)
```

```python
import numpy as np
import pandas as pd
j=[]
for i in score:
    h=np.argmax(i)
    j.append(h)

f=pd.read_csv('test.csv')
f.shape

k=pd.DataFrame(j,columns=['sirna'])

test_df_kaggle=pd.concat([f,k],axis=1)
fin_df=test_df_kaggle.iloc[:,[0,4]]
fin_df_=fin_df.dropna()
fin_df_.to_csv('kaggle_img_.csv',sep=',',index=False)


pd.read_csv('kaggle_img_.csv')
```

Test Prediction with (1)-Epoch



All Other "Testing" has failed due to timing out of VM, memory issues, saving the output file, increasing batch size or layers all have problems with 1-T4 GPU with 52GB RAM

Conclusion:

Model training a tradtional *3 Layer CNN*, poses unforseen challenges: optimization of resources, time out errors, memory requirements are just a few of the concerns that arise. Interestingly, enough creating callbacks to store your model have inherent flaws as well. This seemed straightforward to code in one line, except it was having issues just storing data for future use without spending additional hours recompiling a model. The amount of *accuracy* significantly diminished when callbacks were used. For instance, without callbacks set for monitoring `accuracy` there was an 87% training rate before prediction whereas using a callback dropped training `accuracy` to around *10%* . This suggests two things: the `first` which was not as obvious, which was without a callback the model only returns the *last training example*, the `other issue` may be related which would have been `overfitting`.