In [294]:

```
import tensorflow as tf
from sklearn.ensemble import RandomForestClassifier
from tensorflow import keras
import pandas as pd
import numpy as np
import ast
from tensorboard.plugins.hparams import api as hp
from tensorboard import notebook
import datetime, os
import math
from tensorflow.contrib.tensor_forest.python import tensor_forest
import cv2
import matplotlib.pyplot as plt
```

In [295]:

```
tf.enable_eager_execution()
```

In [296]:

```
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [772]:

```
def convertToInt(this_output):
    if this_output == [0, 1, 0, 0, 0]:
        return 1
    else:
        return 0
```

In [773]:

```
def convertToQuadrant(x, y):
    if x > 0 and y < 0:
        return [1,0,0,0]
    if x < 0 and y < 0:
        return [0,1,0,0]
    if x > 0 and y < 0:
        return [0,0,1,0]
    else:
        return [0,0,0,1]
```

In [774]:

```
def convertToAngle(x,y):
    if x == 0:
        x += 0.1
    if y == 0:
        y += 0.1
    if x < 0 and y < 0:
        rads = (math.atan(-x/-y)) + (math.pi/2)
    elif x < 0 and y > 0:
        rads = (math.atan(y/-x)) + (math.pi)
    elif x > 0 and y < 0:
        rads = (math.atan(-y/x))
    elif x > 0 and y > 0:
        rads = (math.atan(x/y)) + ((3 * math.pi)/2)
    return [rads]
```

In [775]:

```
def populate_training_testing(train_data, test_data):

    testing_inputs = []
    testing_outputs = []
    training_inputs = []
    training_outputs = []
    test_array = []

    for i,row in test_data.iterrows():
        features = populate_features(row)
        this_test = [row['token']]
        this_test.extend([features, row['labels'], row['bounding box']])
        test_array.append(this_test)

        this_output = ast.literal_eval(row['labels'])
        this_output = ast.literal_eval(row['labels'])
        testing_outputs.append(convertToInt(this_output))

        testing_inputs.append(features)

    for i,row in train_data.iterrows():
        features = populate_features(row)
        this_output = ast.literal_eval(row['labels'])

        training_outputs.append(convertToInt(this_output))

        training_inputs.append(features)

    return training_inputs, training_outputs, testing_inputs, testing_outputs
```

In [776]:

```
def train_model(input_neurons, hidden_layers, output_neurons, opt, dropout):
    print(input_neurons)
    nn_model = tf.keras.models.Sequential()
    nn_model.add(keras.layers.InputLayer(input_shape=(1,input_neurons)))
    for neurons in hidden_layers:
        print(neurons)
        nn_model.add(tf.keras.layers.Dense(neurons, activation=tf.nn.relu))
    print(output_neurons)
    nn_model.add(tf.keras.layers.Dropout(dropout))
    nn_model.add(keras.layers.Dense(output_neurons, activation=tf.nn.softmax)

    nn_model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

    return nn_model
```

In [777]:

```
def result_breakdown(predictions, test_array):
    print(len(predictions))
    false_positive = []
    true_negative = []
    correct_matches = []
    label_matches = []
    failures = []
    for i in range(len(testing_inputs)):
        label_match = testing_outputs[i] == 1 and predictions[i][0][1] > 0.5
        correct = (testing_outputs[i] == 0 and predictions[i][0][0] > 0.5) or
        false_pos = testing_outputs[i] == 0 and predictions[i][0][1] >= 0.5
        true_neg = testing_outputs[i] == 1 and predictions[i][0][0] >= 0.5
        match = []
        match.extend([test_array[i][0]])
        match.extend([test_array[i][1], test_array[i][2], test_array[i][3], p
        if label_match:
            label_matches.append(match)
        if false_pos:
            false_positive.append(match)
        if true_neg:
            true_negative.append(match)
        if correct:
            correct_matches.append(match)
        if not correct:
            failures.append(match)
    return label_matches, false_positive, true_negative, correct_matches, fai
```

In [778]:

```
def sort_accuracy(result):
    return sorted(result, key=lambda x: x[4][1], reverse=True)
```

In [779]:

```python
def visualizeFalseMatches(filename, false_positives):
    document_path = "BCData/BC/" + filename
    img = cv2.imread(document_path, cv2.IMREAD_COLOR)

    top_index = int(len(false_positive) * 0.2)
    end_index = int(len(false_positive) * 0.8)

#     top_fifth = false_positives[0:top_index]
#     middle = false_positives[top_index:end_index]
#     bottom_fifth = false_positives[end_index:]

    for i in range(len(false_positives)):
        token = false_positives[i]
        bbox = ast.literal_eval(token[3])
        x1,y1,x2,y2 = int(bbox[0]),int(bbox[1]),int(bbox[2]),int(bbox[3])
        color = None
        accuracy = token[4]
        cv2.rectangle(img, (x1,y1), (x2,y2), color, 2)
        if i < top_index:
            color = (255,0,0)
        elif i >= top_index and i < end_index:
            color = (255,255,0)
        else:
            color = (255,255,255)
        cv2.rectangle(img, (x1,y1), (x2,y2), color, 2)

    plt.figure(figsize=(30,30))
    plt.imshow(img)
    plt.show()
```

In [793]:

```python
def populate_features(row):
    xy_distance = ast.literal_eval(row['xy_distance'])
    relative_xy = ast.literal_eval(row['relative xy position'])
    line_neighbors = ast.literal_eval(row['line neighbors'])
    phrase_dist = ast.literal_eval(row['min phrase distance'])
    matches = ast.literal_eval(row['levenshtein_matches'])
    surround_neighbors = ast.literal_eval(row['neighbors'])
    phrases = ast.literal_eval(row['completed phrases'])
    features = []
    features.extend([relative_xy[1]])
#     features.extend(convertToAngle(xy_distance[0],xy_distance[1]))
#     features.extend(convertToQuadrant(xy_distance[0],xy_distance[1]))
#     features.extend([matches[1], neighbors[1], phrases[1]])
    features.extend([matches[1],phrases[1],
                     line_neighbors[2], line_neighbors[3]])
    return features
```

In [794]:

```python
# feature_list  = ['y position','levenshtein match', 'completed phrases','mot
#                  'father line neighbors']
feature_list  = ['Angle','levenshtein match', 'completed phrases',
                 'surround neighbors','mother line neighbors', 'father line r
```

In [795]:

```
def get_train_test_sets(file_list):
    # new_features = pd.read_csv("/Users/abasker/Desktop/Past Output Files/cl
    # original_features = pd.read_csv("/Users/abasker/Desktop/Past Output Fil
    original_features = pd.read_csv('../idea_projects/featureextraction/Data/
    new_features = pd.read_csv('../idea_projects/featureextraction/Data/new_f
    completed_phrases_col = new_features['completed phrases']
    phrase_dist_col = new_features['min phrase distance']
    original_features["min phrase distance"] = phrase_dist_col
    original_features["completed phrases"] = completed_phrases_col

    nn_data = original_features

    test_data = nn_data[(nn_data['filename'].isin(file_list)) & (nn_data['tok
    train_data = nn_data[~nn_data['filename'].isin(file_list)]
    train_data = train_data[train_data['labels'] != '[0, 0, 0, 0, 1]']

    return train_data, test_data
```

In [796]:

```
def reshape(input_data):
    return np.array(input_data).reshape(len(input_data),1, len(input_data[0])
```

In [797]:

```
def train_rf(training_inputs, training_outputs):
    shape = training_inputs.shape
    rf_train_data = training_inputs.reshape(shape[0], shape[2])
    clf = RandomForestClassifier(n_estimators=100,n_jobs=2, random_state=0)
    clf.fit(rf_train_data, training_outputs)
    return clf
```

In [798]:

```
def get_nn_results(file_list, data):
    training_inputs, testing_inputs, training_outputs, testing_outputs = data

    # input_neurons, hidden_layers, output_neurons, optimizer, dropout
    input_neurons = training_inputs.shape[2]
    nn_model = train_model(input_neurons, [12, 4], 2, 'adam', 0.1)

    nn_model.fit(training_inputs, np.array(training_outputs), epochs=10)
    test_loss, test_acc = nn_model.evaluate(testing_inputs, testing_outputs)
    print('Test accuracy:', test_acc)
    predictions = nn_model.predict(testing_inputs)
    nn_result = result_breakdown(predictions, test_array)

    return nn_result
```

In [799]:

```
def get_rf_results(file_list, data):
    training_inputs, testing_inputs, training_outputs, testing_outputs = data

    clf = train_rf(training_inputs, training_outputs)

    test_shape = testing_inputs.shape
    testing = testing_inputs.reshape(test_shape[0],test_shape[2])
    p = clf.predict_proba(testing)
    new_pred = [[i] for i in p]
    clf_result = result_breakdown(new_pred, test_array)

    rf_dataframe = pd.DataFrame()
    rf_dataframe['Feature'] = feature_list
    rf_dataframe['Importance'] = clf.feature_importances_
    print(rf_dataframe)

    return clf_result
```

In [800]:

```
file_list = ["Ahree's birth certificate.pdf-page-1.jpg"]
train_data, test_data = get_train_test_sets(file_list)
training_inputs, training_outputs, testing_inputs, testing_outputs, test_arra
training_inputs, testing_inputs = reshape(training_inputs), reshape(testing_i
model_data = training_inputs, testing_inputs, training_outputs, testing_outpu
```

In [801]:

```
train_data
```

Out[801]:

	filename	token	bounding box	labels	xy_distance	levenshtein_
17	0868_001.pdf-page-1.jpg	CHILD	[510.0, 1183.0, 590.0, 1224.0]	[1, 0, 0, 0, 0]	[-980.0, -776.5]	[0.0, 4.0,
18	0868_001.pdf-page-1.jpg	-	[599.0, 1184.0, 609.0, 1224.0]	[1, 0, 0, 0, 0]	[-926.0, -776.0]	[5.0, 4.0,
19	0868_001.pdf-page-1.jpg	NAME	[609.0, 1184.0, 691.0, 1225.0]	[1, 0, 0, 0, 0]	[-880.0, -775.5]	[4.0, 2.0,
67	0868_001.pdf-page-1.jpg	MOTHER	[517.0, 1538.0, 633.0, 1587.0]	[0, 0, 1, 0, 0]	[-955.0, -417.5]	[6.0, 4.0,

In [802]:

```
nn_label_match, nn_false_positive, nn_true_negative, nn_correct_matches, nn_
```

```
5
12
4
2
Train on 3098 samples
Epoch 1/10
3098/3098 [==============================] - 1s 228us/sample - loss: 0.5285
- acc: 0.7415
Epoch 2/10
3098/3098 [==============================] - 0s 36us/sample - loss: 0.3354
- acc: 0.7435
Epoch 3/10
3098/3098 [==============================] - 0s 34us/sample - loss: 0.2505
- acc: 0.7277
Epoch 4/10
3098/3098 [==============================] - 0s 62us/sample - loss: 0.1925
- acc: 0.7287
Epoch 5/10
3098/3098 [==============================] - 0s 43us/sample - loss: 0.1609
- acc: 0.7225
Epoch 6/10
3098/3098 [==============================] - 0s 45us/sample - loss: 0.1443
- acc: 0.7152
Epoch 7/10
3098/3098 [==============================] - 0s 35us/sample - loss: 0.1414
- acc: 0.6897
Epoch 8/10
3098/3098 [==============================] - 0s 42us/sample - loss: 0.1312
- acc: 0.6841
Epoch 9/10
3098/3098 [==============================] - 0s 47us/sample - loss: 0.1275
- acc: 0.6859
Epoch 10/10
3098/3098 [==============================] - 0s 34us/sample - loss: 0.1220
- acc: 0.6837
107/107 [==============================] - 0s 1ms/sample - loss: 0.0560 - a
cc: 0.9524
Test accuracy: 0.95239586
107
```

In [803]:

```
sort_accuracy(nn_label_match)
```

Out[803]:

```
[['DATE',
  [0.30833333333333335, 0.0, 3, 0, 0],
  '[0, 1, 0, 0, 0]',
  '[212.0, 1181.0, 376.0, 1261.0]',
  array([0.02212774, 0.9778723 ], dtype=float32)],
 ['BIRTH',
  [0.30833333333333335, 0.0, 2, 0, 0],
  '[0, 1, 0, 0, 0]',
  '[519.0, 1181.0, 695.0, 1261.0]',
  array([0.05407757, 0.9459224 ], dtype=float32)]]
```

```
In [804]:  sort_accuracy(nn_false_positive)
```

```
Out[804]: [['DATE',
            [0.7037878787878787, 0.0, 3, 1, 1],
            '[0, 0, 0, 0, 1]',
            '[224.0, 2744.0, 377.0, 2830.0]',
            array([0.04888941, 0.9511106 ], dtype=float32)]]
```

```
In [805]:  nn_true_negative
```

```
Out[805]: []
```

```
In [806]:  sort_accuracy(nn_correct_matches)
```

```
Out[806]: [['DATE',
            [0.30833333333333335, 0.0, 3, 0, 0],
            '[0, 1, 0, 0, 0]',
            '[212.0, 1181.0, 376.0, 1261.0]',
            array([0.02212774, 0.9778723 ], dtype=float32)],
           ['BIRTH',
            [0.30833333333333335, 0.0, 2, 0, 0],
            '[0, 1, 0, 0, 0]',
            '[519.0, 1181.0, 695.0, 1261.0]',
            array([0.05407757, 0.9459224 ], dtype=float32)],
           ['Bow',
            [0.8625, 4.0, 2, 0, 0],
            '[0, 0, 0, 0, 1]',
            '[2818.0, 3357.0, 2952.0, 3474.0]',
            array([0.5503256 , 0.44967443], dtype=float32)],
           ['DATE',
            [0.871969696969697, 0.0, 0, 0, 0],
            '[0, 0, 0, 0, 1]',
            '[647.0, 3426.0, 749.0, 3480.0]',
            array([0.73510325, 0.26489678], dtype=float32)],
```

```
In [686]:  rf_label_match, rf_false_positive, rf_true_negative, rf_correct_matches, rf_
```

```
423
                     Feature  Importance
0                 y position    0.051607
1          levenshtein match    0.266173
2          completed phrases    0.475034
3    minimum phrase distance    0.062272
4      mother line neighbors    0.080301
5      father line neighbors    0.064614
```

In [683]:

```
rf_label_match
```

Out[683]:

```
[['BIRT',
  [0.22581845238095238, 1.0, 2, 0, 0],
  '[0, 1, 0, 0, 0]',
  '[1888.0, 901.0, 1957.0, 920.0]',
  array([0., 1.])],
 ['DATE',
  [0.22495039682539683, 0.0, 3, 0, 0],
  '[0, 1, 0, 0, 0]',
  '[1759.0, 872.0, 1834.0, 942.0]',
  array([0., 1.])]]
```

In [520]:

```
sort_accuracy(rf_false_positive)
```

Out[520]:

```
[['DATE',
  [0.22718253968253968, 0.0, 3, 0.013406078130422867, 0, 0],
  '[0, 0, 0, 0, 1]',
  '[1757.0, 906.0, 1830.0, 926.0]',
  array([0., 1.])],
 ['BIRTH',
  [0.2238343253968254, 0.0, 2, 0.01806602235178407, 0, 0],
  '[0, 0, 0, 0, 1]',
  '[1893.0, 868.0, 1973.0, 937.0]',
  array([0., 1.])],
 ['BIRTH',
  [0.38132440476190477, 0.0, 2, 0.01594043952499905, 0, 0],
  '[0, 0, 0, 0, 1]',
  '[481.0, 1520.0, 549.0, 1555.0]',
  array([0., 1.])],
 ['BIRTH',
  [0.3790922619047619, 0.0, 2, 0.02133144139026578, 0, 0],
  '[0, 0, 0, 0, 1]',
  '[1709.0, 1505.0, 1792.0, 1552.0]',
  array([0., 1.])]
```

In [554]:

```
!rm -rf ./logs/
```

In [555]:

```
logs_base_dir = "./logs"
os.makedirs(logs_base_dir, exist_ok=True)
%tensorboard --logdir {logs_base_dir}
```

```
Reusing TensorBoard on port 6010 (pid 66461), started 5 days, 2:36:08 ago.
(Use '!kill 66461' to kill it.)
```

In [556]:

```
HP_NUM_UNITS_1 = hp.HParam('num_units_1', hp.Discrete([8,12,16]))
HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.1, 0.2))
HP_NUM_UNITS_2 = hp.HParam('num_units_2', hp.Discrete([4,6,8]))
# HP_NUM_LAYERS = hp.HParam('num_layers', hp.Discrete([1,2,3]))
HP_DROPOUT = hp.HParam('dropout', hp.RealInterval(0.1, 0.2))
HP_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'sgd']))

METRIC_ACCURACY = 'accuracy'

with tf.contrib.summary.create_file_writer('logs/hparam_tuning').as_default(
    hp.hparams_config(
        hparams=[HP_NUM_UNITS_1, HP_NUM_UNITS_2, HP_DROPOUT, HP_OPTIMIZER],
        metrics=[hp.Metric(METRIC_ACCURACY, display_name='Accuracy')],
    )
```

In [557]:

```
def train_test_model(hparams, data):
    training_inputs, testing_inputs, training_outputs, testing_outputs = data
    nn_model = tf.keras.models.Sequential()
    nn_model.add(keras.layers.InputLayer(input_shape=(1,training_inputs.shape
#     for x in range(hparams[HP_NUM_LAYERS]):
#         nn_model.add(tf.keras.layers.Dense(hparams[HP_NUM_UNITS], activatio
    nn_model.add(tf.keras.layers.Dense(hparams[HP_NUM_UNITS_1], activation=tf
    nn_model.add(tf.keras.layers.Dropout(hparams[HP_DROPOUT]))
    nn_model.add(tf.keras.layers.Dense(hparams[HP_NUM_UNITS_2], activation=tf
    nn_model.add(keras.layers.Dense(2, activation=tf.nn.softmax))

    nn_model.compile(
      optimizer=hparams[HP_OPTIMIZER],
      loss='sparse_categorical_crossentropy',
      metrics=['accuracy'],
    )
#     logdir = os.path.join(logs_base_dir, datetime.datetime.now().strftime('

    logdir = os.path.join(logs_base_dir, "Layer_1:" + str(hparams[HP_NUM_UNIT
                          "Layer_2:" + str(hparams[HP_NUM_UNITS_2]) +
                          "Optimizer:" + str(hparams[HP_OPTIMIZER]))

    nn_model.fit(training_inputs, training_outputs, epochs=5,
        callbacks=[
            tf.keras.callbacks.TensorBoard(logdir),
            hp.KerasCallback(logdir, hparams),
    ])
    _, accuracy = nn_model.evaluate(testing_inputs, testing_outputs)
    return accuracy
```

In [558]:

```
def run(run_dir, hparams, model_data):
    with tf.contrib.summary.create_file_writer(run_dir).as_default():
        hp.hparams(hparams)
        accuracy = train_test_model(hparams, model_data)
        tf.summary.scalar(METRIC_ACCURACY, accuracy)
#         tf.summary.scalar(METRIC_ACCURACY, accuracy,step=1)
```

In [551]:

```
def run_hyperparameter_tuning(model_data):
    session_num = 0
    for num_units_1 in HP_NUM_UNITS_1.domain.values:
        for num_units_2 in HP_NUM_UNITS_2.domain.values:
            for dropout_rate in (HP_DROPOUT.domain.min_value, HP_DROPOUT.doma
                for optimizer in HP_OPTIMIZER.domain.values:
                    hparams = {
    #                       HP_NUM_LAYERS: num_layers,
                      HP_NUM_UNITS_1: num_units_1,
                      HP_NUM_UNITS_2: num_units_2,
                      HP_DROPOUT: dropout_rate,
                      HP_OPTIMIZER: optimizer,
                    }
                    run_name = "run-%d" % session_num
                    print('--- Starting trial: %s' % run_name)
                    print({h.name: hparams[h] for h in hparams})
                    run('logs/hparam_tuning/' + run_name, hparams, model_data
                    session_num += 1
```

In [552]:

```
run_hyperparameter_tuning(model_data)
```

```
--- Starting trial: run-0
{'num_units_1': 8, 'num_units_2': 4, 'dropout': 0.1, 'optimizer': 'ada
m'}
Train on 3092 samples
Epoch 1/5
3092/3092 [==============================] - 1s 167us/sample - loss: 0.5
581 - acc: 0.7911
Epoch 2/5
3092/3092 [==============================] - 0s 37us/sample - loss: 0.34
87 - acc: 0.7520
Epoch 3/5
3092/3092 [==============================] - 0s 32us/sample - loss: 0.27
72 - acc: 0.7204
Epoch 4/5
3092/3092 [==============================] - 0s 31us/sample - loss: 0.24
46 - acc: 0.7130
Epoch 5/5
3092/3092 [==============================] - 0s 31us/sample - loss: 0.22
17 - acc: 0.6988
```

In [ ]:

```
```