Step 0: Input

Get a table and define a label, x, y, and test column(s)

The test string will be matched as a prefix for the column name, specify the entire name to use a single column

Selecting multiple columns allows for correlating similar tests or higher channel counts

cSessionId is only needed for agg tests, to get test limits

Data will be written out to tables prefixed with temp

By default only the finished dataset and the results are kept, but intermediate tables are written for re-runs after a fail

```
%load_ext autoreload
%autoreload 2
#
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

channels = 3
d = 80 # Latent Space Dimension
test_str = 'TEST_PREFIX'
label = 'U1'
x = label + '_x'
y = label + '_y'
cSessionId = 'cSessionId' #used to tie tests to their limits
table = 'TABLENAME'
temp = 'TEMP_TABLENAME'

dlt_sdf = spark.sql("SELECT * FROM {}".format(table))
test_lst = [c for c in dlt_sdf.columns if c.startswith(test_str)]
cols = test_lst + [label, x, y, cSessionId]

dlt_sdf = dlt_sdf.select(cols)

display(dlt_sdf)
```

Step 1: Impute

Imputes the missing sections, while trying to preserve the shape of the wafer.

Writes out to temp + impute

```
from vae_pipeline.impute import get_interpolated

df = get_interpolated( # this took 40 - 60 minutes on a large cluster
   dlt_sdf,
   test_lst,
   label,
   x,
   y
 )
 display(df)
spark.sql("DROP TABLE IF EXISTS {}impute".format(temp))
df.write.mode('overwrite').saveAsTable('{}impute'.format(temp))
```

Step 2: Aggregate Tests

Creates an column of aggregated test failures

Can be trained on or visualized and compared to patterns outputted from tests

```
from vae_pipeline.agg_tests import prepare_test_data, agg_tests
#from pyspark.sql import SparkSession
#spark = SparkSession.builder.getOrCreate()
dlt_sdf = spark.sql("SELECT * FROM {}".format(table))

trimmed_tst_list, test_data = prepare_test_data(dlt_sdf, test_lst, spark)
#aggregate tests
out_sdf = agg_tests(dlt_sdf, trimmed_tst_list, test_data)

display(out_sdf)

spark.sql("DROP TABLE IF EXISTS {}agg_tests".format(temp))
out_sdf.write.mode('overwrite').saveAsTable('{}agg_tests'.format(temp))
```

Step 3: Group correlated tests

Take all the given tests and sort in groups

Number of groups is the number of channels

The tests in a group will be averaged

Drop threshold removes wafers that are missing more than that percentage of values

Threshold is the amount of correlation required to group tests

```
from vae_pipeline.correlated_tests import get_clusters

dlt_sdf = spark.sql("SELECT * FROM {}impute".format(temp))
us = dlt_sdf.select(label).distinct().collect()

clusters, c_sdf = get_clusters( # took ~15 minutes on e2, shows an error but not breaking
   dlt_sdf,
   channels,
   test_str,
   label,
   x,
   y,
   drop_threshold=0.5,
   threshold=0.8
)

spark.sql("DROP TABLE IF EXISTS {}corr_matrix".format(temp))
c_sdf.write.mode('overwrite').saveAsTable('{}corr_matrix'.format(temp))

# Convert clusters list to a DataFrame and save it
clusters_df = spark.createDataFrame([(i,) for i in clusters], ["cluster"])
spark.sql("DROP TABLE IF EXISTS {}clusters".format(temp))
```

Step 4: Create Wafermaps

Take clusters or tests and write to a pivot table of values

Creates an image like data structure

```
clusters_df.write.mode('overwrite').saveAsTable('{}clusters'.format(temp))
from vae_pipeline.pre_vae import get_dataset

# Convert clusters DataFrame back to a list of lists
clusters_df = spark.sql("SELECT * FROM {}clusters".format(temp))
clusters = [row.cluster for row in clusters_df.collect()]

dlt_sdf = spark.sql("SELECT * FROM {}impute".format(temp))

dataset, sizes = get_dataset(   # took about an hour
   dlt_sdf,
   clusters,
   coln=label,
   x=x,
   y=y
)

# Convert dataset list to a DataFrame and save it
dataset_df = spark.createDataFrame(dataset, [label, 'values'])
spark.sql("DROP TABLE IF EXISTS {}dataset".format(temp))
dataset_df.write.mode('overwrite').saveAsTable('{}dataset'.format(temp))

sizes_df = spark.createDataFrame(enumerate(sizes), ["index", "size"])
spark.sql("DROP TABLE IF EXISTS {}sizes".format(temp))
sizes_df.write.mode('overwrite').saveAsTable('{}sizes'.format(temp))
```

Step 5: Run VAE

Run the training loop

There are a lot of options here to be adjusted

If you are getting NaN as the training loss, the dataset is likely broken or contains missing values.

```
from vae_pipeline.vae import run_vae

dlt_sdf = spark.sql("SELECT * FROM {}dataset".format(temp))

sizes = spark.sql("SELECT size FROM {}sizes ORDER BY index".format(temp)).collect()
sizes = [row.size for row in sizes]

print(sizes)

model = run_vae( # maybe 20 minutes
   dlt_sdf,
   sizes,
   colname=label,
   save_chkpt='checkpoint.pth',
   learning_rate=3e-3,
   d=d,
   channels=channels,
   outc=2,
   padding=0,
   stride=1,
   kernel=3,
   fc1_connections1=128,
   fc1_connections2=64,
   fc1_connections3=32,
   batch_size=32,
   epochs=100
 )
```

Step 6: Metrics and logging

Loads the model from saved weights

Gets a similarity score for every wafer relationship

Writes the results and summarized distribution

```
from vae_pipeline.post_vae import evaluate_vae

dlt_sdf = spark.sql("SELECT * FROM {}dataset".format(temp))

stats, df, udf = evaluate_vae(
 model,
 dlt_sdf,
 d=d,
 load_chkpt='checkpoint.pth',
 col=label,
 channels=channels
 )

# save to db
sparkdf = spark.createDataFrame(df)
display(sparkdf)
# drop if exists
spark.sql("DROP TABLE IF EXISTS {}similarity".format(temp))
sparkdf.write.mode("overwrite").saveAsTable("{}similarity".format(temp))

# delete the intermediate tables
spark.sql("DROP TABLE IF EXISTS {}impute".format(temp))
spark.sql("DROP TABLE IF EXISTS {}agg_tests".format(temp))
spark.sql("DROP TABLE IF EXISTS {}corr_matrix".format(temp))
spark.sql("DROP TABLE IF EXISTS {}clusters".format(temp))
# spark.sql("DROP TABLE IF EXISTS {}dataset".format(temp))
# spark.sql("DROP TABLE IF EXISTS {}sizes".format(temp))

print(stats)
```

Step 7: Group Similarity Matrix

Enforce matrix symmetry and modify scores

```
from vae_pipeline.nn_output_grouping import clusterNN_hierarchical,
clusterNN_kmeans

nn_out = spark.sql('select * from {}similarity'.format(temp))
clustered_nn_sdf, core_samples,  num_clusters = clusterNN_hierarchical(nn_out,
spark, threshold=0.9)

# print(core_samples)
display(clustered_nn_sdf)
```

Step 8: Visualization

Requires some external packages

Uses similarity score to create a force based network graph

Connecting wafers with a score above a threshold and a strength of that score

Then uses Kmeans on the resultant locations and displays samples

```
!pip install datashader
!pip install dask
!pip install scikit-image
#%restart_python

from vae_pipeline.plots import network

network(spark, temp + 'dataset', temp + 'similarity', label)
```