Test

June 18, 2018

```
In [53]: import cv2
         import numpy as np
         from matplotlib import pyplot as plt
         #%matplotlib ipympl
         import glob as gb
         import csv

In [83]: res = 28

In [55]: total = []

         img_path = gb.glob("data/*.JPG")
         i = 0
         for path in img_path:
             pl = path.lower()

             img1 = cv2.imread(path, cv2.IMREAD_GRAYSCALE)#'data/Resized_256_256xBroken_0.JPG',
             if img1 is None:                      #img1
                         continue
             res1= cv2.resize(img1,(res,res))               #
             res1 = cv2.equalizeHist(res1)
             res1_1 = res1.flatten()/255.0 #res1_1 = res1.reshape(1,784)/255.0       #
             #im_data = np.concatenate((im_data, res1_1))
             res1_1_1 = res1_1.tolist()                       #numpy.narraylist
             total.append(res1_1_1)
```

1 Label 10 classes

```
In [56]: label10 = []
         #im_data = np.array([])
         img_path = gb.glob("data/*.JPG")
         i = 0
         for path in img_path:
             pl = path.lower()
             if "dripping" in pl:
                 if "dripping_30oct" in pl:
                     label = 0
```

```
            else:
                label = 1
        elif "jetting" in pl:
            if "extreme" in pl:
                label = 2
            elif "30oct" in pl:
                label = 3
            else:
                label = 4
        elif "wetting" in pl:
            if "extreme" in pl:
                label = 5
            elif "30oct" in pl:
                label = 6
            else:
                label = 7
        elif "broken" in pl:
            if "new" in pl:
                label = 8
            else:
                label = 9
        else:
            label = 10

        label10.append(label)
```

2 Label 4 classes

In [60]:
```
label4 = []
img_path = gb.glob("data/*.JPG")
for path in img_path:
    pl = path.lower()
    if "dripping" in pl:
        label = 0
    elif "jetting" in pl:
        label = 1
    elif "wetting" in pl:
        label = 2
    else:
        label = 3

    label4.append(label)
```

3 Save Image Data

In [62]:
```
im_data = np.array(total)
im_data.tofile('data/img'+str(res)+'.bin')
```

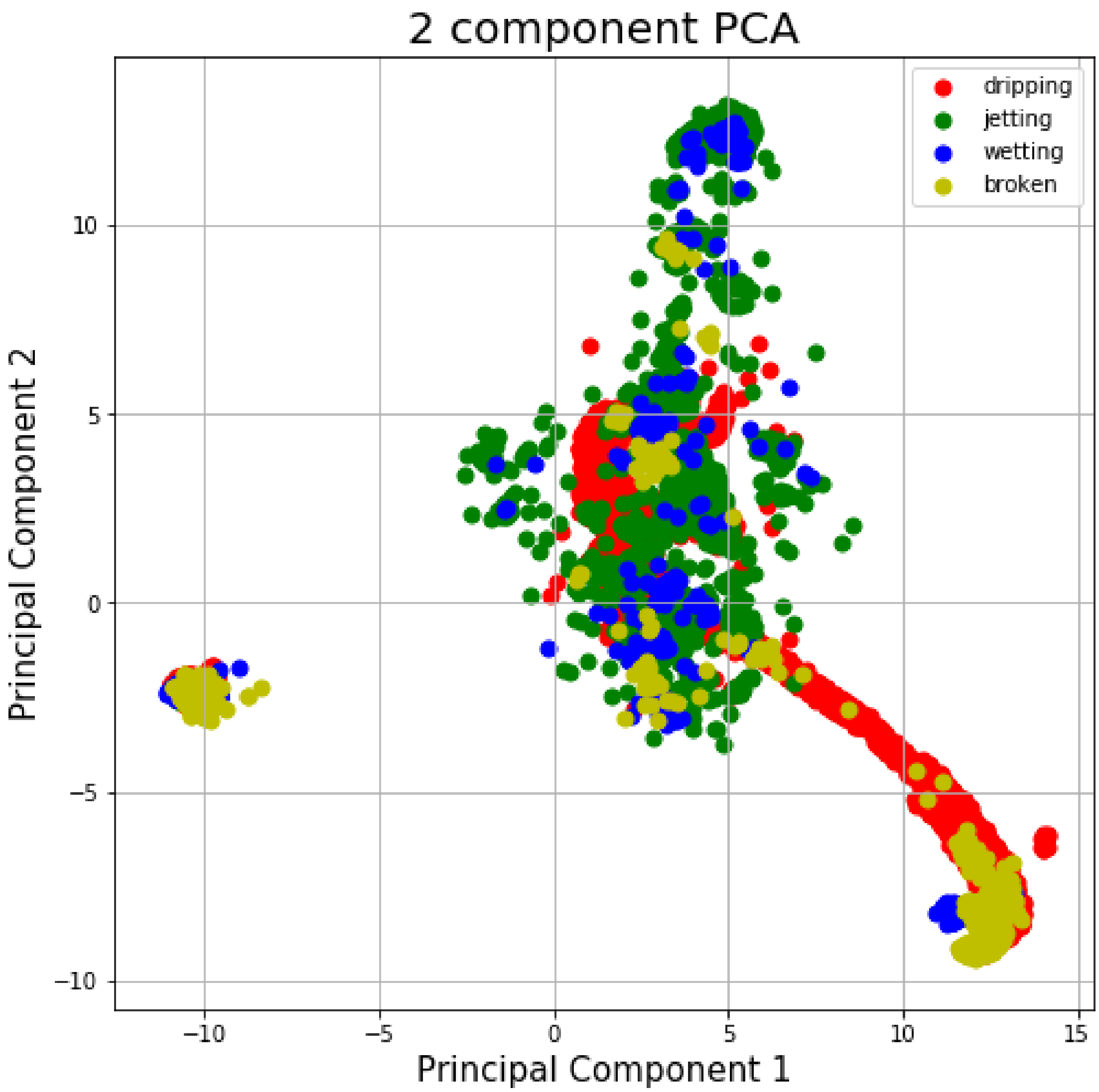
```
im_lb4 = np.array(label4)
im_lb4.tofile('data/label4.bin')
im_lb10 = np.array(label10)
im_lb10.tofile('data/label10.bin')
```

4 Load Image Data

```
In [63]: # Load IMG
         res2 = res*res
         data = np.fromfile('data/img'+str(res)+'.bin',  dtype=np.float64)
         im_data = data.reshape(-1, res2)
         # Load Label
         im_lb4 = np.fromfile('data/label4.bin', dtype=np.int64)
         im_lb10 = np.fromfile('data/label10.bin', dtype=np.int64)
```

5 2 component PCA 4 Classes

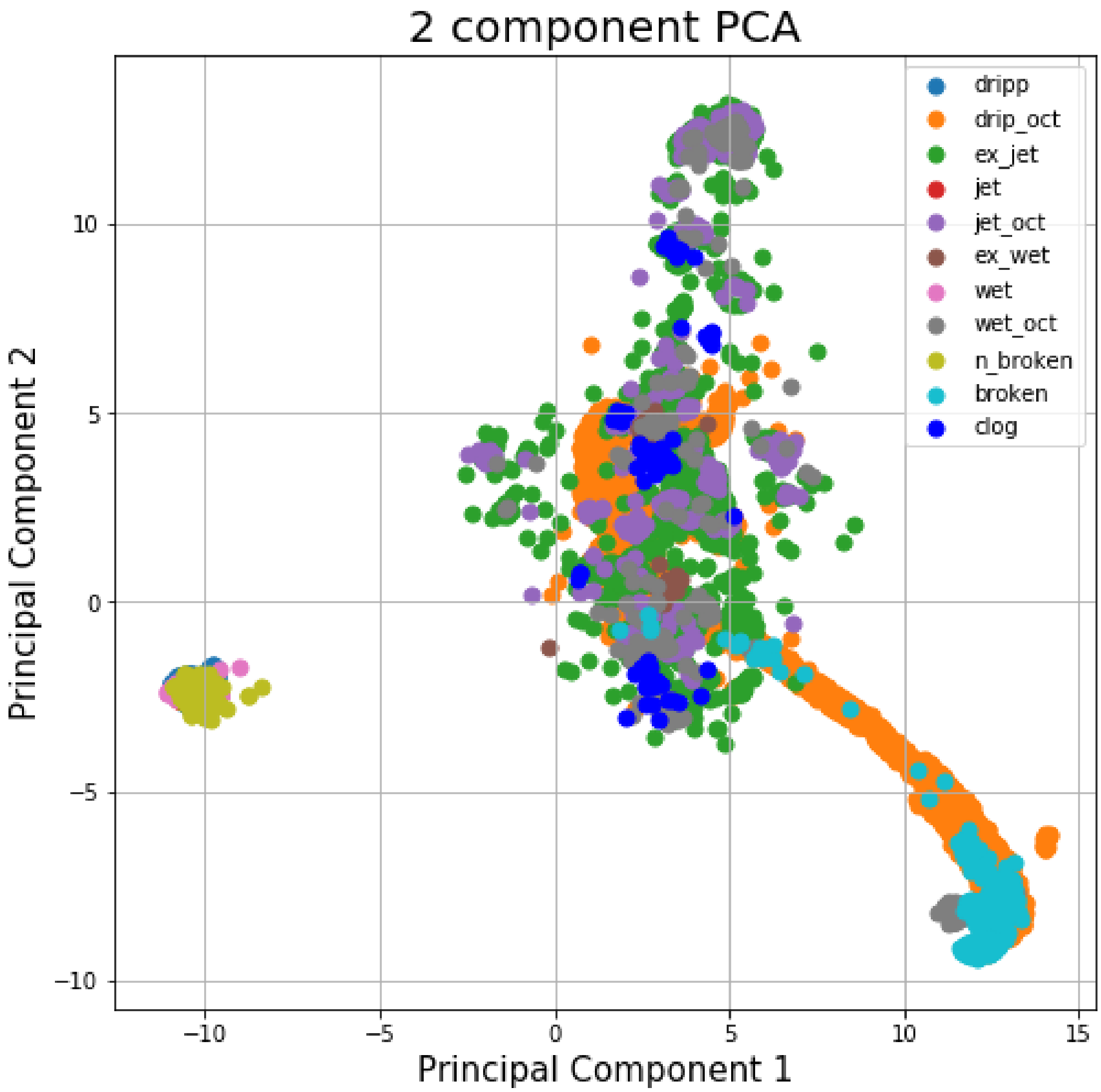
```
In [64]: from sklearn.decomposition import PCA
         pca = PCA(n_components=2)
         principalComponents = pca.fit_transform(im_data)

         fig = plt.figure(figsize = (8,8))
         ax = fig.add_subplot(1,1,1)
         ax.set_xlabel('Principal Component 1', fontsize = 15)
         ax.set_ylabel('Principal Component 2', fontsize = 15)
         ax.set_title('2 component PCA', fontsize = 20)
         targets = [0,1,2,3]
         states = ['dripping', 'jetting', 'wetting', 'broken']
         colors = ['r', 'g', 'b', 'y']
         for target, color, state in zip(targets,colors,states):
             ##indicesToKeep = finalDf['target'] == target
             ind = im_lb4 == target
             ax.scatter(principalComponents[ind,0],#finalDf.loc[indicesToKeep, 'principal compon
                        principalComponents[ind,1],#, finalDf.loc[indicesToKeep, 'principal comp
                        c = color,
                        s = 50)
         ax.legend(states)
         ax.grid()
```



6 2 component PCA 10 Classes

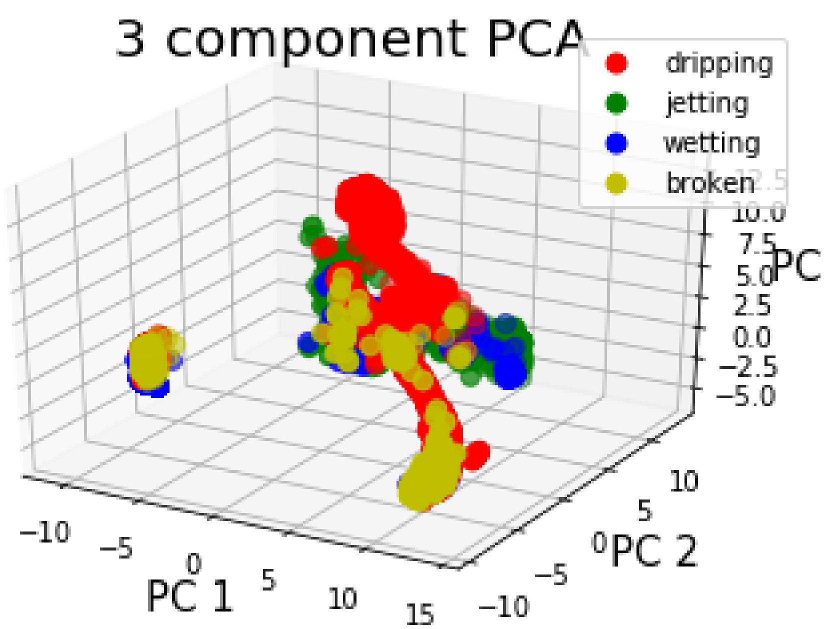
```
In [65]: from sklearn.decomposition import PCA
         pca = PCA(n_components=2)
         principalComponents = pca.fit_transform(im_data)

         fig = plt.figure(figsize = (8,8))
         ax = fig.add_subplot(1,1,1)
         ax.set_xlabel('Principal Component 1', fontsize = 15)
         ax.set_ylabel('Principal Component 2', fontsize = 15)
         ax.set_title('2 component PCA', fontsize = 20)
         targets = [0,1,2,3,4,5,6,7,8,9,10]
```

```
states = ['dripp', 'drip_oct', 'ex_jet', 'jet', 'jet_oct', 'ex_wet', 'wet', 'wet_oct',
colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown',

for target, color, state in zip(targets,colors,states):
    ##indicesToKeep = finalDf['target'] == target
    ind = im_lb10 == target
    ax.scatter(principalComponents[ind,0],#finalDf.loc[indicesToKeep, 'principal compon
               principalComponents[ind,1],#, finalDf.loc[indicesToKeep, 'principal comp
               c = color,
               s = 50)
ax.legend(states)
ax.grid()
```



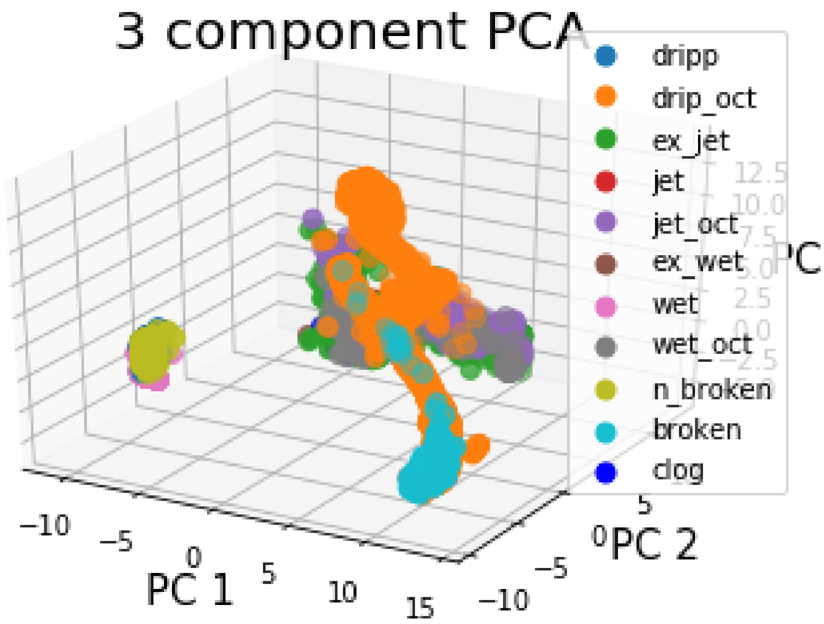
7 3 component PCA 4 Classes

```
In [66]: from sklearn.decomposition import PCA
         pca = PCA(n_components=3)
         principalComponents = pca.fit_transform(im_data)

         #%matplotlib ipympl  # Interactive Mode, Delay due to data size
         %matplotlib inline
         from mpl_toolkits.mplot3d import Axes3D
         fig = plt.figure()
         ax = fig.add_subplot(111, projection='3d')
         ax.set_xlabel('PC 1', fontsize = 15)
         ax.set_ylabel('PC 2', fontsize = 15)
         ax.set_zlabel('PC 3', fontsize = 15)
         ax.set_title('3 component PCA', fontsize = 20)
         targets = [0,1,2,3]
         states = ['dripping', 'jetting', 'wetting', 'broken']
         colors = ['r', 'g', 'b', 'y']
         for target, color, state in zip(targets,colors,states):
             ##indicesToKeep = finalDf['target'] == target
             ind = im_lb4 == target
             ax.scatter(principalComponents[ind,0],#finalDf.loc[indicesToKeep, 'principal compon
                        principalComponents[ind,1],#, finalDf.loc[indicesToKeep, 'principal comp
                        principalComponents[ind,2],
                        c = color,
                        s = 50)
         ax.legend(states)
         ax.grid()
```



8 3 component PCA 10 Classes

In [67]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
principalComponents = pca.fit_transform(im_data)

#%matplotlib ipympl  # Interactive Mode, Delay due to data size
%matplotlib inline
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('PC 1', fontsize = 15)
ax.set_ylabel('PC 2', fontsize = 15)
ax.set_zlabel('PC 3', fontsize = 15)
ax.set_title('3 component PCA', fontsize = 20)

targets = [0,1,2,3,4,5,6,7,8,9,10]
states = ['dripp', 'drip_oct', 'ex_jet', 'jet', 'jet_oct', 'ex_wet', 'wet', 'wet_oct',
colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown',

for target, color, state in zip(targets,colors,states):
    ##indicesToKeep = finalDf['target'] == target
    ind = im_lb10 == target
    ax.scatter(principalComponents[ind,0],#finalDf.loc[indicesToKeep, 'principal compon
               principalComponents[ind,1],#, finalDf.loc[indicesToKeep, 'principal comp
               principalComponents[ind,2],
               c = color,
               s = 50)
ax.legend(states)
ax.grid()
```



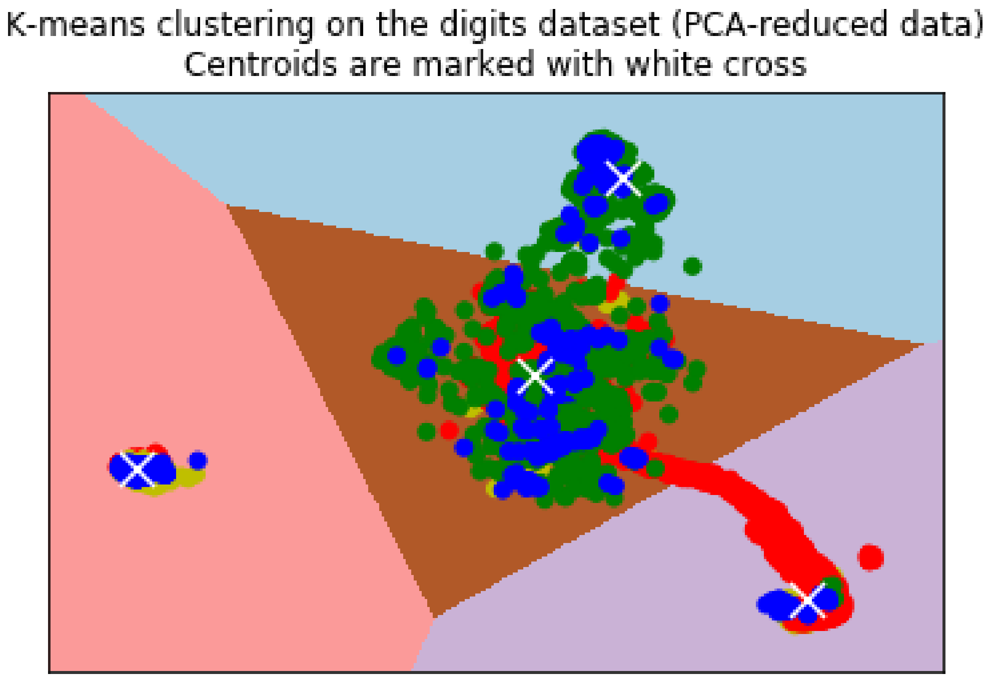
9 K Means on Flattened Daata

In [14]:
```
from sklearn.decomposition import PCA
from time import time

from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.preprocessing import scale


n_samples, n_features = im_data.shape
n_digits = len(np.unique(im_lb))
labels = im_lb

sample_size = 300

colors = ['r', 'g', 'b', 'y']
cb = [colors[i] for i in labels]

print("n_digits: %d, \t n_samples %d, \t n_features %d"
      % (n_digits, n_samples, n_features))


print(82 * '_')
print('init\t\ttime\tinertia\thomo\tcompl\tv-meas\tARI\tAMI\tsilhouette')
```

```python
def bench_k_means(estimator, name, data):
    t0 = time()
    estimator.fit(data)
    print('%-9s\t%.2fs\t%i\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f\t%.3f'
          % (name, (time() - t0), estimator.inertia_,
             metrics.homogeneity_score(labels, estimator.labels_),
             metrics.completeness_score(labels, estimator.labels_),
             metrics.v_measure_score(labels, estimator.labels_),
             metrics.adjusted_rand_score(labels, estimator.labels_),
             metrics.adjusted_mutual_info_score(labels,  estimator.labels_),
             metrics.silhouette_score(data, estimator.labels_,
                                      metric='euclidean',
                                      sample_size=sample_size)))

bench_k_means(KMeans(init='k-means++', n_clusters=n_digits, n_init=10),
              name="k-means++", data=im_data)

bench_k_means(KMeans(init='random', n_clusters=n_digits, n_init=10),
               name="random", data=im_data)

# in this case the seeding of the centers is deterministic, hence we run the
# kmeans algorithm only once with n_init=1
pca = PCA(n_components=n_digits).fit(im_data)
bench_k_means(KMeans(init=pca.components_, n_clusters=n_digits, n_init=1),
              name="PCA-based",
              data=im_data)
print(82 * '_')

# #############################################################################
# Visualize the results on PCA-reduced data

reduced_data = PCA(n_components=2).fit_transform(im_data)
kmeans = KMeans(init='k-means++', n_clusters=n_digits, n_init=10)
kmeans.fit(reduced_data)

# Step size of the mesh. Decrease to increase the quality of the VQ.
h = .02     # point in the mesh [x_min, x_max]x[y_min, y_max].

# Plot the decision boundary. For that, we will assign a color to each
x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Obtain labels for each point in mesh. Use last trained model.
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
```

```
Z = Z.reshape(xx.shape)
plt.figure(1)
plt.clf()
plt.imshow(Z, interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap=plt.cm.Paired,
           aspect='auto', origin='lower')

#plt.plot(reduced_data[:, 0], reduced_data[:, 1], 'k.', markersize=2)
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], color = cb)

# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1],
            marker='x', s=169, linewidths=3,
            color='w', zorder=10)
plt.title('K-means clustering on the digits dataset (PCA-reduced data)\n'
          'Centroids are marked with white cross')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

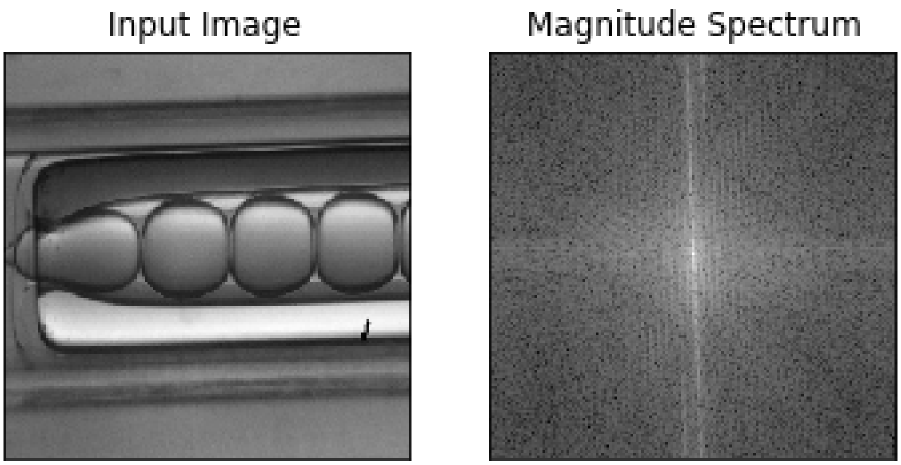
```
n_digits: 4,          n_samples 40346,         n_features 784
_________________________________________________________________________________
init                 time         inertia        homo         compl         v-meas          ARI
k-means++          18.10s         847213        0.391         0.274         0.322         0.185
random             21.74s         847213        0.391         0.274         0.322         0.185
PCA-based          1.81s         847213        0.391         0.274         0.322         0.185
_________________________________________________________________________________
```



10 PCA in Fourier Domain

In [68]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

In [34]:
```
img = cv2.imread('data/Resized_256_256xBroken_10.JPG',cv2.IMREAD_GRAYSCALE)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))

plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```



11 Process and Save IMGs in Fourier Domain

```
In [69]: totalf = []

         img_path = gb.glob("data/*.JPG")
         i = 0
         for path in img_path:
             pl = path.lower()

             img1 = cv2.imread(path, cv2.IMREAD_GRAYSCALE)#'data/Resized_256_256xBroken_0.JPG',
             if img1 is None:                     #img1
                         continue
             res1= cv2.resize(img1,(res,res))              #
             res1 = cv2.equalizeHist(res1)
             #res1_1 = res1.flatten()/255.0 #res1_1 = res1.reshape(1,784)/255.0       #
             #im_data = np.concatenate((im_data, res1_1))
             f = np.fft.fft2(res1)
             fshift = np.fft.fftshift(f)
             res1_1 = 20*np.ma.log(np.abs(fshift))
             res1_1_1 = res1_1.flatten().tolist()                    #numpy.narraylist
             totalf.append(res1_1_1)
```

12 Save Fourier Image

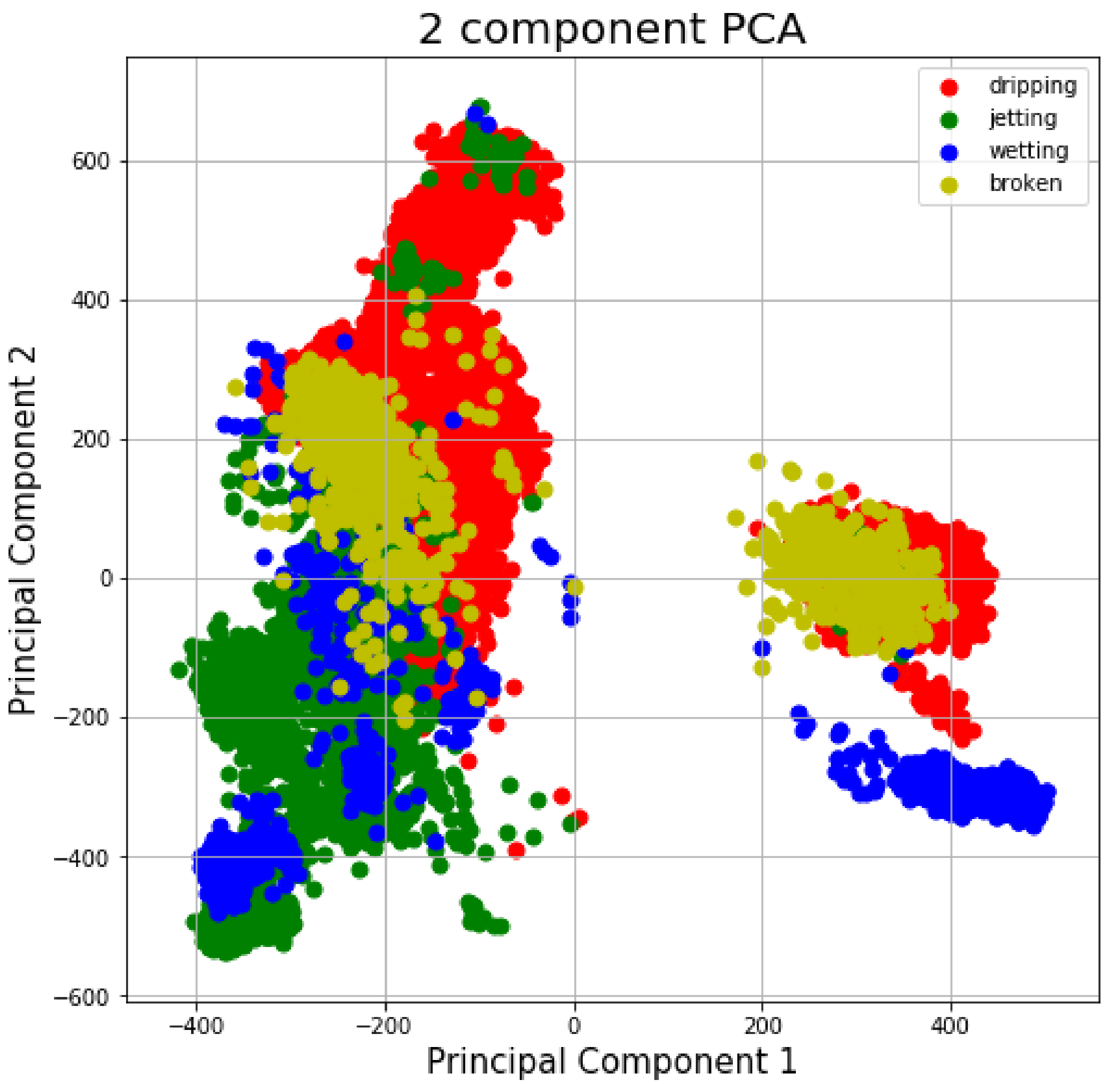
```
In [70]: im_data = np.array(totalf)
         im_dataf = im_data.astype(float)
         im_dataf.tofile('data/imgf'+str(res)+'.bin')
```

13 Load Fourier Image

In [84]:
```
# Load IMG
res2 = res*res
data = np.fromfile('data/imgf'+str(res)+'.bin',  dtype=np.float64)
im_dataf = data.reshape(-1, res2)

# Load Label
im_lb4 = np.fromfile('data/label4.bin', dtype=np.int64)
im_lb10 = np.fromfile('data/label10.bin', dtype=np.int64)

# Replacing NAN WITH AVG
im_dataf[np.isnan(im_dataf)] = np.nanmean(im_dataf)
#im_dataf[np.is(im_dataf)] = np.nanmean(im_dataf)
```

14 PCA 2 components on Fourier Domain 4 Classes

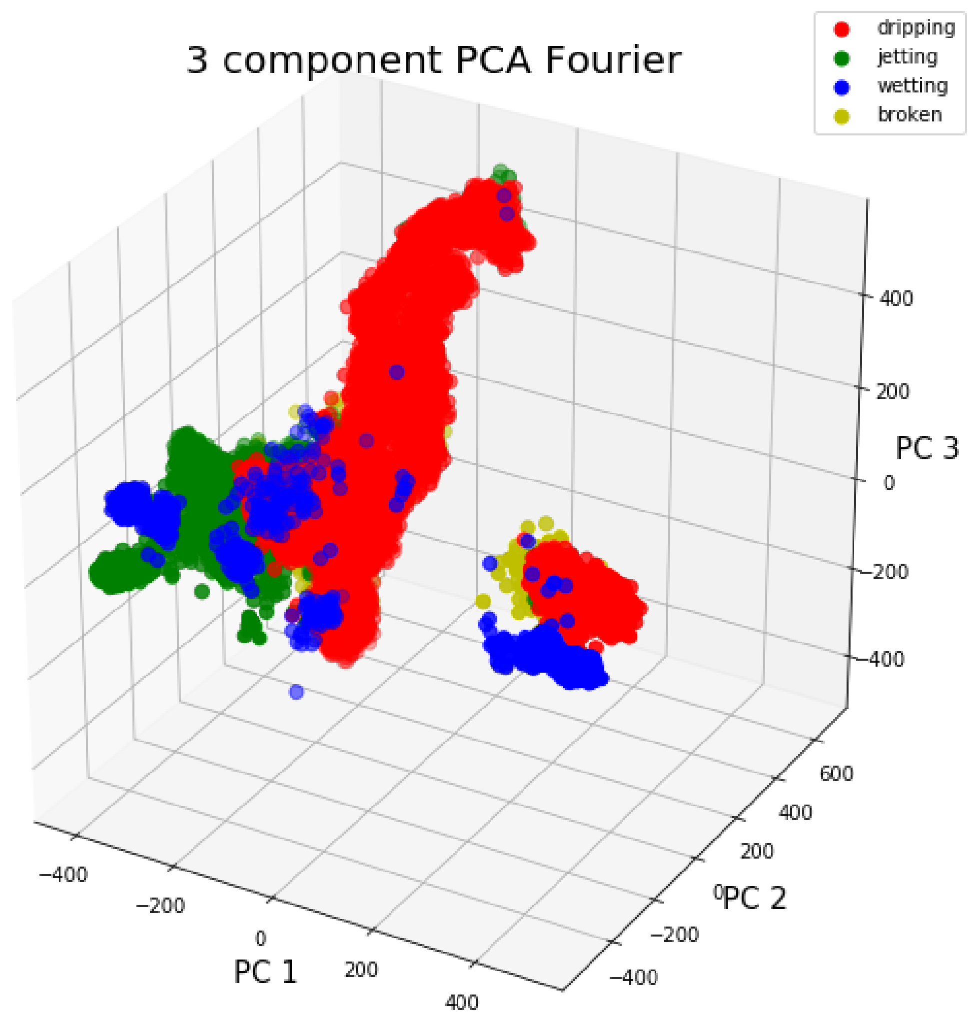
In [75]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(im_dataf)

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA Fourier', fontsize = 20)
targets = [0,1,2,3]
states = ['dripping', 'jetting', 'wetting', 'broken']
colors = ['r', 'g', 'b', 'y']
for target, color, state in zip(targets,colors,states):
    ##indicesToKeep = finalDf['target'] == target
    ind = im_lb4 == target
    ax.scatter(principalComponents[ind,0],#finalDf.loc[indicesToKeep, 'principal compon
               principalComponents[ind,1],#, finalDf.loc[indicesToKeep, 'principal comp
               c = color,
               s = 50)
ax.legend(states)
ax.grid()
```



15 PCA 3 components on Fourier Domain 4 Classes

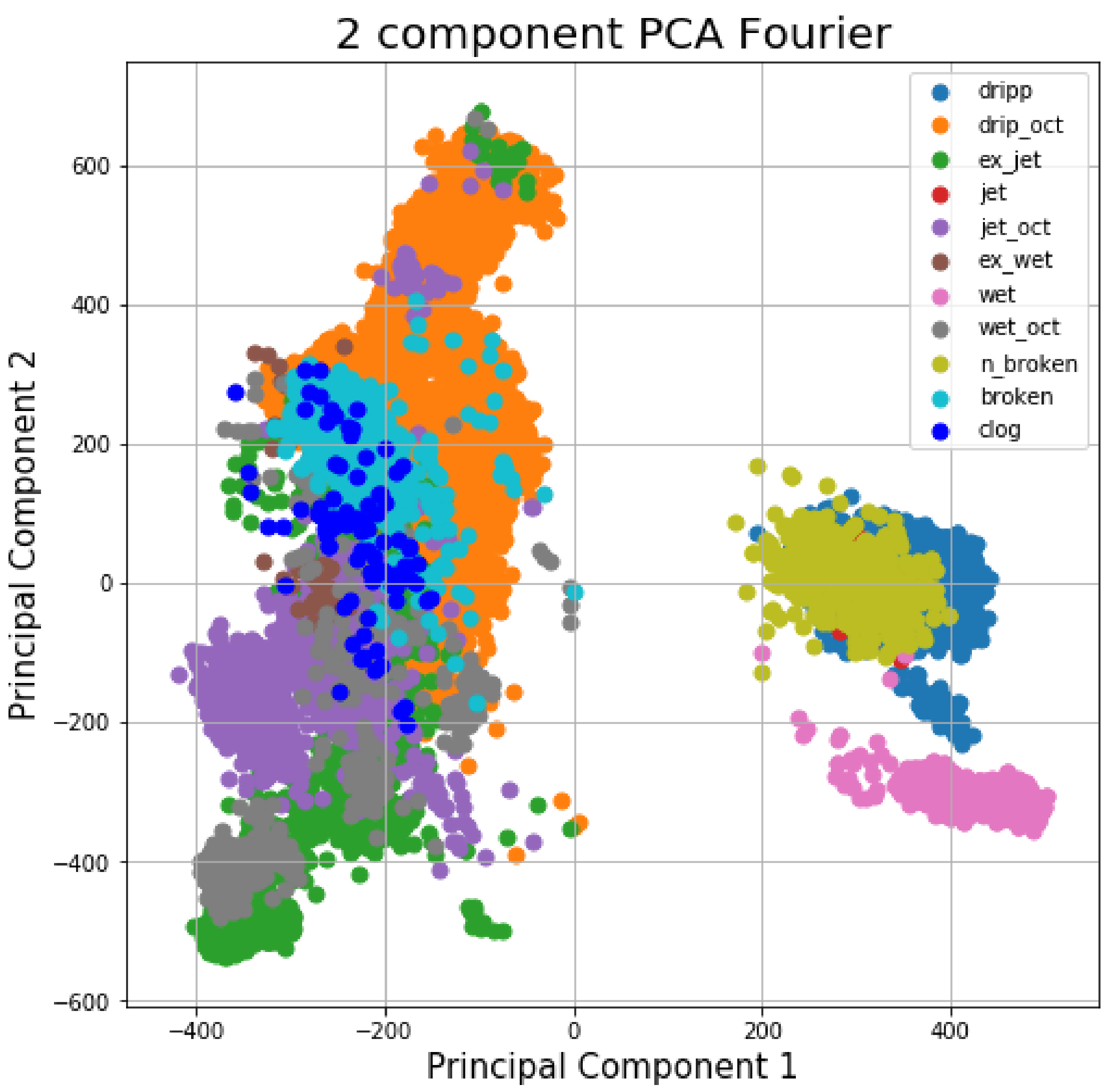
```
In [77]: from sklearn.decomposition import PCA
         pca = PCA(n_components=3)
         principalComponents = pca.fit_transform(im_dataf)


         #%matplotlib ipympl
         %matplotlib inline
         from mpl_toolkits.mplot3d import Axes3D
         fig = plt.figure(figsize = (10,10))
         ax = fig.add_subplot(111, projection='3d')
         ax.set_xlabel('PC 1', fontsize = 15)
```

```
ax.set_ylabel('PC 2', fontsize = 15)
ax.set_zlabel('PC 3', fontsize = 15)
ax.set_title('3 component PCA Fourier', fontsize = 20)
targets = [0,1,2,3]
states = ['dripping', 'jetting', 'wetting', 'broken']
colors = ['r', 'g', 'b', 'y']
for target, color, state in zip(targets,colors,states):
    ##indicesToKeep = finalDf['target'] == target
    ind = im_lb4 == target
    ax.scatter(principalComponents[ind,0],#finalDf.loc[indicesToKeep, 'principal compon
               principalComponents[ind,1],#, finalDf.loc[indicesToKeep, 'principal comp
               principalComponents[ind,2],
               c = color,
               s = 50)
ax.legend(states)
ax.grid()
```



16 PCA 2 components on Fourier Domain 10 Classes

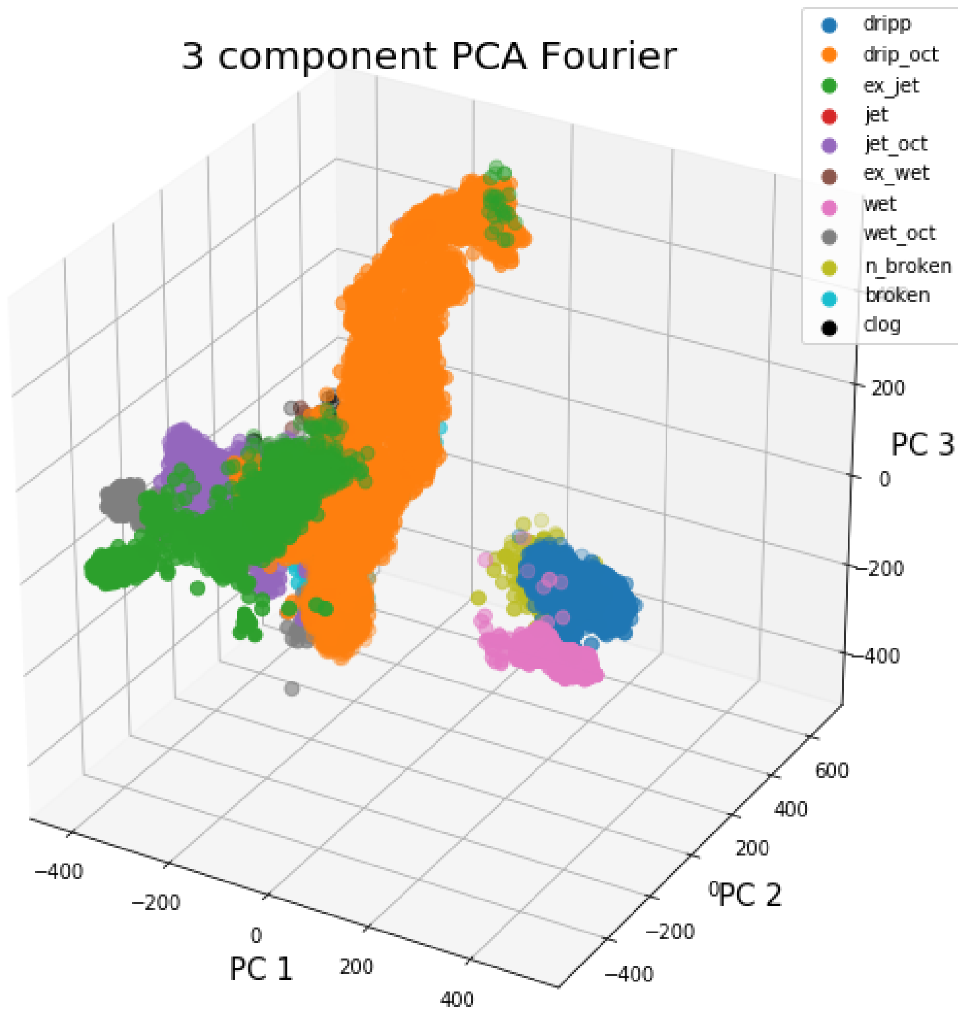
In [78]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(im_dataf)

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA Fourier', fontsize = 20)

targets = [0,1,2,3,4,5,6,7,8,9,10]
states = ['dripp', 'drip_oct', 'ex_jet', 'jet', 'jet_oct', 'ex_wet', 'wet', 'wet_oct',
colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown',

for target, color, state in zip(targets,colors,states):
    ##indicesToKeep = finalDf['target'] == target
    ind = im_lb10 == target
    ax.scatter(principalComponents[ind,0],#finalDf.loc[indicesToKeep, 'principal compon
               principalComponents[ind,1],#, finalDf.loc[indicesToKeep, 'principal comp
               c = color,
               s = 50)
ax.legend(states)
ax.grid()
```



17 PCA 3 components on Fourier Domain 10 Classes Size 64

In [79]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=3)
principalComponents = pca.fit_transform(im_dataf)


#%matplotlib ipympl
%matplotlib inline
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize = (10,10))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('PC 1', fontsize = 15)
```

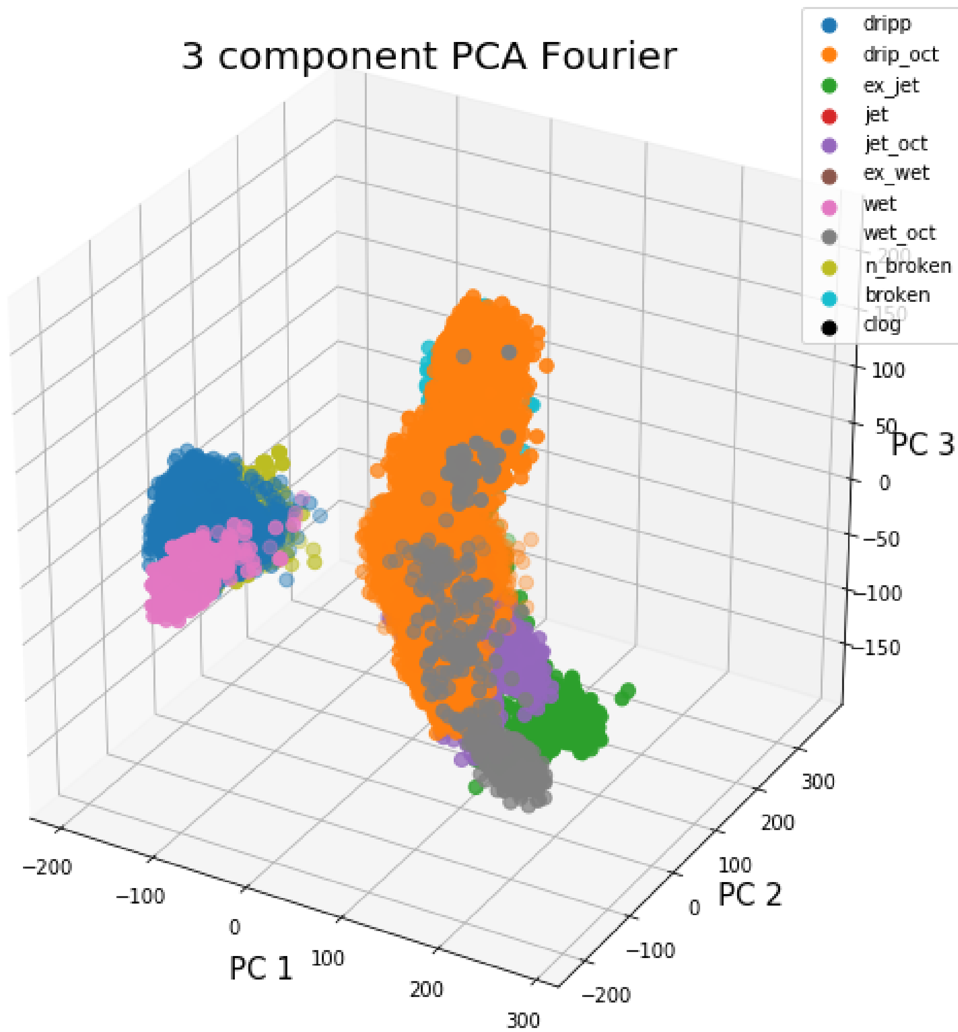
```python
ax.set_ylabel('PC 2', fontsize = 15)
ax.set_zlabel('PC 3', fontsize = 15)
ax.set_title('3 component PCA Fourier', fontsize = 20)

targets = [0,1,2,3,4,5,6,7,8,9,10]
states = ['dripp', 'drip_oct', 'ex_jet', 'jet', 'jet_oct', 'ex_wet', 'wet', 'wet_oct',
colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown',

for target, color, state in zip(targets,colors,states):
    ##indicesToKeep = finalDf['target'] == target
    ind = im_lb10 == target
    ax.scatter(principalComponents[ind,0],#finalDf.loc[indicesToKeep, 'principal compon
               principalComponents[ind,1],#, finalDf.loc[indicesToKeep, 'principal comp
               principalComponents[ind,2],
               c = color,
               s = 50)
ax.legend(states)
ax.grid()
```



18 PCA 3 components on Fourier Domain 10 Classes Size 28

```
In [86]: # 28
         from sklearn.decomposition import PCA
         pca = PCA(n_components=3)
         principalComponents = pca.fit_transform(im_dataf)


         #%matplotlib ipympl
         %matplotlib inline
         from mpl_toolkits.mplot3d import Axes3D
         fig = plt.figure(figsize = (10,10))
         ax = fig.add_subplot(111, projection='3d')
```

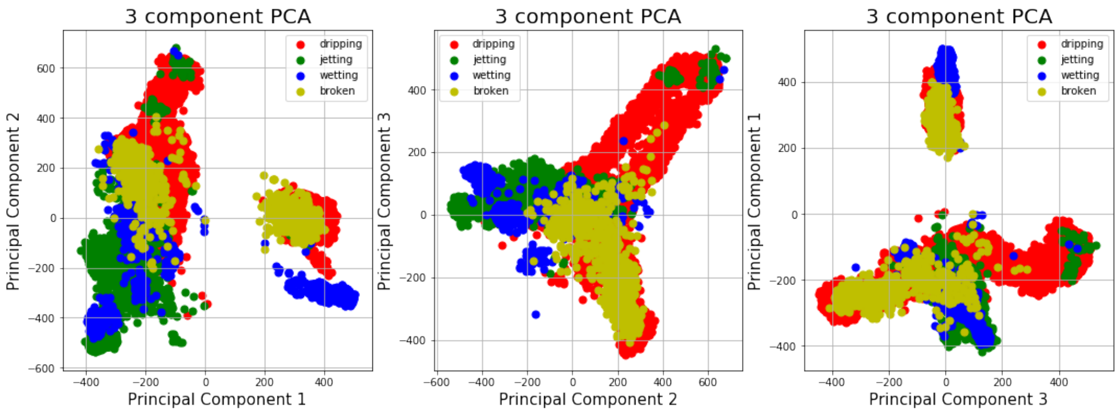
```
ax.set_xlabel('PC 1', fontsize = 15)
ax.set_ylabel('PC 2', fontsize = 15)
ax.set_zlabel('PC 3', fontsize = 15)
ax.set_title('3 component PCA Fourier', fontsize = 20)

targets = [0,1,2,3,4,5,6,7,8,9,10]
states = ['dripp', 'drip_oct', 'ex_jet', 'jet', 'jet_oct', 'ex_wet', 'wet', 'wet_oct',
colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown',

for target, color, state in zip(targets,colors,states):
    ##indicesToKeep = finalDf['target'] == target
    ind = im_lb10 == target
    ax.scatter(principalComponents[ind,0],#finalDf.loc[indicesToKeep, 'principal compon
               principalComponents[ind,1],#, finalDf.loc[indicesToKeep, 'principal comp
               principalComponents[ind,2],
               c = color,
               s = 50)
ax.legend(states)
ax.grid()
```



19 PCA 3 components 3 Views 4 Classes Fourier Size 64

```
In [80]: from sklearn.decomposition import PCA
         pca = PCA(n_components=3)
         principalComponents = pca.fit_transform(im_dataf)

         fig = plt.figure(figsize = (18,6))
         for i in range(3):
             ax = fig.add_subplot(1,3,i+1)

             ax.set_xlabel('Principal Component ' + str(i%3+1), fontsize = 15)
             ax.set_ylabel('Principal Component ' +str((i+1)%3+1), fontsize = 15)
```

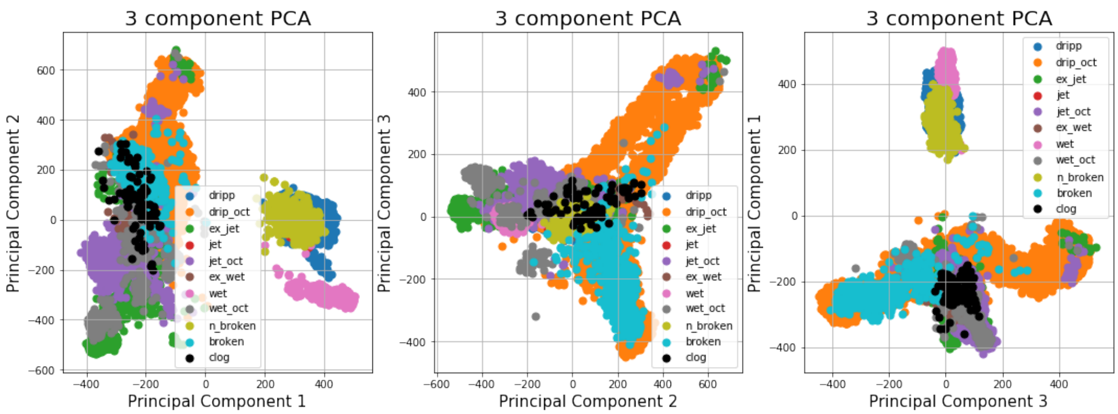
```
    ax.set_title('3 component PCA', fontsize = 20)
    targets = [0,1,2,3]
    states = ['dripping', 'jetting', 'wetting', 'broken']
    colors = ['r', 'g', 'b', 'y']
    for target, color, state in zip(targets,colors,states):

        ind = im_lb4 == target
        ax.scatter(principalComponents[ind,i%3],#finalDf.loc[indicesToKeep, 'principal
                   principalComponents[ind,(i+1)%3],#, finalDf.loc[indicesToKeep, 'prin
                   c = color,
                   s = 50)
    ax.legend(states)
    ax.grid()
```



20 PCA 3 components 3 Views 10 Classes Fourier Size 64

```
In [81]: from sklearn.decomposition import PCA
         pca = PCA(n_components=3)
         principalComponents = pca.fit_transform(im_dataf)

         fig = plt.figure(figsize = (18,6))
         for i in range(3):
             ax = fig.add_subplot(1,3,i+1)

             ax.set_xlabel('Principal Component ' + str(i%3+1), fontsize = 15)
             ax.set_ylabel('Principal Component ' +str((i+1)%3+1), fontsize = 15)

             ax.set_title('3 component PCA', fontsize = 20)

             targets = [0,1,2,3,4,5,6,7,8,9,10]
             states = ['dripp', 'drip_oct', 'ex_jet', 'jet', 'jet_oct', 'ex_wet', 'wet', 'wet_oc
             colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brow
```

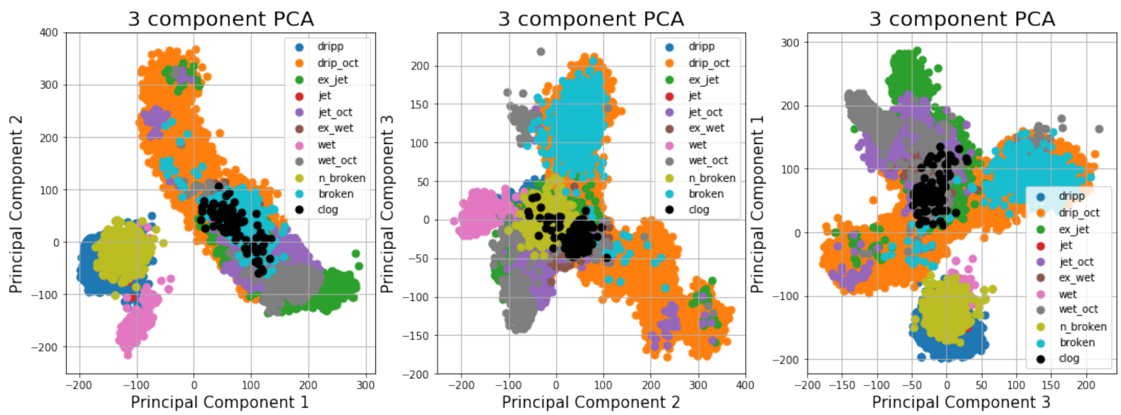
```
    for target, color, state in zip(targets,colors,states):

        ind = im_lb10 == target
        ax.scatter(principalComponents[ind,i%3],#finalDf.loc[indicesToKeep, 'principal
                   principalComponents[ind,(i+1)%3],#, finalDf.loc[indicesToKeep, 'prin
                   c = color,
                   s = 50)
    ax.legend(states)
    ax.grid()
```



21 PCA 3 components on Fourier Domain 10 Classes Size 28

```
In [85]: # 28 times 28

         from sklearn.decomposition import PCA
         pca = PCA(n_components=3)
         principalComponents = pca.fit_transform(im_dataf)

         fig = plt.figure(figsize = (18,6))
         for i in range(3):
             ax = fig.add_subplot(1,3,i+1)

             ax.set_xlabel('Principal Component ' + str(i%3+1), fontsize = 15)
             ax.set_ylabel('Principal Component ' +str((i+1)%3+1), fontsize = 15)

             ax.set_title('3 component PCA', fontsize = 20)

             targets = [0,1,2,3,4,5,6,7,8,9,10]
             states = ['dripp', 'drip_oct', 'ex_jet', 'jet', 'jet_oct', 'ex_wet', 'wet', 'wet_oc
             colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brow
```

```
for target, color, state in zip(targets,colors,states):

    ind = im_lb10 == target
    ax.scatter(principalComponents[ind,i%3],#finalDf.loc[indicesToKeep, 'principal
               principalComponents[ind,(i+1)%3],#, finalDf.loc[indicesToKeep, 'prin
               c = color,
               s = 50)
ax.legend(states)
ax.grid()
```



22 To test, different preprocessing/ rotation blur/ normalization

23 To test, which feature each cluster stands for

24 To test, TDA