In [2]:

```
import imagehash
import pandas as pd
import numpy as np
import requests
from PIL import Image
import os
from imagededup.methods import PHash

from imagededup.methods import CNN
from imagededup.utils import plot_duplicates
import csv

import matplotlib.pyplot as plt
import networkx as nx
import scipy.linalg as la
import math
```

MemeRank Check In

So our project is designed to determine as a more organic method of determining a subreddit rank based on meme content shared with other subreddits. The subreddit which contains the most memes with formats found in other subreddits will float to the top of our ranking system.

In [5]:

```
Memes = pd.read_csv("MemeDatabase.csv", encoding='UTF-8',skip_blank_lines=False)
```

In [6]:

```
column = Memes["Subreddit"]
thing=Memes.Subreddit.unique()
print(len(thing))
```

```
76
```

So we have 76 unique Subreddits that we are pulling data from, from this it seems fair that we should be able to use a matrix for the Pagerank model.

In [9]:

```
cleanMemes=Memes.dropna()
condition = cleanMemes["Imagelink"].str.contains(".jpg")
cleanerMemes = cleanMemes[condition]
```

Many of the memes are NA or gif so we went ahead and removed those from the database to clean the dataset.

In [ ]:

```
tol=20
for i,m in enumerate(cleanerMemes):
    image_data = Image.open(requests.get("https://memegenerator.net/img/instances/31131375/spiderman-is-doomed.jpg", stream=True).raw)
    hashed1 = imagehash.phash(image_data)
    image_data = Image.open(requests.get("https://img.tineye.com/result/595d5d88b1f875202380301704d2155b6aeca25d13306f3a462982ed3e856837?size=160", stream=True).raw)
    hashed2 = imagehash.phash(image_data)
```

In [29]:

```
image_data = Image.open(requests.get("https://memegenerator.net/img/instances/31131375/spiderman-is-doomed.jpg", stream=True).raw)
```

```
hashed1 = imagehash.phash(image_data)
image_data = Image.open(requests.get("https://img.tineye.com/result/595d5d88b1f875202380301704d2155b6aeca25d13306f3a462982ed3e856837?size=160", stream=True).raw)
hashed2 = imagehash.phash(image_data)
print(hashed2-hashed1)
```

```
10
```

We should use less than between 10 and 20 for successful comparisons, anything over 25 is definitely not a match.

In [11]:

```
tol=15
image_base = Image.open(requests.get(cleanerMemes['Imagelink'][1], stream=True).raw)
hashBase=imagehash.phash(image_base)
matches=[]
for index, row in cleanerMemes.iterrows():
    if index>10:
        break
    try:
        image_data = Image.open(requests.get(row['Imagelink'], stream=True).raw)
        image_data.save(os.path.join('/Users/Sid/Documents/FestoProject/MemeProject/Images', row['Imagelink']), quality=85)
    except:
        continue
    hashed1 = imagehash.phash(image_data)
    if (hashBase-hashed1<tol) :
        matches.append(row['Imagelink'])
```

In [12]:

```
for index, row in cleanerMemes.iterrows():
    if index<50000:
        continue
    if index>50000:
        break
    try:
        image_data = Image.open(requests.get(row['Imagelink'], stream=True).raw)
        image_data.save(os.path.join('/Users/Sid/Documents/FestoProject/MemeProject/Images', 'image'+str(index)+'.jpg'), quality=85)
    except:
        continue
```

First we ripped the entire image set from Reddit so that we could run the CNN over the image data locally.

In [7]:

```
image_data = Image.open(requests.get("https://memegenerator.net/img/instances/31131375/spiderman-is-doomed.jpg", stream=True).raw)
image_data.save(os.path.join('/Users/Sid/Documents/FestoProject/MemeProject/Images', 'image.jpg'), quality=85)
```

In [ ]:

```
phasher = CNN()

# Generate encodings for all images in an image directory
encodings = phasher.encode_images(image_dir='/Users/Sid/Documents/FestoProject/MemeProject/Images')

# Find duplicates using the generated encodings
duplicates = phasher.find_duplicates(encoding_map=encodings)

# plot duplicates obtained for a given file using the duplicates dictionary
```

In [9]:

```
plot_duplicates(image_dir='/Users/Sid/Documents/FestoProject/MemeProject/Images',
```

```
                duplicate_map=duplicates,
                filename='image1.jpg')
```

```
/Users/Sid/opt/anaconda3/lib/python3.8/site-packages/imagededup/utils/plotter.py:66: Matp
lotlibDeprecationWarning: savefig() got unexpected keyword argument "dpi" which is no lon
ger supported as of 3.3 and will become an error two minor releases later
  gs.tight_layout(fig)
```

So it looks like our CNN worked, now we'll export our duplicates dictionary as a link table in csv format so that we can plug it into our modified PageRank algorithm.

In [13]:

```
cleanDict=duplicates.copy()
for key in duplicates:
    if len(duplicates[key])>0:
        for k in duplicates[key]:
            try:
                if cleanDict[key] is not None:
                    cleanDict.pop(k,None)
            except:
                i=1
print(len(cleanDict))
print(len(duplicates))
```

```
23800
27778
```

In [12]:

```
with open('memeProject.csv', 'w', newline='') as csvfile:
    spamwriter = csv.writer(csvfile, delimiter=',',quotechar='|', quoting=csv.QUOTE_MINI
MAL)
    for key in cleanDict:
        if len(cleanDict[key])>0:
            for k in cleanDict[key]:
                kk=k.replace('.jpg', '')
                kk=kk.replace('image', '')
                kkk=key.replace('.jpg', '')
                kkk=kkk.replace('image', '')
                try:
                    output1=cleanerMemes[cleanerMemes.index==int(kk)]["Subreddit"].value
s[0]
                    output2=cleanerMemes[cleanerMemes.index==int(kkk)]["Subreddit"].valu
es[0]
                    spamwriter.writerow([output1, output2])
                except:
                    print(kk)
```

In [75]:

```
maxD=0
outKey=0
for key in cleanDict:
    if len(cleanDict[key])>0:
        if len(cleanDict[key])>maxD:
```

```
            maxD=len(cleanDict[key])
            print(key)
            outKey=key
print(maxD)
print(outKey)
```

```
image10055.jpg
image1011.jpg
image10281.jpg
image10737.jpg
282
image10737.jpg
```

In [61]:

```
print(len(duplicates["image10737.jpg"]))
```

```
282
```

In [ ]:

```
for key in sorted(cleanDict, key=lambda k: len(cleanDict[k]), reverse=True):
    print(key)
```

In [ ]:

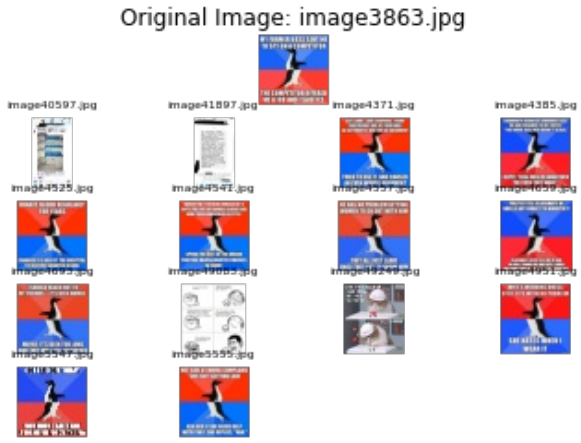
```
cleanDict.pop("image10737.jpg",None)
```

In [42]:

```
res=list(sorted(duplicates, key=lambda k: len(duplicates[k]), reverse=True))
```

In [81]:

```
plot_duplicates(image_dir='/Users/Sid/Documents/FestoProject/MemeProject/Images',
                duplicate_map=duplicates,
                filename="image3863.jpg")
```



So first we removed duplicate entries of reduced meme template formats to a single entry containing all of the matched memes like shown in the image above. We can see that there is some slight mismatch but for the most part the memes above are all the same format. We then sorted them based on the number of duplicates in each format.

Weighted page rank

So now that we've effectively extended our data so now we have clusterings of each of the parameters of the original data set we will use our csv generated above and plug that into our modified page rank algorithm below.

In [14]:

```
dfMemes = pd.read_csv("Database.csv", encoding='UTF-8',skip_blank_lines=False)
```

```
subredditIndices = {}
index = 0
for subreddit in dfMemes["Subreddit"].unique():
    subredditIndices[subreddit] = index
    subredditIndices[index] = subreddit
    index += 1
```

In [15]:

```
dfLinks = pd.read_csv('memeProject.csv', header=None, encoding='utf8')
dfLinks.columns = ['i','j']

adjacencyList = [[] for n in range(index)]

for _, row in dfLinks.iterrows():
    subredditI = row["i"]
    subredditJ = row['j']
    i = subredditIndices[subredditI]
    j = subredditIndices[subredditJ]

    #links are bidirectional
    #multiple links in memeProject.csv are added multiple times so they are counted later
    adjacencyList[j].append(i)
    adjacencyList[i].append(j)
```

In [16]:

```
from matplotlib.pyplot import figure

pd.set_option('display.max_rows', 500)

def stochasticMatrix(adjacency_list):
    n = len(adjacency_list)
    matrix = np.zeros((n, n))
    for col, line in enumerate(adjacency_list):
        if len(line) > 0:
              #page has existing links
            for index in line:

                matrix[index][col] += 1/(len(line)) #add a portion of the final weight
        else:
            #page not found
            for index in range(n):
                matrix[index][col] = 1/n
    return matrix

#transitionMatrix = (1-beta)/n * m + beta*stochasitcMatrix
def transitionMatrix(stochastic_matrix):
    n = len(stochastic_matrix)
    beta = 0.85 #teleportation factor
    m = np.ones((n, n)) #matrix of 1's
    part1 = np.multiply(((1-beta)/n), m)
    part2 = np.multiply(beta, stochastic_matrix)
    transition_matrix = np.add(part1, part2)
    return transition_matrix

def pageRank(transition_matrix):
    n = len(transition_matrix)
    err = 0.0005
    v1 = np.ones(n)
    v1 = np.multiply((1/n), v1)
    v2 = np.matmul(transition_matrix, v1)

    while not within_err(v1, v2, err):
        #ensures differences between v1 and v2 is under the err bound
        v1 = v2
        v2 = np.matmul(transition_matrix, v1)
    return (v2.tolist())

def within_err(v1, v2, err):
```

```
    diff_vector = np.subtract(v2, v1)
    for diff in diff_vector:
        if abs(diff) > err:
            return False
    return True


m = stochasticMatrix(adjacencyList)


t = transitionMatrix(m)


p = pageRank(t)


N = 10
#Return index values(company ids) for the 10 highest page ranks
topSubredditID = sorted(range(len(p)), key = lambda sub: p[sub])[-N:]

topSubreddits = [subredditIndices[subredditID] for subredditID in topSubredditID]
```

In [17]:

```
print(topSubreddits)
```

```
['gymmemes', 'badmemes', 'bee_irl', 'me_irl', 'christianmemes', 'kenm', 'bonehurtingjuice
', 'adviceanimals', '2meirl4meirl', 'blackpeopletwitter']
```

So now we have our top ten memeist subreddits. A quick visit to any of these subreddits will confirm that they have an abundance of memes (perhaps to no ones surprise). One interesting thing however is that these subreddits are not necessarily the most popular based on traditional metrics such as number of users or upvotes. That being said it is well known that these statistics (users and upvotes) can be easily gamed on any social network, so this suggests that our algorithm worked for more organically ranking subreddits.

In [ ]:

```

```