Name: Tshenolo Daumas

Student Number: 19643128

In [1]:

```
%matplotlib inline
%load_ext autoreload
%autoreload 2
```

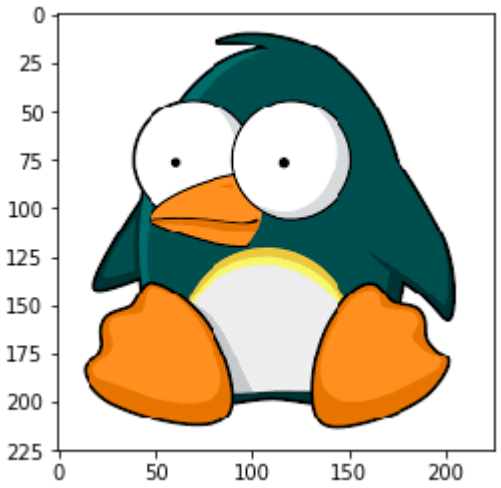
In [2]:

```
# Import different modules for using with the notebook
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from IPython.display import HTML
from IPython.display import display
from IPython.display import Image

from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.metrics import confusion_matrix
from sklearn.cluster import KMeans
from sklearn.mixture import GMM

from numpy.random import randint
from kmeans_resource import Kmeans as KM
from skimage import io
from ipywidgets import interact
```

Perform k-means on an image

In [3]:

```
im = io.imread('./cartoon.png')
plt.imshow(im/np.max(im))
plt.show()
```



Clustering of colors

Note that the colors in the image above are clustered around only a few colors. **You need to find the clusters using the k-means algorithm in scikit-learn. Reproducing the image using only 3 colors gives something like the following image.**

How many clusters do you need for a perfect reproduction?

Note: Before you produce the plot using plt.imshow(), make sure the image data is op type "unint8", i.e. set dtype = "uint8"

Mark: 1

In [4]:

```
r,c,d= np.shape(im)
im_flat= im.reshape(r*c,d)
```

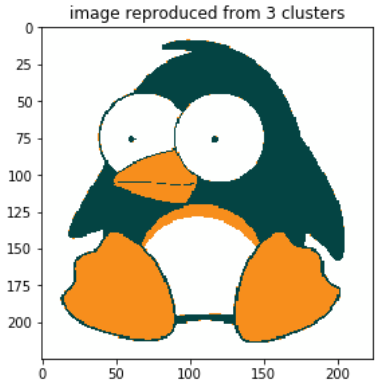
In [5]:

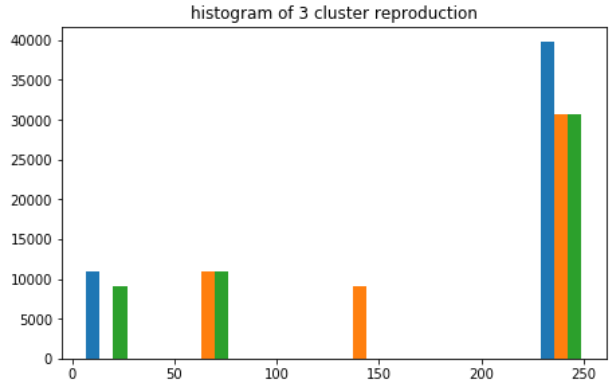
```
kmeans= KMeans(n_clusters= 3, random_state= 42)
kmeans= kmeans.fit(im_flat)
centers= kmeans.cluster_centers_
print(all(kmeans.labels_ == kmeans.predict(im_flat)))
```

```
True
```

In [6]:

```
im_rep_flat= np.empty(np.shape(im_flat))
for i in range(r*c):
    im_rep_flat[i,:]= centers[kmeans.labels_[i],:]

im_rep= im_rep_flat.reshape(r,c,d)

plt.figure(figsize=(16, 10))

plt.subplot(221)
plt.imshow(im_rep/np.max(im_rep))
plt.title("image reproduced from 3 clusters")

plt.subplot(222)
plt.hist(im_rep_flat)
plt.title("histogram of 3 cluster reproduction")
plt.show()
```





In [7]:

```
clusters= [5,10,34,54]
#sp= [221,222,223,224]
for j in clusters:

    plt.figure(figsize=(16, 10))

    kmeans= KMeans(n_clusters= j, random_state= 42)
    kmeans= kmeans.fit(im_flat)
    centers= kmeans.cluster_centers_

    im_rep_flat= np.empty(np.shape(im_flat))

    for i in range(r*c):
        im_rep_flat[i,:]= centers[kmeans.labels_[i],:]

    im_rep= im_rep_flat.reshape(r,c,d)

    plt.subplot(221)
    plt.imshow(im_rep/np.max(im_rep))
    plt.title("image reproduced from {} clusters".format(j))

    plt.subplot(222)
    plt.hist(im_rep_flat)
    plt.title("histogram of {} cluster reproduction".format(j))
    plt.show()

plt.figure(figsize=(16, 10))
plt.subplot(221)
plt.imshow(im/np.max(im))
plt.title("original")

plt.subplot(222)
plt.hist(im_rep_flat)
plt.title("histogram of original image")
plt.show()
```

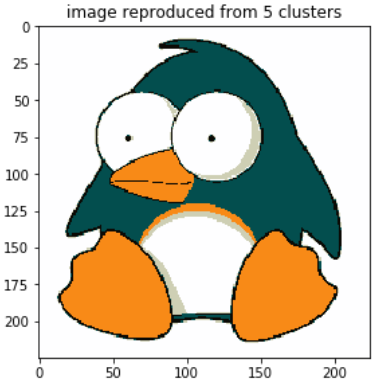
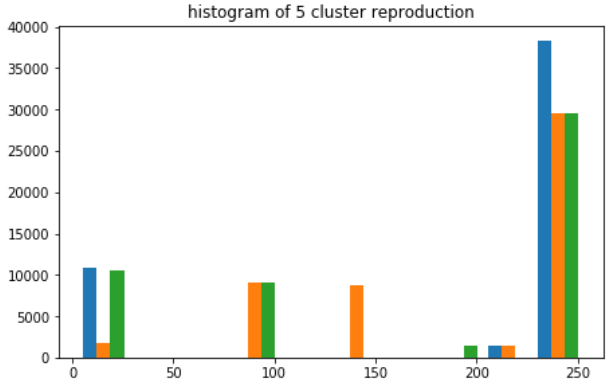


image reproduced from 5 clusters


histogram of 5 cluster reproduction

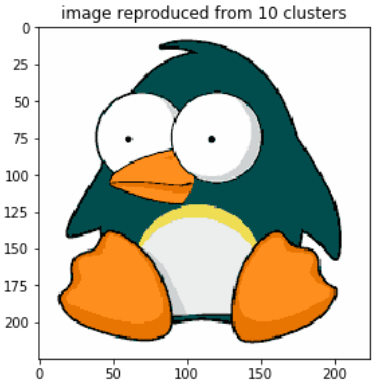
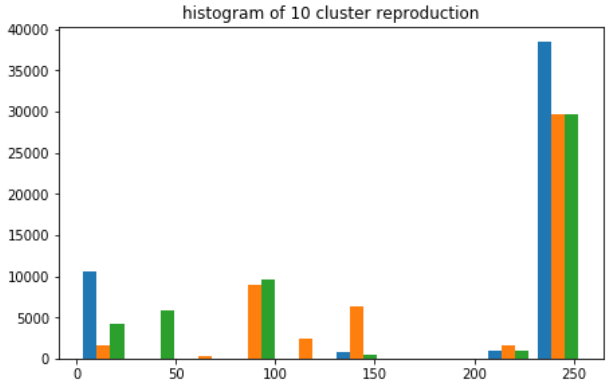


image reproduced from 10 clusters


histogram of 10 cluster reproduction

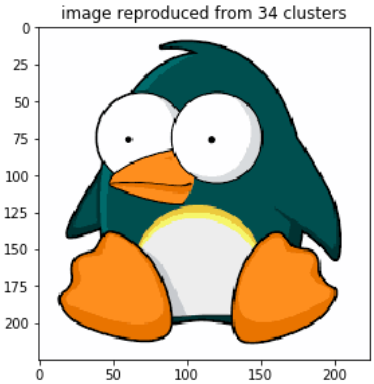
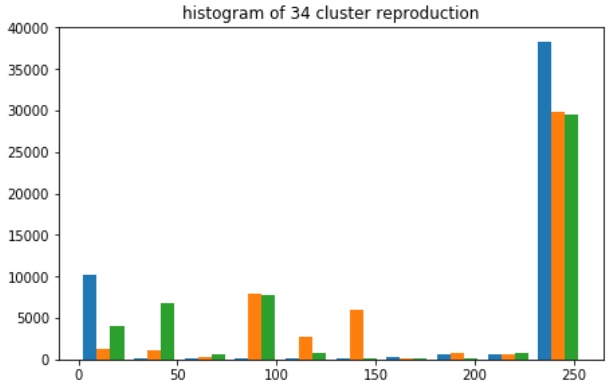


image reproduced from 34 clusters


histogram of 34 cluster reproduction

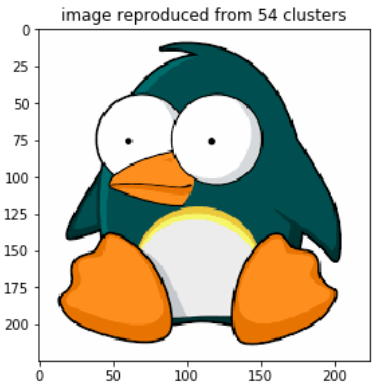
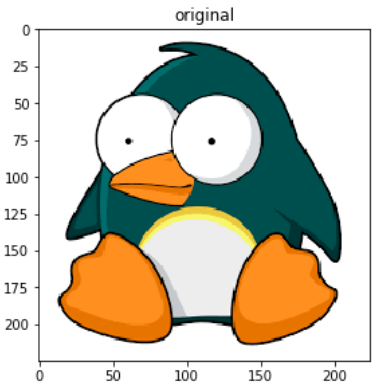
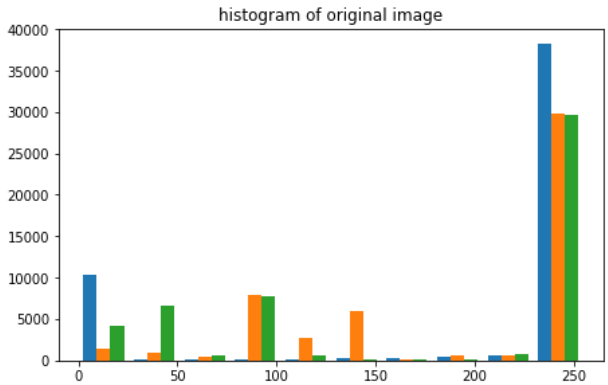


image reproduced from 54 clusters


histogram of 54 cluster reproduction


original


histogram of original image

In [8]:

```
print(np.all(np.uint8(im_rep)==im))
```

```
False
```

Interpretation

The histogram analysis shows that the higher the number of clusters, the closer the reproduced image is to the original image.
However, it also shows that it is unlikely that there is a number of clusters existent to give a "perfect reproduction". The histograms show even as high as 54 clusters, the proportions of the 3 components of the pixel values look visually identical but the reproduced image matrix is still not equal to the original image matrix.
Theoretically, we know that, there are 256 values for each of the 3 pixel components, therefore, there are 256^3 options for each pixel, thus 256^3 clusters required. Futhermore, upon investigation, not all 256 possible pixel values are used for any of the 3 pixel dimensions (components), therefore the lower bound of the number of clusters that are required would theoretically be the number of different combinations present in the image matrix along the dimension of the 3 pixel components. The upper bound being 256^3

Clustering of digits data

For the next problem you use k-means to cluster the digits data in the scikit-learn library.

Although the digits data comes with labels, for your first task you ignore the labels and just see how well k-means clusters the different digits.

Before you get to that, though, this is too good an opportunity to investigate the use of PCA and LDA on higher dimensional data!

Load the digit datasets

The datasets are loaded into a dictionary.

In [9]:

```
# Import `datasets` from `sklearn`
from sklearn import datasets

# Load in the `digits` data
digits = datasets.load_digits()
print(digits.keys())
```

```
dict_keys(['data', 'target', 'target_names', 'images', 'DESCR'])
```

In [10]:

```
# Find the number of unique labels
number_digits = len(np.unique(digits.target))

print (number_digits)
```

```
10
```

Inspect the different digit images

Project onto 2 PCA components

Use the scikit-learn module and project the digits data onto 2 principal components. Display the clusters in different colors using the known labels.

Marks: 1

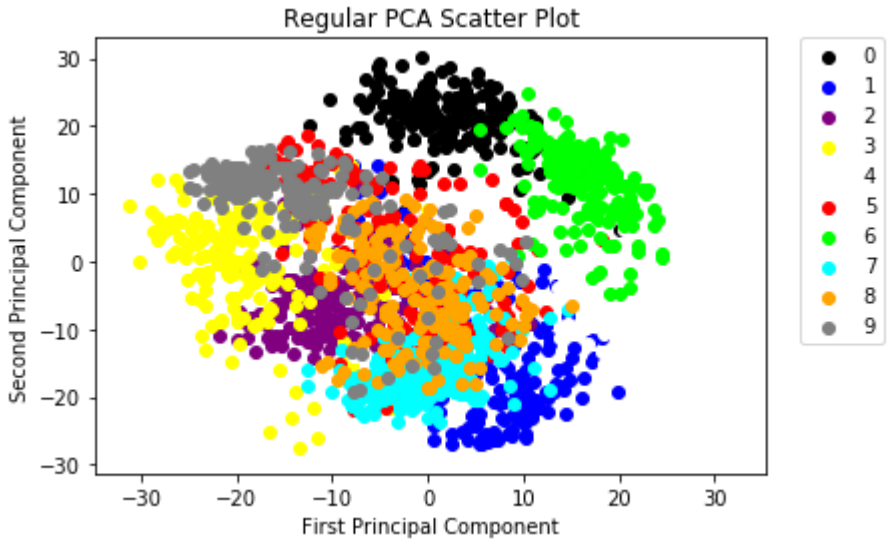
In [11]:

```
# Create a regular PCA model
# insert code
data= digits.data
pca= PCA(n_components=2)

# Fit and transform the data to the model
# insert code
reduced_data_pca = pca.fit(data).transform(data)
```

Using the code in the following code cell (it will not run as is because you need to calculate reduced_data_pca in the code cell above), to find the following

In [12]:

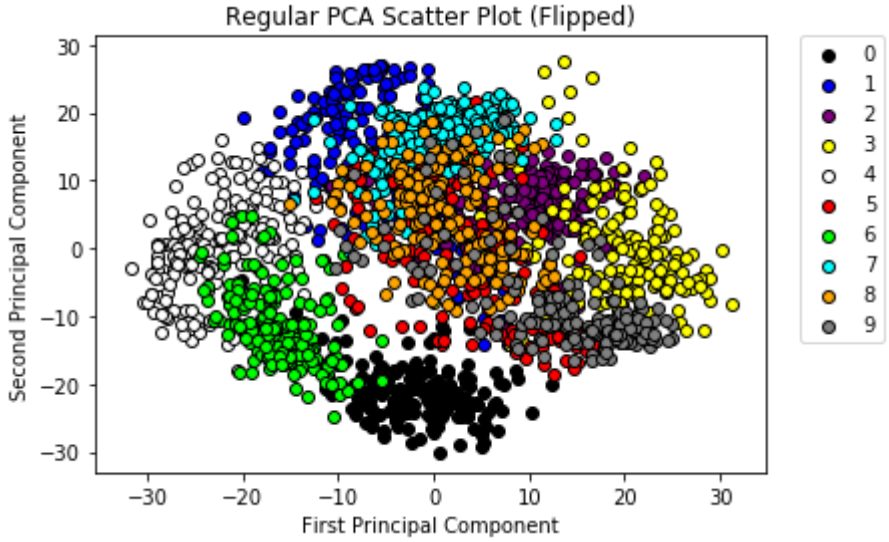
```
# This code will not execute because you have to provide data
colors = ['black', 'blue', 'purple', 'yellow', 'white', 'red', 'lime', 'cyan',
'orange', 'gray']
for i in range(len(colors)):
    x = reduced_data_pca[:, 0][digits.target == i]
    y = reduced_data_pca[:, 1][digits.target == i]
    plt.scatter(x, y, c=colors[i])
plt.legend(digits.target_names, bbox_to_anchor=(1.05, 1), loc=2,
borderaxespad=0.)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title("Regular PCA Scatter Plot")
plt.show()
```



Plot is reflected along both the x- and y- axes. I will need to flip it to make the plot right side up. Also, data points do not have edges, therefore (4) which is white, is not visible due to the white background, edgecolor= 'black' needs to be added to the plt.scatter() command.

In [13]:

```
# This code will not execute because you have to provide data
colors = ['black', 'blue', 'purple', 'yellow', 'white', 'red', 'lime', 'cyan',
'orange', 'gray']
for i in range(len(colors)):
    x = -reduced_data_pca[:, 0][digits.target == i]
    y = -reduced_data_pca[:, 1][digits.target == i]
    plt.scatter(x, y, c=colors[i], edgecolors= 'black')
plt.legend(digits.target_names, bbox_to_anchor=(1.05, 1), loc=2,
borderaxespad=0.)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title("Regular PCA Scatter Plot (Flipped)")
plt.show()
```



LDA

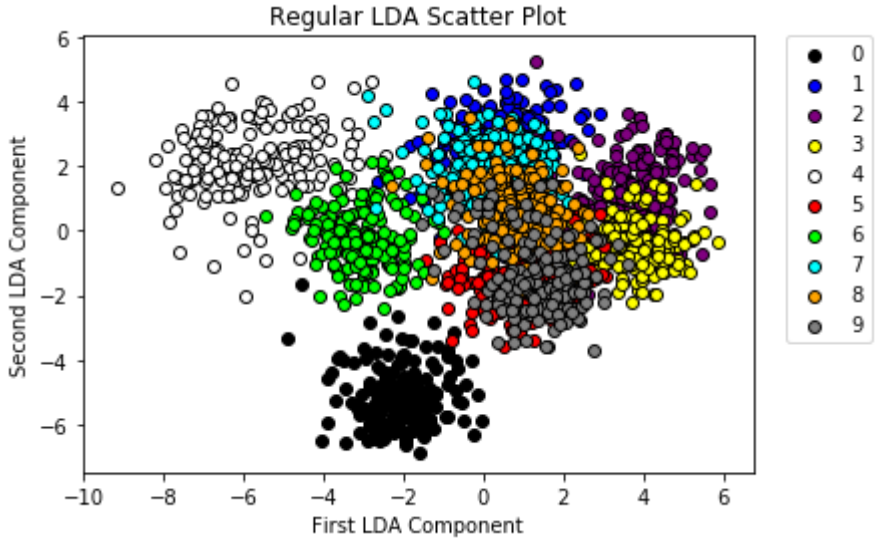
Now do the same as for PCA but now project onto 2 LDA components to get the following:

Marks: 0

In [14]:

```
lda= LDA(n_components=2)
reduced_data_lda= lda.fit(digits.data,digits.target).transform(digits.data)

# This code will not execute because you have to provide data
colors = ['black', 'blue', 'purple', 'yellow', 'white', 'red', 'lime', 'cyan',
'orange', 'gray']
for i in range(len(colors)):
    x = reduced_data_lda[:, 0][digits.target == i]
    y = reduced_data_lda[:, 1][digits.target == i]
    plt.scatter(x, y, c=colors[i],edgecolors='black')
plt.legend(digits.target_names, bbox_to_anchor=(1.05, 1), loc=2,
borderaxespad=0.)
plt.xlabel('First LDA Component')
plt.ylabel('Second LDA Component')
plt.title("Regular LDA Scatter Plot")
plt.show()
```

```
/home/it/.local/lib/python3.5/site-packages/sklearn/discriminant_ana
lysis.py:387: UserWarning: Variables are collinear.
  warnings.warn("Variables are collinear.")
```



Clustering the digits data

Split the data into test and training sets

scikit-learn provides a convenient function to split given data into training, validation and test sets in its model_selection package. Let's do that first.

In [15]:

```
# Import `train_test_split`
from sklearn.model_selection import train_test_split

# Split the `digits` data into training and test sets
X_train, X_test, y_train, y_test, images_train, images_test = train_test_split(d
igits.data, digits.target, digits.images, test_size=0.25, random_state=42)
```

In [16]:

```
# Number of training features
n_samples, n_features = X_train.shape

# Print out `n_samples`
print(n_samples)

# Print out `n_features`
print(n_features)

# Number of Training labels
n_digits = len(np.unique(y_train))
print (n_digits)

# Inspect `y_train`
print(len(y_train))
```

```
1347
64
10
1347
```

K-means model

Create the k-means model and display the means.

Marks: 0

In [17]:

```
# Create the KMeans model
# insert the code. Make sure you set init='k-means++', and random_state=42
kmeans = KMeans(n_clusters= 10,init= 'k-means++',random_state= 42)

# Fit the training data to the model
# insert code
kmeans= kmeans.fit(X_train)

# Retrieve the cluster centres
# insert code
cluster_centers= kmeans.cluster_centers_
```

In [18]:

```
# This cell will not execute because you have to provide the data.

# Figure size in inches
fig = plt.figure(figsize=(8, 3))

# Add title
fig.suptitle('Cluster Center Images', fontsize=14, fontweight='bold')

# For all labels (0-9)
for i in range(10):
    # Initialize subplots in a grid of 2X5, at i+1th position
    ax = fig.add_subplot(2, 5, 1 + i)
    # Display images
    ax.imshow(cluster_centers[i].reshape((8, 8)), cmap=plt.cm.binary, interpolat
ion='kaiser')
    # Don't show the axes
    plt.axis('off')

# Show the plot
plt.show()
```

Confusion matrix

Predict the labels using the k-means model you created above. Compare against the known labels using a confusion matrix.

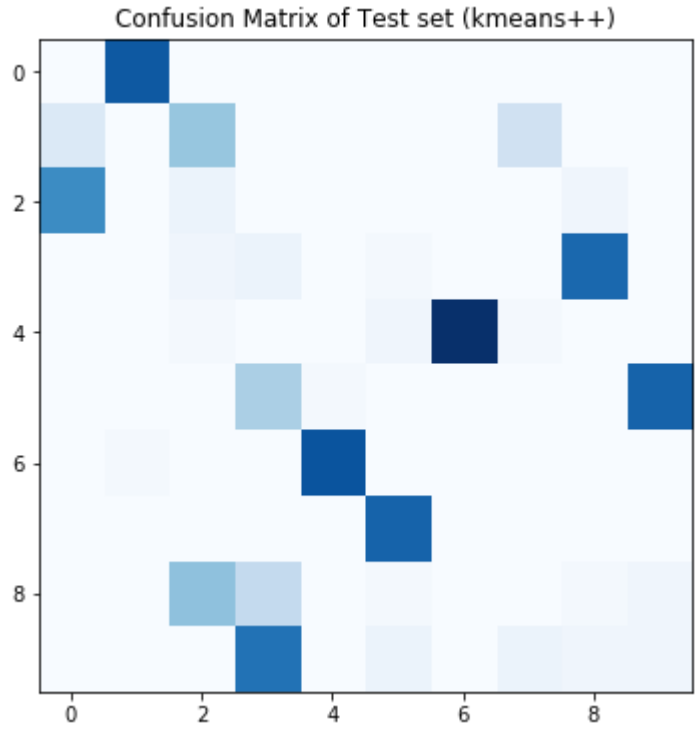
Explain what you see, and comment on the accuracy of the predicted labels.

Marks: 1

In [19]:

```
# Predict the labels for `X_test`
# Insert code
Y_pred= kmeans.predict(X_test)

# Print out the confusion matrix with `confusion_matrix()`
# insert code
CM= confusion_matrix(y_test, Y_pred)
print(CM)

plt.figure(figsize=(6,6))
plt.imshow(CM,interpolation='nearest',cmap='Blues')
plt.title('Confusion Matrix of Test set (kmeans++)')
plt.show()
```

```
[[ 0 43  0  0  0  0  0  0  0  0]
 [ 7  0 20  0  0  0  0 10  0  0]
 [33  0  3  0  0  0  0  0  2  0]
 [ 0  0  2  3  0  1  0  0 40  0]
 [ 0  0  1  0  0  2 51  1  0  0]
 [ 0  0  0 17  1  0  0  0  0 41]
 [ 0  1  0  0 44  0  0  0  0  0]
 [ 0  0  0  0  0 41  0  0  0  0]
 [ 0  0 21 13  0  1  0  0  1  2]
 [ 0  0  0 38  0  3  0  3  2  2]]
```



Interpretation

The confusion matrix is not diagonally dominant, this shows that the kmeans algorithm with the 'kmeans++' initialization method results in a bad (confused) model with poor accuracy, that severely misclassifies the Test data too often. It is not usable for this problem and other initialization methods must be investigated that would cause the algorithm to converge more and classify data points more accurately.

Really bad

You should have observed serious problems with the confusion matrix above. **Next try to initialize the k-means algorithm by providing one random sample from each class as the initial mean estimate for each cluster.** Remember that the class labels are actually available in this case.

Again plot the class means as above, predict the labels of the training data, and show the confusion matrix. Is there any improvement?

Marks: 1

In [20]:

```
init_means= np.empty((n_digits,n_features) , dtype= np.float64)

for i in range(n_digits):
    clust= X_test[np.where(y_test==i)]
    init_means[i,:]= clust[randint(len(clust)),:]

kmeans = KMeans(n_clusters= 10, init= init_means, random_state= 42)
kmeans= kmeans.fit(X_train)
cluster_centers= kmeans.cluster_centers_

Y_pred= kmeans.predict(X_test)
Y_train_labels= kmeans.predict(X_train)

CM= confusion_matrix(y_test, Y_pred)
CM_train= confusion_matrix(y_train, Y_train_labels)
```

```
/home/it/.local/lib/python3.5/site-packages/sklearn/cluster/k_means
_.py:889: RuntimeWarning: Explicit initial center position passed: p
erforming only one init in k-means instead of n_init=10
  return_n_iter=True)
```

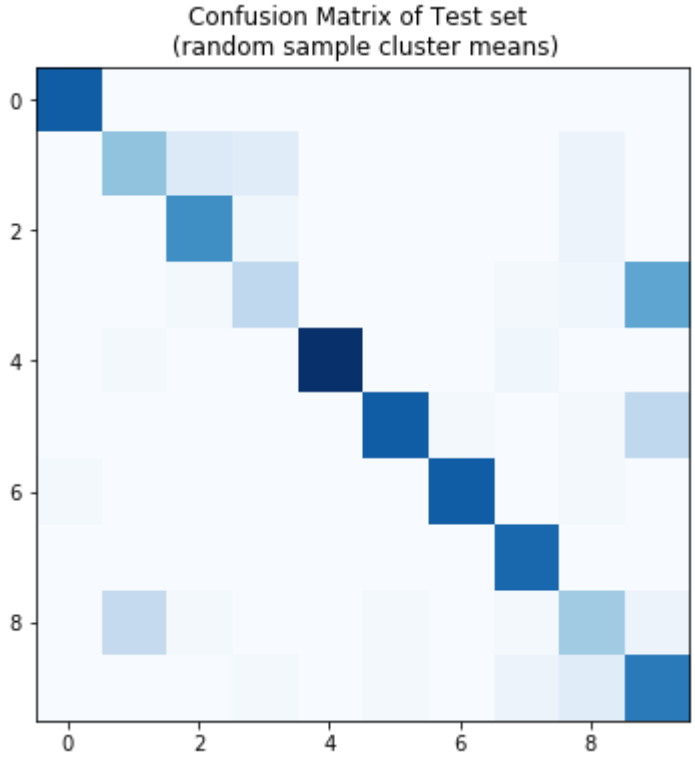
In [21]:

```
## Printing the cluster centers for the provided initial means
fig = plt.figure(figsize=(8, 3))
fig.suptitle('Cluster Center Images (provided sample)', fontsize=14,
fontweight='bold')

for i in range(10):
    ax = fig.add_subplot(2, 5, 1 + i)
    ax.imshow(cluster_centers[i].reshape((8, 8)), cmap=plt.cm.binary, interpolat
ion='kaiser')
    plt.axis('off')
plt.show()
```

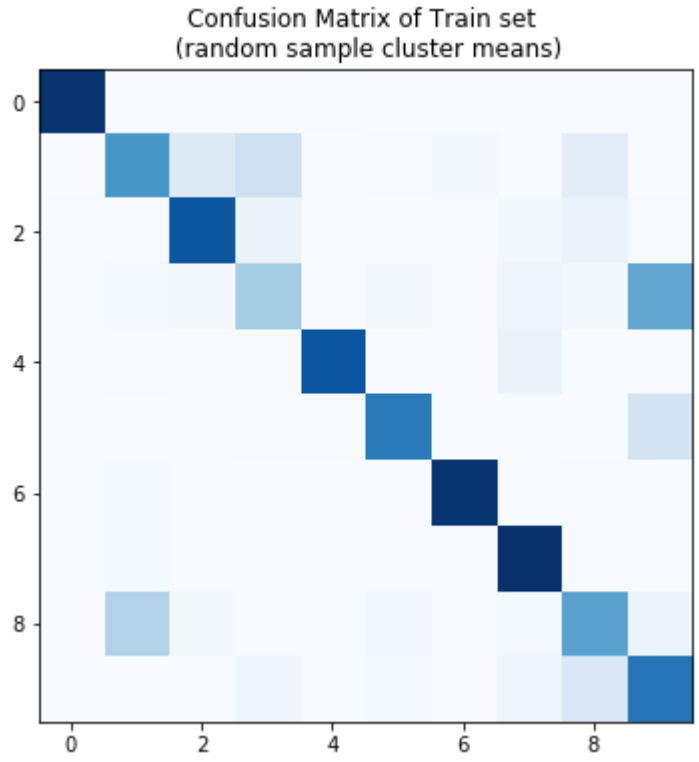
In [22]:

```
## Printing the confusion matrices for both train and test data
plt.figure(figsize=(6,6))
print("Confusion matrix of predicted Test set labels (fitted on Train set)")
print(CM)

plt.imshow(CM,interpolation='nearest',cmap='Blues')
plt.title('Confusion Matrix of Test set \n (random sample cluster means)')
plt.show()

plt.figure(figsize=(6,6))
print("Confusion matrix of predicted Train set labels (fitted on train set)")
print(CM_train)

plt.imshow(CM_train,interpolation='nearest',cmap='Blues')
plt.title('Confusion Matrix of Train set \n (random sample cluster means)')
plt.show()
```

```
Confusion matrix of predicted Test set labels (fitted on Train set)
[[43  0  0  0  0  0  0  0  0  0]
 [ 0 21  7  6  0  0  0  0  3  0]
 [ 0  0 33  2  0  0  0  0  3  0]
 [ 0  0  1 14  0  0  0  1  2 28]
 [ 0  1  0  0 52  0  0  2  0  0]
 [ 0  0  0  0  0 43  1  0  1 14]
 [ 1  0  0  0  0  0 43  0  1  0]
 [ 0  0  0  0  0  0  0 41  0  0]
 [ 0 13  1  0  0  1  0  1 19  3]
 [ 0  0  0  1  0  1  0  3  6 37]]
```




```
Confusion matrix of predicted Train set labels (fitted on train set)
[[134   0   0   0   1   0   0   0   0   0]
 [  0  82  17  28   0   1   3   0  14   0]
 [  1   0 116   9   0   0   0   3   9   1]
 [  0   2   3  48   0   3   0   6   3  72]
 [  0   1   0   0 116   0   0   9   0   0]
 [  0   0   0   1   1  97   0   0   0  24]
 [  0   2   0   0   0   0 133   0   1   0]
 [  0   2   0   0   0   0   0 136   0   0]
 [  0  43   3   0   0   4   1   2  75   8]
 [  0   0   0   5   0   2   0   6  19 100]]
```



There is a very significant improvement! The chosen initial cluster means were chosen from each cluster themselves, meaning that convergence is automatically improved because the initial means are either close to, or are the actual(intended final-) means for the clusters.
The initial cluster means are more likely to be closer to the true mean than with the 'random' and 'kmeans++' approach of the algorithm.

Semi-supervised training

In assignment 2, you studied labeled data, and were introduced to the supervised learning process. So far, this assignment has considered how data can be organized if no labels are available, so-called unsupervised learning.

In practice it often happens that one has some labeled data available, in addition to (an often large amount of) unlabeled data. The question is how to make use of the additional information. This is known as semi-supervised learning.

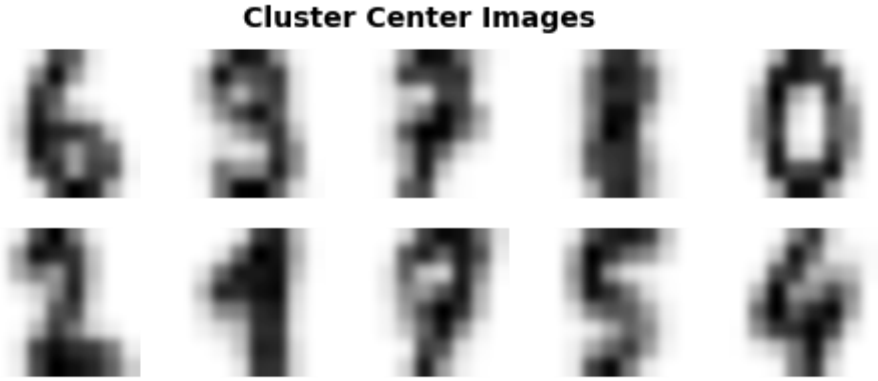
You are now going to adapt the k-means algorithm for semi-supervised learning. For this you will use the kmeans.py package provided. The relevant code for semi-supervised learning was removed, but it still runs for unsupervised learning. The few code cells below execute it in unsupervised mode to give you an idea how it functions (not unlike the scikit-learn package).

In [38]:

```
# Random initialization
km = KM(codes=10,itr=20)
km.fit(X_train)
cluster_centers = km.get_means
```

In [39]:

```
# Figure size in inches
fig = plt.figure(figsize=(8, 3))

# Add title
fig.suptitle('Cluster Center Images', fontsize=14, fontweight='bold')

# For all labels (0-9)
for i in range(10):
    # Initialize subplots in a grid of 2X5, at i+1th position
    ax = fig.add_subplot(2, 5, 1 + i)
    # Display images
    ax.imshow(cluster_centers[i].reshape((8, 8)), cmap=plt.cm.binary, interpolat
ion='kaiser')
    # Don't show the axes
    plt.axis('off')

# Show the plot
plt.show()
```



Creating Data

To use semi-supervised learning, we need some labeled data. To do this, we select an equal number of samples from each class, using the test set.

In [40]:

```
# The number of samples per class
n_samples = 2

# The same number of samples is generated for each class. This is just
# a convenience, any number will do. Note: Assume at least one sample for
# each class
X_lbl = np.zeros((10*n_samples,64))
y_lbl = np.zeros((10*n_samples,))
for i in range(10):
    dat = X_test[y_test==i,:]
    samples = np.random.randint(0,dat.shape[0]-1,n_samples)

    X_lbl[i*n_samples:(i+1)*n_samples] = dat[samples]
    y_lbl[i*n_samples:(i+1)*n_samples] = i
```

Look at the data that we generated.

Modify code

Modify the given k-means code to allow semi-supervised learning. *Note*: You have to provide code anywhere you encounter the **pass** keyword; include *only* the code you added in your report, clearly indicating which portions correspond to the first, second, and third occurence of the **pass** keyword.

If your modification is successful, your cluster centres should look something like the image below. **Predict the labels in the training set and print the confusion matrix.**

Marks: 4

In [41]:

```
# This cell will not execute unless you have made the necessary modification.

km = KM(codes=10,itr=20)
km.fit(X= X_train,X_label= X_lbl,y=y_lbl)
km_cluster_centers = km.get_means
```

In [42]:

```
# Figure size in inches
fig = plt.figure(figsize=(8, 3))

# Add title
fig.suptitle('Cluster Center Images', fontsize=14, fontweight='bold')

# For all labels (0-9)
for i in range(10):
    # Initialize subplots in a grid of 2X5, at i+1th position
    ax = fig.add_subplot(2, 5, 1 + i)
    # Display images
    ax.imshow(km_cluster_centers[i].reshape((8, 8)), cmap=plt.cm.binary, interpo
lation='kaiser')
    # Don't show the axes
    plt.axis('off')

# Show the plot
plt.show()
```

Modified K-means code

pass #1

means= self.init_means*lbl*(X_label,y)

pass #2

dat= np.vstack((X_label[y==i,:],X[labels==i,:]))

pass #3

samps= X_label[y==i,:]
means[i]= samps[np.random.randint(0,samps.shape[0]),:]

Confusion matrix

In this case we don't have the identifiability problem as indicated by the confusion matrix.

In [43]:

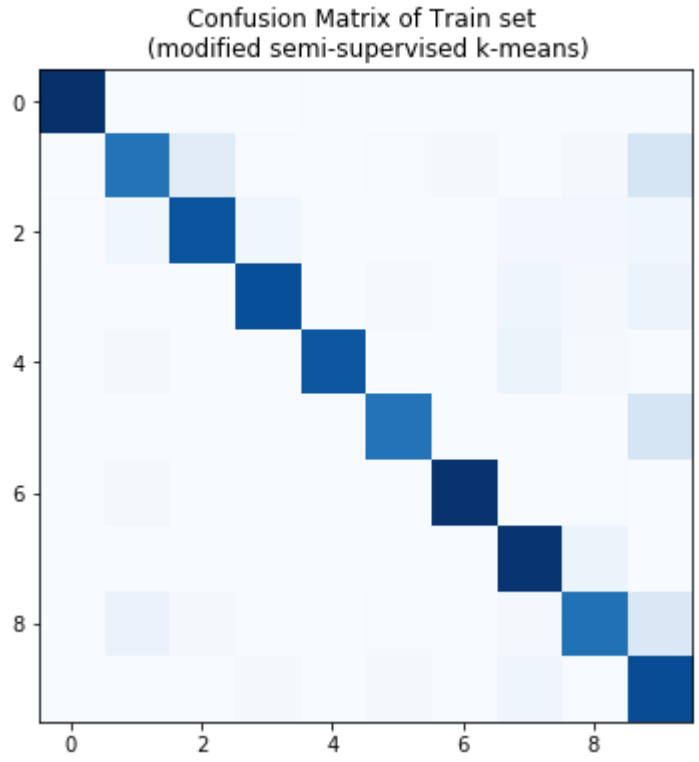
```
y_pred = km.predict(X_test)
print(confusion_matrix(y_test, y_pred))
```

```
[[43  0  0  0  0  0  0  0  0  0]
 [ 0 27  7  0  0  0  0  0  0  3]
 [ 0  1 33  1  0  0  0  0  3  0]
 [ 0  0  0 40  0  0  0  1  3  2]
 [ 0  2  0  0 51  0  0  2  0  0]
 [ 0  0  0  0  0 45  1  0  0 13]
 [ 1  0  0  0  0  0 44  0  0  0]
 [ 0  0  0  0  0  0  0 40  1  0]
 [ 0  3  0  0  0  1  0  0 31  3]
 [ 0  0  0  0  0  1  0  3  1 43]]
```

In [44]:

```
Y_pred= km.predict(X_train)
CM_train= confusion_matrix(y_train, Y_pred)

plt.figure(figsize=(6,6))
print("Confusion matrix of predicted Train set labels")
print(CM_train)

plt.imshow(CM_train,interpolation='nearest',cmap='Blues')
plt.title('Confusion Matrix of Train set \n (modified semi-supervised k-means)')
plt.show()
```

```
Confusion matrix of predicted Train set labels
[[134   0   0   0   1   0   0   0   0   0]
 [  0  99  15   1   0   1   3   0   3  23]
 [  1   5 115   5   0   0   0   4   4   5]
 [  0   0   1 118   0   2   0   6   3   7]
 [  0   3   0   0 114   0   0   7   2   0]
 [  0   0   0   1   1  99   0   0   0  22]
 [  0   3   0   0   0   0 132   0   1   0]
 [  0   0   0   0   0   0   0 131   7   0]
 [  0   9   3   0   0   1   1   3 100  19]
 [  0   0   0   3   0   3   0   6   1 119]]
```



Explanation

The modified code for semi-supervised k-means clustering seems to have worked well and accuracy is fair. This is shown by the diagonally dominant confusion matrix. It performs quite satisfactorily and classifies most of the data adequately.

GMM

Next, fit the data to a 10 component GMM model, using the scikit-learn package.

Once you have built the GMM model, extract the means as well as the covariances of the 10 GMM components. Display the means to get something like the image below.

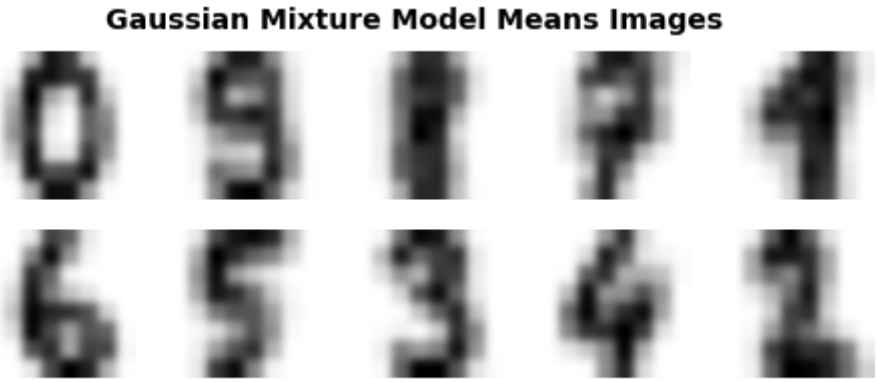
Marks: 1

In [ ]:

```
data = digits.data

np.random.seed(1)
# Insert GMM code

gmm= GMM(n_components= 10).fit(data)
means= gmm.means_
covariances= gmm.covars_

## I did run this block, I just needed to clear the output because it was a leng
thy WARNING that would appear
## in the pdf too.
```

In [46]:

```
fig = plt.figure(figsize=(8, 3))
fig.suptitle('Gaussian Mixture Model Means Images', fontsize=14, fontweight='bol
d')

for i in range(10):
    ax = fig.add_subplot(2, 5, 1 + i)
    ax.imshow(means[i].reshape((8, 8)), cmap=plt.cm.binary, interpolation='kaise
r')
    plt.axis('off')

plt.show()
```



Generate samples

Use the means and covariances of the different components and draw a sample from each component. Display the samples in an image.

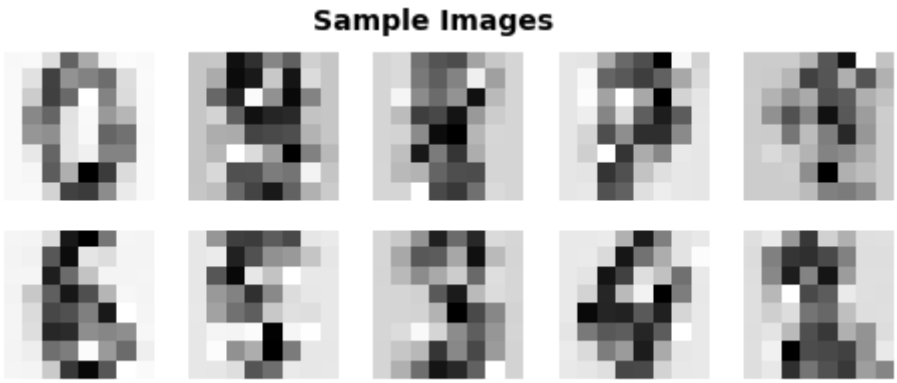
Hint: Read the documentation of scipy.stats.multivariate_normal

Marks: 1

In [47]:

```
from scipy.stats import multivariate_normal as mn

print(data.shape)

samples= np.zeros((10,64))
for i in range(10):
    samples[i] = mn.rvs(mean= means[i], cov= covariances[i])
    #print(sample.shape)


fig = plt.figure(figsize=(8, 3))
fig.suptitle('Sample Images', fontsize=14, fontweight='bold')

for i in range(10):
    ax = fig.add_subplot(2, 5, 1 + i)
    ax.imshow(samples[i].reshape((8, 8)), cmap=plt.cm.binary)
    plt.axis('off')

plt.show()
```

```
(1797, 64)
```



Plagiarism declaration

Add the plagiarism declaration from the University's Plagiarism Policy in this cell.

I know that plagiarism is wrong.
Plagiarism is to use another's work (even if it is summarised, translated or rephrased) and pretend that it is one's own.
This assignment is my own work.
Each contribution to and quotation (e.g. "cut and paste") in this assignment from the work(s) of other people has been explicitly attributed, and has been cited and referenced.
In addition to being explicitly attributed, all quotations are enclosed in inverted commas, and long quotations are additionally in indented paragraphs.
I have not allowed, and will not allow, anyone to use my work (in paper, graphics, electronic, verbal or any other format) with the intention of passing it off as his/her own work.
I know that a mark of zero may be awarded to assignments with plagiarism and also that no opportunity be given to submit an improved assignment.
I know that students involved in plagiarism will be reported to the Registrar and/or the Central Disciplinary Committee.
Name: Tshenolo Thato Daumas
Student Numer: 19643128
Signature: TTE Daumas
Date: 24 April 2017

In [ ]: