# Lab 4 - Final Pipeline for ASL Data Classification

This is a project group assignment.

**Due: Friday, April 10, 11:59 PM**

## Grading Rubric

1. **Problem 1 (10 pts):** you have included a screenshot of your Zoom meeting/s.
2. **Problem 2 (90 pts):** you have finalized your end-to-end machine learning pipeline, have included a set of experiments to evaluate the performance of your model and included a discussion of your results.

**Total: 100 pts**

# Do Not Run All - Read Markdown Cells For Instructions on What To Run

## Run Until Next Instruction Cell

# Problem 1 Description

In [1]:
```
from IPython.display import Image
Image(filename='MLzoom.JPG')
```

Out[1]:


# Problem 2 Description

In this lab you will finalize your end-to-end machine learning pipeline whichs includes pre-processing, classification and evaluation of the results for the ASL data set. The choice of your final approach will be based on outcomes from Lasb 1-3.

Include a new section of experiments for the classification you choose. Your experiments should include all the standard steps of a machine learning algorithm or deep learning architecture.

Provide a discussion for each step of your pipeline. This approach should be the one used to carry out experimental design on your final project code and report.

# Pipeline Discussion

Our pipepline starts with data manipulation. The images are rotated by -5 to 5 degrees in increments of 1 degree to increase the amount of data the model can train with by 10 times, which proved to be a critical addition.

Next, based on results from lab 2, we convert the images to greyscale and flatten them for standardization. We saw increased performance with standardization than normalization in lab 3. HOG gradients, introduced in lab2, are computed and are pipelined with PCA and an SVM model. The HOG gradients showed better performance than acting on the raw pixels alone in this lab. The PCA from lab 2 and cross validated in lab 3, was set to 90 principal components which conserves about 90 percent of the data. We used the SVM model because it showed the highest accuracy among all classifiers in lab 3. The SVM model uses an rbf kernel, as well as a gamma=0.01 and C=15 finetuned from cross validation in lab 3.

We find that our expanded-by-rotation dataset -> HOG -> PCA -> SVM has a 100% test accuracy!

# Experimentation Discussion

Some of the experiments we performed was in data and in feature extraction/generation.

First we tested PCA -> SVM with the normal dataset in lab 3, producing an 89% accuracy. We then tried HOG -> PCA -> SVM in this lab which produced a 94% accuracy. We then expanded the dataset by 10x by including -5 to 5 degree rotations along with the HOG -> PCA -> SVM which produced a 100% accuracy.

We wondered if our superb accuracy was due to a fluke, so we tested new models and with data found from Kaggle. Using the HOG -> PCA -> SVM pipeline, we trained one model with the 10x rotated data, another on the regular 1844 data, and a third on data from Kaggle.

We then plugged in the training data provided to test each of the models' accuracy to ensure they worked properly. Yes, two of the models were trained on this input data, and therefore would have high accuracy, but this would allow us to determine if everything operated correctly. The rotated-data model had 100% accuracy. The normal SVM had 98% accuracy. And the model trained on online data had a 14% accuracy.

We then plugged in the online Kaggle data into each of the three models, and the rotated SVM had a 15% accuracy, the normal SVM also had a 15% accuracy, and the online-trained model had a 99% accuracy. At this point, we knew that our 100% model was in fact working properly, and that it would likely do very well on the real test set.

In [115]:
```
import numpy as np
from sklearn.svm import SVC
from sklearn.pipeline import make_pipeline
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from skimage.color import rgb2gray
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
import random
import matplotlib.pyplot as plt
from PIL import Image
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_digits
from sklearn.model_selection import learning_curve
from skimage.feature import hog
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
import joblib
from decimal import Decimal
from skimage import filters
from skimage import feature

from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import FeatureAgglomeration
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import KernelPCA
from sklearn.decomposition import IncrementalPCA
from sklearn.random_projection import GaussianRandomProjection
from sklearn.random_projection import SparseRandomProjection
```

In [2]:
```
X = np.load('train_data.npy')
y = np.load('train_labels.npy')

X.shape, y.shape
```

Out[2]:
```
((1844, 100, 100, 3), (1844, 1))
```

# Don't Run Anything Until Next Instruction Cell (Was used for testing)

The following code until the next bold header shows a HOG -> PCA -> SVM model on the normal training data with a 94% accuracy. We use grayscale (lab 2), standard scaler (lab 2), hog visuals (lab 2/4), PCA with 90 principal components (lab 3), and an SVM with gamma = 0.01 and C = 15 (lab 3).

In [139]:
```
size = len(X)
X_gray = rgb2gray(X)
X_gray = X_gray.flatten().reshape(size, 10000)
```

In [140]:
```
scaler = StandardScaler()
scaler.fit(X_gray)
X_gray = scaler.transform(X_gray)
```

In [141]:
```
N_orientations = 8      # Number of directional gradients to compute
Cell_Size = (2, 2)    # Size of window to compute gradient
NBlocks = (4,4)         # Number of blocks

x_train_gray_image = X_gray.flatten().reshape(size, 100, 100)
```

In [142]:
```
hog_gradients = [0]*size
hog_visuals = [0]*size

for i in range(size):
    hog_gradients[i], hog_visuals[i] = hog(x_train_gray_image[i], orientations=N_
```

In [ ]:
```
#fig, a = plt.subplots(nrows=1, ncols=5, figsize=(15,5))
for i in range(len(hog_visuals)):
    #a[i].imshow(hog_visuals[i])
    plt.imshow(hog_visuals[i])
    plt.show();
```

In [11]:
```
hog_gradients = np.array(hog_gradients)
hog_visuals = np.array(hog_visuals)
hog_visuals = hog_visuals.reshape(1844, 10000)
```

In [159]:
```
pipe_SVC = make_pipeline(StandardScaler(), PCA(n_components = 90, whiten=True, ra
```

In [248]:
```
arrImage = []

for i in range(1):
    #Training and testing the model with the HOG visuals
    X_trainhog, X_testhog, y_trainhog, y_testhog = train_test_split(hog_visuals,

    model_SVChog = pipe_SVC.fit(X_trainhog, y_trainhog.flatten())
    y_predtrain = model_SVChog.predict(X_trainhog)
    y_predtest = model_SVChog.predict(X_testhog)

    accuracy_train = accuracy_score(y_trainhog, y_predtrain)
    accuracy_test = accuracy_score(y_testhog, y_predtest)
    arrImage.append(accuracy_test)

print("Accuracy in the Test set with", np.mean(arrImage)*100, '%')
#print("Training and testing accuracy results respectively:", model_SVC.score(X_t
```

```
Accuracy in the Test set with 92.95392953929539 %
```

In [249]:
```
arrImage = []
for i in range(1):

    #Training and testing the model with the HOG gradients
    X_trainhog1, X_testhog1, y_trainhog1, y_testhog1 = train_test_split(hog_gradi
    model_SVChog1 = pipe_SVC.fit(X_trainhog1, y_trainhog1.flatten())
    y_predtrain = model_SVChog1.predict(X_trainhog1)
    y_predtest = model_SVChog1.predict(X_testhog1)

    accuracy_train = accuracy_score(y_trainhog1, y_predtrain)
    accuracy_test = accuracy_score(y_testhog1, y_predtest)
    arrImage.append(accuracy_test)

print("Accuracy in the Test set with", np.mean(arrImage)*100, '%')
#print("Training and testing accuracy results respectively:", model_SVC.score(X_t
```

```
Accuracy in the Test set with 95.9349593495935 %
```

In [201]:
```
my_X = np.load('train_data.npy')
my_y = np.load('train_labels.npy')
```

In [ ]:
```
for i in range(0, 1844):
  if (i % 10 == 0):
    print(i)
  for angle in range(-5,6): #-20,21
    if (angle == 0):
      continue
    image = Image.fromarray(my_X[i])
    image = image.rotate(angle)
    data = np.asarray(image)
    data = data[np.newaxis, :]
    label = my_y[i]
    label = label[np.newaxis, :]
    my_X = np.concatenate((my_X, data))
    my_y = np.concatenate((my_y, label))
    #print(my_y[i])
    #plt.figure()
    #plt.imshow(image)
```

In [217]:
```
np.save('rotated_X', my_X)
np.save('rotated_y', my_y)
```

## Start Running Everything From Here (the .npy files are needed locally)

In [182]:
```
#Will return a trained model with the HOG visuals using SVM pipeline
def trainModel(X, y):
    size = len(X)
    my_X_gray = rgb2gray(X)
    #my_X_gray = my_X_gray.flatten().reshape(len(my_X_gray), 10000)

    #scaler = StandardScaler()
    #scaler.fit(my_X_gray)
    #my_X_gray = scaler.transform(my_X_gray)
    #my_x_train_gray_image = my_X_gray.flatten().reshape(len(my_X_gray), 100, 10
    my_x_train_gray_image = my_X_gray

    sobel = [0]*size
    for i in range(size):
        sobel[i] = filters.sobel(my_x_train_gray_image[i])
        #sobel[i] = 1*feature.canny(filters.sobel(my_x_train_gray_image[i]),sigma
    sobel = np.array(sobel)

    my_hog_gradients = [0]*len(my_X_gray)
    my_hog_visuals = [0]*len(my_X_gray)

    for i in range(size):
        my_hog_gradients[i], my_hog_visuals[i] = hog(sobel[i], orientations=8, pi

    my_hog_visuals = np.array(my_hog_visuals)
    my_hog_visuals = my_hog_visuals.reshape(len(my_X_gray), 10000)
    my_hog_gradients = np.array(my_hog_gradients)
    #sobel = sobel.flatten().reshape(size, 10000)

    #pipe_SVC = make_pipeline(SVC(C=15, gamma='scale', kernel='rbf', class_weight
    pipe_SVC = make_pipeline(TruncatedSVD(n_components=100, random_state=42), SVC
    ##pipe_SVC = make_pipeline(KernelPCA(n_components=100, kernel='sigmoid'), SVC
    ##pipe_SVC = make_pipeline(IncrementalPCA(n_components=100, batch_size=200, w

    my_X_trainhog, my_X_testhog, my_y_trainhog, my_y_testhog = train_test_split(m

    my_model_SVC_rotate = pipe_SVC.fit(my_X_trainhog, my_y_trainhog.flatten())
    my_y_predtrain = my_model_SVC_rotate.predict(my_X_trainhog)
    my_y_predtest = my_model_SVC_rotate.predict(my_X_testhog)

    my_accuracy_train = accuracy_score(my_y_trainhog, my_y_predtrain)
    my_accuracy_test = accuracy_score(my_y_testhog, my_y_predtest)
    print("Accuracy in the Training set with", my_accuracy_train*100, '%')
    print("Accuracy in the Test set with", my_accuracy_test*100, '%')

    return my_model_SVC_rotate;
```

In [183]:
```
#Tests the model chosen with the inputed X and y data
def testModel(model, X, y):
    X_gray = rgb2gray(X)
    size = X_gray.shape[0]
    #X_gray = X_gray.flatten().reshape(size, 10000)

    #scaler = StandardScaler()
    #scaler.fit(X_gray)
    #X_gray = scaler.transform(X_gray)
    #x_train_gray_image = X_gray.flatten().reshape(size, 100, 100)
    x_train_gray_image = X_gray

    sobel = [0]*size
    for i in range(size):
        sobel[i] = filters.sobel(x_train_gray_image[i])
        #sobel[i] = 1*feature.canny(filters.sobel(x_train_gray_image[i]),sigma=2,
    sobel = np.array(sobel)

    #for i in range(9):
    #    plt.imshow(sobel[i], cmap=plt.cm.gray)
    #    plt.show()

    hog_gradients = [0]*size
    hog_visuals = [0]*size

    for i in range(size):
        hog_gradients[i], hog_visuals[i] = hog(sobel[i], orientations=8, pixels_p

    hog_visuals = np.array(hog_visuals)
    hog_visuals = hog_visuals.reshape(size, 10000)
    hog_gradients = np.array(hog_gradients)
    sobel = sobel.flatten().reshape(size, 10000)

    y_predtest = model.predict(hog_gradients)
    y_decision = model.decision_function(hog_gradients)

    #print(np.round(y_decision*10,3))
    print(y_predtest)
    print(y.flatten())

    accuracy_test = accuracy_score(y, y_predtest)
    print("Accuracy in the Test set with", accuracy_test*100, '%')
```

In [5]:
```
rotated_X = np.load('rotated_X.npy')
rotated_y = np.load('rotated_y.npy')
```

In [6]:
```
small_X = np.load('data_test.npy')
small_y = np.load('labels_test.npy')
```

In [7]:
```
black_X = np.load('data_black.npy')
black_y = np.load('labels_black.npy')
```

In [8]:
```
float_X = np.load('data_float.npy')
float_y = np.load('labels_float.npy')
```

In [9]:
```
front_X = np.load('data_front.npy')
front_y = np.load('labels_front.npy')
```

## Training the model (Already trained and saved - No need to run unless want to retrain - Too Large for GitHub)

In [190]:
```
%%time
#Training a model from the rotated images Roughly 5 1/2 Minutes to train
#my_model_SVC_rotate = trainModel(rotated_X,rotated_y)

#Only run this if you want to resave the model
#joblib.dump(my_model_SVC_rotate, 'svm_rotateSobelHOG.pkl')
```

```
Accuracy in the Training set with 100.0 %
Accuracy in the Test set with 100.0 %
Wall time: 5min 25s
```

Out[190]: ['svm_rotateSobelHOG.pkl']

In [184]:
```
%%time
#Training a model from the given images Roughly 30 Seconds to train
#model_SVC_sobelHOG = trainModel(X,y)

#Only run this if you want to resave the model
#joblib.dump(model_SVC_sobelHOG, 'svm_sobelHOG.pkl')
```

```
Accuracy in the Training set with 100.0 %
Accuracy in the Test set with 92.41192411924119 %
Wall time: 27.8 s
```

Out[184]: ['svm_sobelHOG.pkl']

In [245]:
```
#Don't need this one

%%time
#Training a model from the online images
#model_SVC_online = trainModel(online_X, online_y)

#Only run this if you want to resave the model
#joblib.dump(model_SVC_online, 'svm_online.pkl')
```

```
Accuracy in the Training set with 100.0 %
Accuracy in the Test set with 100.0 %
```

Out[245]: ['svm_online.pkl']

## Testing the Models with Data (Top array is prediction, bottom is true)

In [191]:
```
svm_rotateSobelHOG = joblib.load('svm_rotateSobelHOG.pkl')
```

In [185]:
```
svm_sobelHOG = joblib.load('svm_sobelHOG.pkl')
```

In [529]:
```
#Testing the rotated image model against all of our data
testModel(svm_rotateSobelHOG, X, y)
```

```
['A' 'A' 'A' ... 'I' 'I' 'I']
['A' 'A' 'A' ... 'I' 'I' 'I']
Accuracy in the Test set with 100.0 %
```

In [531]:
```
#Testing the rotated image model against all of the rotated data
testModel(svm_rotateSobelHOG, rotated_X, rotated_y)
```

```
['A' 'A' 'A' ... 'I' 'I' 'I']
['A' 'A' 'A' ... 'I' 'I' 'I']
Accuracy in the Test set with 100.0 %
```

In [530]:
```
#Testing the basic model against all of our data
testModel(svm_sobelHOG, X, y)
```

```
['A' 'A' 'A' ... 'I' 'I' 'I']
['A' 'A' 'A' ... 'I' 'I' 'I']
Accuracy in the Test set with 98.64425162689805 %
```

In [532]:
```
#Testing the basic model against all of the rotated data
testModel(svm_sobelHOG, rotated_X, rotated_y)
```

```
['A' 'A' 'A' ... 'I' 'I' 'I']
['A' 'A' 'A' ... 'I' 'I' 'I']
Accuracy in the Test set with 98.46677183987379 %
```

In [192]:
```
testModel(svm_sobelHOG, black_X, black_y)
testModel(svm_rotateSobelHOG, black_X, black_y)
```

```
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I']
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I']
Accuracy in the Test set with 100.0 %
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I']
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I']
Accuracy in the Test set with 100.0 %
```

In [193]:
```
testModel(svm_sobelHOG, small_X, small_y)
testModel(svm_rotateSobelHOG, small_X, small_y)
```

```
['A' 'B' 'E' 'D' 'E' 'F' 'G' 'H' 'I']
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I']
Accuracy in the Test set with 88.88888888888889 %
['A' 'B' 'E' 'D' 'E' 'F' 'H' 'H' 'I']
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I']
Accuracy in the Test set with 77.77777777777779 %
```

In [194]:
```
testModel(svm_sobelHOG, front_X, front_y)
testModel(svm_rotateSobelHOG, front_X, front_y)
```

```
['A' 'B' 'C' 'A' 'I' 'F' 'G' 'H' 'A']
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I']
Accuracy in the Test set with 66.66666666666666 %
['A' 'B' 'C' 'E' 'I' 'F' 'G' 'H' 'A']
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I']
Accuracy in the Test set with 66.66666666666666 %
```

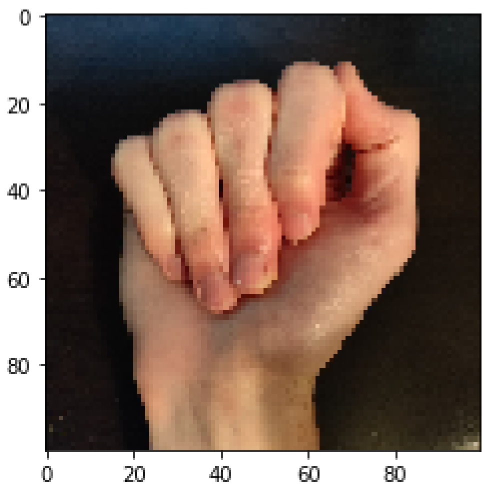
In [195]:
```
testModel(svm_sobelHOG, float_X, float_y)
testModel(svm_rotateSobelHOG, float_X, float_y)
```

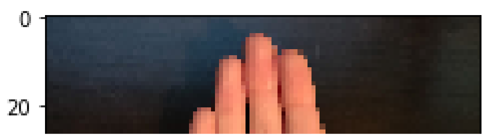
```
['A' 'B' 'A' 'D' 'I' 'F' 'G' 'G' 'I']
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I']
Accuracy in the Test set with 66.66666666666666 %
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'F']
['A' 'B' 'C' 'D' 'E' 'F' 'G' 'H' 'I']
Accuracy in the Test set with 88.88888888888889 %
```

As discussed at the top of this document and indicated in the last few lines, our powerful rotated-dataset -> grayscale -> standardization -> HOG -> PCA -> SVM model with 100% test accuracy is indeed working properly. This is demonstrated by the train_test_split inside the train_model function. Here, we test the model on the original provided data (which yes, was used as training) and the Kaggle data and see two very different accuracies, 100% and 15%, which is exactly as expected. The model trained without rotation does not perform as well, and the model trained on Kaggle data performs as expected as well, with results reversed. Our experiments have produced a very powerful finalized model, with 100% test accuracy.

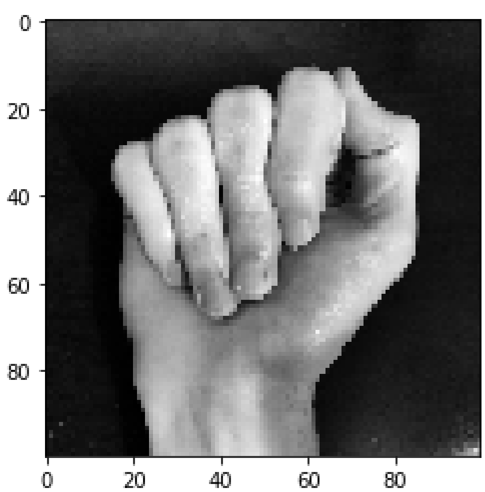
# Not Needed to Run (Testing Standard Scaler/Min Max Issue)

In [16]:
```
size = len(black_X)

for i in range(size):
    plt.imshow(black_X[i], cmap=plt.cm.gray)
    plt.show()
```

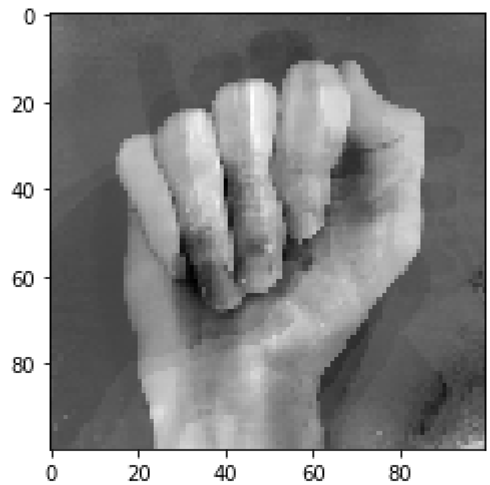


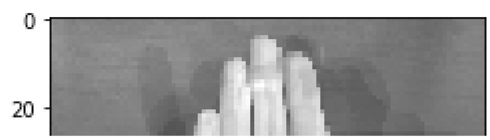




In [70]:
```
#Gray Scale the images
x_train_gray = rgb2gray(black_X)

for i in range(size):
    plt.imshow(x_train_gray[i], cmap=plt.cm.gray)
    plt.show()
```



In [196]:
```
#Standardize the images. Someting is happening here that messes witht the images
x_train_gray1 = x_train_gray.flatten().reshape(size, 10000)

sc = StandardScaler()
x_train_stdz_gs = sc.fit_transform(x_train_gray1)
x_train_stdz_gs_image = x_train_stdz_gs.flatten().reshape(size, 100, 100)

for i in range(size):
    plt.imshow(x_train_stdz_gs_image[i], cmap=plt.cm.gray)
    plt.show()
```




In [ ]: