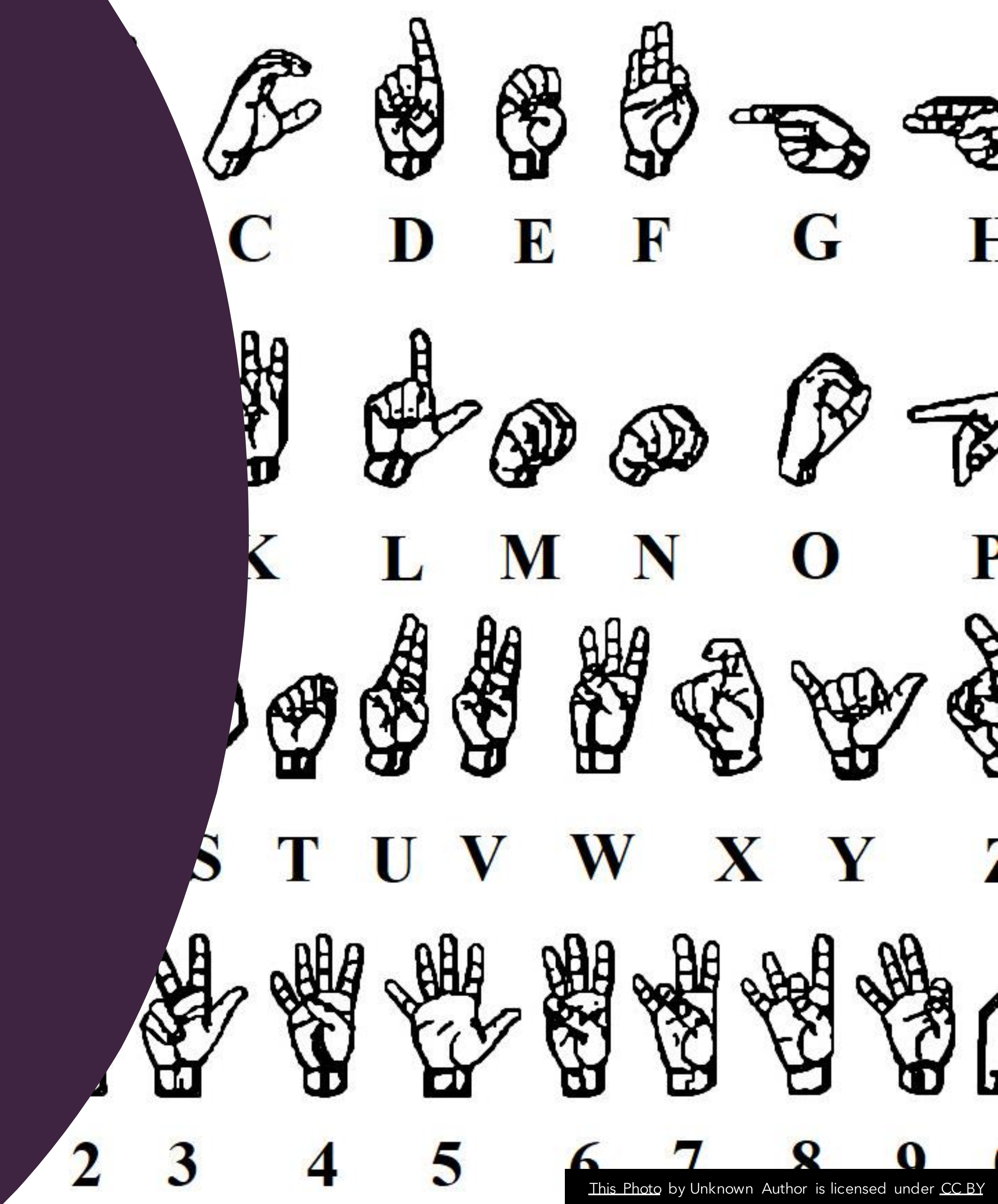
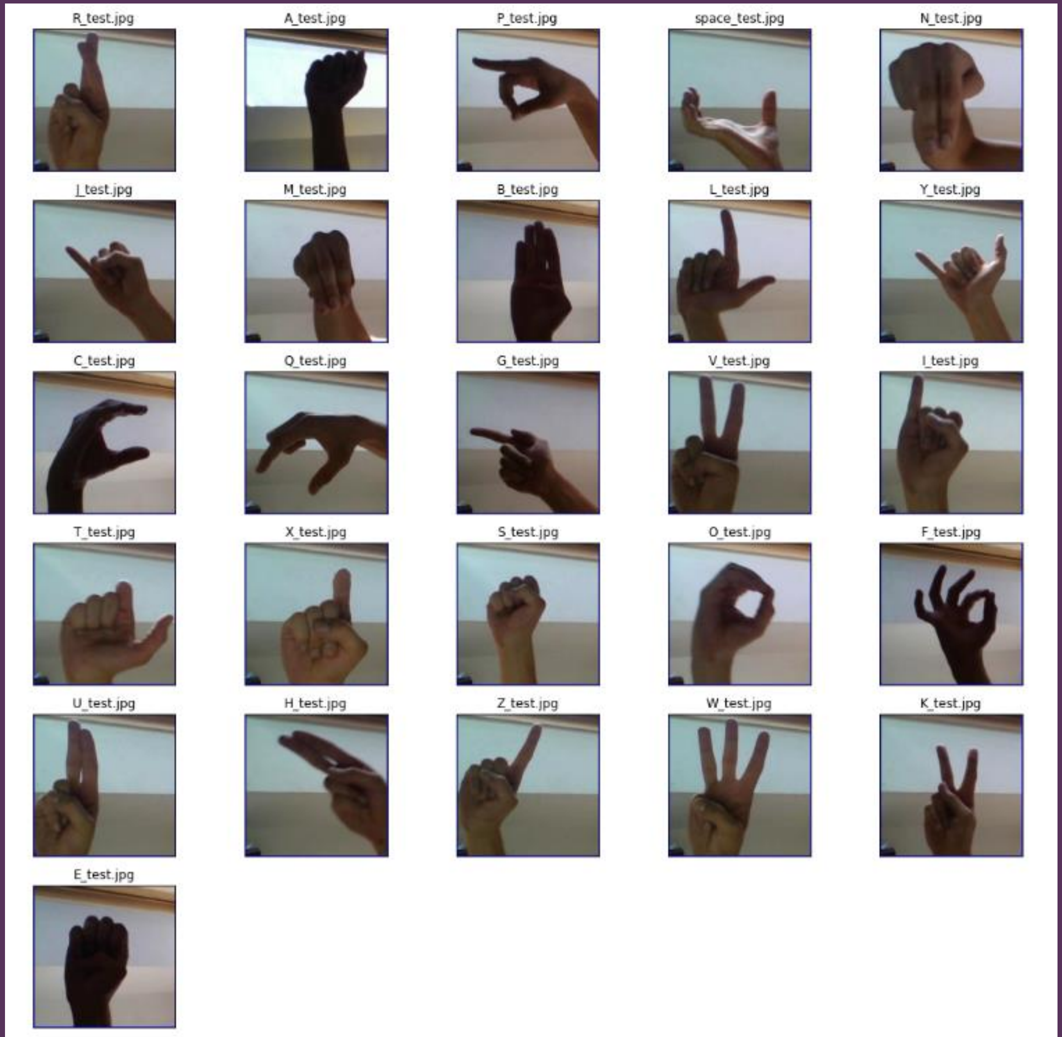
Sign Language

Gestures to interpret

Ketan Patel, Noah Guralnik,
Matthew Benvenuto





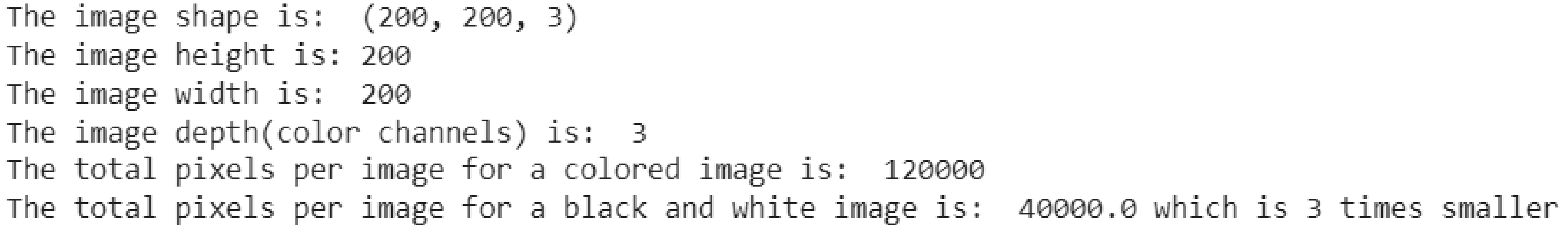
The ASL Sign Language Gestures

- Here is a list of all of our gestures that we are using to build an Eigen-sign Matrix.
- Each column of the Eigen-Sign Matrix will be a unique image (sign).

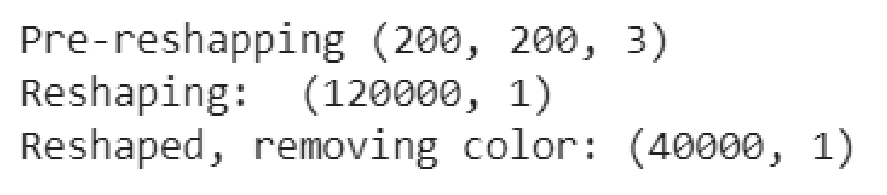
One Image

Analyzing the mathematical structure of an individual image

The structure after removing color and creating a vector for an image

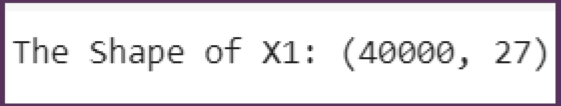


```
The image shape is:  (200, 200, 3)
The image height is: 200
The image width is:  200
The image depth(color channels) is:  3
The total pixels per image for a colored image is:  120000
The total pixels per image for a black and white image is:  40000.0 which is 3 times smaller
```



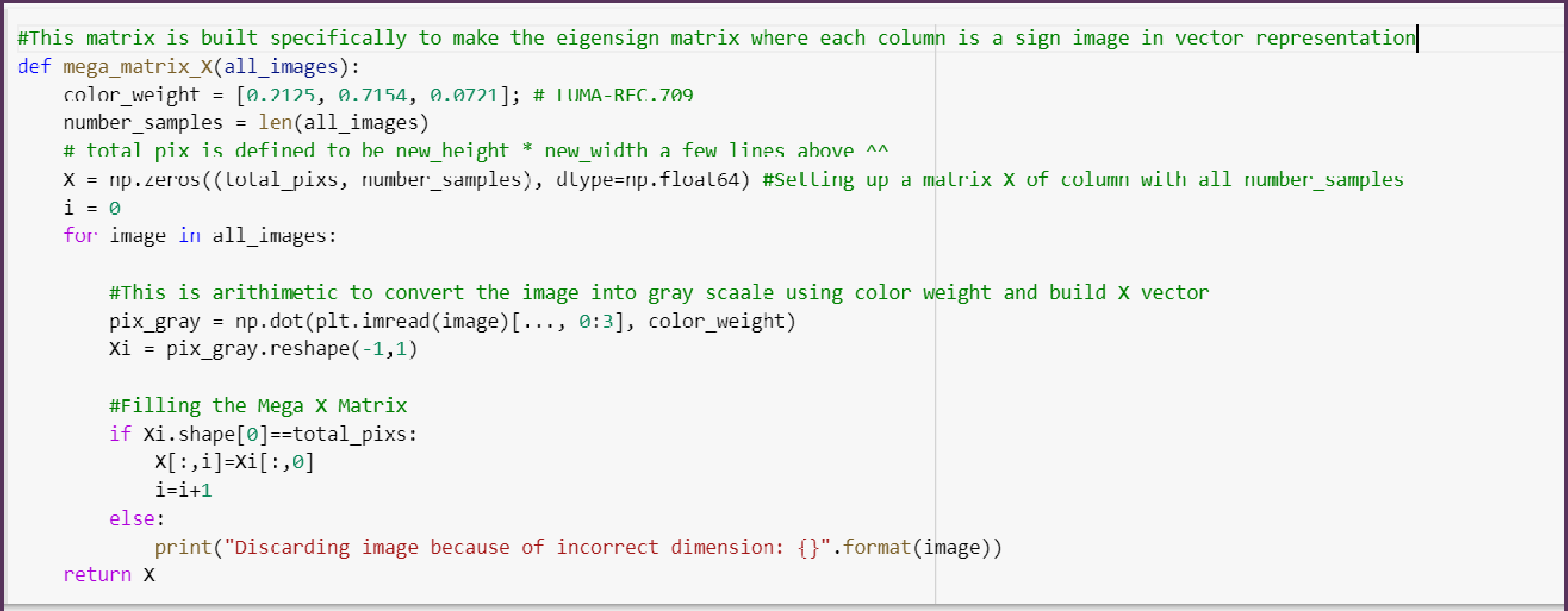
```
Pre-reshapping (200, 200, 3)
Reshaping:  (120000, 1)
Reshaped, removing color: (40000, 1)
```

Mega Matrix X



```
The Shape of X1: (40000, 27)
```

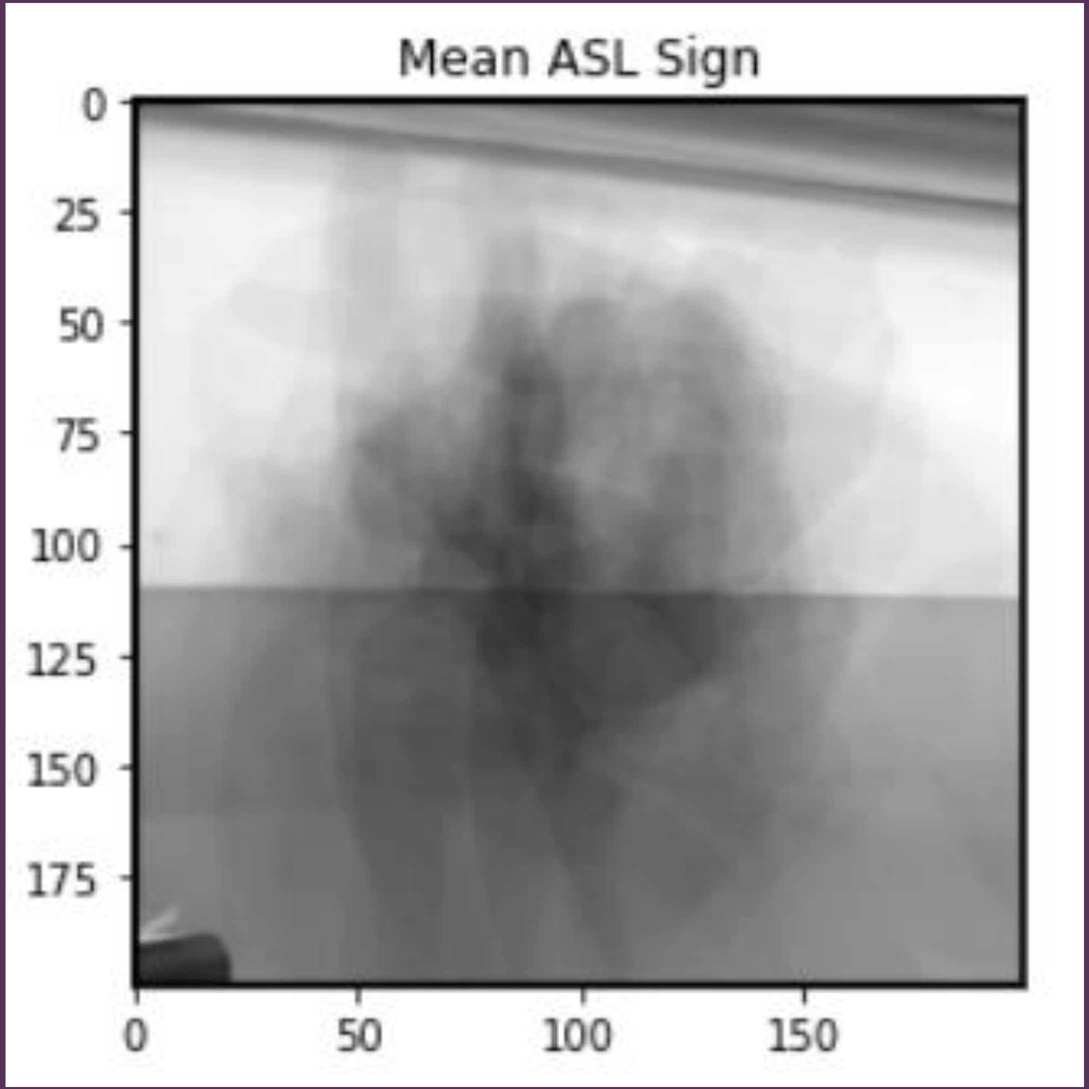
- This matrix is the key to the whole design.
  - Applied for sign recognition
  - Applied for sign detection



```
#This matrix is built specifically to make the eigensign matrix where each column is a sign image in vector representation
def mega_matrix_X(all_images):
    color_weight = [0.2125, 0.7154, 0.0721]; # LUMA-REC.709
    number_samples = len(all_images)
    # total pix is defined to be new_height * new_width a few lines above ^^
    X = np.zeros((total_pixs, number_samples), dtype=np.float64) #Setting up a matrix X of column with all number_samples
    i = 0
    for image in all_images:

        #This is arithimetic to convert the image into gray scaale using color weight and build X vector
        pix_gray = np.dot(plt.imread(image)[..., 0:3], color_weight)
        Xi = pix_gray.reshape(-1,1)

        #Filling the Mega X Matrix
        if Xi.shape[0]==total_pixs:
            X[:,i]=Xi[:,0]
            i=i+1
        else:
            print("Discarding image because of incorrect dimension: {}".format(image))
    return X
```

You might be thinking what is that?

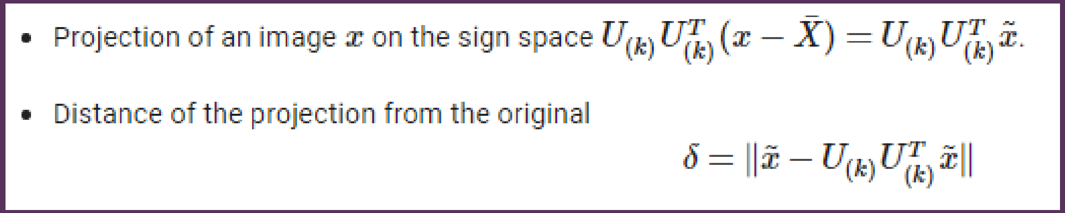
- Is that a pinky?
- Is that the Flash waving his hand?
- Is it Casper the ghost?
- Actually, this is just the Mean Sign of our ASL sign language gestures



This Photo by Unknown Author is licensed under CC BY-SA-NC

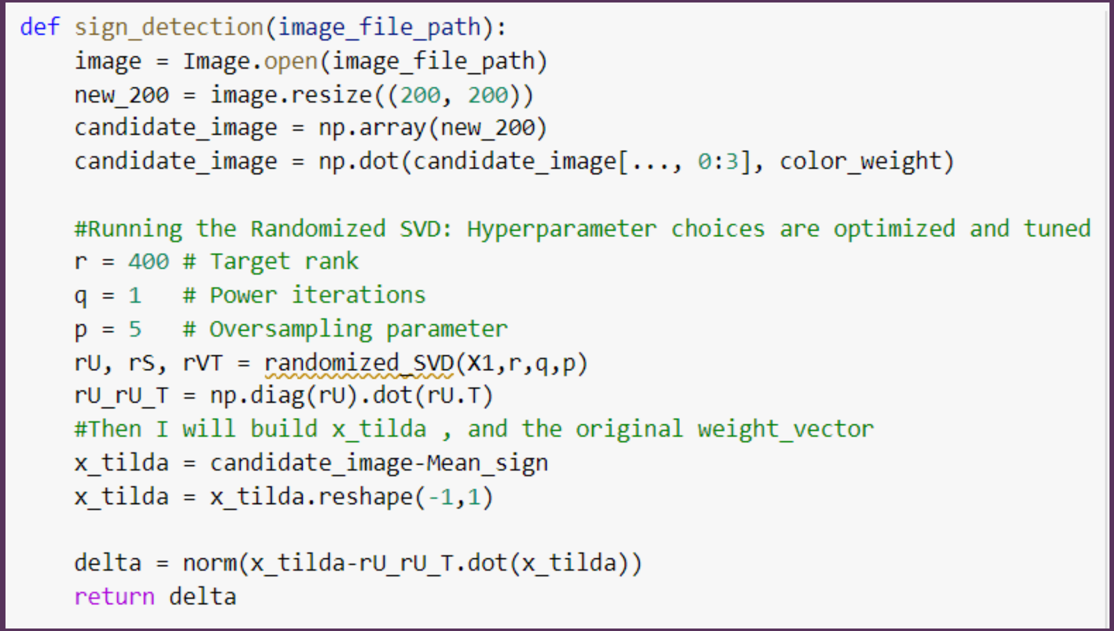
This Photo by Unknown Author is licensed under CC BY-SA

Sign Detection Implemented in Python



- Projection of an image x on the sign space $U_{(k)}U_{(k)}^T(x - \bar{X}) = U_{(k)}U_{(k)}^T\tilde{x}$.
- Distance of the projection from the original

$$\delta = \|\tilde{x} - U_{(k)}U_{(k)}^T\tilde{x}\|$$



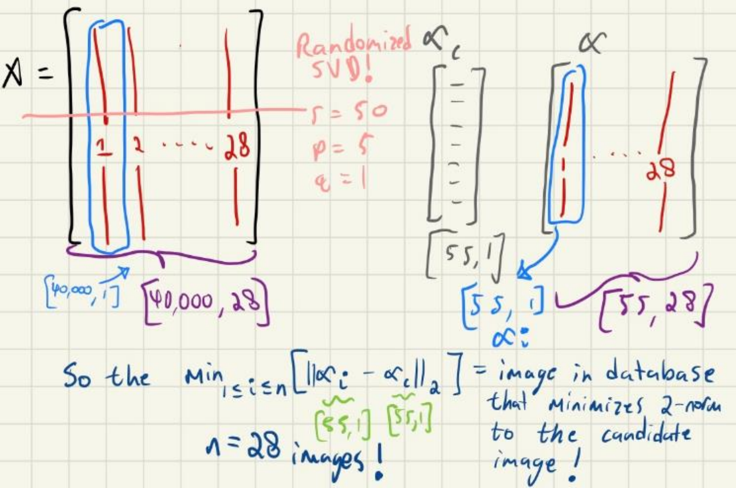
```python
def sign_detection(image_file_path):
    image = Image.open(image_file_path)
    new_200 = image.resize((200, 200))
    candidate_image = np.array(new_200)
    candidate_image = np.dot(candidate_image[..., 0:3], color_weight)

    #Running the Randomized SVD: Hyperparameter choices are optimized and tuned
    r = 400 # Target rank
    q = 1   # Power iterations
    p = 5   # Oversampling parameter
    rU, rS, rVT = randomized_SVD(X1,r,q,p)
    rU_rU_T = np.diag(rU).dot(rU.T)
    #Then I will build x_tilda , and the original weight_vector
    x_tilda = candidate_image-Mean_sign
    x_tilda = x_tilda.reshape(-1,1)

    delta = norm(x_tilda-rU_rU_T.dot(x_tilda))
    return delta
```

L2 Norm|| (x_c - mean face)- proj(x_c on sign space) ||

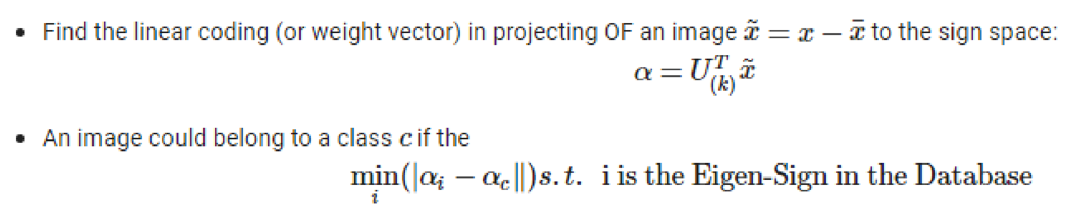
where x_c = candidate image

Image Recognition

- We are applying randomized SVD to the X1 mega matrix so that we can project some candidate weight vectors and compare to the database.
- Simultaneously we are resizing, reshaping, a colored candidate image into a (200,200) image so that math is aligned.
- We are iterating through the columns of



X =
1
2
28
[40,000, 1]
[40,000, 28]
Randomized SVD!
r = 50
p = 5
q = 1
α_c
[55, 1]
α
28
[55, 1]
α_i
[55, 28]
So the $\min_{1\le i\le n}\left[\|\alpha_i - \alpha_c\|_2\right]$ = image in database that minimizes 2-norm to the candidate image!
[55,1]
[55,1]
n = 28 images!

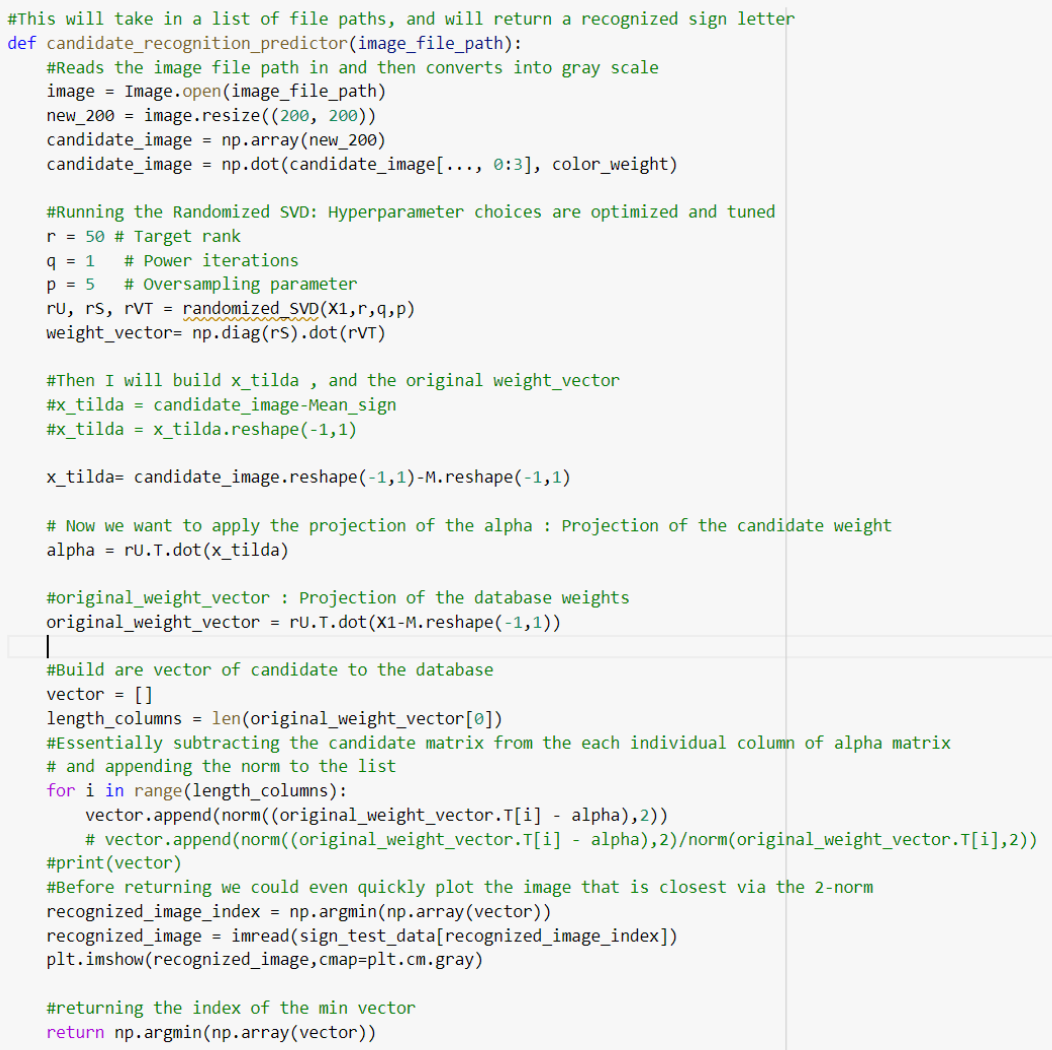


- Find the linear coding (or weight vector) in projecting OF an image $\tilde{x} = x - \bar{x}$ to the sign space:

$$\alpha = U_{(k)}^T \tilde{x}$$

- An image could belong to a class c if the

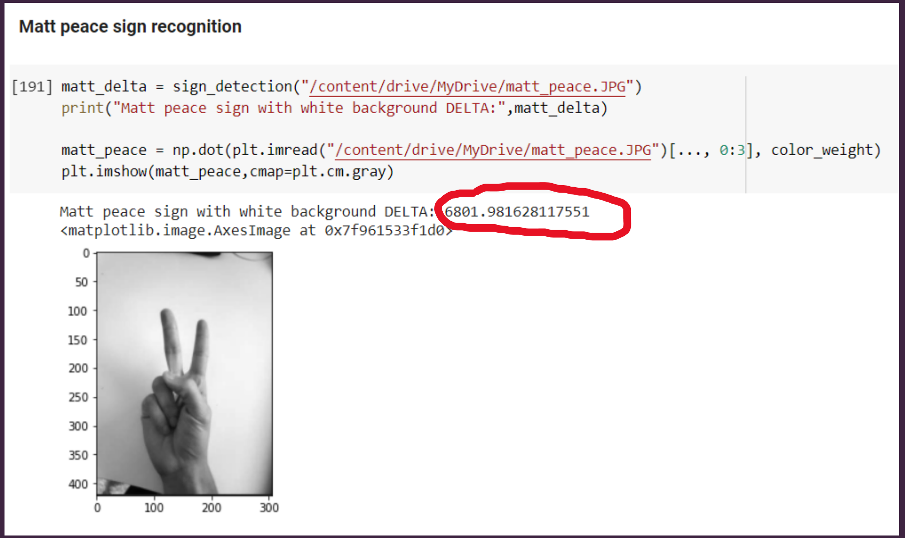
$$\min_i(|\alpha_i - \alpha_c\|) s.t.\ \text{ i is the Eigen-Sign in the Database}$$



```
#This will take in a list of file paths, and will return a recognized sign letter
def candidate_recognition_predictor(image_file_path):
    #Reads the image file path in and then converts into gray scale
    image = Image.open(image_file_path)
    new_200 = image.resize((200, 200))
    candidate_image = np.array(new_200)
    candidate_image = np.dot(candidate_image[..., 0:3], color_weight)

    #Running the Randomized SVD: Hyperparameter choices are optimized and tuned
    r = 50 # Target rank
    q = 1   # Power iterations
    p = 5   # Oversampling parameter
    rU, rS, rVT = randomized_SVD(X1,r,q,p)
    weight_vector= np.diag(rS).dot(rVT)

    #Then I will build x_tilda , and the original weight_vector
    #x_tilda = candidate_image-Mean_sign
    #x_tilda = x_tilda.reshape(-1,1)

    x_tilda= candidate_image.reshape(-1,1)-M.reshape(-1,1)

    # Now we want to apply the projection of the alpha : Projection of the candidate weight
    alpha = rU.T.dot(x_tilda)

    #original_weight_vector : Projection of the database weights
    original_weight_vector = rU.T.dot(X1-M.reshape(-1,1))

    #Build are vector of candidate to the database
    vector = []
    length_columns = len(original_weight_vector[0])
    #Essentially subtracting the candidate matrix from the each individual column of alpha matrix
    # and appending the norm to the list
    for i in range(length_columns):
        vector.append(norm((original_weight_vector.T[i] - alpha),2))
        # vector.append(norm((original_weight_vector.T[i] - alpha),2)/norm(original_weight_vector.T[i],2))
    #print(vector)
    #Before returning we could even quickly plot the image that is closest via the 2-norm
    recognized_image_index = np.argmin(np.array(vector))
    recognized_image = imread(sign_test_data[recognized_image_index])
    plt.imshow(recognized_image,cmap=plt.cm.gray)

    #returning the index of the min vector
    return np.argmin(np.array(vector))
```

Given the Candidate Image of Matthew Benvenuto's Peace Sign

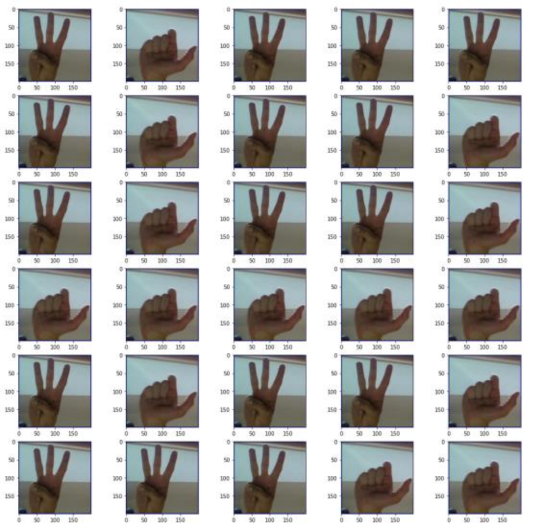
- The algorithm for 30 different runs measuring the L2 norm against the candidate image produced similar looking images but was not fully accurate!



Matt peace sign recognition

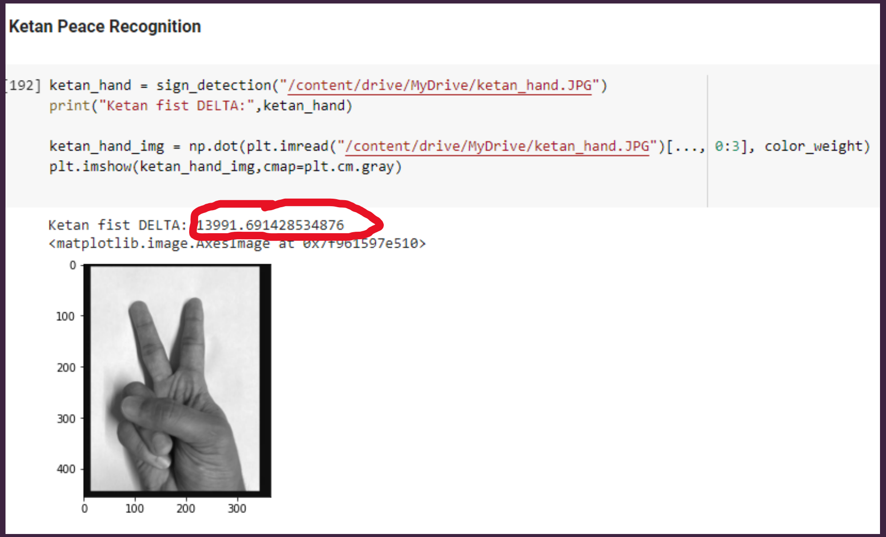
```
[191] matt_delta = sign_detection("/content/drive/MyDrive/matt_peace.JPG")
      print("Matt peace sign with white background DELTA:",matt_delta)

      matt_peace = np.dot(plt.imread("/content/drive/MyDrive/matt_peace.JPG")[..., 0:3], color_weight)
      plt.imshow(matt_peace,cmap=plt.cm.gray)
```

```
Matt peace sign with white background DELTA: 6801.981628117551
<matplotlib.image.AxesImage at 0x7f961533f1d0>
```



Given the Candidate Image of Ketan Patel's Peace Sign

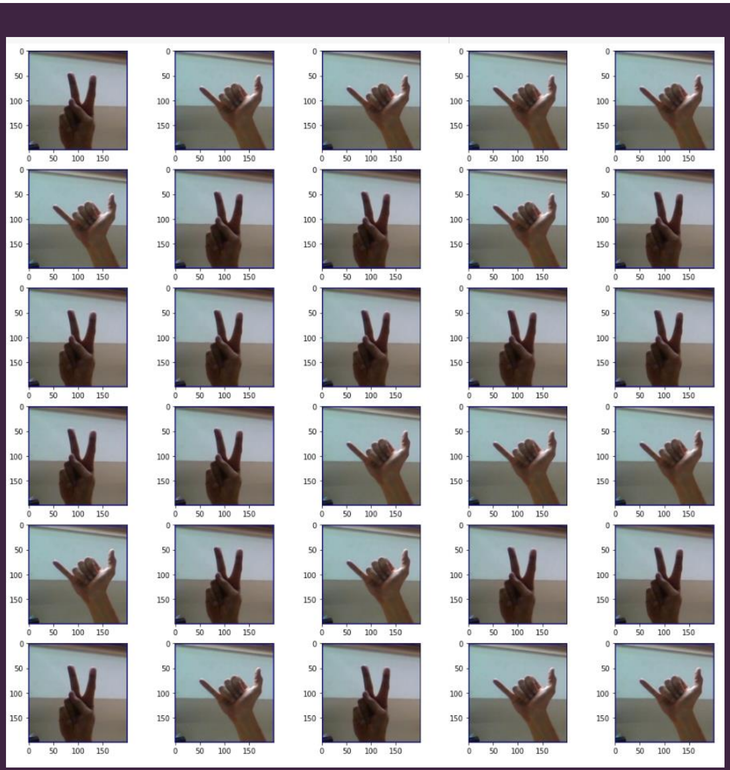
- The algorithm for 30 different runs measuring the L2 norm against the candidate image produced similar looking images and was super accurate recognizing the peace sign!



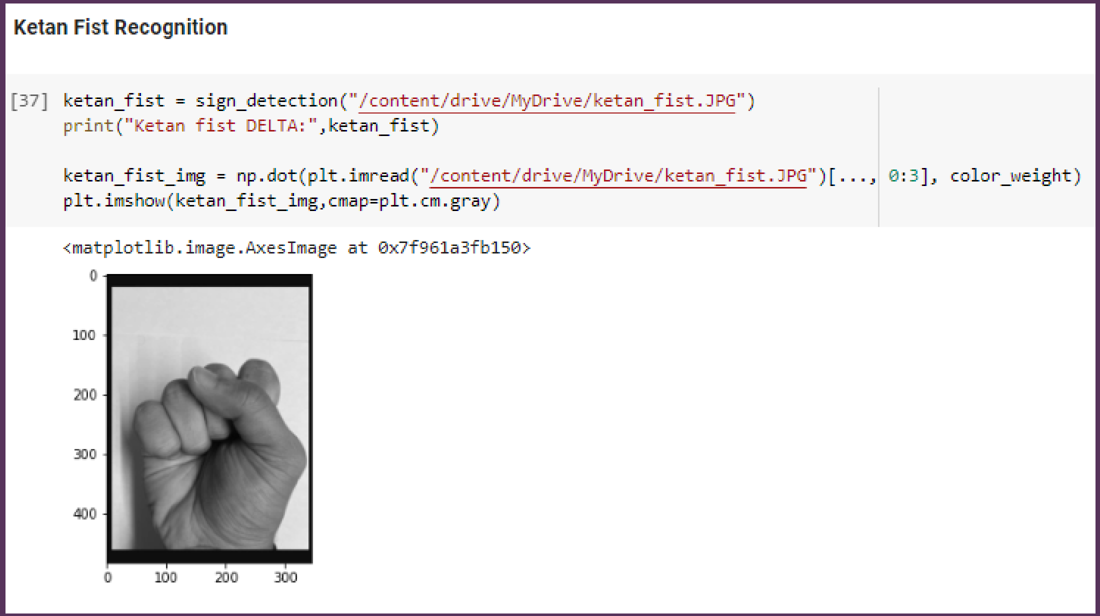
Ketan Peace Recognition

```
[192] ketan_hand = sign_detection("/content/drive/MyDrive/ketan_hand.JPG")
      print("Ketan fist DELTA:",ketan_hand)

      ketan_hand_img = np.dot(plt.imread("/content/drive/MyDrive/ketan_hand.JPG")[..., 0:3], color_weight)
      plt.imshow(ketan_hand_img,cmap=plt.cm.gray)
```

```
Ketan fist DELTA: 13991.691428534876
<matplotlib.image.AxesImage at 0x7f961597e510>
```



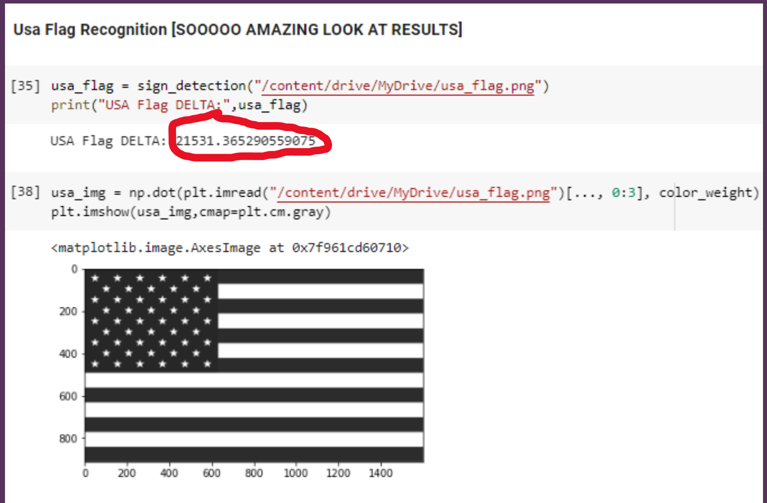
The Fist Recognition was the best yet, not exact but accurate!



Ketan Fist Recognition

```
[37] ketan_fist = sign_detection("/content/drive/MyDrive/ketan_fist.JPG")
     print("Ketan fist DELTA:",ketan_fist)

     ketan_fist_img = np.dot(plt.imread("/content/drive/MyDrive/ketan_fist.JPG")[..., 0:3], color_weight)
     plt.imshow(ketan_fist_img,cmap=plt.cm.gray)

     <matplotlib.image.AxesImage at 0x7f961a3fb150>
```

If we were to use a different candidate image like a black and white USA flag, what would you expect?

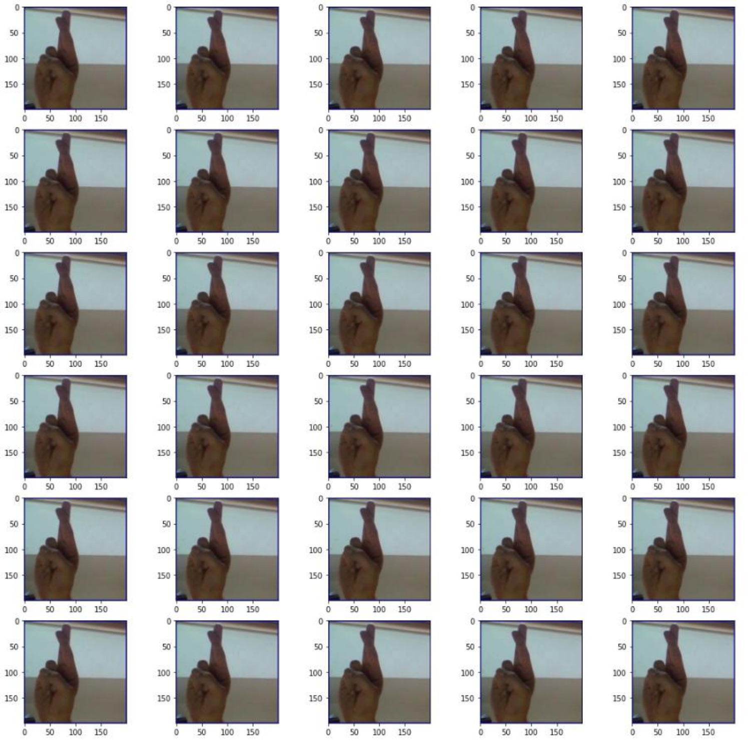


Usa Flag Recognition [SOOOOO AMAZING LOOK AT RESULTS]

```
[35] usa_flag = sign_detection("/content/drive/MyDrive/usa_flag.png")
     print("USA Flag DELTA:",usa_flag)

     USA Flag DELTA: 21531.365290559075

[38] usa_img = np.dot(plt.imread("/content/drive/MyDrive/usa_flag.png")[..., 0:3], color_weight)
     plt.imshow(usa_img,cmap=plt.cm.gray)

     <matplotlib.image.AxesImage at 0x7f961cd60710>
```

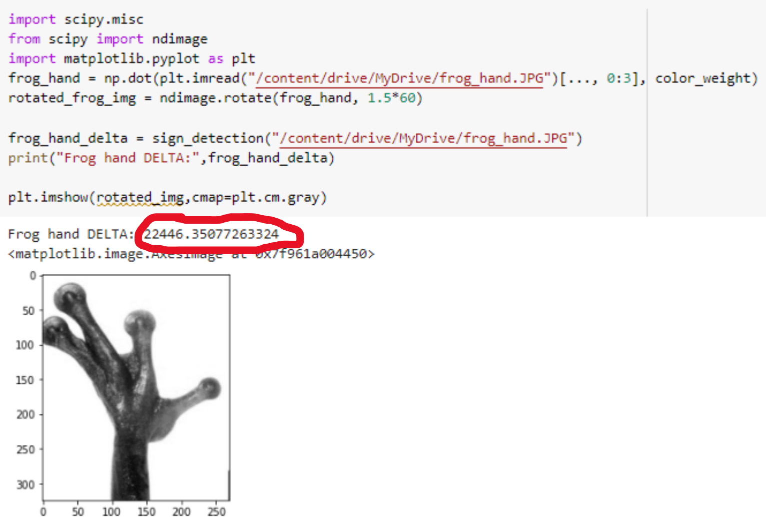
Photo by Luke Michael on Unsplash

Photo by Nathan Fertig on Unsplash



Frog Hand

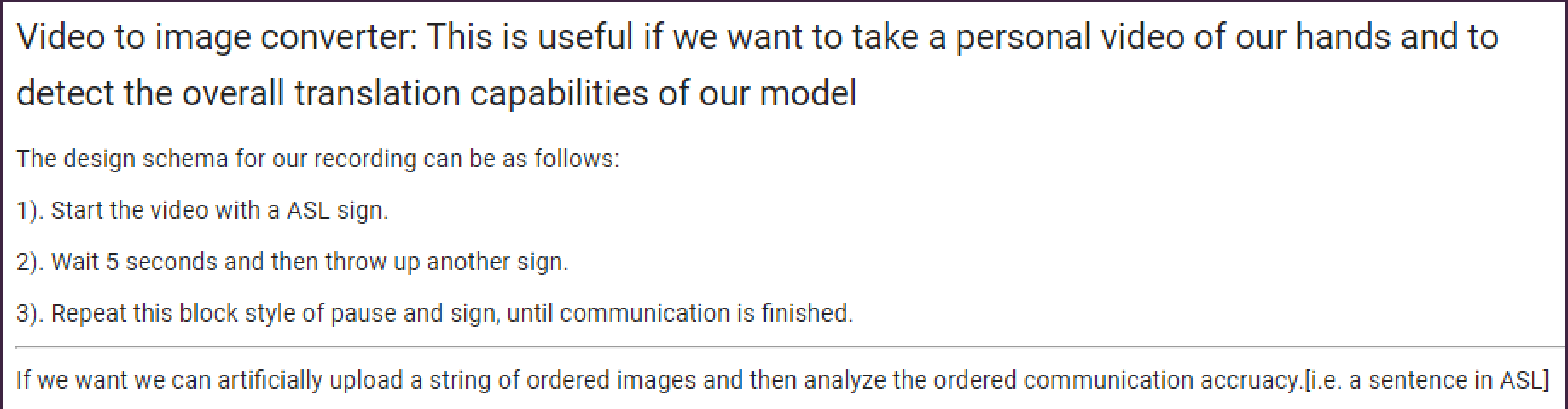
- Image Recognition results make sense choosing the gesture with most curves, and the one that best represents forearm in picture.



```
import scipy.misc
from scipy import ndimage
import matplotlib.pyplot as plt
frog_hand = np.dot(plt.imread("/content/drive/MyDrive/frog_hand.JPG")[..., 0:3], color_weight)
rotated_frog_img = ndimage.rotate(frog_hand, 1.5*60)

frog_hand_delta = sign_detection("/content/drive/MyDrive/frog_hand.JPG")
print("Frog hand DELTA:",frog_hand_delta)

plt.imshow(rotated_img,cmap=plt.cm.gray)
```

```
Frog hand DELTA: 22446.35077263324
<matplotlib.image.AxesImage at 0x7f961a004450>
```

Video to Image Converter: This is a cool application of this Eigen-Signs



Video to image converter: This is useful if we want to take a personal video of our hands and to detect the overall translation capabilities of our model

The design schema for our recording can be as follows:

1). Start the video with a ASL sign.

2). Wait 5 seconds and then throw up another sign.

3). Repeat this block style of pause and sign, until communication is finished.

If we want we can artificially upload a string of ordered images and then analyze the ordered communication accruacy.[i.e. a sentence in ASL]

Lets try recording a live video and cutting the image frame by frame! There is some accuracy but could be better

Video Frame by Frame Sliced

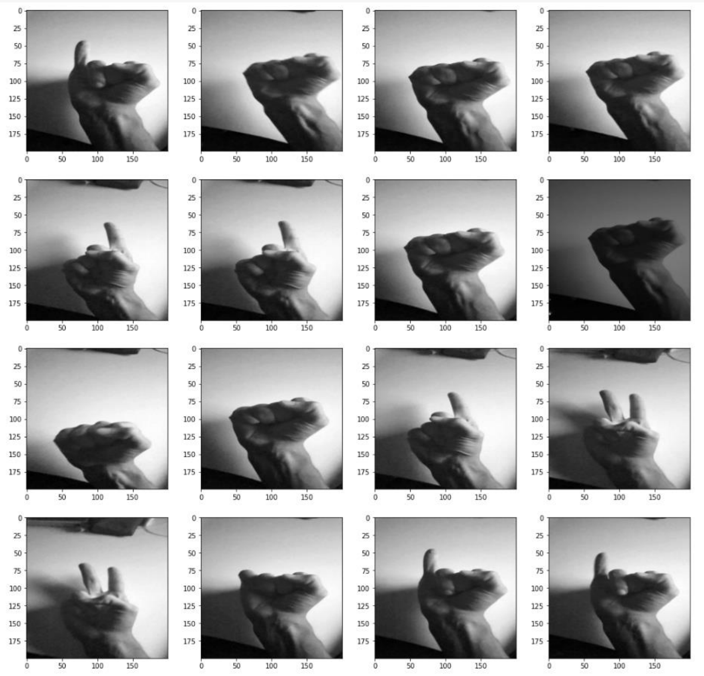
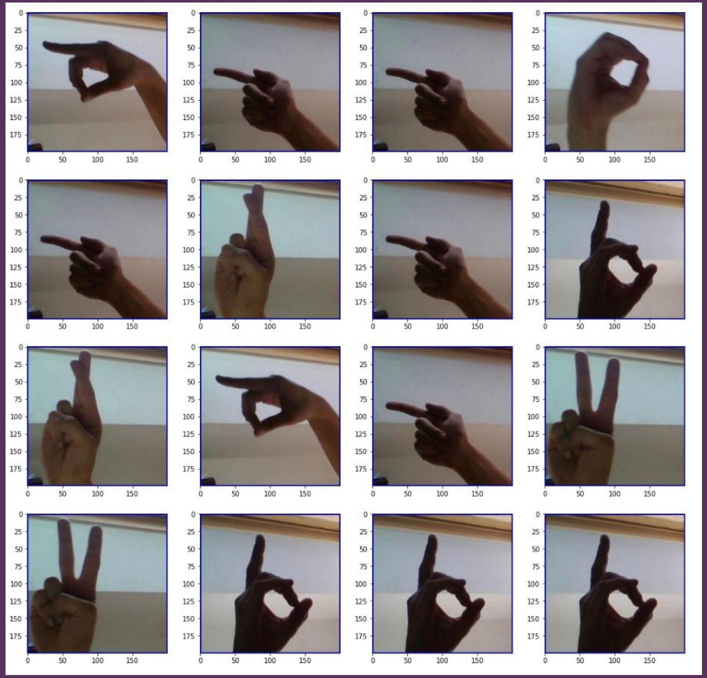


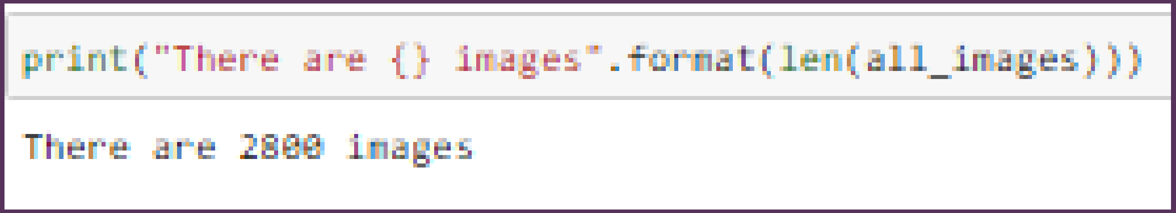
Image Recognition

Corresponding Candidate in Database



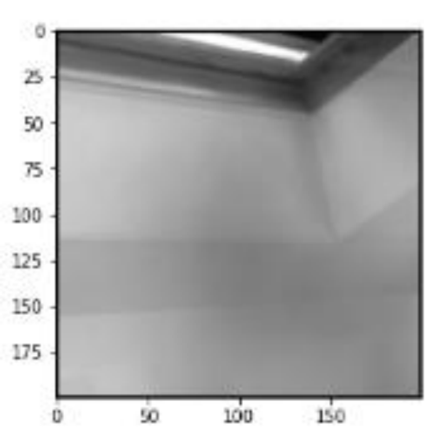
The Video Model's gesture recognition did not produce the best quality images, but even still the gesture model managed pretty well, **bottom two rows are close , and third column nice**

2800 images

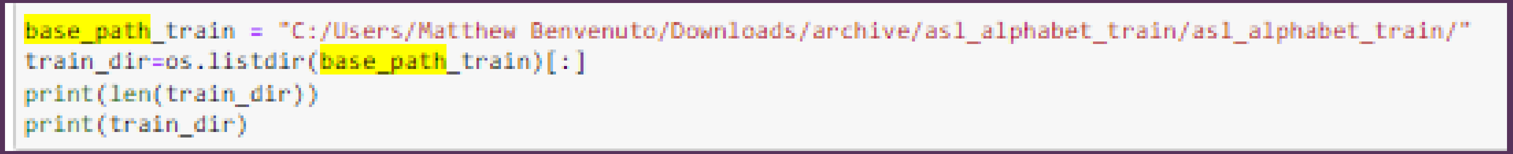


```
print("There are {} images".format(len(all_images)))
```

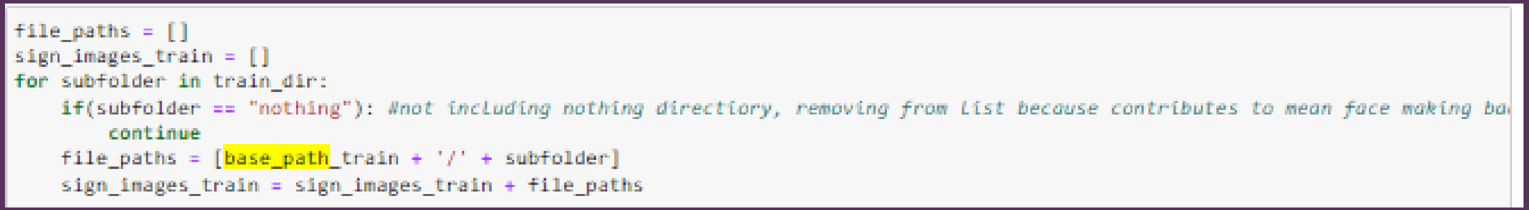
```
There are 2800 images
```



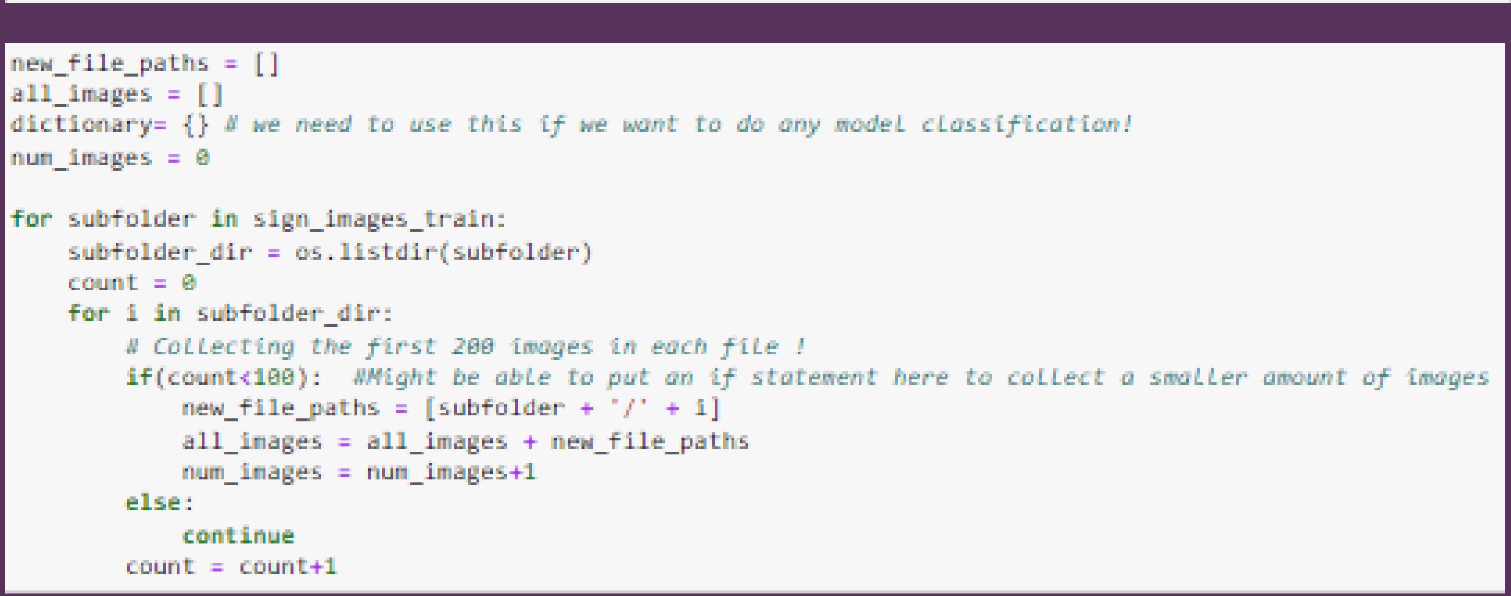
Not so good mean sign, cause for bad results!



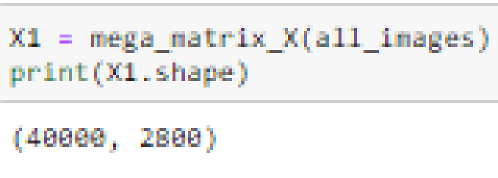
```
base_path_train = "C:/Users/Matthew Benvenuto/Downloads/archive/asl_alphabet_train/asl_alphabet_train/"
train_dir=os.listdir(base_path_train)[:]
print(len(train_dir))
print(train_dir)
```



```
file_paths = []
sign_images_train = []
for subfolder in train_dir:
    if(subfolder == "nothing"): #not including nothing directiory, removing from list because contributes to mean face making ba
        continue
    file_paths = [base_path_train + '/' + subfolder]
    sign_images_train = sign_images_train + file_paths
```



```
new_file_paths = []
all_images = []
dictionary= {} # we need to use this if we want to do any model classification!
num_images = 0

for subfolder in sign_images_train:
    subfolder_dir = os.listdir(subfolder)
    count = 0
    for i in subfolder_dir:
        # Collecting the first 200 images in each file !
        if(count<100):  #Might be able to put an if statement here to collect a smaller amount of images
            new_file_paths = [subfolder + '/' + i]
            all_images = all_images + new_file_paths
            num_images = num_images+1
        else:
            continue
        count = count+1
```



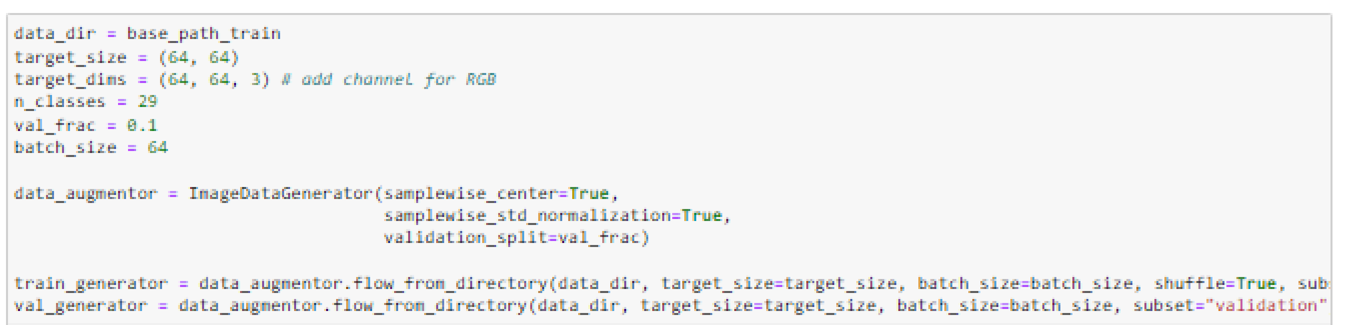
```
X1 = mega_matrix_X(all_images)
print(X1.shape)
```

```
(40000, 2800)
```

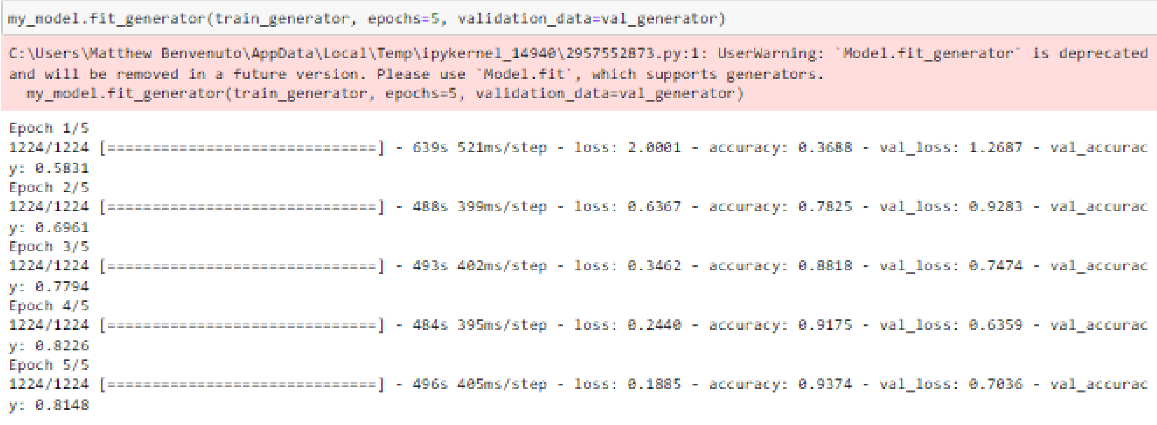
Future Directions

- Using more data points, 2800 training gestures and getting a more accurate mean sign!
- Optimizing the video to image frame converter so that the video stream can recognize gestures more precisely.
- Using adaptive thresholding, edge detection, transforming aligned images, blurring background, and hand detection to control the variables effecting the errors.

Machine Learning Model Architecture: After 5 epochs- 93% accuracy



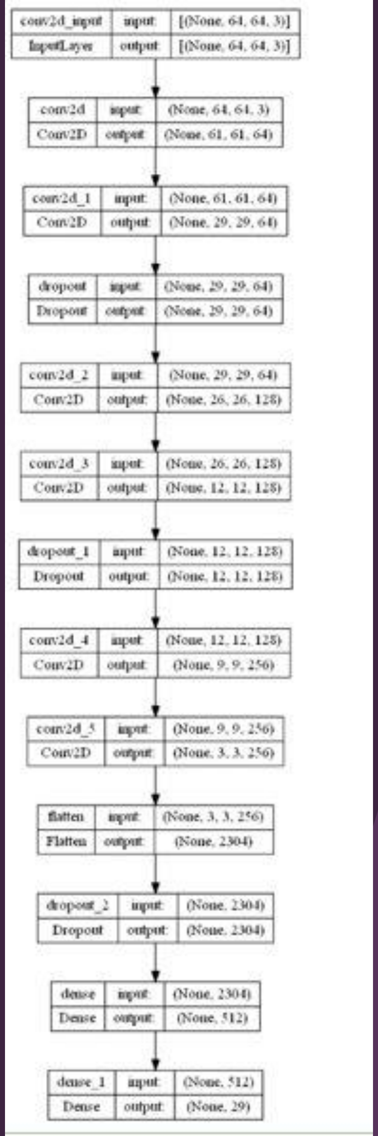
```
data_dir = base_path_train
target_size = (64, 64)
target_dims = (64, 64, 3) # add channel for RGB
n_classes = 29
val_frac = 0.1
batch_size = 64

data_augmentor = ImageDataGenerator(samplewise_center=True,
                                    samplewise_std_normalization=True,
                                    validation_split=val_frac)

train_generator = data_augmentor.flow_from_directory(data_dir, target_size=target_size, batch_size=batch_size, shuffle=True, sub
val_generator = data_augmentor.flow_from_directory(data_dir, target_size=target_size, batch_size=batch_size, subset="validation"
```



```
my_model.fit_generator(train_generator, epochs=5, validation_data=val_generator)
```

```
C:\Users\Matthew Benvenuto\AppData\Local\Temp\ipykernel_14940\2957552873.py:1: UserWarning: `Model.fit_generator` is deprecated
and will be removed in a future version. Please use `Model.fit`, which supports generators.
  my_model.fit_generator(train_generator, epochs=5, validation_data=val_generator)
```

```
Epoch 1/5
1224/1224 [==============================] - 639s 521ms/step - loss: 2.0001 - accuracy: 0.3688 - val_loss: 1.2687 - val_accurac
y: 0.5831
Epoch 2/5
1224/1224 [==============================] - 488s 399ms/step - loss: 0.6367 - accuracy: 0.7825 - val_loss: 0.9283 - val_accurac
y: 0.6961
Epoch 3/5
1224/1224 [==============================] - 493s 402ms/step - loss: 0.3462 - accuracy: 0.8818 - val_loss: 0.7474 - val_accurac
y: 0.7794
Epoch 4/5
1224/1224 [==============================] - 484s 395ms/step - loss: 0.2440 - accuracy: 0.9175 - val_loss: 0.6359 - val_accurac
y: 0.8226
Epoch 5/5
1224/1224 [==============================] - 496s 405ms/step - loss: 0.1885 - accuracy: 0.9374 - val_loss: 0.7036 - val_accurac
y: 0.8148
```



```
conv2d_input | input:  | [(None, 64, 64, 3)]
InputLayer   | output: | [(None, 64, 64, 3)]

conv2d | input:  | (None, 64, 64, 3)
Conv2D | output: | (None, 61, 61, 64)

conv2d_1 | input:  | (None, 61, 61, 64)
Conv2D   | output: | (None, 29, 29, 64)

dropout | input:  | (None, 29, 29, 64)
Dropout | output: | (None, 29, 29, 64)

conv2d_2 | input:  | (None, 29, 29, 64)
Conv2D   | output: | (None, 26, 26, 128)

conv2d_3 | input:  | (None, 26, 26, 128)
Conv2D   | output: | (None, 12, 12, 128)

dropout_1 | input:  | (None, 12, 12, 128)
Dropout   | output: | (None, 12, 12, 128)

conv2d_4 | input:  | (None, 12, 12, 128)
Conv2D   | output: | (None, 9, 9, 256)

conv2d_5 | input:  | (None, 9, 9, 256)
Conv2D   | output: | (None, 3, 3, 256)

flatten | input:  | (None, 3, 3, 256)
Flatten | output: | (None, 2304)

dropout_2 | input:  | (None, 2304)
Dropout   | output: | (None, 2304)

dense | input:  | (None, 2304)
Dense | output: | (None, 512)

dense_1 | input:  | (None, 512)
Dense   | output: | (None, 29)
```

Work Cited

- All images have been properly cited within the slides and below:
  - This Photo by Unknown Author is licensed under CC BY-SA-NC
  - Photo by Luke Michael on Unsplash
  - Photo by Nathan Fertig on Unsplash