

Digital Image Processing Final Project Report: ASL Hand Gesture Image Classification using Convolutional Neural Network

By Arunkumar Ramachandran

ECE 5256

4/28/2021



Contents

Table of Figures	4
Table of Tables	5
1. Introduction	6
2. Goal	6
3. Dataset	6
4. Preprocessing Dataset	7
4.1. Train-Test Split:	7
4.2. Normalize Function	7
4.2.1. Normalizing train and test data by dividing it with 255	7
4.2.2. Normalize Function using Mean And Standard Deviation:	8
4.3. Resize the images:	9
5. Data Augmentation:	10
5.1. Horizontal and Vertical Shift Augmentation:	11
5.1.1. Horizontal Shift Augmentation:	11
5.1.2. Vertical Shift Augmentation:	12
5.2. Horizontal and Vertical Flip Augmentation:	13
5.2.1. Horizontal Flip Augmentation:	13
5.2.2. Vertical Flip Augmentation:	14
5.3. Random Rotation:	15
5.4. Random Brightness Augmentation:	16
5.5. Random Zoom Augmentation:	17
5.6. Data Augmentation on different network architectures:	18
6. Network Architecture	19
7. Modern CNN Network Architecture	19
8. Ensemble Learning:	21
8.1. Mini VGG:	21
8.2. VGG16:	23
8.3. AlexNet:	24
8.4. VGG19:	24
9. Activation Function	26
10. Loss Function	26
11. Tune hyperparameters	26

12.	Optimizer	27
13.	Kernel Weight Initialization Method:	27
14.	Final Code	28
15.	Discussions.....	30

Table of Figures

Figure 1 Images in my ASL hand gesture dataset	6
Figure 2 Train-Test split to tune hyperparameters.....	7
Figure 3 Normalization method dividing train and test data with 255.	8
Figure 4 Normalization Using Mean and Standard Deviation.	8
Figure 5 Testing Different sizes for Resizing.	9
Figure 6 Images after resizing.	9
Figure 7 Code containing Images after resizing to (75x75)	10
Figure 8 Images after resizing to (75x75).....	10
Figure 9 A code used in Horizontal shift augmentation.....	11
Figure 10 Output obtained after using Horizontal shift augmentation.....	12
Figure 11 Code used in vertical shift augmentation.	12
Figure 12 Output obtained after using vertical shift augmentation.....	13
Figure 13 Code used in Horizontal flip augmentation.	14
Figure 14 Output obtained after using horizontal flip augmentation.	14
Figure 15 Code used in Vertical flip augmentation.....	15
Figure 16 Output obtained after using vertical shift augmentation.....	15
Figure 17 Code used in random rotation augmentation.	16
Figure 18 Output obtained after rotation augmentation.....	16
Figure 19 Code used in random brightness augmentation.	17
Figure 20 Output obtained after using random brightness augmentation.	17
Figure 21 Code used in random zoom augmentation.	18
Figure 22 Output obtained after using random zoom augmentation.	18
Figure 23 Identity Block Of ResNet50 Architecture.	20
Figure 24 Convolutional Block Of ResNet50 Architecture.	20
Figure 25 ResNet50 Architecture containing both identity and convolution block.	21
Figure 26 Ensemble of nets output of MiniVGG.	22
Figure 27 Snapshot Ensemble of nets output of MIniVGG.	22
Figure 28 Ensemble of nets output of VGG16.	23
Figure 29 Snapshot Ensemble of nets output of VGG16.	23
Figure 30 Ensemble of nets output of AlexNet.....	24
Figure 31 Snapshot Ensemble of nets output of AlexNet.....	24
Figure 32 Ensemble of nets output of VGG19.	25
Figure 33 Snapshot Ensemble of nets output of VGG19.	25
Figure 34 Tuning Hyperparameter.....	27
Figure 35 70000 dataset values sent for training final model.	28
Figure 36 Final Accuracy obtained after training model.....	29
Figure 37 Graph of Final model obtained after training.	29

Table of Tables

Table 1 Network Architecture and accuracy values after using image generator.	19
Table 2 Network Architecture and their accuracy.	19
Table 3 Comparison between Ensemble of Nets and Snapshot Ensemble.	26
Table 4 Activation Function and their accuracy.	26
Table 5 Loss Function and their accuracy.	26
Table 6 Optimizer and their accuracy.	27
Table 7 Kernel Initializer and their accuracy.	28
Table 8 Best values obtained on tuning the hyperparameters.	30

1. Introduction

Gestures are a form of nonverbal communication in which visible bodily actions are used to communicate important messages, either in speech or together and in parallel with spoken words. Gestures include movement of the hands, face, or other parts of the body. Physical non-verbal communication such as purely expressive displays, proxemics, or displays of joint attention differ from gestures, which communicate specific messages.

2. Goal

The main goal of this project is to classify images according to their respective labels and train the model to obtain good accuracy.

3. Dataset

The data set is used from the site: <https://www.kaggle.com/grassknoted/asl-alphabet>

There are 87K images, which are of 200x200x3 pixels. The image of the classification can be shown as follows:

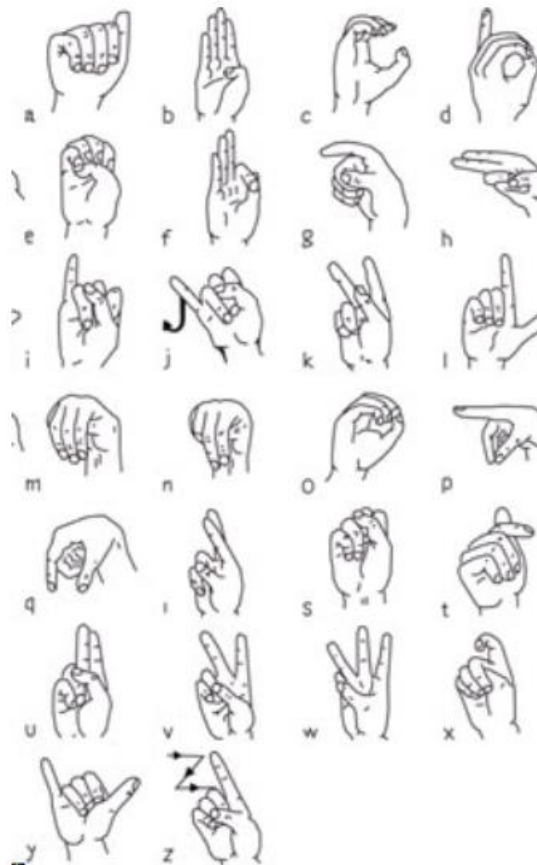


Figure 1 Images in my ASL hand gesture dataset

The dataset contains 29 classes, of which 26 are for letters A-Z and three classes for SPACE, DELETE, and NOTHING. I feel the dataset is challenging because of the diversity in the classes, and I might have to resize the images and make sure I do not lose much of the details from the image.

4. Preprocessing Dataset

This section describes the assumptions and procedures used to set up the analysis and obtain solutions for them.

4.1. Train-Test Split:

To tune the hyperparameters, a total of 10,000 values were sent and the train test split used can be shown in the following figure:

```
[ ] 1 #####
2 #Normalizing 1
3
4 (trainX, valX, trainY, valY) = train_test_split(X[:10000], y[:10000], test_size=0.20, random_state=1)
5
6 trainX = trainX.astype('float32')/255.0
7 valX = valX.astype('float32')/255.0
8
9 print(trainX.shape)
10 print(trainY.shape)
11 print(valX.shape)
12 print(valY.shape)

(8000, 75, 75, 3)
(8000,)
(2000, 75, 75, 3)
(2000,)
```

Figure 2 Train-Test split to tune hyperparameters.

As seen in the above figure, 10,000 datapoints were split and 8000 datapoints were used for train set and 2000 was used for validation/test set.

4.2. Normalize Function

In this project, two normalization methods were used. The two normalization methods used are:

- Normalizing train and test data by dividing it with 255
- Min-Max Function

4.2.1. Normalizing train and test data by dividing it with 255

The train and test data were normalized by dividing it by 255. The function can be depicted as:

- $\text{trainX}/255$
- $\text{valX}/255$

```

[ ] 1 X = data['features']
    2 y = data['labels']

Normalization Methods

Normalizing Method 1

[ ] 1 #####
    2 #Normalizing 1
    3
    4 (trainX, valX, trainY, valY) = train_test_split(X[:10000], y[:10000], test_size=0.20, random_state=1)
    5
    6 trainX = trainX.astype('float32')/255.0
    7 valX = valX.astype('float32')/255.0
    8
    9 print(trainX.shape)
   10 print(trainY.shape)
   11 print(valX.shape)
   12 print(valY.shape)

(8000, 75, 75, 3)
(8000,)
(2000, 75, 75, 3)
(2000,)

```

Figure 3 Normalization method dividing train and test data with 255.

4.2.2. Normalize Function using Mean And Standard Deviation:

The test and train set data were normalized by using the mean and standard deviation method.

```

[ ] 1 from torchvision import transforms
    2
    3 (trainX, valX, trainY, valY) = train_test_split(X[:10000], y[:10000], test_size=0.20, random_state=1)
    4
    5 #Mean and std deviation for trainX
    6 mean = trainX.mean(axis=(0,1,2))
    7 std = trainX.std(axis=(0,1,2))
    8 print(mean,std)
    9
   10 #Mean and std for valX
   11 mean_2 = valX.mean(axis=(0,1,2))
   12 std_2 = valX.std(axis=(0,1,2))
   13 print(mean_2,std_2)
   14
   15 #Normalize trainX
   16
   17 train_transform = transforms.Compose([transforms.ToPILImage(),
   18                                     transforms.ToTensor(),
   19                                     transforms.Normalize(mean,std)])
   20 #Normalize valX
   21 valid_transform = transforms.Compose([transforms.ToPILImage(),
   22                                     transforms.ToTensor(),
   23                                     transforms.Normalize(mean_2,std_2)])
   24
   25
   26 print(trainX.shape)
   27 print(valX.shape)

[131.42447873 127.0583638 132.15750736 [66.65213616 64.49961312 57.4409279 ]
 [132.03767502 127.70541129 132.75568729] [66.42018716 64.28376307 57.1662313 ]
 (8000, 75, 75, 3)
 (2000, 75, 75, 3)

```

Figure 4 Normalization Using Mean and Standard Deviation.

4.3. Resize the images:

To pick the right image size, random values were given to test how the images look after resizing, and an appropriate size was selected for resizing all images. The images contained in the train and test dataset have been resized from (200*200*3) to (75*75*3).

```
☆
Help Last saved at 6:05 PM
+ Code + Text
Connect Editing
Testing different images by resizing

1 resize = [(30,30),(50,50),(75,75),(80,80),(85,85),(90,90),(95,95),(100,100),(110,110)]
2
3
4
5 image = cv2.cvtColor(cv2.imread('/content/drive/My Drive/ASL/asl_alphabet_train/asl_alphabet_train/A/A1000.jpg'), cv2.COLOR_BGR2RGB)
6
7 h,ax = plt.subplots(3,3)
8 h.subplots_adjust(0,0,2,2)
9 index = 0
10 for i in range(0,3,1):
11     for j in range(0,3,1):
12
13         # rnd_number = randint(0,len(Images))
14         ax[i,j].imshow(cv2.resize(image,resize[index]))
15         ax[i,j].set_title('Size: ' + str(resize[index]))
16         index +=1
```

Figure 5 Testing Different sizes for Resizing.

The output after resizing the images in folder A can be shown as follows:

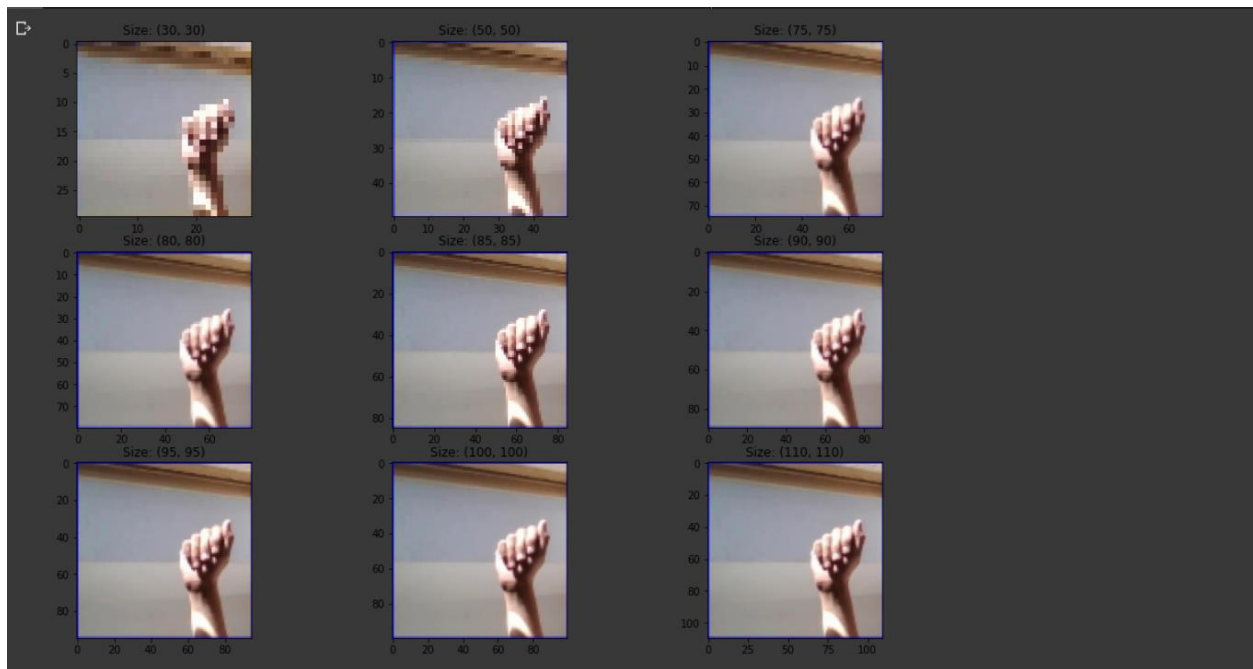


Figure 6 Images after resizing.

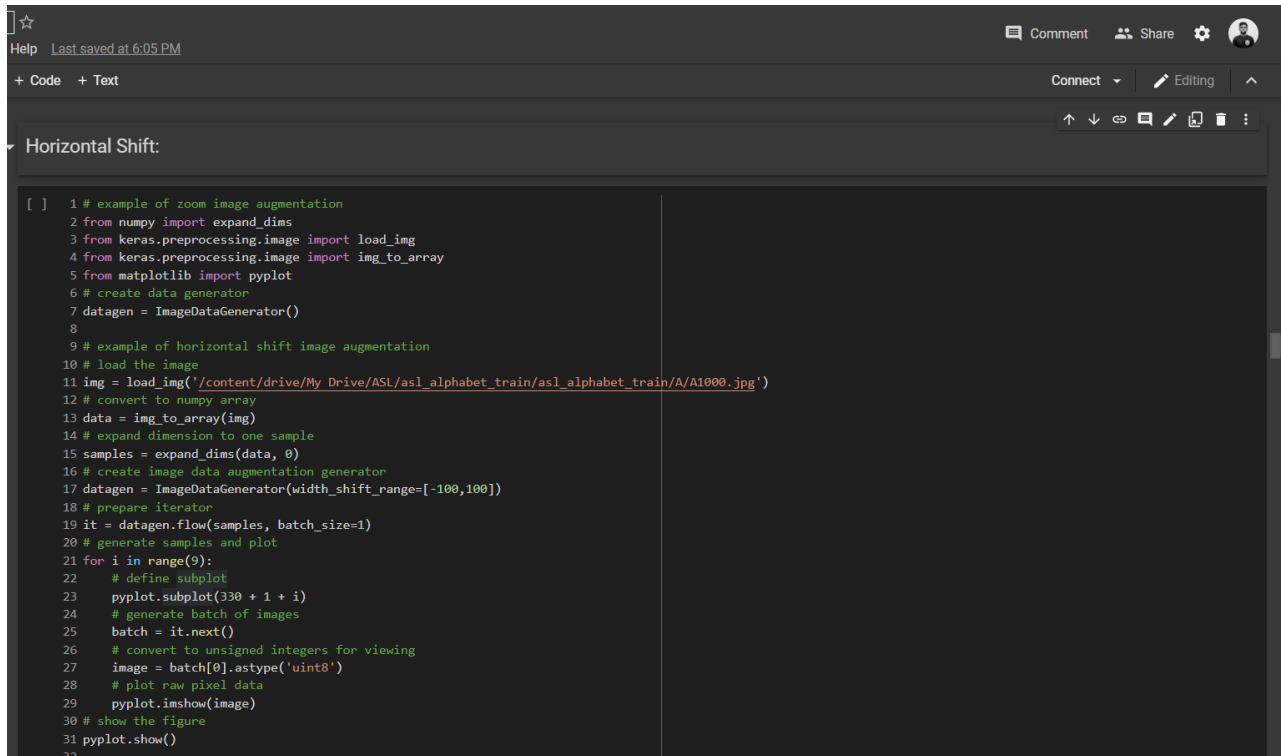
As seen above, the images in folder A (containing hand gestures for alphabet A) have been resized according to the given sizes. Amongst them, size (75x75) was selected as a lot of information was not missing from the images, and it looked just about the right size. After selecting (75x75), I ran another code, as shown below, to check how other images looked. They can be seen as follows:

5.1. Horizontal and Vertical Shift Augmentation:

A horizontal and vertical shift feature was used, and the images were tested with different values and then plotted.

5.1.1. Horizontal Shift Augmentation:

A horizontal shift augmentation used the “**width_shift_range**” function so that depending on the value provided, the images tend to move horizontally. The function and the range of values provided can be shown as follows:

The image shows a Jupyter Notebook interface with a dark theme. At the top, there's a toolbar with icons for Help, Comment, Share, and a user profile. Below the toolbar, there's a tab labeled 'Horizontal Shift:'. The main area contains a code cell with the following Python code:

```
[ ] 1 # example of zoom image augmentation
2 from numpy import expand_dims
3 from keras.preprocessing.image import load_img
4 from keras.preprocessing.image import img_to_array
5 from matplotlib import pyplot
6 # create data generator
7 datagen = ImageDataGenerator()
8
9 # example of horizontal shift image augmentation
10 # load the image
11 img = load_img('/content/drive/My Drive/ASL/asl_alphabet_train/asl_alphabet_train/A/A1000.jpg')
12 # convert to numpy array
13 data = img_to_array(img)
14 # expand dimension to one sample
15 samples = expand_dims(data, 0)
16 # create image data augmentation generator
17 datagen = ImageDataGenerator(width_shift_range=[-100,100])
18 # prepare iterator
19 it = datagen.flow(samples, batch_size=1)
20 # generate samples and plot
21 for i in range(9):
22     # define subplot
23     pyplot.subplot(330 + 1 + i)
24     # generate batch of images
25     batch = it.next()
26     # convert to unsigned integers for viewing
27     image = batch[0].astype('uint8')
28     # plot raw pixel data
29     pyplot.imshow(image)
30 # show the figure
31 pyplot.show()
32
```

Figure 9 A code used in Horizontal shift augmentation.

The output of the augmentation can be shown as follows:

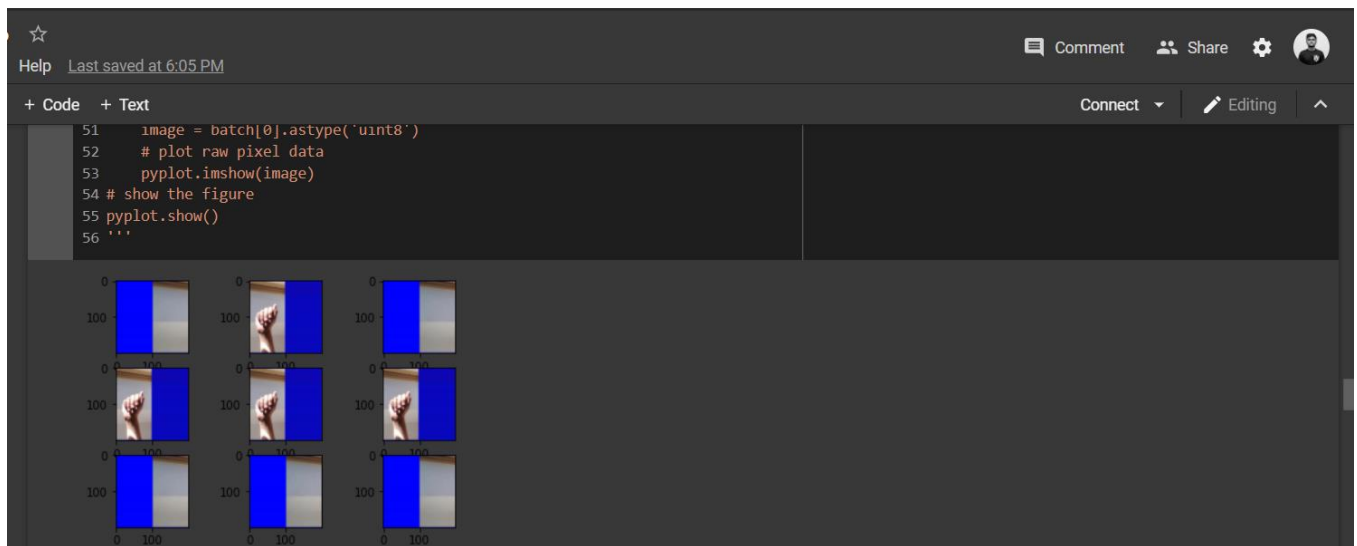


Figure 10 Output obtained after using Horizontal shift augmentation.

5.1.2. Vertical Shift Augmentation:

A Vertical shift augmentation used “the **height_shift_range**” function so that depending on the value provided, the images tend to move vertically. The function and the range of values provided can be shown in the following figure.

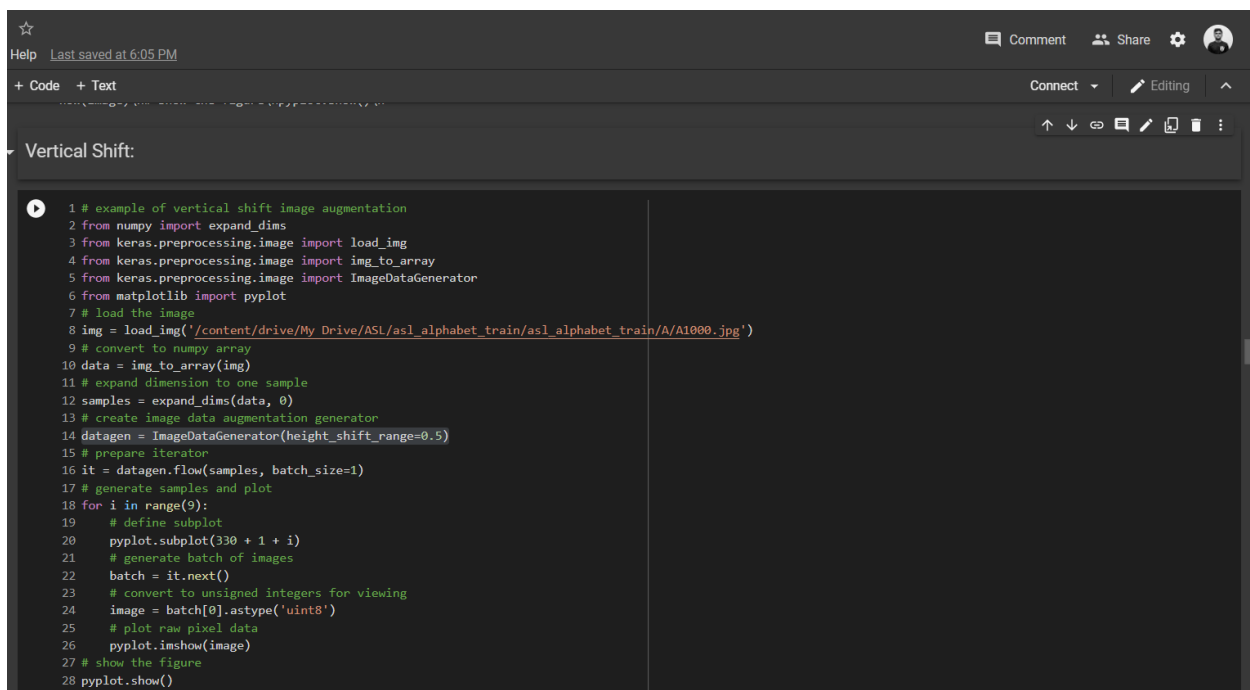


Figure 11 Code used in vertical shift augmentation.

The output of the augmentation can be shown as follows:

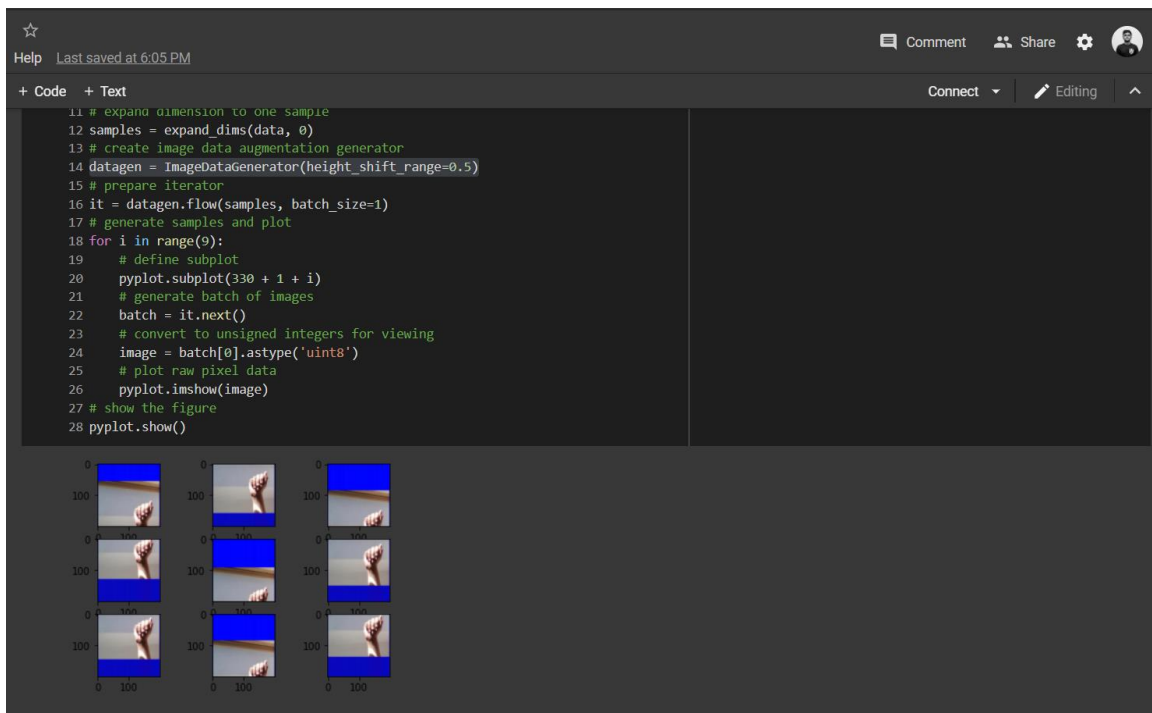


Figure 12 Output obtained after using vertical shift augmentation.

5.2. Horizontal and Vertical Flip Augmentation:

A horizontal and vertical flip feature was used, and the images were tested with different values and then plotted.

5.2.1. Horizontal Flip Augmentation:

A horizontal flip augmentation used the “**horizontal_flip**” function so that depending on the value provided, the images tend to flip horizontally. The function and a Boolean value (True) provided can be shown as follows:

```

[ ] 1 # example of horizontal flip image augmentation
2 from numpy import expand_dims
3 from keras.preprocessing.image import load_img
4 from keras.preprocessing.image import img_to_array
5 from keras.preprocessing.image import ImageDataGenerator
6 from matplotlib import pyplot
7
8 # load the image
9 img = load_img('/content/drive/My Drive/ASL/asl_alphabet_train/asl_alphabet_train/A/A1000.jpg')
10 # convert to numpy array
11 data = img_to_array(img)
12 # expand dimension to one sample
13 samples = expand_dims(data, 0)
14 # create image data augmentation generator
15 datagen = ImageDataGenerator(horizontal_flip=True)
16 # prepare iterator
17 it = datagen.flow(samples, batch_size=1)
18 # generate samples and plot
19 for i in range(9):
20     # define subplot
21     pyplot.subplot(330 + 1 + i)
22     # generate batch of images
23     batch = it.next()
24     # convert to unsigned integers for viewing
25     image = batch[0].astype('uint8')
26     # plot raw pixel data
27     pyplot.imshow(image)
28 # show the figure
29 pyplot.show()

```

Figure 13 Code used in Horizontal flip augmentation.

The output of the augmentation can be shown as follows:

```

22 # generate batch of images
23 batch = it.next()
24 # convert to unsigned integers for viewing
25 image = batch[0].astype('uint8')
26 # plot raw pixel data
27 pyplot.imshow(image)
28 # show the figure
29 pyplot.show()

```

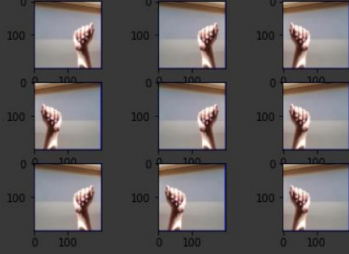


Figure 14 Output obtained after using horizontal flip augmentation.

5.2.2. Vertical Flip Augmentation:

A vertical flip augmentation used the “**vertical_flip**” function so that depending on the value provided, the images tend to move vertically. The function and a Boolean value (True) provided can be shown as follows:

```

[ ] 1 # load the image
2 img = load_img('/content/drive/My Drive/ASL/asl_alphabet_train/asl_alphabet_train/A/A1000.jpg')
3 # convert to numpy array
4 data = img_to_array(img)
5 # expand dimension to one sample
6 samples = expand_dims(data, 0)
7 # create image data augmentation generator
8 datagen = ImageDataGenerator(vertical_flip=True)
9 # prepare iterator
10 it = datagen.flow(samples, batch_size=1)
11 # generate samples and plot
12 for i in range(9):
13     # define subplot
14     pyplot.subplot(330 + 1 + i)
15     # generate batch of images
16     batch = it.next()
17     # convert to unsigned integers for viewing
18     image = batch[0].astype('uint8')
19     # plot raw pixel data
20     pyplot.imshow(image)
21 # show the figure
22 pyplot.show()

```

Figure 15 Code used in Vertical flip augmentation.

The output of the augmentation can be shown as follows:

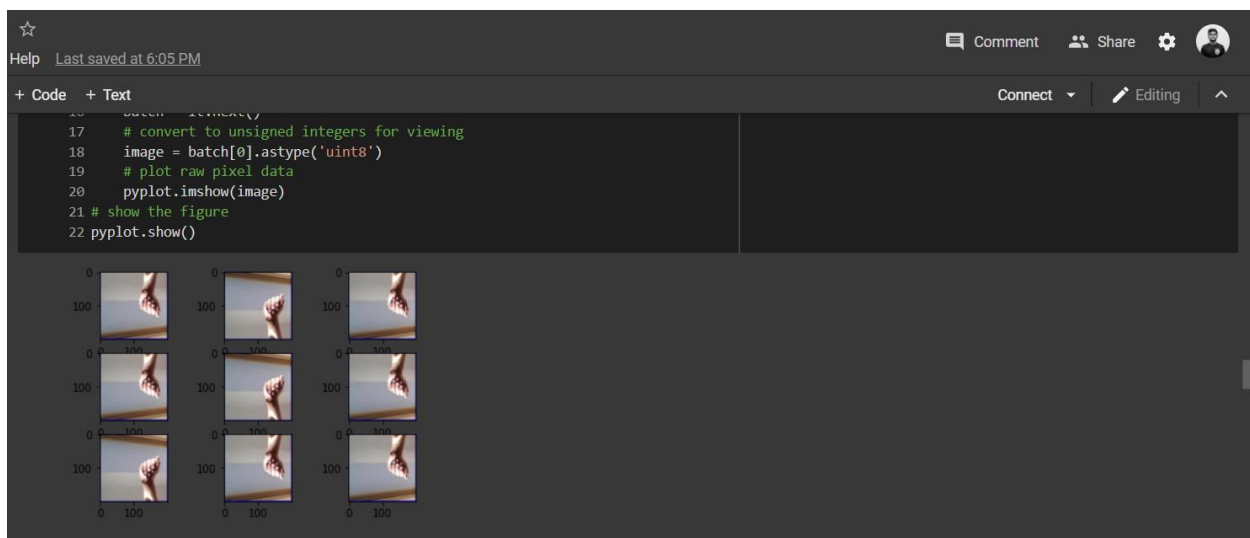


Figure 16 Output obtained after using vertical shift augmentation.

5.3. Random Rotation:

A random rotation augmentation used the “**rotation_range**” function so that depending on the value provided, the images tend to rotate. The function and the range of values provided can be shown as follows:

```

1 # example of random rotation image augmentation
2
3 # load the image
4 img = load_img('/content/drive/My Drive/ASL/asl_alphabet_train/asl_alphabet_train/A/A1000.jpg')
5 # convert to numpy array
6 data = img_to_array(img)
7 # expand dimension to one sample
8 samples = expand_dims(data, 0)
9 # create image data augmentation generator
10 datagen = ImageDataGenerator(rotation_range=180)
11 # prepare iterator
12 it = datagen.flow(samples, batch_size=1)
13 # generate samples and plot
14 for i in range(9):
15     # define subplot
16     pyplot.subplot(330 + 1 + i)
17     # generate batch of images
18     batch = it.next()
19     # convert to unsigned integers for viewing
20     image = batch[0].astype('uint8')
21     # plot raw pixel data
22     pyplot.imshow(image)
23 # show the figure
24 pyplot.show()

```

Figure 17 Code used in random rotation augmentation.

The output of the augmentation can be shown as follows:

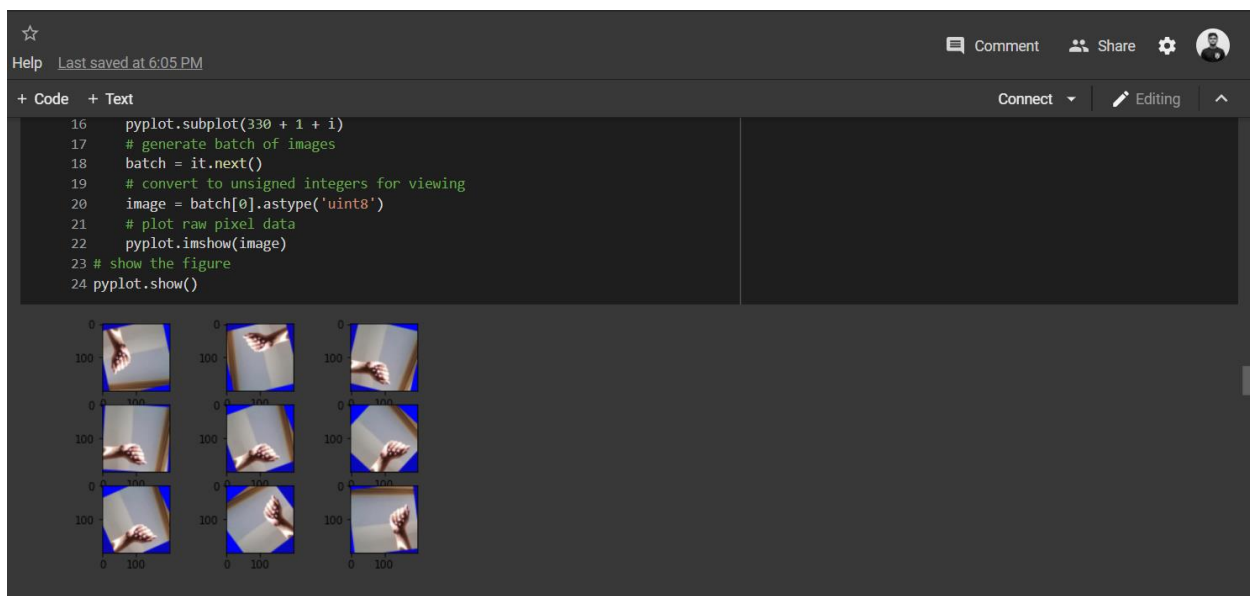


Figure 18 Output obtained after rotation augmentation.

5.4. Random Brightness Augmentation:

A random brightness augmentation used the “**brightness_range**” function so that depending on the value provided, the images tend to rotate. The function and the range of values provided can be shown as follows:

☆

Help

Last saved at 6:05 PM

+ Code

+ Text

Connect

Editing

Random Brightness Augmentation:

1 # example of brighting image augmentation

2

3 # load the image

4 img = load_img('/content/drive/My Drive/ASL/asl_alphabet_train/asl_alphabet_train/A/A1000.jpg')

5 # convert to numpy array

6 data = img_to_array(img)

7 # expand dimension to one sample

8 samples = expand_dims(data, 0)

9 # create image data augmentation generator

10 datagen = ImageDataGenerator(brightness_range=[0.2,1.0])

11 # prepare iterator

12 it = datagen.flow(samples, batch_size=1)

13 # generate samples and plot

14 for i in range(9):

15 # define subplot

16 pyplot.subplot(330 + 1 + i)

17 # generate batch of images

18 batch = it.next()

19 # convert to unsigned integers for viewing

20 image = batch[0].astype('uint8')

21 # plot raw pixel data

22 pyplot.imshow(image)

23 # show the figure

24 pyplot.show()

Figure 19 Code used in random brightness augmentation.

The output of the augmentation can be shown as follows:

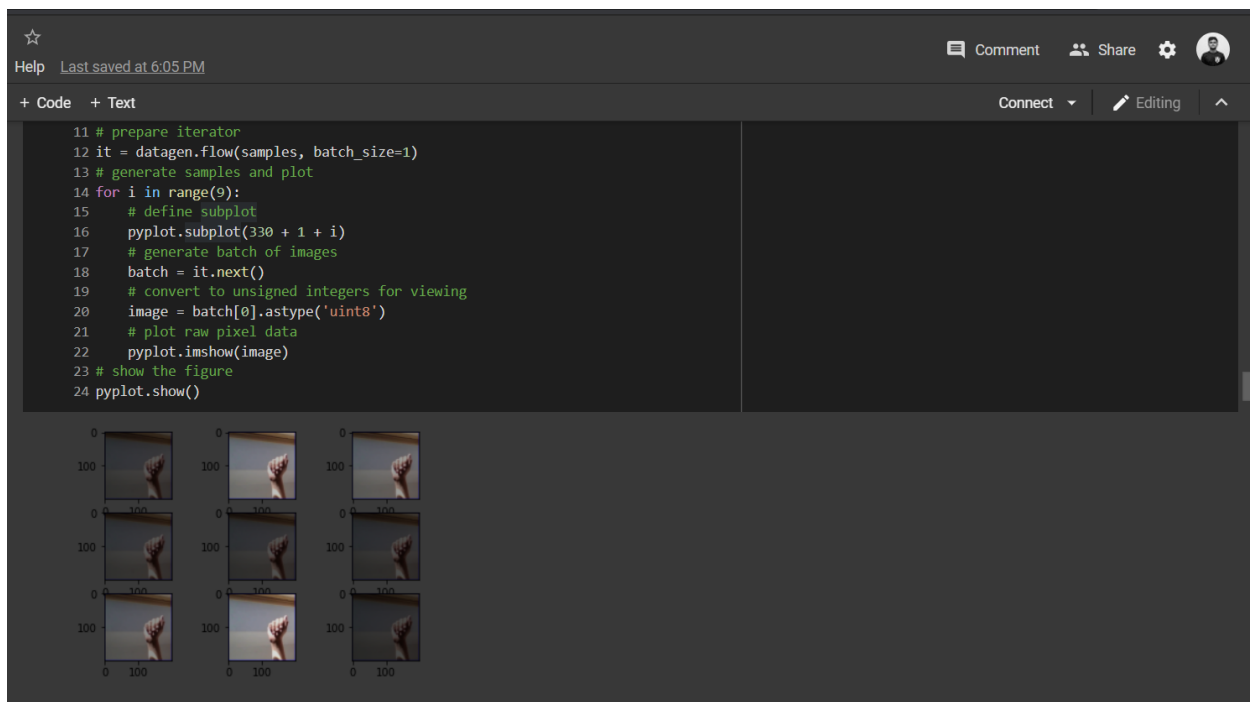


Figure 20 Output obtained after using random brightness augmentation.

5.5. Random Zoom Augmentation:

A random brightness augmentation used the “**zoom_range**” function so that depending on the value provided, the images tend to rotate. The function and the range of values provided can be shown as follows:

```
☆
Help Last saved at 6:05 PM
+ Code + Text
Connect Editing
Random Zoom Augmentation:

1 # example of zoom image augmentation
2
3 # load the image
4 img = load_img('/content/drive/My Drive/ASL/asl_alphabet_train/asl_alphabet_train/A/A1000.jpg')
5 # convert to numpy array
6 data = img_to_array(img)
7 # expand dimension to one sample
8 samples = expand_dims(data, 0)
9 # create image data augmentation generator
10 datagen = ImageDataGenerator(zoom_range=[0.5,1.0])
11 # prepare iterator
12 it = datagen.flow(samples, batch_size=1)
13 # generate samples and plot
14 for i in range(9):
15     # define subplot
16     pyplot.subplot(330 + 1 + i)
17     # generate batch of images
18     batch = it.next()
19     # convert to unsigned integers for viewing
20     image = batch[0].astype('uint8')
21     # plot raw pixel data
22     pyplot.imshow(image)
23 # show the figure
24 pyplot.show()
```

Figure 21 Code used in random zoom augmentation.

The output of the augmentation can be shown as follows:

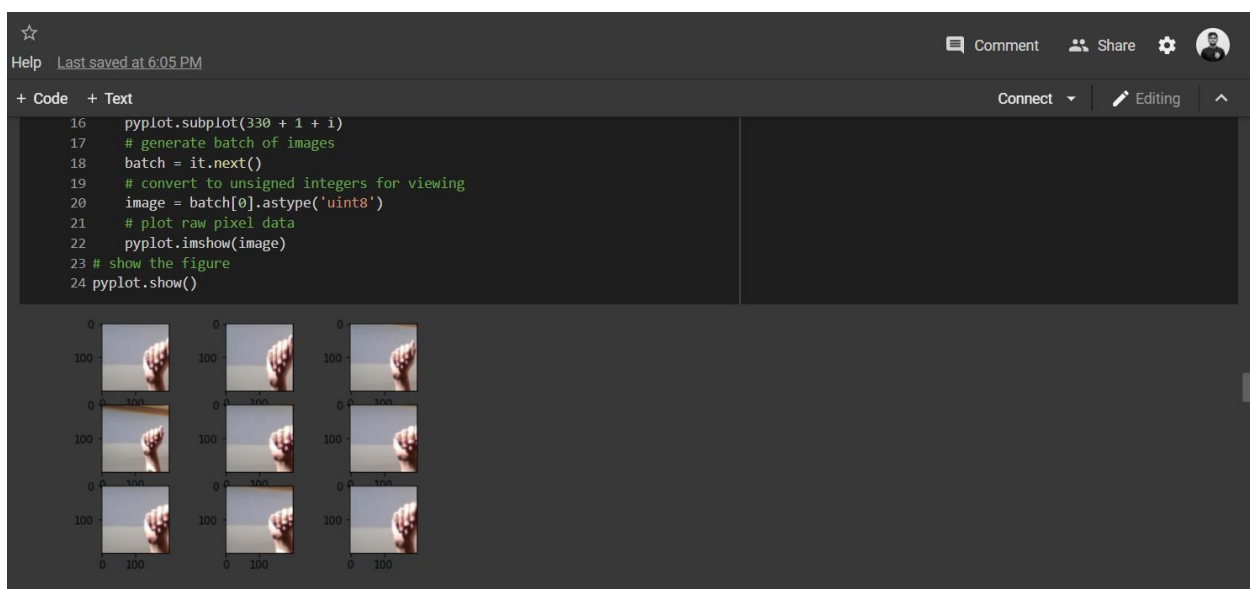


Figure 22 Output obtained after using random zoom augmentation.

5.6. Data Augmentation on different network architectures:

ImageDataGenerator function was used for different nets to calculate the accuracy values on the train set. The nets used and their accuracy can be given as follows:

Table 1 Network Architecture and accuracy values after using image generator.

Network Architecture	Accuracy
VGG19	0.85
VGG16	0.93
AlexNet	0.78
ResNet50	0.69

6. Network Architecture

Different CNN Network architectures were used before tuning the hyperparameters. They are as follows:

- LeNet
- AlexNet
- VGG16
- VGG19
- MiniVGG

Out of these, VGG19 gave the highest accuracy. After selecting that, the rest of the hyperparameters were tuned. The accuracy values obtained for each can be shown in the following table:

Table 2 Network Architecture and their accuracy.

Network Architecture	Accuracy
LeNet	0.90
AlexNet	0.90
VGG16	0.94
VGG19	0.98
MiniVGG	0.91

7. Modern CNN Network Architecture

To test a modern CNN architecture, I used a ResNet50 network architecture and trained the model. The ResNet50 is used primarily to avoid vanishing gradients. It uses the shortcut or skip connection, allowing the gradient to be directly backpropagated to earlier layers. The code is divided into two blocks:

a) Identity Code:

Introduce Conv 2D, back normalization, activation function as a 1st component, 2nd component, and 3rd component, and later the shortcut and input are added together.

b) Conv Code:

The code ensures that the input and output dimensions don't match up.

The code can be shown as follows:

```

NeuralNetwork_Project2.ipynb
File Edit View Insert Runtime Tools Help Last saved at 6:05 PM
Code + Text Connect Editing

ResNet 50 (2015)

[ ] 1 def identity_block(X, f, filters, stage, block):
2     """
3     Implementation of the identity block as defined in Figure 3
4
5     Arguments:
6     X -- input tensor of shape (n, n_H_prev, n_W_prev, n_C_prev)
7     f -- integer, specifying the shape of the middle CONV's window for the main path
8     filters -- python list of integers, defining the number of filters in the CONV layers of the main path
9     stage -- integer, used to name the layers, depending on their position in the network
10    block -- string/character, used to name the layers, depending on their position in the network
11
12    Returns:
13    X -- output of the identity block, tensor of shape (n_H, n_W, n_C)
14    """
15
16    # defining name basis
17    conv_name_base = 'res' + str(stage) + block + '_branch'
18    bn_name_base = 'bn' + str(stage) + block + '_branch'
19
20    # Retrieve Filters
21    F1, F2, F3 = filters
22
23    # Save the input value. You'll need this later to add back to the main path.
24    X_shortcut = X
25
26    # First component of main path
27    X = Conv2D(filters = F1, kernel_size = (f, f), strides = (1,1), padding = 'valid', name = conv_name_base + '2a', kernel_initializer = glorot_uniform(seed=0))(X)
28    X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
29    X = Activation('relu')(X)
30
31    # Second component of main path (3 lines)
32    X = Conv2D(filters = F2, kernel_size = (f, f), strides = (1,1), padding = 'same', name = conv_name_base + '2b', kernel_initializer = glorot_uniform(seed=0))(X)
33    X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
34    X = Activation('relu')(X)
35
36    # Third component of main path (2 lines)
37    X = Conv2D(filters = F3, kernel_size = (1, 1), strides = (1,1), padding = 'valid', name = conv_name_base + '2c', kernel_initializer = glorot_uniform(seed=0))(X)
38    X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)
39
40    # Final step: Add shortcut value to main path, and pass it through a RELU activation (n2 lines)
41    X = Add()([X, X_shortcut])
42    X = Activation('relu')(X)
43
44    return X
45
46

```

Figure 23 Identity Block Of ResNet50 Architecture.

```

NeuralNetwork_Project2.ipynb
File Edit View Insert Runtime Tools Help Last saved at 6:05 PM
Code + Text Connect Editing

[ ] 1 def convolutional_block(X, f, filters, stage, block, s = 2):
2     """
3     Implementation of the convolutional block as defined in Figure 4
4
5     Arguments:
6     X -- input tensor of shape (n, n_H_prev, n_W_prev, n_C_prev)
7     f -- integer, specifying the shape of the middle CONV's window for the main path
8     filters -- python list of integers, defining the number of filters in the CONV layers of the main path
9     stage -- integer, used to name the layers, depending on their position in the network
10    block -- string/character, used to name the layers, depending on their position in the network
11    s -- integer, specifying the stride to be used
12
13    Returns:
14    X -- output of the convolutional block, tensor of shape (n_H, n_W, n_C)
15    """
16
17    # defining name basis
18    conv_name_base = 'res' + str(stage) + block + '_branch'
19    bn_name_base = 'bn' + str(stage) + block + '_branch'
20
21    # Retrieve Filters
22    F1, F2, F3 = filters
23
24    # Save the input value
25    X_shortcut = X
26
27    ##### MAIN PATH #####
28
29    # First component of main path
30    X = Conv2D(F1, (f, f), strides = (s,s), name = conv_name_base + '2a', kernel_initializer = glorot_uniform(seed=0))(X)
31    X = BatchNormalization(axis = 3, name = bn_name_base + '2a')(X)
32    X = Activation('relu')(X)
33
34    # Second component of main path (3 lines)
35    X = Conv2D(filters = F2, kernel_size = (f, f), strides = (1,1), padding = 'same', name = conv_name_base + '2b', kernel_initializer = glorot_uniform(seed=0))(X)
36    X = BatchNormalization(axis = 3, name = bn_name_base + '2b')(X)
37    X = Activation('relu')(X)
38
39    # Third component of main path (2 lines)
40    X = Conv2D(filters = F3, kernel_size = (1, 1), strides = (1,1), padding = 'valid', name = conv_name_base + '2c', kernel_initializer = glorot_uniform(seed=0))(X)
41    X = BatchNormalization(axis = 3, name = bn_name_base + '2c')(X)
42
43    ##### SHORTCUT PATH ##### (2 lines)
44    X_shortcut = Conv2D(filters = F3, kernel_size = (1, 1), strides = (s,s), padding = 'valid', name = conv_name_base + '1',
45                        kernel_initializer = glorot_uniform(seed=0))(X_shortcut)
46    X_shortcut = BatchNormalization(axis = 3, name = bn_name_base + '1')(X_shortcut)
47
48    # Final step: Add shortcut value to main path, and pass it through a RELU activation (n2 lines)
49    X = Add()([X, X_shortcut])
50    X = Activation('relu')(X)
51
52
53

```

Figure 24 Convolutional Block Of ResNet50 Architecture.

The last code contains the Resnet50 function, which contains both the identity and the convolution block.

```

NeuralNetwork_Project12.ipynb
File Edit View Insert Runtime Tools Help Last saved at 6:05 PM
Code + Text
Connect Editing

1 def ResNet50(input_shape=(75, 75, 3), classes=29):
2     """
3     Implementation of the popular ResNet50 the following architecture:
4     CONV2D -> BATCHNORM -> RELU -> MAXPOOL -> CONVLOCK -> IDBLOCK*2 -> CONVLOCK -> IDBLOCK*2 -> AVGPOOL -> TOPLAYER
5     -> CONVLOCK -> IDBLOCK*5 -> CONVLOCK -> IDBLOCK*2 -> AVGPOOL -> TOPLAYER
6
7     Arguments:
8     input_shape -- shape of the images of the dataset
9     classes -- integer, number of classes
10
11     Returns:
12     model -- a Model() instance in Keras
13     """
14
15     # Define the input as a tensor with shape input_shape
16     X_input = Input(input_shape)
17
18     # Zero-Padding
19     X = ZeroPadding2D((3, 3))(X_input)
20
21     # Stage 1
22     X = Conv2D(64, (7, 7), strides=(2, 2), name='conv1', kernel_initializer=glorot_uniform(seed=0))(X)
23     X = BatchNormalization(axis=3, name='bn_conv1')(X)
24     X = Activation('relu')(X)
25     X = MaxPooling2D((3, 3), strides=(2, 2))(X)
26
27     # Stage 2
28     X = convolutional_block(X, f=3, filters=[64, 64, 256], stage=2, block='a', s=1)
29     X = identity_block(X, 3, [64, 64, 256], stage=2, block='b')
30     X = identity_block(X, 3, [64, 64, 256], stage=2, block='c')
31
32     ## START CODE HERE ##
33
34     # Stage 3 (4 lines)
35     X = convolutional_block(X, f=3, filters=[128, 128, 512], stage=3, block='a', s=2)
36     X = identity_block(X, 3, [128, 128, 512], stage=3, block='b')
37     X = identity_block(X, 3, [128, 128, 512], stage=3, block='c')
38     X = identity_block(X, 3, [128, 128, 512], stage=3, block='d')
39
40     # Stage 4 (4 lines)
41     X = convolutional_block(X, f=3, filters=[256, 256, 1024], stage=4, block='a', s=2)
42     X = identity_block(X, 3, [256, 256, 1024], stage=4, block='b')
43     X = identity_block(X, 3, [256, 256, 1024], stage=4, block='c')
44     X = identity_block(X, 3, [256, 256, 1024], stage=4, block='d')
45     X = identity_block(X, 3, [256, 256, 1024], stage=4, block='e')
46     X = identity_block(X, 3, [256, 256, 1024], stage=4, block='f')
47
48     # Stage 5 (4 lines)
49     X = convolutional_block(X, f=3, filters=[512, 512, 2048], stage=5, block='a', s=2)
50     X = identity_block(X, 3, [512, 512, 2048], stage=5, block='b')
51     X = identity_block(X, 3, [512, 512, 2048], stage=5, block='c')
52

```

Figure 25 ResNet50 Architecture containing both identity and convolution block.

A final accuracy of 0.90 was obtained on running the code.

8. Ensemble Learning:

8.1. Mini VGG:

The code of my ensemble using Mini VGG can be shown as follows:

```

1 from imutils import paths
2 import cv2
3 import glob
4 import h5py
5 import imutils
6 import matplotlib.pyplot as plt
7 import numpy as np
8 import os
9 import pickle
10 import progressbar
11 import random
12 tf.keras.backend.clear_session()
13
14 trainY = to_categorical(trainV, 29)
15
16 valY = to_categorical(valV, 29)
17
18 numberOfModels = 5
19 epochs = 25
20 labelNames = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z","del","nothing","space"]
21
22 for i in range(numberOfModels):
23     print('Net', i, 'is being trained...')
24
25     # choose the optimizer
26     #lr = SGD(lr = 0.01, decay = 0.1 / epochs, momentum = 0.9, nesterov = True)
27     opt = Adam()
28
29     # compile the model
30     model = MiniVGNet.build(75, 75, 3, 29)
31     model.compile(loss = 'categorical_crossentropy', optimizer = opt, metrics = ['accuracy'])
32
33     # train the model
34     aug = ImageDataGenerator(rotation_range = 10, width_shift_range = 0.1, height_shift_range = 0.1, horizontal_flip = True, fill_mode = 'nearest')
35     #H = model.fit(aug.flow(trainX, trainY, batch_size = 128), validation_data = (valX, valY), epochs = epochs, steps_per_epoch = len(trainX) // 64, verbose = 1)
36     H = model.fit(trainX, trainY, validation_split = 0.20, batch_size = 128, epochs = epochs, verbose = 1)
37
38     # save the model
39     p = ['content/drive/MyDrive/ASL/models1', 'content/drive/MyDrive/ASL/model_0_model'.format(i)]
40     model.save(os.path.sep.join(p))
41
42     # evaluate the network
43     predictions = model.predict(valX, batch_size=128)
44     report = classification_report(valY.argmax(axis=1), predictions.argmax(axis=1), target_names=labelNames)
45     '''
46     # save the classification report to file
47     p = ['content/drive/MyDrive/ASL/models1', 'content/drive/MyDrive/ASL/model_0_model'.format(i)]
48     f = open(os.path.sep.join(p), "w")
49     f.write(report)
50     f.close()
51     '''

```

Figure 26 Ensemble of nets output of MiniVG.

To test a different type of ensemble, a snapshot ensemble was used to train my model:

```

1 tf.keras.backend.clear_session()
2 from tensorflow.keras.callbacks import LearningRateScheduler
3 from tensorflow.keras.callbacks import ModelCheckpoint
4 models = 3
5 initialLearningRate = 0.2
6 epochs = 25
7
8 trainY = to_categorical(trainV, 29)
9 valY = to_categorical(valV, 29)
10
11 labelNames = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z","del","nothing","space"]
12
13 # code for a learning rate scheduler
14 def shiftedCosineLearning(epoch):
15     maxEpochs = epochs
16     baseLearningRate = initialLearningRate
17
18     alpha = (initialLearningRate/2)*(np.cos(np.pi*epoch/mod(epoch - 1, np.ceil(epochs/models)))/np.ceil(epochs/models)) + 1)
19
20     # return the learning rate
21     return alpha
22
23 callbacks = [LearningRateScheduler(shiftedCosineLearning)]
24
25 # choose the optimizer
26 opt = SGD(lr = initialLearningRate)
27 #pt = Adam()
28
29 # compile the model
30 model = MiniVGNet.build(75, 75, 3, 29)
31 model.compile(loss = 'categorical_crossentropy', optimizer = opt, metrics = ['accuracy'])
32
33 # train the model
34 checkpoint = ModelCheckpoint('model(epoch:80).model', period=np.ceil(epochs/models))
35 #aug = ImageDataGenerator(rotation_range = 10, width_shift_range = 0.1, height_shift_range = 0.1, horizontal_flip = True, fill_mode = 'nearest')
36 #H = model.fit(aug.flow(trainX, trainY, batch_size = 64), validation_data = (valX, valY), epochs = epochs,
37 #             #callbacks = callbacks, steps_per_epoch = len(trainX) // 64, verbose = 1)
38 H = model.fit(trainX, trainY, validation_data = (valX, valY), batch_size = 128, epochs = epochs, verbose = 1)
39
40 # evaluate the network
41 predictions = model.predict(valX, batch_size=128)
42 report = classification_report(valY.argmax(axis=1), predictions.argmax(axis=1), target_names=labelNames)
43
44 valY = valY.argmax(axis=1)
45 trainY = trainY.argmax(axis=1)

```

Figure 27 Snapshot Ensemble of nets output of MiniVG.

8.2. VGG16:

The code of my ensemble in VGG16 can be shown as follows:

```
Help Last saved at 6:05 PM
+ Code + Text
Connect Editing
VGG16:

[ ] 1 from imutils import paths
2 import cv2
3 import glob
4 import h5py
5 import imutils
6 import matplotlib.pyplot as plt
7 import numpy as np
8 import os
9 import pickle
10 import progressbar
11 import random
12 tf.keras.backend.clear_session()
13
14 trainY = to_categorical(trainY, 29)
15
16 valY = to_categorical(valY, 29)
17
18 numberOfModels = 3
19 epochs = 25
20 labelNames = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "del", "nothing", "space"]
21
22 for i in range(numberOfModels):
23     print("Net", i, "is being trained...")
24
25     # choose the optimizer
26     # opt = SGD(lr = 0.01, decay = 0.1 / epochs, momentum = 0.9, nesterov = True)
27     opt = Adam()
28
29     # compile the model
30     model = VGGNet16.build(75, 75, 3, 29)
31     model.compile(loss = 'categorical_crossentropy', optimizer = opt, metrics = ['accuracy'])
32
33     # train the model
34     aug = ImageDataGenerator(rotation_range = 10, width_shift_range = 0.1, height_shift_range = 0.1, horizontal_flip = True, fill_mode = 'nearest')
35     # H = model.fit(aug.flow(trainX, trainY, batch_size = 128), validation_data = (valX, valY), epochs = epochs, steps_per_epoch = len(trainX) // 64, verbose = 1)
36     H = model.fit(trainX, trainY, validation_split = 0.20, batch_size = 128, epochs = epochs, verbose = 1)
37
38     # save the model
39     p = ["content/drive/MyDrive/ASL/models2", "content/drive/MyDrive/ASL/models2/model_1.model".format(i)]
40     model.save(os.path.sep.join(p))
41
42     # evaluate the network
43     predictions = model.predict(valX, batch_size=128)
44     report = classification_report(valY.argmax(axis=1), predictions.argmax(axis=1), target_names=labelNames)
```

Figure 28 Ensemble of nets output of VGG16.

To test a different type of ensemble, a snapshot ensemble was used to train my model:

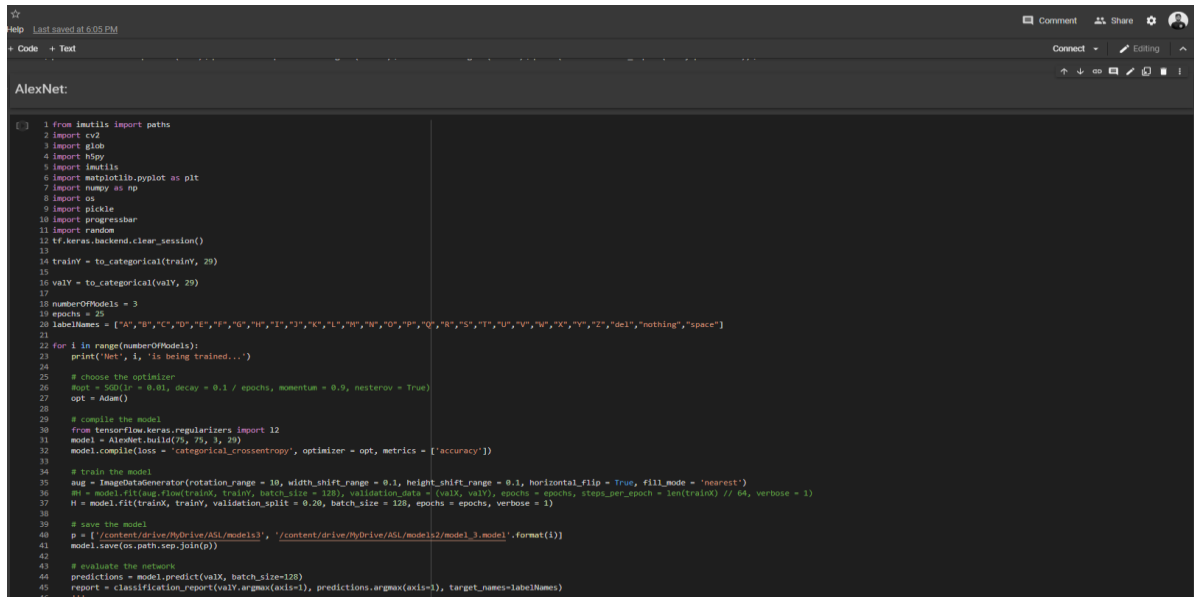
```
Help Last saved at 6:05 PM
+ Code + Text
Connect Editing
Snapshot Ensemble:

[ ] 1 tf.keras.backend.clear_session()
2 from tensorflow.keras.callbacks import LearningRateScheduler
3 from tensorflow.keras.callbacks import ModelCheckpoint
4 models = 3
5 initialLearningRate = 0.2
6 epochs = 25
7
8 trainY = to_categorical(trainY, 29)
9 valY = to_categorical(valY, 29)
10
11 labelNames = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "del", "nothing", "space"]
12
13 # code for a learning rate scheduler
14 def shiftedCosineLearning(epoch):
15     maxEpochs = epochs
16     baseLearningRate = initialLearningRate
17
18     alpha = (initialLearningRate/2)*(np.cos(np.pi*np.mod(epoch - 1, np.ceil(epochs/models))/np.ceil(epochs/models)) + 1)
19
20     # return the learning rate
21     return alpha
22
23 callbacks = [LearningRateScheduler(shiftedCosineLearning)]
24
25 # choose the optimizer
26 opt = SGD(lr = initialLearningRate)
27 # opt = Adam()
28
29 # compile the model
30 model = MiniVGGNet.build(75, 75, 3, 29)
31 model.compile(loss = 'categorical_crossentropy', optimizer = opt, metrics = ['accuracy'])
32
33 # train the model
34 checkpoint = ModelCheckpoint('model(epoch:80).model', period=np.ceil(epochs/models))
35 # aug = ImageDataGenerator(rotation_range = 10, width_shift_range = 0.1, height_shift_range = 0.1, horizontal_flip = True, fill_mode = 'nearest')
36 # H = model.fit(aug.flow(trainX, trainY, batch_size = 64), validation_data = (valX, valY), epochs = epochs,
37 #               callbacks = callbacks, steps_per_epoch = len(trainX) // 64, verbose = 1)
38 H = model.fit(trainX, trainY, validation_data = (valX, valY), batch_size = 128, epochs = epochs, verbose = 1)
39
40 # evaluate the network
41 predictions = model.predict(valX, batch_size=128)
42 report = classification_report(valY.argmax(axis=1), predictions.argmax(axis=1), target_names=labelNames)
43
44 valY = valY.argmax(axis=1)
45 trainY = trainY.argmax(axis=1)
```

Figure 29 Snapshot Ensemble of nets output of VGG16.

8.3. AlexNet:

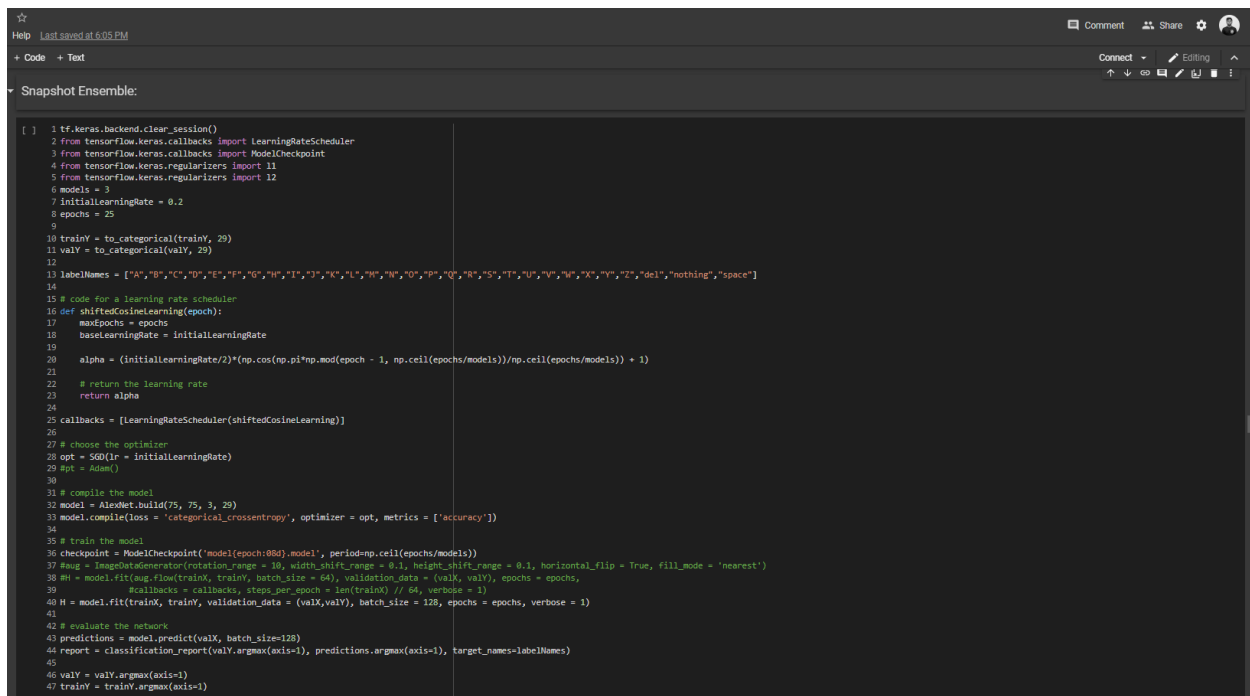
The code of my ensemble in AlexNet can be shown as follows:

A screenshot of a code editor window titled "AlexNet:". The code is written in Python and implements an ensemble of AlexNet models. It includes imports for various libraries like tf.keras, cv2, and matplotlib. The code defines a function to train a single AlexNet model with specific parameters like rotation range, width shift range, and height shift range. It then loops through a range of models (1 to 25) to train and save each one. Finally, it evaluates the ensemble by predicting on a validation set and reporting the accuracy.

```
1 from itertools import paths
2 import cv2
3 import glob
4 import KSPy
5 import matplotlib.pyplot as plt
6 import numpy as np
7 import os
8 import pickle
9 import random
10 import progressbar
11 import random
12 if tf.keras.backend.clear_session():
13     trainY = to_categorical(trainY, 29)
14     valY = to_categorical(valY, 29)
15
16 numberOfModels = 3
17 epochs = 25
18 labelNames = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z","del","nothing","space"]
19
20 for i in range(numberOfModels):
21     print('Net', i, 'is being trained...')
22
23     # choose the optimizer
24     opt = Adam(lr = 0.01, decay = 0.1 / epochs, momentum = 0.9, nesterov = True)
25
26     # compile the model
27     from tensorflow.keras.regularizers import l2
28     model = AlexNet.build(75, 75, 3, 29)
29     model.compile(loss = 'categorical_crossentropy', optimizer = opt, metrics = ['accuracy'])
30
31     # train the model
32     aug = ImageDataGenerator(rotation_range = 10, width_shift_range = 0.1, height_shift_range = 0.1, horizontal_flip = True, fill_mode = 'nearest')
33     M = model.fit(aug.flow(trainX, trainY, batch_size = 128), validation_data = (valX, valY), epochs = epochs, steps_per_epoch = len(trainX) // 64, verbose = 1)
34
35     # save the model
36     p = ['content/drive/MyDrive/ASL/models3', '/content/drive/MyDrive/ASL/models2/model_3.model'].format(i)
37     model.save(os.path.join(p))
38
39     # evaluate the network
40     predictions = model.predict(valX, batch_size=128)
41     report = classification_report(valY.argmax(axis=1), predictions.argmax(axis=1), target_names=labelNames)
42     print(report)
```

Figure 30 Ensemble of nets output of AlexNet.

To test a different type of ensemble, a snapshot ensemble was used to train my model:

A screenshot of a code editor window titled "Snapshot Ensemble:". The code is written in Python and implements a snapshot ensemble of AlexNet models. It includes imports for various libraries like tf.keras, tensorflow.keras.callbacks, and tensorflow.keras.regularizers. The code defines a function to train a single AlexNet model with specific parameters like initial learning rate and snapshot interval. It then loops through a range of models (1 to 25) to train and save each one. Finally, it evaluates the ensemble by predicting on a validation set and reporting the accuracy.

```
1 tf.keras.backend.clear_session()
2 from tensorflow.keras.callbacks import LearningRateScheduler
3 from tensorflow.keras.callbacks import ModelCheckpoint
4 from tensorflow.keras.regularizers import l2
5 from tensorflow.keras.regularizers import l2
6 models = 3
7 initialLearningRate = 0.2
8 epochs = 25
9
10 trainY = to_categorical(trainY, 29)
11 valY = to_categorical(valY, 29)
12
13 labelNames = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z","del","nothing","space"]
14
15 # code for a learning rate scheduler
16 def shiftedCosineLearning(epoch):
17     # epochs = epochs
18     baseLearningRate = initialLearningRate
19
20     alpha = (initialLearningRate/2)*(np.cos(np.pi*np.mod(epoch - 1, np.ceil(epochs/models))/np.ceil(epochs/models)) + 1)
21
22     # return the learning rate
23     return alpha
24
25 callbacks = [LearningRateScheduler(shiftedCosineLearning)]
26
27 # choose the optimizer
28 opt = Adam(lr = initialLearningRate)
29 opt = Adam()
30
31 # compile the model
32 model = AlexNet.build(75, 75, 3, 29)
33 model.compile(loss = 'categorical_crossentropy', optimizer = opt, metrics = ['accuracy'])
34
35 # train the model
36 checkpoint = ModelCheckpoint('model(epoch:880).model', period=np.ceil(epochs/models))
37 # aug = ImageDataGenerator(rotation_range = 10, width_shift_range = 0.1, height_shift_range = 0.1, horizontal_flip = True, fill_mode = 'nearest')
38 M = model.fit(aug.flow(trainX, trainY, batch_size = 64), validation_data = (valX, valY), epochs = epochs,
39             callbacks = callbacks, steps_per_epoch = len(trainX) // 64, verbose = 1)
40 H = model.fit(trainX, trainY, validation_data = (valX, valY), batch_size = 128, epochs = epochs, verbose = 1)
41
42 # evaluate the network
43 predictions = model.predict(valX, batch_size=128)
44 report = classification_report(valY.argmax(axis=1), predictions.argmax(axis=1), target_names=labelNames)
45
46 valY = valY.argmax(axis=1)
47 trainY = trainY.argmax(axis=1)
```

Figure 31 Snapshot Ensemble of nets output of AlexNet.

8.4. VGG19:

The code of my ensemble in VGG19 can be shown as follows:


```

☆
Help Last saved at 6:05 PM
+ Code + Text
Connect - Editing ^
VGG19:

[ ] 1 from imutils import paths
2 import cv2
3 import glob
4 import h5py
5 import imutils
6 import matplotlib.pyplot as plt
7 import numpy as np
8 import os
9 import pickle
10 import progressbar
11 import random
12 tf.keras.backend.clear_session()
13
14 trainY = to_categorical(trainY, 29)
15
16 valY = to_categorical(valY, 29)
17
18 numberOFModels = 3
19 epochs = 25
20 labelNames = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z","del","nothing","space"]
21
22 for i in range(numberOFModels):
23     print('Net', i, 'is being trained...')
24
25     # choose the optimizer
26     #opt = SGD(lr = 0.01, decay = 0.1 / epochs, momentum = 0.9, nesterov = True)
27     opt = Adam()
28
29     # compile the model
30     from tensorflow.keras.regularizers import l2
31     model = VGGNet19.build(75, 75, 3, 29)
32     model.compile(loss = 'categorical_crossentropy', optimizer = opt, metrics = ['accuracy'])
33
34     # train the model
35     aug = ImageDataGenerator(rotation_range = 18, width_shift_range = 0.1, height_shift_range = 0.1, horizontal_flip = True, fill_mode = 'nearest')
36     #H = model.fit(aug.flow(trainX, trainY, batch_size = 128), validation_data = (valX, valY), epochs = epochs, steps_per_epoch = len(trainX) // 64, verbose = 1)
37     H = model.fit(trainX, trainY, validation_split = 0.20, batch_size = 128, epochs = epochs, verbose = 1)
38
39     # save the model
40     p = ['content/drive/MyDrive/ASL/models4', '/content/drive/MyDrive/ASL/models2/model_4_model'.format(i)]
41     model.save(os.path.sep.join(p))
42
43     # evaluate the network
44     predictions = model.predict(valX, batch_size=128)
45     report = classification_report(valY.argmax(axis=1), predictions.argmax(axis=1), target_names=labelNames)
46     ...
47     # save the classification report to file
48     p = ['content/drive/MyDrive/ASL/models4', '/content/drive/MyDrive/ASL/models4/model_4_model'.format(i)]

```

Figure 32 Ensemble of nets output of VGG19.

To test a different type of ensemble, a snapshot ensemble was used to train my model:

```

☆
Help Last saved at 6:05 PM
+ Code + Text
Connect - Editing ^
Snapshot Ensemble:

[ ] 1 tf.keras.backend.clear_session()
2 from tensorflow.keras.callbacks import LearningRateScheduler
3 from tensorflow.keras.callbacks import ModelCheckpoint
4 models = 3
5 initialLearningRate = 0.2
6 epochs = 25
7
8 trainY = to_categorical(trainY, 29)
9 valY = to_categorical(valY, 29)
10
11 labelNames = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z","del","nothing","space"]
12
13 # code for a learning rate scheduler
14 def shiftedCosineLearning(epoch):
15     maxEpochs = epochs
16     baseLearningRate = initialLearningRate
17
18     alpha = (initialLearningRate/2)*(np.cos(np.pi*np.mod(epoch - 1, np.ceil(epochs/models)))/np.ceil(epochs/models)) + 1)
19
20     # return the learning rate
21     return alpha
22
23 callbacks = [LearningRateScheduler(shiftedCosineLearning)]
24
25 # choose the optimizer
26 opt = SGD(lr = initialLearningRate)
27 #pt = Adam()
28
29 # compile the model
30 model = VGGNet19.build(75, 75, 3, 29)
31 model.compile(loss = 'categorical_crossentropy', optimizer = opt, metrics = ['accuracy'])
32
33 # train the model
34 checkpoint = ModelCheckpoint('model(epoch:880).model', period=np.ceil(epochs/models))
35 #aug = ImageDataGenerator(rotation_range = 18, width_shift_range = 0.1, height_shift_range = 0.1, horizontal_flip = True, fill_mode = 'nearest')
36 #H = model.fit(aug.flow(trainX, trainY, batch_size = 64), validation_data = (valX, valY), epochs = epochs,
37 #             #callbacks = callbacks, steps_per_epoch = len(trainX) // 64, verbose = 1)
38 H = model.fit(trainX, trainY, validation_data = (valX, valY), batch_size = 128, epochs = epochs, verbose = 1)
39
40 # evaluate the network
41 predictions = model.predict(valX, batch_size=128)
42 report = classification_report(valY.argmax(axis=1), predictions.argmax(axis=1), target_names=labelNames)
43
44 valY = valY.argmax(axis=1)
45 trainY = trainY.argmax(axis=1)

```

Figure 33 Snapshot Ensemble of nets output of VGG19.

The accuracy values obtained can be shown in the table below:

Table 3 Comparison between Ensemble of Nets and Snapshot Ensemble.

Network Architecture	Ensemble Of Nets	Snapshot Ensemble
MiniVGG	0.74	0.94
VGG16	0.92	0.84
AlexNet	0.77	0.41
VGG19	0.97	0.67

9. Activation Function

The activation function used in this project are as follows:

Table 4 Activation Function and their accuracy.

Activation Function	Accuracy
Sigmoid	0.49
ReLU	0.79
tanh	0.74
Softmax	0.03

Amongst these, ReLU showed better accuracy than the other activation functions.

10. Loss Function

The Loss functions used in this project are as follows:

Table 5 Loss Function and their accuracy.

Loss Function	Accuracy
Mean_squared_Error	0.80
Categorical_crossentropy	0.89
Binary_categorical_crossentropy	0.87

Amongst these, category_crossentropy was selected as it gave a high accuracy compared to others. According to an online source, category_crossentropy is an ideal choice for multiclass image classification problems.

11. Tune hyperparameters

After selecting the right network architecture, activation, and loss function, the hyperparameters were tuned. To tune the hyperparameters, specific values for dropout, kernel size, layer size, and L1/L2 were given to choose the best one out of these values depending on the accuracy they printed. The values used are as shown below:

```

[ ] 1 #####
2 # Tune the hyperparameters
3
4 Drop_out = [0,0.01, 0.2 , 0.3 , 0.4, 0.5,0.6,0.7,0.8,0.9]
5 Filter_size = [ 4,16, 32, 64, 128, 256] #Number of Filters 1
6 Filter_size2 = [ 4,16, 32, 64, 128, 256 ] #Number of Filters 2
7 Filter_size3 = [ 4,16, 32, 64, 128, 256 ] #Number of filters 3
8 Filter_size4 = [ 4,16, 32, 64, 128, 256 ] #Number of filters 4
9 Layer_Size= [(1,1), (3,3), (5,5), (7,7),(9,9),(11,11)] #Kernel/Filter size
10 l1 = [1e-1 , 1e-2 , 1e-3 , 0 , 1e-4, 1e-5] #Kernel Regularizer
11 l2 = [1e-1 , 1e-2 , 1e-3 , 0 , 1e-4, 1e-5]#Kernel Regularizer
12 l_1 = [1e-1 , 1e-2 , 1e-3 , 0 , 1e-4, 1e-5] #Activity Regularizer
13 l_2 = [1e-1 , 1e-2 , 1e-3 , 0 , 1e-4, 1e-5]#Activity Regularizer
14 poolsize = [(1,1),(2,2),(3,3)]

```

Figure 34 Tuning Hyperparameter.

As seen in the above figure, to select a value for dropout, I have initialized the ‘Drop_out’ variable with some values, i.e., from 0 to 0.9. Based on these values, a for loop was implemented on my network architecture, and based on the highest test accuracy obtained, the value corresponding to that accuracy was selected. The chosen values and their respected accuracy can be shown in table 8.

12. Optimizer

The optimizers used in this project can be shown as follows:

Table 6 Optimizer and their accuracy.

Optimizer	Accuracy
Adam	0.67
SGD (0.01)	0.65
SGD (0.001)	0.11
Adagrad	0.04
Adadelat	0.04
Rmsprop	0.92
Adamax	0.35
Nadam	0.65

Amongst these, Rmsprop gave the highest accuracy. Although it did give a good accuracy, Adam was used as Rmsprop for some reason gave unstable values. Now, after tuning all the hyperparameters, a final code was run to check the accuracy.

13. Kernel Weight Initialization Method:

The kernel weight initialization methods used in this project are as follows:

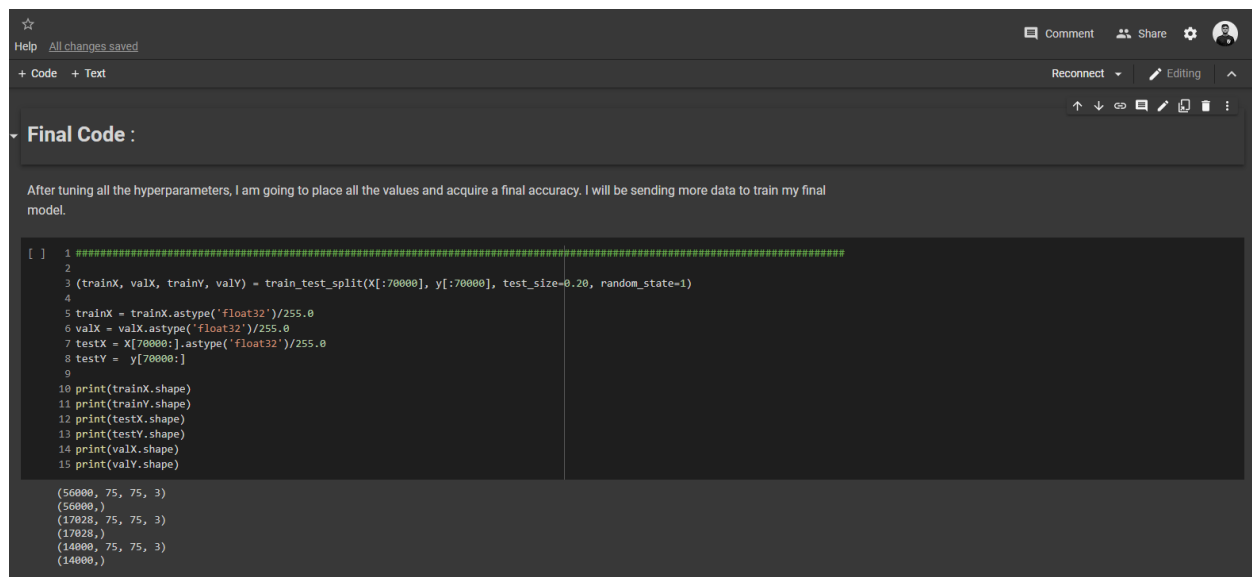
Table 7 Kernel Initializer and their accuracy.

Kernel Initializer	Accuracy
Glorot_Uniform	0.78
Random_normal	0.61
Random_uniform	0.78
Truncated_Normal	0.83
Zeros	0.03
Ones	0.05
Glorot_normal	0.89
Constant	0.03

Amongst these, glorot_normal gave the highest accuracy and hence it was used to tune my final model.

14. Final Code

Using the above values, and with the activation, network architecture, weight initialization methods and tuned hyperparameters, a final code is drafted to check the accuracy of the test, train, and validation. Before running the final code, more datapoints were used. A total of 70,000 images from the dataset were used as shown in the figure below:



The screenshot shows a Jupyter Notebook with the following content:

Help All changes saved

+ Code + Text

Reconnect Editing

Final Code :

After tuning all the hyperparameters, I am going to place all the values and acquire a final accuracy. I will be sending more data to train my final model.

```
[ ] 1 #####
2
3 (trainX, valX, trainY, valY) = train_test_split(X[:70000], y[:70000], test_size=0.20, random_state=1)
4
5 trainX = trainX.astype('float32')/255.0
6 valX = valX.astype('float32')/255.0
7 testX = X[70000:].astype('float32')/255.0
8 testY = y[70000:]
9
10 print(trainX.shape)
11 print(trainY.shape)
12 print(testX.shape)
13 print(testY.shape)
14 print(valX.shape)
15 print(valY.shape)
```

```
(56000, 75, 75, 3)
(56000,)
(17028, 75, 75, 3)
(17028,)
(14000, 75, 75, 3)
(14000,)
```

Figure 35 70000 dataset values sent for training final model.

As seen in the above figure, 56000 dataset values were used for training, 17028 were used for test set and 14000 datasets was used for validation. The accuracy obtained can be shown in the following figure:

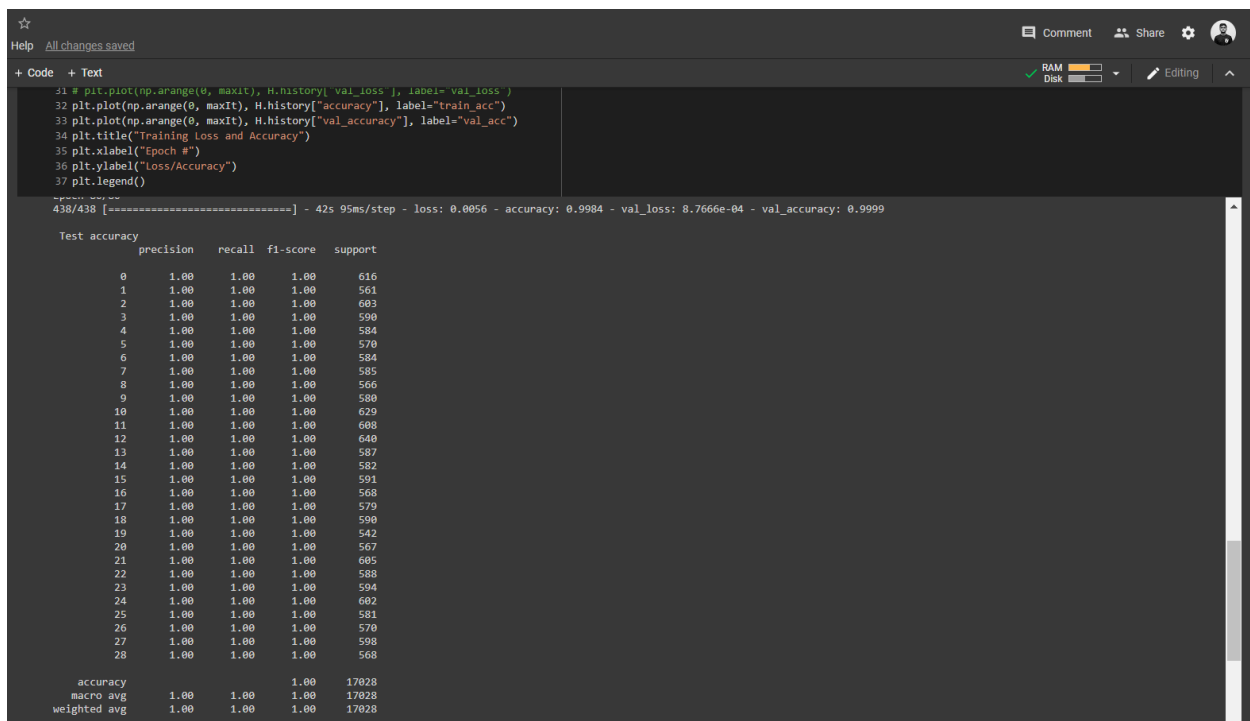


Figure 36 Final Accuracy obtained after training model

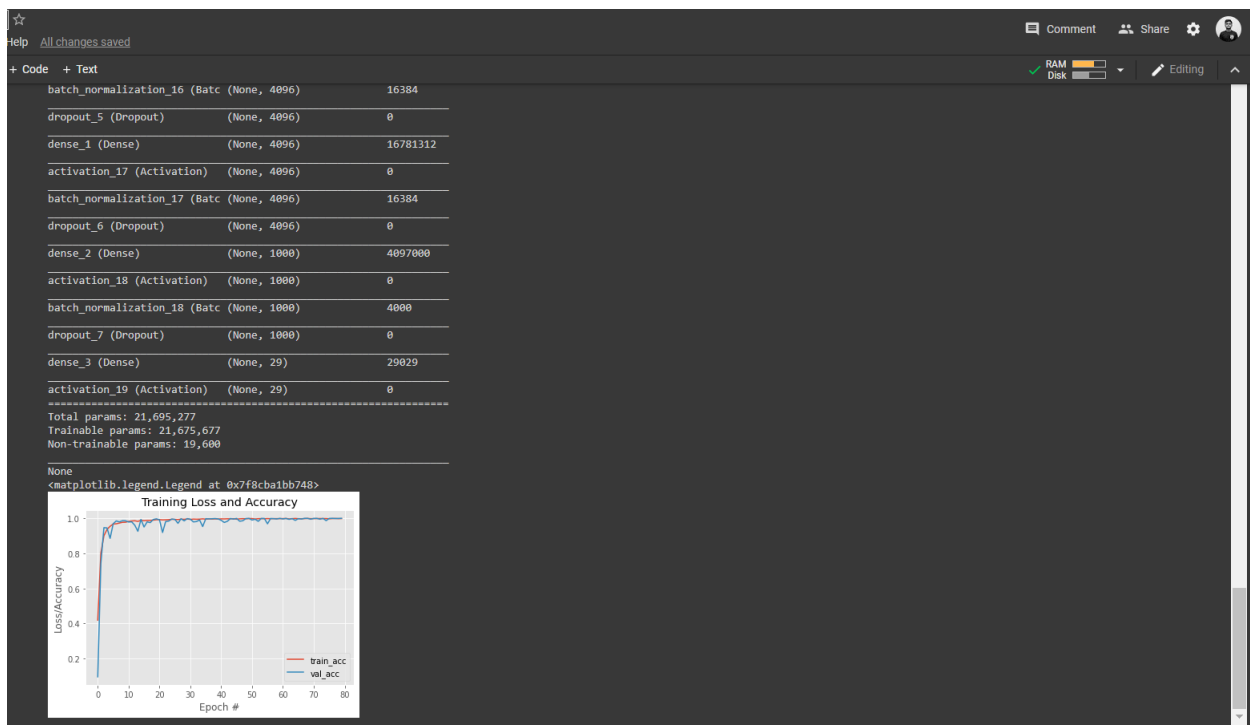


Figure 37 Graph of Final model obtained after training.

As seen in the above figure, the accuracy obtained for the test, train, and validation are:

- Train set – 0.98
- Test set – 1.00
- Validation Set – 0.99

15. Discussions

As seen above, the accuracy values touched 99 percent. The best values selected for tuning the hyperparameters can be shown in the following table.

Table 8 Best values obtained on tuning the hyperparameters.

Hyperparameters	Value Selected	Accuracy
Dropout	0.2	0.93
Filter_size (Kernel Size)	32	0.92
Filter_size2(Kernel Size)	16	0.90
Filter_size3(Kernel Size)	64	0.90
Filter_size4(Kernel Size)	32	0.85
Layer Size	(3,3)	0.86
L1 & L2 (Kernel Regularizer)	L1 & L2 = 0	0.83 & 0.79
L_1 & L_2 (Activity Regularizer)	L1 & L2 = $1 * e^{-5}$	0.60 & 0.78
Pool size	(2,2)	0.86