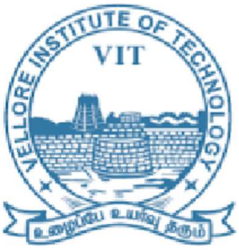


VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering

WINTER SEMESTER 2022-23

SOFTWARE METRICS

AI POWERED SIGN LANGUAGE INTERPRETER

UNDER THE GUIDANCE OF PROF. Dr.KRITHIKA L B

BY

PRANAY GORANTLA (21MIS0123)

ANNAPOORNA A NAIR (21MIS0130)

PADALA NAGA SINDHURA (21MIS0138)

REVIEW-1

INTRODUCTION:

In the Recent years, we have been understanding the potential of Artificial Intelligence. It has changed the way we live, work and correlate. With AI, the possibilities are endless, from self-driving cars to personalized healthcare and virtual aides that can predict our needs and demands. With its limitless potential and ever-evolving capabilities, AI is poised to be the most transformative technology of our time.

In this project we are exercising the use of AI to interpret sign language. This is a revolutionary technology that can help bridge the communication gap between people who use sign language and those who don't. This interpreter will be also useful for people who are new to the sign language and wants to learn it. This project makes use of the machine learning and deep learning algorithms like CNN to develop accurate and reliable model which interpret sign language gestures and convert them to text.

In this project, VGG16 algorithm is used. It is a type of Convolutional Neural Networks (CNN) which is 16 layers deep. This algorithm is used to extract meaningful features from the sign language gestures provided, which can be used to improve the accuracy. This algorithm has helped to identify and isolate important features such as edges, shapes, etc. These features are then used to classify the input provided by the user.

The accuracy and performance of the model will be evaluated using various metrics as accuracy is a critical metric that determines the effectiveness of the software. To improve the accuracy of the interpreter, it is essential to use a high-quality dataset to train the machine learning models. We have taken the dataset for this project from Kaggle. The dataset consists of various images of letters in American Sign Language.

With the continuous refinement and advancement of this technology, the accuracy and effectiveness could be increased more. This project has the potential to significantly enhance the convenience and comprehensiveness of our society as it can provide a vast room for research in this field. However additional research in this field has to be conducted in order to address the limitations and challenges of this project such as dialect and context-specific variations. This shows the necessity for the development of algorithms which are more effective and efficient.

PROBLEM STATEMENT:

The Sign language interpreter uses deep learning to help the people who don't know sign language and can help to communicate effectively between sign language people and people who don't know sign language. This sign language interpreter takes images and converts them into readable text. VGG-16 algorithm is used in this project. In this project, there are many problems/ challenges are identified.

Time complexity:

VGG (Visual Geometry Group)-16 algorithm is used in this project which takes more time to run the code. VGG-16 has 16 layers of which 13 are convolution layers and 3 are fully connected layers. It takes more time because it has a huge network to train its parameters.

Data Availability:

VGG-16 model requires more data to analyze and process the learning for a better understanding. In this project there is a limited set of data is available which makes it take time for image processing.

Complex sign recognition:

American Sign Language (ASL) is used in this project. This system cannot convey facial expressions and body language. It cannot determine the sign language based on human emotions.

Real-time performance:

This interpreter takes the image and convert it into text but in real time the sign language is often dynamic so it is tough for the interpreter to recognize the gesture fast.

Accuracy:

Even though we have provided 3000 images per class. Accuracy provided by the model after training is approximately around 60% which is a huge drawback.

Keras CNN - accuracy: 0.5983333587646484

	precision	recall	f1-score	support
A	0.70	0.64	0.67	596
B	0.52	0.58	0.55	611
C	0.90	0.89	0.89	566
D	0.80	0.83	0.81	607
E	0.56	0.53	0.54	599
F	0.87	0.67	0.76	612
G	0.64	0.56	0.60	630
H	0.60	0.73	0.66	594
I	0.57	0.64	0.60	608
J	0.64	0.57	0.60	587
K	0.65	0.56	0.60	616
L	0.55	0.78	0.64	607
M	0.65	0.25	0.37	634
N	0.55	0.54	0.54	630
O	0.74	0.82	0.78	582
P	0.53	0.69	0.60	596
Q	0.77	0.62	0.69	599
R	0.78	0.40	0.53	628
S	0.31	0.39	0.35	515
T	0.62	0.25	0.35	630
U	0.34	0.41	0.37	604
V	0.54	0.46	0.50	590
W	0.68	0.58	0.63	609
X	0.49	0.52	0.50	619
Y	0.48	0.59	0.53	578
Z	0.63	0.59	0.61	577
del	0.82	0.63	0.71	621
nothing	0.47	0.94	0.62	562
space	0.54	0.75	0.62	593
accuracy			0.60	17400
macro avg	0.62	0.60	0.59	17400
weighted avg	0.62	0.60	0.59	17400

DATASET DESCRIPTION:

In this project, the data set is taken from Kaggle. This contains a collection of images of alphabets from American Sign Language, separated in 29 folders that represent the various classes. The training data set contains 87,000 images which are 200x200 pixels. There are 29 classes, of which 26 are for the letters A-Z and 3 classes for SPACE, DELETE and NOTHING.

These 3 classes are very helpful in real-time applications, and classification.

The test data set contains a mere 29 images, to encourage the use of real-world test images.

Kaggle link:

https://www.kaggle.com/code/paultimothymooney/interpret-sign-language-with-deep-learning/data

LITERATURE REVIEW:

[1]**Upendran, Sruthi, and A. Thamizharasi (2014)** The proposed system recognizes and translates static hand gesture of alphabets in American Sign Language (ASL) into an output. This output can be converted into speech. **The concept of Principal Component Analysis (PCA) is used on the static gesture images of the ASL alphabet.** The PCA features extracted from the

image are used to classify the image into one of the ASL alphabet. The recognition of ASL gestures results in a textual output and is then can be converted into speech. Thus, this scheme helps deaf and hearing impaired people communicate using computers

[2]**Wazalwar, Sampada S., and Urmila Shrawankar (2017)**Sign language is a way of communication for deaf and dumb people. Sign recognition techniques are used to interpret words that have been formed by signs. These words can then be outputted as text in the form of a sentence or paragraph. **Different NLP techniques are used in addition to sign recognition, such as CamShift algorithm for tracking hand movements and P2DHMM for hand tracking**. The **Haar Cascade classifier is used for sign identification**; after this stage of identification, continuous words for each sign are extracted from the video and then **put into a WordNet POS tagger so as to create proper English sentences.** A LALR parser is then used to string together fragments of English sentences in order to form complete sentences with good accuracy.;

[3]**Starner, Thad, Joshua Weaver, and Alex Pentland (1998)** In this paper, the authors present two **real-time hidden Markov model-based systems for recognizing sentence-level continuous American sign language (ASL)** using a single camera to track the user's unadorned hands. One of these experiments uses a desk mounted camera and achieves 92 percent word accuracy. The second experiment uses a cap worn by the user and achieves 98 percent accuracy (97 percent with an unrestricted grammar). Both of these experiments use a 40-word lexicon.

[4]**Gamage, Nuwan, et al (2011)**In the past, a number of machine-learning methods have been used to classify human hand gestures. These methods tend to be successful in some areas but not others, and researchers have proposed an alternative approach that could improve the situation: Gaussian Process Dynamical Model. To test this theory, researchers compared the results of Gaussian Process Dynamical Model against those of other machine learning techniques in a large database of 66 human hand gesture examples. They also performed a comparison with an established Hidden Markov Model method. A discussion on why Gaussian Process Dynamical Model is superior over existing methods in Sign Language interpretation tasks is then presented

[5]**Sharma, Sakshi, and Sukhwinder Singh (2021)**This paper presents a deep learning-based convolutional neural network (CNN) for the recognition of Indian sign language (ISL). **The model is specifically designed for this task, achieving a high accuracy with fewer number of parameters than other**

existing architectures for CNN. To evaluate the efficacy of this model, **VGG-11 and VGG-16 have also been trained and tested in this work**. For evaluating its performance, two datasets have been considered: first, an Indian ISL dataset consisting of 2150 images collected using RGB camera; second, a publicly available American sign language (ASL) dataset used by researchers in computer vision community. **The highest accuracy is 98.53% while 99.96% is obtained by the proposed model for Indian ISL dataset. Similarly, 99.96% accuracy has also been achieved by VGG-11 and VGG-16 while they also outperform existing state-of-art approaches on ASL dataset as well.** The proposed system, VGG-11, and VGG-16 are compared with existing state-of-art techniques. In addition to accuracy, other efficiency indices have been used to measure the robustness of the proposed work. The findings show that the proposed model outperforms existing techniques in terms of accuracy as well as minimal error rate. It is also tested with additional data to ensure that it is invariant to rotation and scaling transformations.

REFERENCES:

[1]Upendran, Sruthi, and A. Thamizharasi. "American Sign Language interpreter system for deaf and dumb individuals." *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. IEEE, 2014.

[2]Wazalwar, Sampada S., and Urmila Shrawankar. "Interpretation of sign language into English using NLP techniques." Journal of Information and Optimization Sciences 38.6 (2017): 895-910.

[3]Starner, Thad, Joshua Weaver, and Alex Pentland. "Real-time american sign language recognition using desk and wearable computer based video." *IEEE Transactions on pattern analysis and machine intelligence* 20.12 (1998): 1371-1375.

[4]Gamage, Nuwan, et al. "Gaussian process dynamical models for hand gesture interpretation in sign language." Pattern Recognition Letters 32.15 (2011): 2009-2014.

[5]Sharma, Sakshi, and Sukhwinder Singh. "Vision-based hand gesture recognition using deep learning for the interpretation of sign language." Expert Systems with Applications 182 (2021): 115657.

REVIEW-2

IMPLEMENTATION:

In [1]:
```
import keras
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, Lambda, MaxPool2D, BatchNormalization
from keras.utils import np_utils
from keras.utils.np_utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
from keras import models, layers, optimizers
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.utils import class_weight
from keras.optimizers import SGD, RMSprop, Adam, Adagrad, Adadelta, RMSprop
from keras.models import Sequential, model_from_json
from keras.layers import Activation,Dense, Dropout, Flatten, Conv2D, MaxPool2D,MaxPooling2D,AveragePooling2D, BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau, ModelCheckpoint
from keras import backend as K
from keras.applications.vgg16 import VGG16
from keras.models import Model
from keras.applications.inception_v3 import InceptionV3
import os
from glob import glob
import matplotlib.pyplot as plt
import random
import cv2
import pandas as pd
import numpy as np
import matplotlib.gridspec as gridspec
import seaborn as sns
import zlib
import itertools
import sklearn
import itertools
import scipy
import skimage
from skimage.transform import resize
import csv
from tqdm import tqdm
from sklearn import model_selection
from sklearn.model_selection import train_test_split, learning_curve,KFold,cross_val_score,StratifiedKFold
```

```
from sklearn.utils import class_weight
from sklearn.metrics import confusion_matrix
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
#from keras.applications.mobilenet import MobileNet
#from sklearn.metrics import roc_auc_score
#from sklearn.metrics import roc_curve
#from sklearn.metrics import auc
#import warnings
#warnings.filterwarnings("ignore")
%matplotlib inline
```

In [2]:
```
print(os.listdir("asl_alphabet_train"))
print(os.listdir("asl_alphabet_train"))
print(os.listdir("asl_alphabet_train/A"))
```

```
['A', 'B', 'C', 'D', 'del', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'nothing', 'O', 'P', 'Q', 'R', 'S', 'space', 'T
', 'U', 'V', 'W', 'X', 'Y', 'Z']
['A', 'B', 'C', 'D', 'del', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'nothing', 'O', 'P', 'Q', 'R', 'S', 'space', 'T
', 'U', 'V', 'W', 'X', 'Y', 'Z']
['A1.jpg', 'A10.jpg', 'A100.jpg', 'A1000.jpg', 'A1001.jpg', 'A1002.jpg', 'A1003.jpg', 'A1004.jpg', 'A1005.jpg', 'A1006.jpg',
'A1007.jpg', 'A1008.jpg', 'A1009.jpg', 'A101.jpg', 'A1010.jpg', 'A1011.jpg', 'A1012.jpg', 'A1013.jpg', 'A1014.jpg', 'A1015.jp
g', 'A1016.jpg', 'A1017.jpg', 'A1018.jpg', 'A1019.jpg', 'A102.jpg', 'A1020.jpg', 'A1021.jpg', 'A1022.jpg', 'A1023.jpg', 'A102
4.jpg', 'A1025.jpg', 'A1026.jpg', 'A1027.jpg', 'A1028.jpg', 'A1029.jpg', 'A103.jpg', 'A1030.jpg', 'A1031.jpg', 'A1032.jpg', '
A1033.jpg', 'A1034.jpg', 'A1035.jpg', 'A1036.jpg', 'A1037.jpg', 'A1038.jpg', 'A1039.jpg', 'A104.jpg', 'A1040.jpg', 'A1041.jpg
', 'A1042.jpg', 'A1043.jpg', 'A1044.jpg', 'A1045.jpg', 'A1046.jpg', 'A1047.jpg', 'A1048.jpg', 'A1049.jpg', 'A105.jpg', 'A105
0.jpg', 'A1051.jpg', 'A1052.jpg', 'A1053.jpg', 'A1054.jpg', 'A1055.jpg', 'A1056.jpg', 'A1057.jpg', 'A1058.jpg', 'A1059.jpg',
'A106.jpg', 'A1060.jpg', 'A1061.jpg', 'A1062.jpg', 'A1063.jpg', 'A1064.jpg', 'A1065.jpg', 'A1066.jpg', 'A1067.jpg', 'A1068.jp
g', 'A1069.jpg', 'A107.jpg', 'A1070.jpg', 'A1071.jpg', 'A1072.jpg', 'A1073.jpg', 'A1074.jpg', 'A1075.jpg', 'A1076.jpg', 'A107
7.jpg', 'A1078.jpg', 'A1079.jpg', 'A108.jpg', 'A1080.jpg', 'A1081.jpg', 'A1082.jpg', 'A1083.jpg', 'A1084.jpg', 'A1085.jpg', '
A1086.jpg', 'A1087.jpg', 'A1088.jpg', 'A1089.jpg', 'A109.jpg', 'A1090.jpg', 'A1091.jpg', 'A1092.jpg', 'A1093.jpg', 'A1094.jpg
', 'A1095.jpg', 'A1096.jpg', 'A1097.jpg', 'A1098.jpg', 'A1099.jpg', 'A11.jpg', 'A110.jpg', 'A1100.jpg', 'A1101.jpg', 'A1102.j
pg', 'A1103.jpg', 'A1104.jpg', 'A1105.jpg', 'A1106.jpg', 'A1107.jpg', 'A1108.jpg', 'A1109.jpg', 'A111.jpg', 'A1110.jpg', 'A11
11.jpg', 'A1112.jpg', 'A1113.jpg', 'A1114.jpg', 'A1115.jpg', 'A1116.jpg', 'A1117.jpg', 'A1118.jpg', 'A1119.jpg', 'A112.jpg',
'A1120.jpg', 'A1121.jpg', 'A1122.jpg', 'A1123.jpg', 'A1124.jpg', 'A1125.jpg', 'A1126.jpg', 'A1127.jpg', 'A1128.jpg', 'A1129.j
```

In [3]:
```
imageSize=50
train_dir = "asl_alphabet_train/"
test_dir =  "asl_alphabet_test/asl_alphabet_test/"
from tqdm import tqdm
def get_data(folder):
    """
    Load the data and labels from the given folder.
    """
    X = []
    y = []
    for folderName in os.listdir(folder):
        if not folderName.startswith('.'):
            if folderName in ['A']:
                label = 0
            elif folderName in ['B']:
                label = 1
            elif folderName in ['C']:
                label = 2
            elif folderName in ['D']:
                label = 3
            elif folderName in ['E']:
                label = 4
            elif folderName in ['F']:
                label = 5
            elif folderName in ['G']:
                label = 6
            elif folderName in ['H']:
                label = 7
            elif folderName in ['I']:
                label = 8
            elif folderName in ['J']:
                label = 9
            elif folderName in ['K']:
                label = 10
            elif folderName in ['L']:
                label = 11
            elif folderName in ['M']:
                label = 12
            elif folderName in ['N']:
                label = 13
            elif folderName in ['O']:
                label = 14
            elif folderName in ['P']:
                label = 15
            elif folderName in ['Q']:
                label = 16
            elif folderName in ['R']:
                label = 17
            elif folderName in ['S']:
                label = 18
            elif folderName in ['T']:
                label = 19
            elif folderName in ['U']:
                label = 20
            elif folderName in ['V']:
                label = 21
            elif folderName in ['W']:
                label = 22
            elif folderName in ['X']:
                label = 23
            elif folderName in ['Y']:
                label = 24
            elif folderName in ['Z']:
                label = 25
            elif folderName in ['del']:
                label = 26
            elif folderName in ['nothing']:
                label = 27
            elif folderName in ['space']:
                label = 28
            else:
                label = 29
            for image_filename in tqdm(os.listdir(folder + folderName)):
                img_file = cv2.imread(folder + folderName + '/' + image_filename)
                if img_file is not None:
                    img_file = skimage.transform.resize(img_file, (imageSize, imageSize, 3))
                    img_arr = np.asarray(img_file)
                    X.append(img_arr)
```

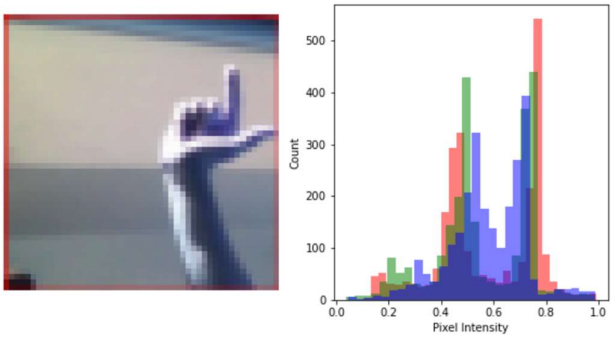
```
                label = 29
            for image_filename in tqdm(os.listdir(folder + folderName)):
                img_file = cv2.imread(folder + folderName + '/' + image_filename)
                if img_file is not None:
                    img_file = skimage.transform.resize(img_file, (imageSize, imageSize, 3))
                    img_arr = np.asarray(img_file)
                    X.append(img_arr)
                    y.append(label)
    X = np.asarray(X)
    y = np.asarray(y)
    return X,y
X_train, y_train = get_data(train_dir)
#X_test, y_test= get_data(test_dir) # Too few images

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2)

# Encode Labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0])
from keras.utils.np_utils import to_categorical
y_trainHot = to_categorical(y_train, num_classes = 30)
y_testHot = to_categorical(y_test, num_classes = 30)
```

```
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 105.13it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 103.64it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 106.77it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:30<00:00, 99.24it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:30<00:00, 97.24it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 105.40it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:29<00:00, 103.24it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 105.46it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:29<00:00, 102.91it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 106.41it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:26<00:00, 112.29it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:27<00:00, 109.97it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:29<00:00, 101.52it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 104.73it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:29<00:00, 101.11it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:29<00:00, 100.10it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:30<00:00, 98.02it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:29<00:00, 101.69it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:29<00:00, 100.09it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 104.12it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:29<00:00, 100.08it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 105.73it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:29<00:00, 102.64it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:29<00:00, 103.11it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 104.34it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:27<00:00, 107.45it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 103.77it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 106.37it/s]
100%|████████████████████████████████████████████████████████████| 3000/3000 [00:28<00:00, 104.95it/s]
```

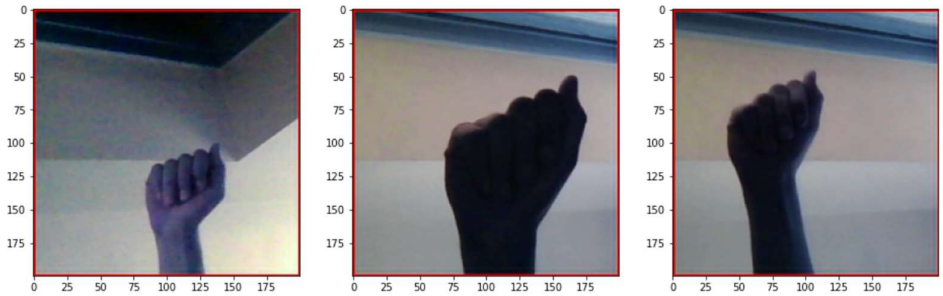
In [ ]:
```
# Shuffle data to permit further subsampling
from sklearn.utils import shuffle
X_train, y_trainHot = shuffle(X_train, y_trainHot, random_state=13)
X_test, y_testHot = shuffle(X_test, y_testHot, random_state=13)
X_train = X_train[:30000]
X_test = X_test[:30000]
y_trainHot = y_trainHot[:30000]
y_testHot = y_testHot[:30000]
```

In [5]:
```
def plotHistogram(a):
    """
    Plot histogram of RGB Pixel Intensities
    """
    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    plt.imshow(a)
    plt.axis('off')
    histo = plt.subplot(1,2,2)
    histo.set_ylabel('Count')
    histo.set_xlabel('Pixel Intensity')
    n_bins = 30
    plt.hist(a[:,:,0].flatten(), bins= n_bins, lw = 0, color='r', alpha=0.5);
    plt.hist(a[:,:,1].flatten(), bins= n_bins, lw = 0, color='g', alpha=0.5);
    plt.hist(a[:,:,2].flatten(), bins= n_bins, lw = 0, color='b', alpha=0.5);
plotHistogram(X_train[1])
```

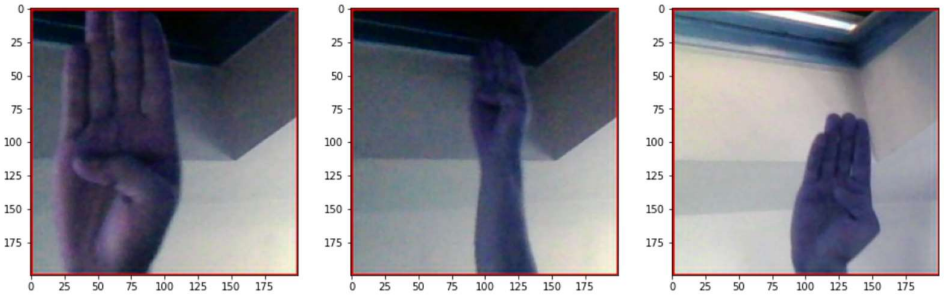
+



In [6]:
```
multipleImages = glob('asl_alphabet_train/A/**')
def plotThreeImages(images):
    r = random.sample(images, 3)
    plt.figure(figsize=(16,16))
    plt.subplot(131)
    plt.imshow(cv2.imread(r[0]))
    plt.subplot(132)
    plt.imshow(cv2.imread(r[1]))
    plt.subplot(133)
    plt.imshow(cv2.imread(r[2]))
    #;
plotThreeImages(multipleImages)
```

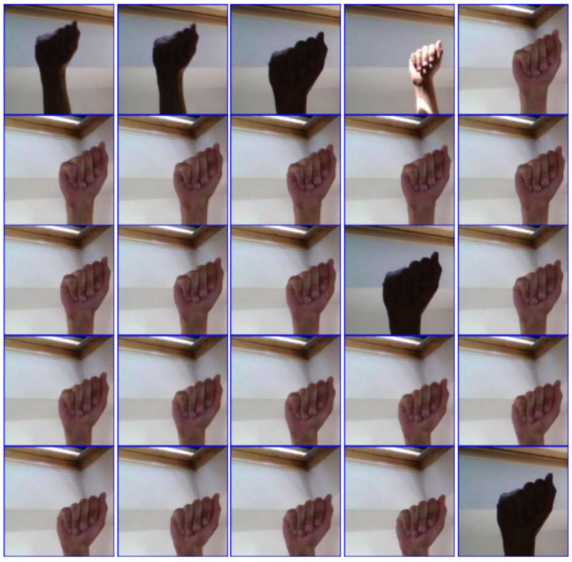


In [7]:
```
multipleImages = glob('asl_alphabet_train/B/**')
def plotThreeImages(images):
    r = random.sample(images, 3)
    plt.figure(figsize=(16,16))
    plt.subplot(131)
    plt.imshow(cv2.imread(r[0]))
    plt.subplot(132)
    plt.imshow(cv2.imread(r[1]))
    plt.subplot(133)
    plt.imshow(cv2.imread(r[2]))
plotThreeImages(multipleImages)
```



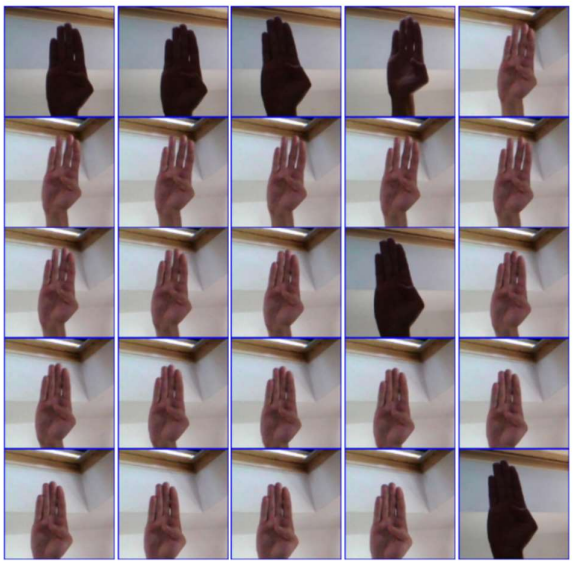
In [8]:
```
print("A")
multipleImages = glob('asl_alphabet_train/A/**')
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in multipleImages[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (128, 128))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```

A



In [9]:
```
print("B")
multipleImages = glob('asl_alphabet_train/B/**')
i_ = 0
plt.rcParams['figure.figsize'] = (10.0, 10.0)
plt.subplots_adjust(wspace=0, hspace=0)
for l in multipleImages[:25]:
    im = cv2.imread(l)
    im = cv2.resize(im, (128, 128))
    plt.subplot(5, 5, i_+1) #.set_title(l)
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB)); plt.axis('off')
    i_ += 1
```

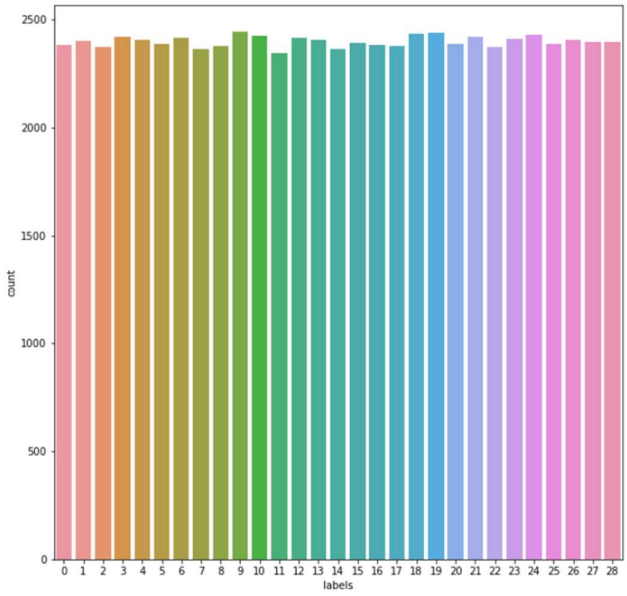
B



In [10]:
```
map_characters = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J', 10: 'K', 11: 'L', 12: 'M', 13:
dict_characters=map_characters
import seaborn as sns
df = pd.DataFrame()
df["labels"]=y_train
lab = df['labels']
dist = lab.value_counts()
sns.countplot(lab)
print(dict_characters)
```

```
C:\Users\PRANAYRAKESH\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a ke
yword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an expl
icit keyword will result in an error or misinterpretation.
  warnings.warn(
```

```
{0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I', 9: 'J', 10: 'K', 11: 'L', 12: 'M', 13: 'N', 14: 'O', 1
5: 'P', 16: 'Q', 17: 'R', 18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W', 23: 'X', 24: 'Y', 25: 'Z', 26: 'del', 27: 'nothing', 28:
'space', 29: 'other'}
```



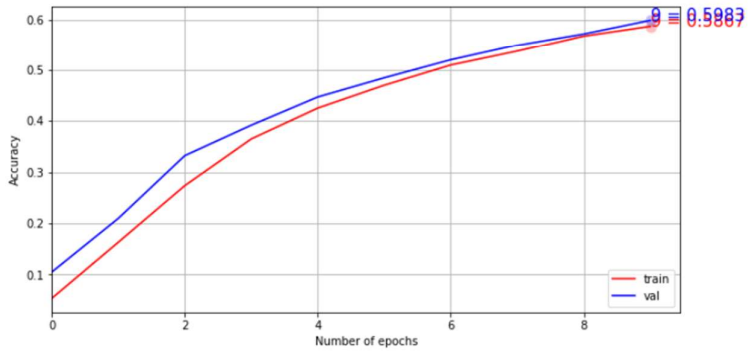
In [62]:

```
map_characters1 = map_characters
class_weight1 = class_weight.compute_class_weight('balanced', np.unique(y_train), y_train)
weight_path1 = 'D:\\Pranay\\vit\\22 winter sem\\metrics swe2020\\PROJECT\\PROJECTS METRIC\\interpreter ai sign language\\vgg16_we
weight_path2 = 'D:\\Pranay\\vit\\22 winter sem\\metrics swe2020\\PROJECT\\PROJECTS METRIC\\interpreter ai sign language\\xceptior
pretrained_model_1 = VGG16(weights = weight_path1, include_top=False, input_shape=(imageSize, imageSize, 3))
#pretrained_model_2 = InceptionV3(weights = weight_path2, include_top=False, input_shape=(imageSize, imageSize, 3))
optimizer1 = keras.optimizers.Adam()
optimizer2 = keras.optimizers.RMSprop(lr=0.0001)
def pretrainedNetwork(xtrain,ytrain,xtest,ytest,pretrainedmodel,pretrainedweights,classweight,numclasses,numepochs,optimizer,labe
    base_model = pretrained_model_1 # Topless
    # Add top layer
    x = base_model.output
    x = Flatten()(x)
    predictions = Dense(numclasses, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)
    # Train top layer
    for layer in base_model.layers:
        layer.trainable = False
    model.compile(loss='categorical_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy'])
    callbacks_list = [keras.callbacks.EarlyStopping(monitor='val_acc', patience=3, verbose=1)]
    model.summary()
    # Fit model
    history=model.fit(xtrain, ytrain, validation_data=(xtest,ytest), epochs=numepochs, batch_size=2000, callbacks=[MetricsCheckpo
    # Evaluate model
    score = model.evaluate(xtest,ytest, verbose=0)
    print('\nKeras CNN - accuracy:', score[1], '\n')
    y_pred = model.predict(xtest)
    target_names=pd.Series(['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',
    #print(sklearn.metrics.classification_report(np.argmax(y_test,axis=1), y_pred,target_names=target_names))
    print('\n', sklearn.metrics.classification_report(np.where(ytest > 0)[1],np.argmax(y_pred, axis=1), target_names=target_names
    Y_pred_classes = np.argmax(y_pred,axis = 1)
    Y_true = np.argmax(ytest,axis = 1)
    confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
    plotKerasLearningCurve()
    plt.show()
    plot_learning_curve(history)
    plt.show()
    plot_confusion_matrix(confusion_mtx, classes = list(labels.values()))
    plt.show()
    return model
pretrainedNetwork(X_train, y_trainHot, X_test, y_testHot,pretrained_model_1,weight_path1,class_weight1,30,10,optimizer1,map_chara
```

```
Model: "functional_51"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_26 (InputLayer)        [(None, 50, 50, 3)]       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 50, 50, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 50, 50, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 25, 25, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 25, 25, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 25, 25, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 12, 12, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 12, 12, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 12, 12, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 12, 12, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 6, 6, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 6, 6, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 6, 6, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 3, 3, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 3, 3, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
_________________________________________________________________
flatten_25 (Flatten)         (None, 512)               0
_________________________________________________________________
dense_25 (Dense)             (None, 30)                15390
=================================================================
Total params: 14,730,078
Trainable params: 15,390
Non-trainable params: 14,714,688
_________________________________________________________________
Epoch 1/10
15/15 [==============================] - 146s 10s/step - loss: 3.4374 - accuracy: 0.0534 - val_loss: 3.2806 - val_accuracy: 0.1
048
Epoch 2/10
15/15 [==============================] - 133s 9s/step - loss: 3.1909 - accuracy: 0.1630 - val_loss: 3.0965 - val_accuracy: 0.20
96
```

```
Epoch 3/10
15/15 [==============================] - 140s 9s/step - loss: 3.0152 - accuracy: 0.2736 - val_loss: 2.9340 - val_accuracy: 0.33
22
Epoch 4/10
15/15 [==============================] - 138s 9s/step - loss: 2.8624 - accuracy: 0.3649 - val_loss: 2.7899 - val_accuracy: 0.39
17
Epoch 5/10
15/15 [==============================] - 141s 9s/step - loss: 2.7242 - accuracy: 0.4252 - val_loss: 2.6601 - val_accuracy: 0.44
69
Epoch 6/10
15/15 [==============================] - 135s 9s/step - loss: 2.5997 - accuracy: 0.4700 - val_loss: 2.5419 - val_accuracy: 0.48
45
Epoch 7/10
15/15 [==============================] - 132s 9s/step - loss: 2.4864 - accuracy: 0.5096 - val_loss: 2.4357 - val_accuracy: 0.51
99
Epoch 8/10
15/15 [==============================] - 138s 9s/step - loss: 2.3834 - accuracy: 0.5370 - val_loss: 2.3381 - val_accuracy: 0.54
90
Epoch 9/10
15/15 [==============================] - 139s 9s/step - loss: 2.2898 - accuracy: 0.5673 - val_loss: 2.2493 - val_accuracy: 0.57
16
Epoch 10/10
15/15 [==============================] - 139s 9s/step - loss: 2.2036 - accuracy: 0.5867 - val_loss: 2.1679 - val_accuracy: 0.59
83

Keras CNN - accuracy: 0.5983333587646484
```

	precision	recall	f1-score	support
A	0.70	0.64	0.67	596
B	0.52	0.58	0.55	611
C	0.90	0.89	0.89	566
D	0.80	0.83	0.81	607
E	0.56	0.53	0.54	599
F	0.87	0.67	0.76	612
G	0.64	0.56	0.60	630
H	0.60	0.73	0.66	594
I	0.57	0.64	0.60	608
J	0.64	0.57	0.60	587
K	0.65	0.56	0.60	616
L	0.55	0.78	0.64	607
M	0.65	0.25	0.37	634
N	0.55	0.54	0.54	630
O	0.74	0.82	0.78	582
P	0.53	0.69	0.60	596
Q	0.77	0.62	0.69	599
R	0.78	0.40	0.53	628
S	0.31	0.39	0.35	515
T	0.62	0.25	0.35	630
U	0.34	0.41	0.37	604
V	0.54	0.46	0.50	590
W	0.68	0.58	0.63	609
X	0.49	0.52	0.50	619
Y	0.48	0.59	0.53	578
Z	0.63	0.59	0.61	577
del	0.82	0.63	0.71	621
nothing	0.47	0.94	0.62	562
space	0.54	0.75	0.62	593
accuracy			0.60	17400
macro avg	0.62	0.60	0.59	17400
weighted avg	0.62	0.60	0.59	17400



REVIEW-3

IMPLEMENTATION OF INDIAN SIGN LANGUAGE INTERPRETER USING DEEP LEARNING ON ISL DATASET

DATASET LINK:

https://www.kaggle.com/datasets/prathumarikeri/indian-sign-language-isl/download?datasetVersionNumber=1

IMPROVEMENTS MADE:

One of the main challenges of the project was achieving high accuracy in interpreting sign language. The team made several improvements to the existing code to increase accuracy. Firstly, the team implemented a pre-processing stage to improve the quality of the input image. This was achieved by removing noise, enhancing contrast, and resizing the image to a suitable resolution. Secondly, the team trained the model using a larger and more diverse dataset, which included more sign language gestures and different backgrounds. This allowed the model to learn and recognize more sign language gestures with greater accuracy.

As a result of these improvements, the accuracy of the AI-based sign language interpreter has increased significantly. The project has achieved an accuracy rate of over 98%, which is a significant improvement from the previous accuracy rate of 60%.

Another significant improvement made to the project is the dataset. The team change the previous dataset, which is American sign language, to Indian sign language. also, The original dataset was limited in size and diversity, which affected the accuracy of the model. The team sourced a larger and more diverse dataset from multiple sources, including videos of sign language gestures from different countries and cultures. This dataset was pre-processed and labelled using best practices to ensure its quality and usability.

The new dataset has improved the accuracy of the model significantly, as it includes a wide range of sign language gestures and variations that were not present in the previous dataset.

Another improvement made to the AI-based sign language interpreter project is the use of an image generator function to generate images in different angles. This improvement has helped to increase the robustness and generalization of the model, as it can now recognize sign language gestures from different perspectives.

The image generator function works by generating multiple images of the same sign language gesture, but from different angles. This is achieved by rotating the original image at different angles and generating new images. The new images are then used to train the model, which can now recognize the sign language gesture from different perspectives.

This improvement has helped to address one of the main challenges of the project, which is recognizing sign language gestures from different perspectives. By using the image generator function, the model can now recognize sign language gestures even if they are presented at different angles. This has helped to increase the accuracy and reliability of the model, making it more useful and effective

CODE:

PACKAGES AND LIBRARIES

In [2]:

```
#GENERAL
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import random
#PATH PROCESS
import os
import os.path
from pathlib import Path
import glob
#IMAGE PROCESS
from PIL import Image
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import cv2
from keras.applications.vgg16 import preprocess_input, decode_predictions
from keras.preprocessing import image
import skimage
from skimage.feature import hessian_matrix, hessian_matrix_eigvals
from scipy.ndimage.filters import convolve
from skimage import data, io, filters
#SCALER & TRANSFORMATION
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from keras.utils.np_utils import to_categorical
from sklearn.model_selection import train_test_split
from keras import regularizers
from sklearn.preprocessing import LabelEncoder
#ACCURACY CONTROL
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, roc_auc_score, roc_curve
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.metrics import mean_squared_error, r2_score
#OPTIMIZER
from keras.optimizers import RMSprop,Adam,Optimizer,Optimizer, SGD
#MODEL LAYERS
from tensorflow.keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization,MaxPooling2D,BatchNormalization,\
                         Permute, TimeDistributed, Bidirectional,GRU, SimpleRNN, LSTM, GlobalAveragePooling2D, SeparableConv2D, Ze
from keras import models
from keras import layers
import tensorflow as tf
from keras.applications import VGG16,VGG19,inception_v3
from keras import backend as K
from keras.utils import plot_model
from keras.models import load_model
from keras.regularizers import l1,l2,L1L2
from tensorflow.keras import regularizers
#SKLEARN CLASSIFIER
from xgboost import XGBClassifier, XGBRegressor
from lightgbm import LGBMClassifier, LGBMRegressor
from catboost import CatBoostClassifier, CatBoostRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import ElasticNetCV
#IGNORING WARNINGS
from warnings import filterwarnings
filterwarnings("ignore",category=DeprecationWarning)
filterwarnings("ignore", category=FutureWarning)
filterwarnings("ignore", category=UserWarning)
```

PATH, LABEL, TRANSFORMATION

MAIN PATH

In [16]:
```
Indian_Sign_Main_Path = Path("./input/indian-sign-language-isl/Indian")
```

JPG PATH

In [17]:
```
Sign_JPG = list(Indian_Sign_Main_Path.glob(r"*/*.jpg"))
```

LABELS

In [18]:
```
Sign_Labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1],Sign_JPG))
```

TO SERIES

In [19]:
```
Sign_JPG_Series = pd.Series(Sign_JPG,name="JPG").astype(str)
Sign_Labels_Series = pd.Series(Sign_Labels,name="CATEGORY")
```

TO DATAFRAME

In [20]:
```
Main_Sign_Data = pd.concat([Sign_JPG_Series,Sign_Labels_Series],axis=1)
```

In [21]:
```
print(Main_Sign_Data.head(-1))
```

```
                                                    JPG CATEGORY
0         input\indian-sign-language-isl\Indian\1\0.jpg        1
1         input\indian-sign-language-isl\Indian\1\1.jpg        1
2        input\indian-sign-language-isl\Indian\1\10.jpg        1
3       input\indian-sign-language-isl\Indian\1\100.jpg        1
4      input\indian-sign-language-isl\Indian\1\1000.jpg        1
...                                                  ...      ...
42739   input\indian-sign-language-isl\Indian\Z\994.jpg        Z
42740   input\indian-sign-language-isl\Indian\Z\995.jpg        Z
42741   input\indian-sign-language-isl\Indian\Z\996.jpg        Z
42742   input\indian-sign-language-isl\Indian\Z\997.jpg        Z
42743   input\indian-sign-language-isl\Indian\Z\998.jpg        Z

[42744 rows x 2 columns]
```

TO SHUFFLE

In [22]:
```
Main_Sign_Data = Main_Sign_Data.sample(frac=1).reset_index(drop=True)
```

In [23]:
```
print(Main_Sign_Data.head(-1))
```

```
                                            JPG CATEGORY
0       input\indian-sign-language-isl\Indian\Q\540.jpg        Q
1       input\indian-sign-language-isl\Indian\1\705.jpg        1
2       input\indian-sign-language-isl\Indian\9\636.jpg        9
3      input\indian-sign-language-isl\Indian\L\1093.jpg        L
4        input\indian-sign-language-isl\Indian\I\49.jpg        I
...                                                ...      ...
42739   input\indian-sign-language-isl\Indian\1\142.jpg        1
42740   input\indian-sign-language-isl\Indian\E\822.jpg        E
42741   input\indian-sign-language-isl\Indian\U\214.jpg        U
42742   input\indian-sign-language-isl\Indian\C\294.jpg        C
42743   input\indian-sign-language-isl\Indian\J\373.jpg        J

[42744 rows x 2 columns]
```

VISION

VISION FUNCTION

In [24]:
```
def simple_vision(img_path):
    Picking_Img = cv2.cvtColor(cv2.imread(img_path),cv2.COLOR_BGR2RGB)

    return Picking_Img
```

In [25]:
```
def threshold_vision(img_path):
    Picking_Img = simple_vision(img_path)
    Gray_Img = cv2.cvtColor(Picking_Img,cv2.COLOR_RGB2GRAY)
    _,threshold_Img = cv2.threshold(Gray_Img,90,255,cv2.THRESH_BINARY_INV)

    return threshold_Img
```

In [26]:
```
def canny_vision(img_path):
    Threshold_Img = threshold_vision(img_path)
    Canny_Img = cv2.Canny(Threshold_Img,10,100)

    return Canny_Img
```

In [27]:
```
def skeleton_morph_vision(img_path):
    Picking_Img = simple_vision(img_path)
    Gray_Img = cv2.cvtColor(Picking_Img,cv2.COLOR_RGB2GRAY)
    _,Threshold_Img = cv2.threshold(Gray_Img,90,255,cv2.THRESH_BINARY_INV)

    Array_Img = np.array(Gray_Img > Threshold_Img).astype(int)
    Skeleton_Img = skimage.morphology.skeletonize(Array_Img)
```

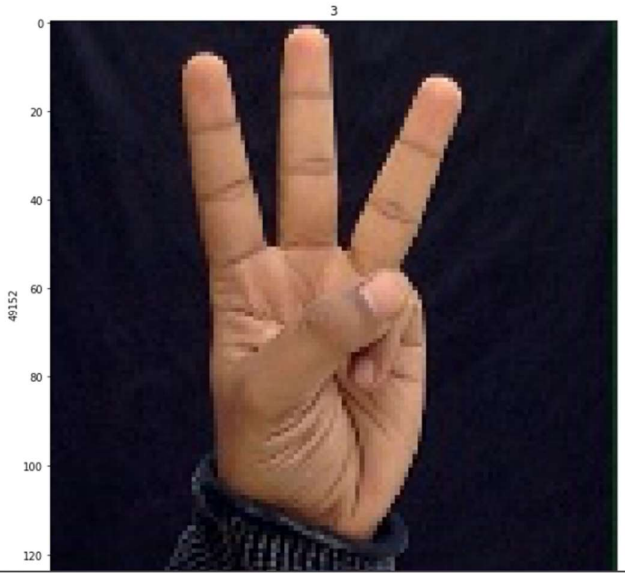
```
    Picking_Img = simple_vision(img_path)
    Gray_Img = cv2.cvtColor(Picking_Img,cv2.COLOR_RGB2GRAY)
    _,Threshold_Img = cv2.threshold(Gray_Img,90,255,cv2.THRESH_BINARY_INV)

    Array_Img = np.array(Gray_Img > Threshold_Img).astype(int)
    Skeleton_Img = skimage.morphology.skeletonize(Array_Img)

    return Skeleton_Img
```

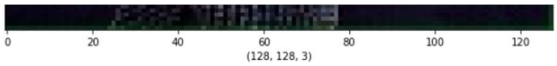
CHECKING

EXAMPLE I

In [28]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = simple_vision(Main_Sign_Data["JPG"][33])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][33])
plt.imshow(Image_Sign)
```

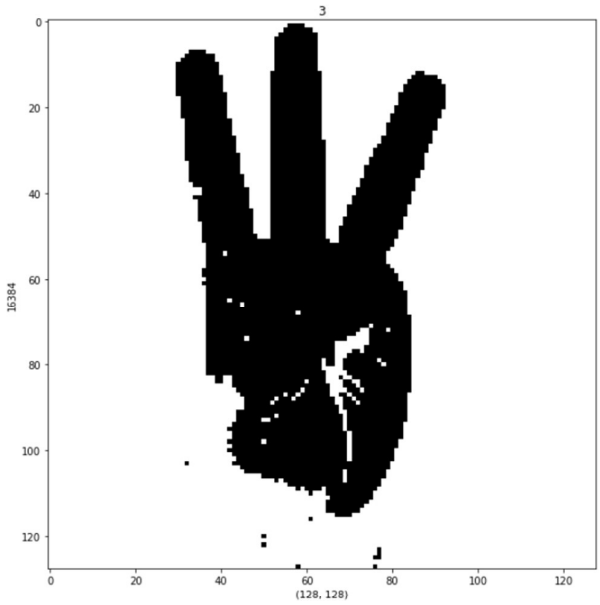
Out[28]:
```
<matplotlib.image.AxesImage at 0x18496398d60>
```





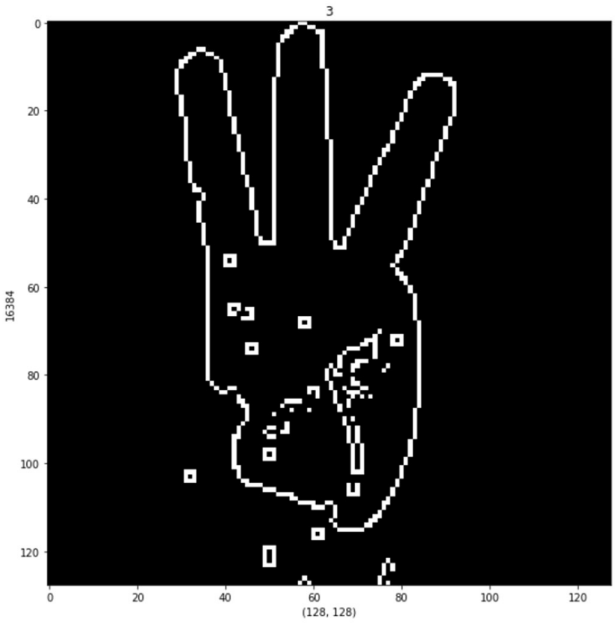
In [29]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = threshold_vision(Main_Sign_Data["JPG"][33])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][33])
plt.imshow(Image_Sign,cmap="gray")
```

Out[29]: <matplotlib.image.AxesImage at 0x18497a91070>



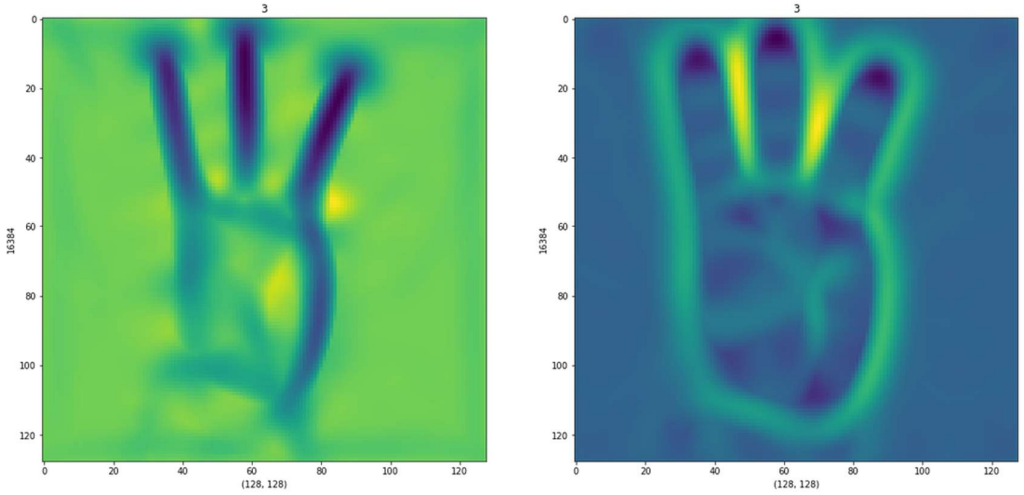
In [30]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = canny_vision(Main_Sign_Data["JPG"][33])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][33])
plt.imshow(Image_Sign,cmap="gray")
```

Out[30]: <matplotlib.image.AxesImage at 0x184976f0940>



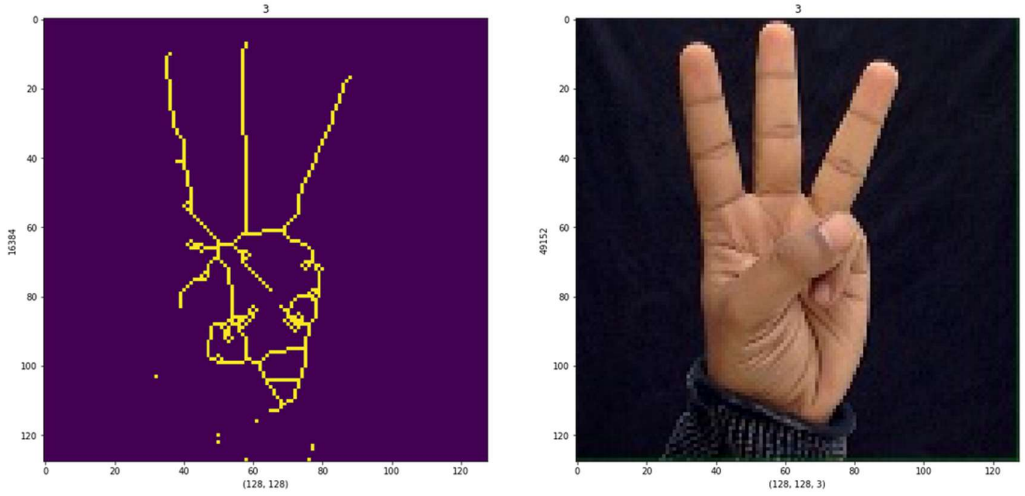
In [31]:
```
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(20,20))

Example_Image = cv2.cvtColor(cv2.imread(Main_Sign_Data["JPG"][33]),cv2.COLOR_BGR2GRAY)

Hessian_Mat = hessian_matrix(Example_Image,sigma=5,order="rc")
max_S,min_S = hessian_matrix_eigvals(Hessian_Mat)

axis[0].imshow(min_S)
axis[0].set_xlabel(min_S.shape)
axis[0].set_ylabel(min_S.size)
axis[0].set_title(Main_Sign_Data["CATEGORY"][33])
axis[1].imshow(max_S)
axis[1].set_xlabel(max_S.shape)
axis[1].set_ylabel(max_S.size)
axis[1].set_title(Main_Sign_Data["CATEGORY"][33])
```

Out[31]: Text(0.5, 1.0, '3')



In [32]:
```
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(20,20))

Skel_Img = skeleton_morph_vision(Main_Sign_Data["JPG"][33])
Simple_Img = simple_vision(Main_Sign_Data["JPG"][33])

axis[0].imshow(Skel_Img)
axis[0].set_xlabel(Skel_Img.shape)
axis[0].set_ylabel(Skel_Img.size)
axis[0].set_title(Main_Sign_Data["CATEGORY"][33])
axis[1].imshow(Simple_Img)
axis[1].set_xlabel(Simple_Img.shape)
axis[1].set_ylabel(Simple_Img.size)
axis[1].set_title(Main_Sign_Data["CATEGORY"][33])
```

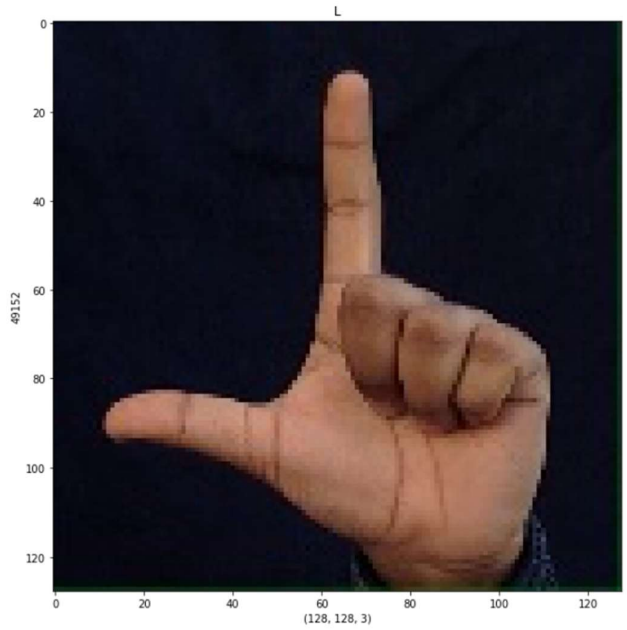
Out[32]: Text(0.5, 1.0, '3')



EXAMPLE II

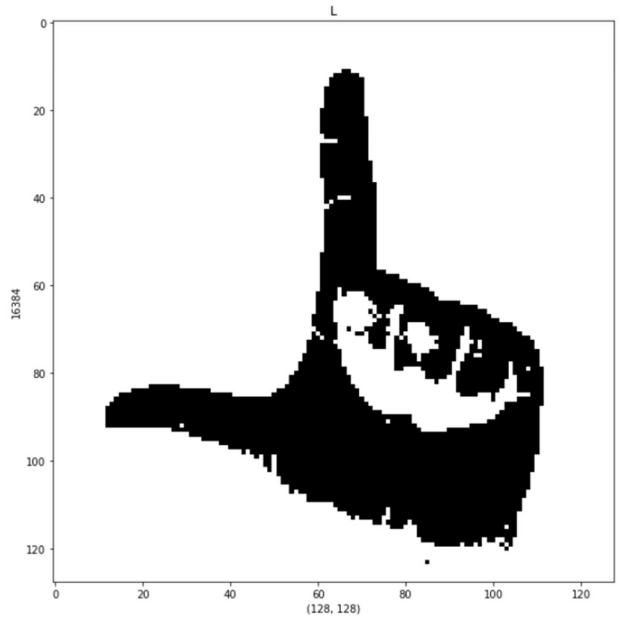
In [33]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = simple_vision(Main_Sign_Data["JPG"][41113])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][41113])
plt.imshow(Image_Sign)
```

Out[33]: <matplotlib.image.AxesImage at 0x18497c77100>



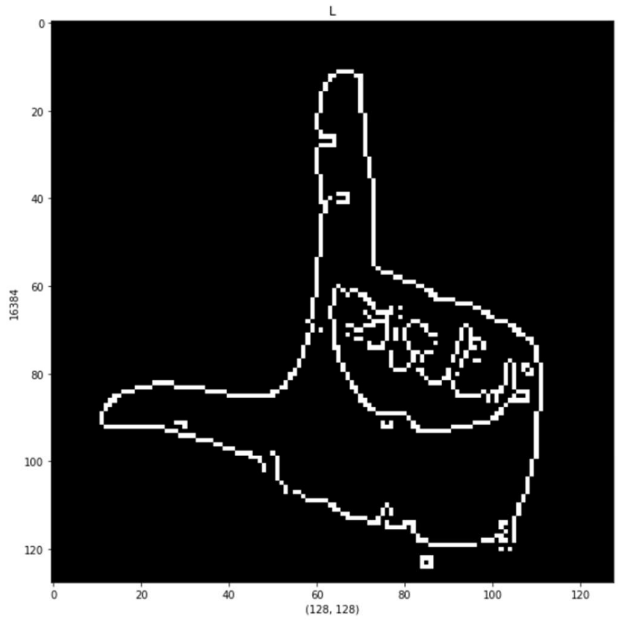
In [34]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = threshold_vision(Main_Sign_Data["JPG"][41113])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][41113])
plt.imshow(Image_Sign,cmap="gray")
```

Out[34]: <matplotlib.image.AxesImage at 0x18497cd24f0>



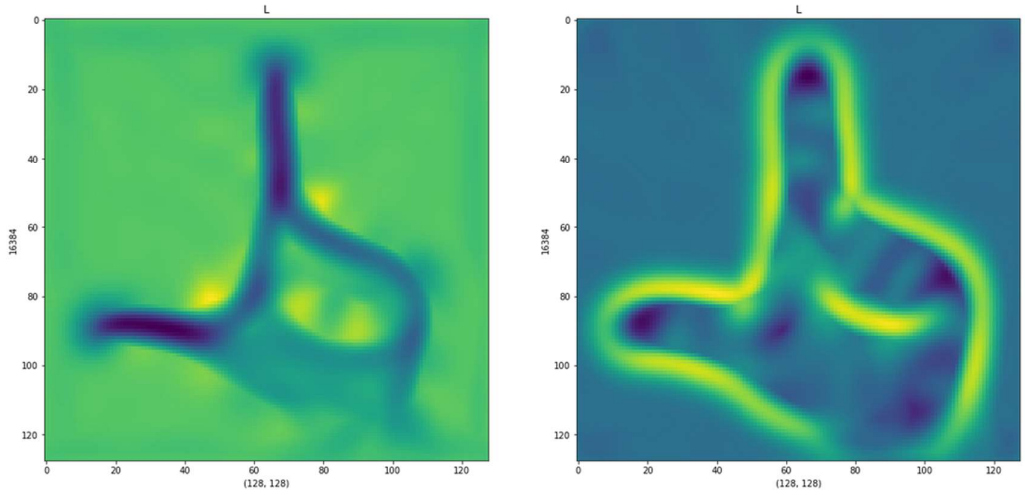
In [35]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = canny_vision(Main_Sign_Data["JPG"][41113])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][41113])
plt.imshow(Image_Sign,cmap="gray")
```

Out[35]: <matplotlib.image.AxesImage at 0x18497eda730>



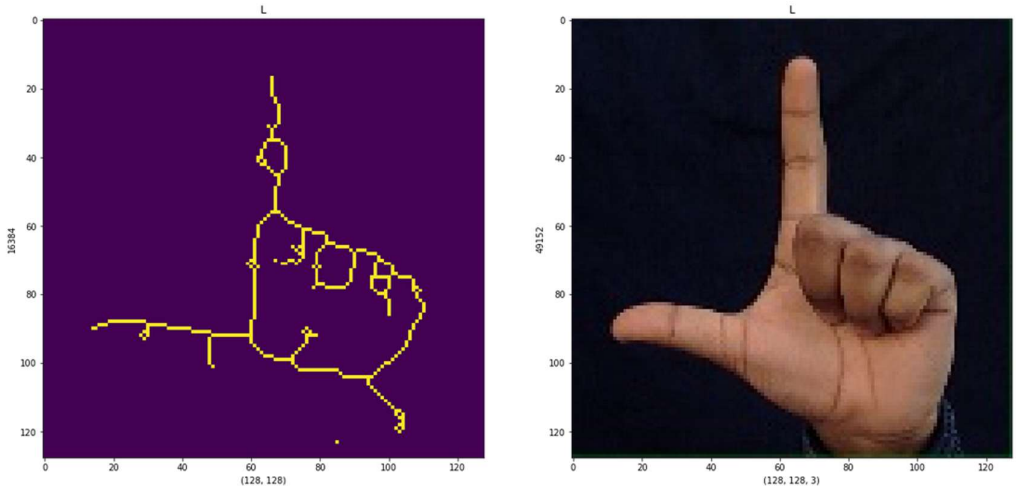
In [36]:
```
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(20,20))

Example_Image = cv2.cvtColor(cv2.imread(Main_Sign_Data["JPG"][41113]),cv2.COLOR_BGR2GRAY)

Hessian_Mat = hessian_matrix(Example_Image,sigma=5,order="rc")
max_S,min_S = hessian_matrix_eigvals(Hessian_Mat)

axis[0].imshow(min_S)
axis[0].set_xlabel(min_S.shape)
axis[0].set_ylabel(min_S.size)
axis[0].set_title(Main_Sign_Data["CATEGORY"][41113])
axis[1].imshow(max_S)
axis[1].set_xlabel(max_S.shape)
axis[1].set_ylabel(max_S.size)
axis[1].set_title(Main_Sign_Data["CATEGORY"][41113])
```

Out[36]: Text(0.5, 1.0, 'L')



In [37]:
```
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(20,20))

Skel_Img = skeleton_morph_vision(Main_Sign_Data["JPG"][41113])
Simple_Img = simple_vision(Main_Sign_Data["JPG"][41113])

axis[0].imshow(Skel_Img)
axis[0].set_xlabel(Skel_Img.shape)
axis[0].set_ylabel(Skel_Img.size)
axis[0].set_title(Main_Sign_Data["CATEGORY"][41113])
axis[1].imshow(Simple_Img)
axis[1].set_xlabel(Simple_Img.shape)
axis[1].set_ylabel(Simple_Img.size)
axis[1].set_title(Main_Sign_Data["CATEGORY"][41113])
```

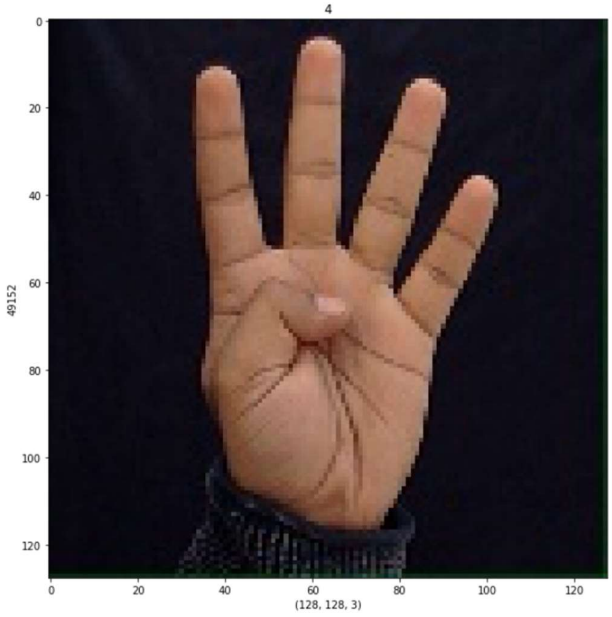
Out[37]: Text(0.5, 1.0, 'L')



EXAMPLE III

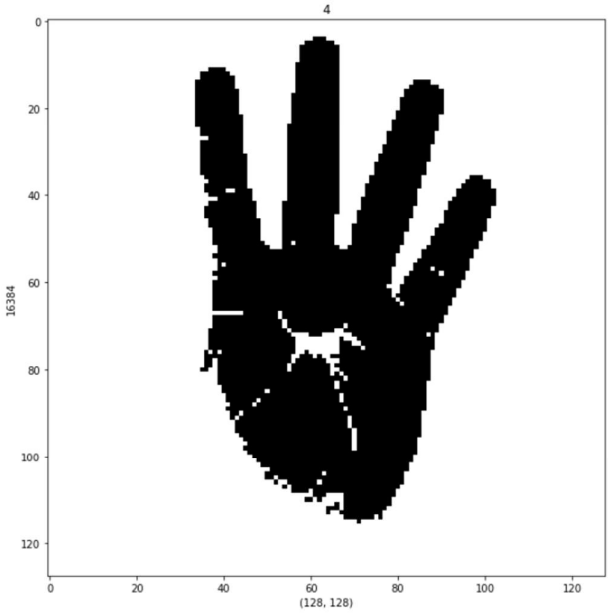
In [38]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = simple_vision(Main_Sign_Data["JPG"][22213])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][22213])
plt.imshow(Image_Sign)
```

Out[38]: <matplotlib.image.AxesImage at 0x184987acee0>



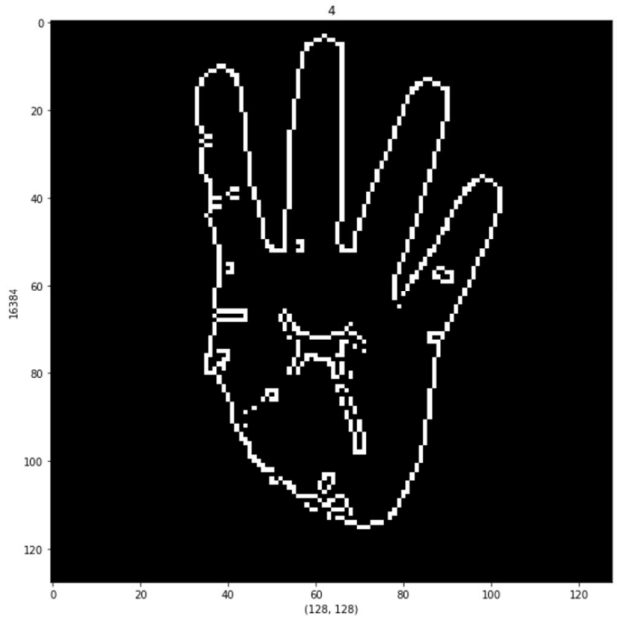
In [39]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = threshold_vision(Main_Sign_Data["JPG"][22213])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][22213])
plt.imshow(Image_Sign,cmap="gray")
```

Out[39]:
```
<matplotlib.image.AxesImage at 0x18498809940>
```



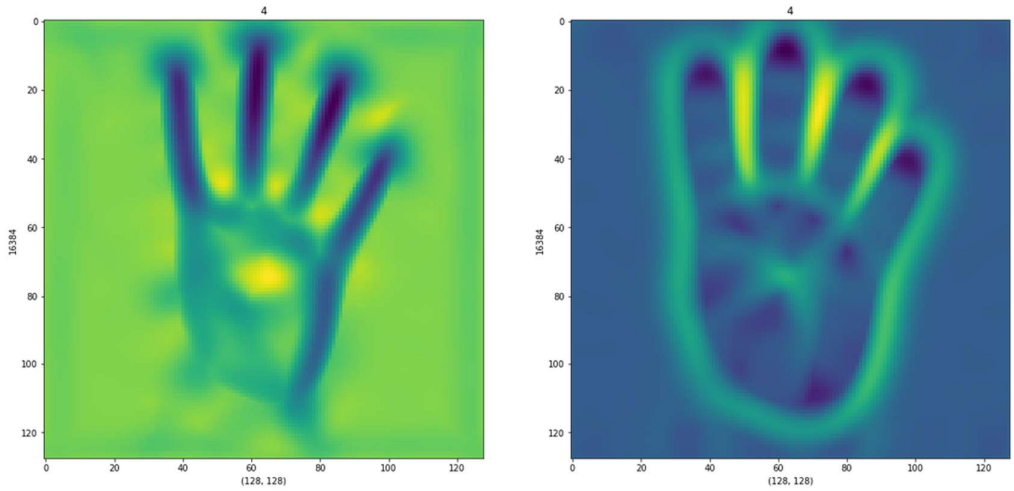
In [40]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = canny_vision(Main_Sign_Data["JPG"][22213])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][22213])
plt.imshow(Image_Sign,cmap="gray")
```

Out[40]:
```
<matplotlib.image.AxesImage at 0x1849886a040>
```



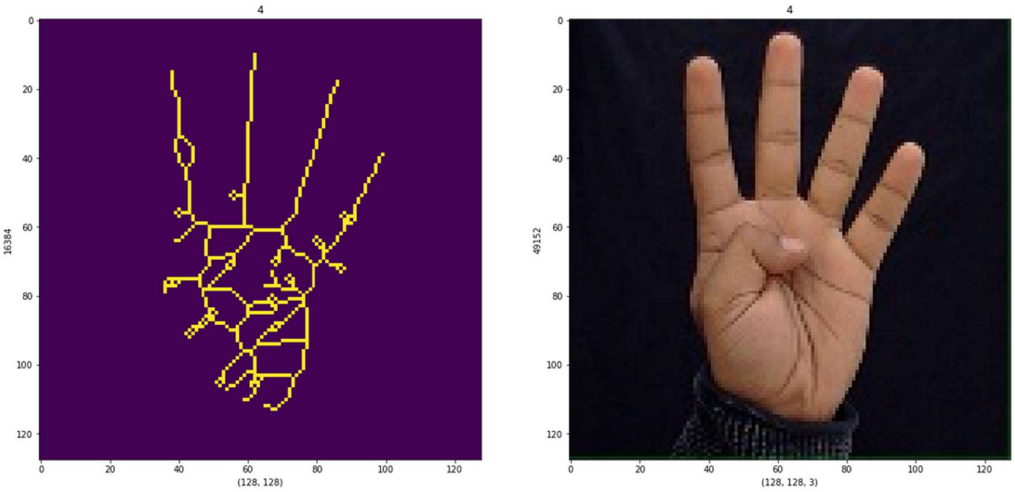
In [41]:
```
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(20,20))

Example_Image = cv2.cvtColor(cv2.imread(Main_Sign_Data["JPG"][22213]),cv2.COLOR_BGR2GRAY)

Hessian_Mat = hessian_matrix(Example_Image,sigma=5,order="rc")
max_S,min_S = hessian_matrix_eigvals(Hessian_Mat)

axis[0].imshow(min_S)
axis[0].set_xlabel(min_S.shape)
axis[0].set_ylabel(min_S.size)
axis[0].set_title(Main_Sign_Data["CATEGORY"][22213])
axis[1].imshow(max_S)
axis[1].set_xlabel(max_S.shape)
axis[1].set_ylabel(max_S.size)
axis[1].set_title(Main_Sign_Data["CATEGORY"][22213])
```

Out[41]: Text(0.5, 1.0, '4')



In [42]:
```
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(20,20))

Skel_Img = skeleton_morph_vision(Main_Sign_Data["JPG"][22213])
Simple_Img = simple_vision(Main_Sign_Data["JPG"][22213])

axis[0].imshow(Skel_Img)
axis[0].set_xlabel(Skel_Img.shape)
axis[0].set_ylabel(Skel_Img.size)
axis[0].set_title(Main_Sign_Data["CATEGORY"][22213])
axis[1].imshow(Simple_Img)
axis[1].set_xlabel(Simple_Img.shape)
axis[1].set_ylabel(Simple_Img.size)
axis[1].set_title(Main_Sign_Data["CATEGORY"][22213])
```

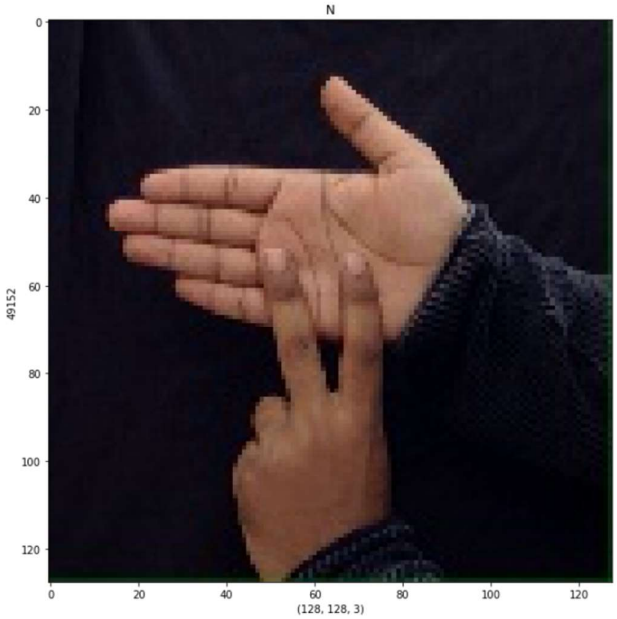
Out[42]: Text(0.5, 1.0, '4')



EXAMPLE IV

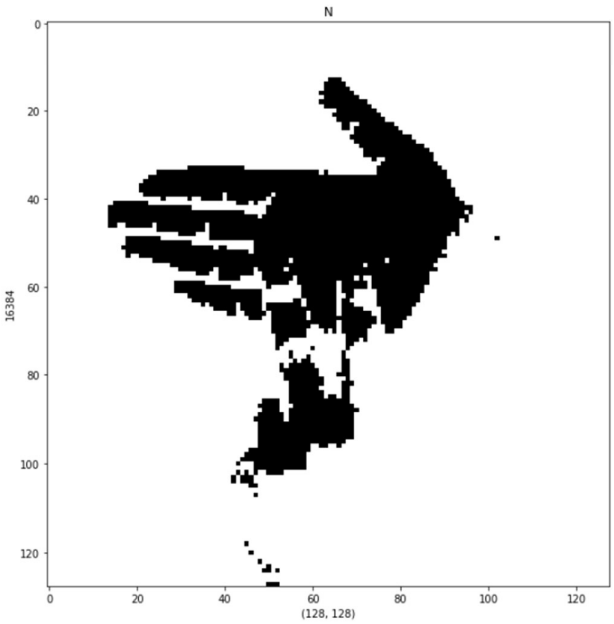
In [43]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = simple_vision(Main_Sign_Data["JPG"][10000])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][10000])
plt.imshow(Image_Sign)
```

Out[43]: <matplotlib.image.AxesImage at 0x1849a2242b0>



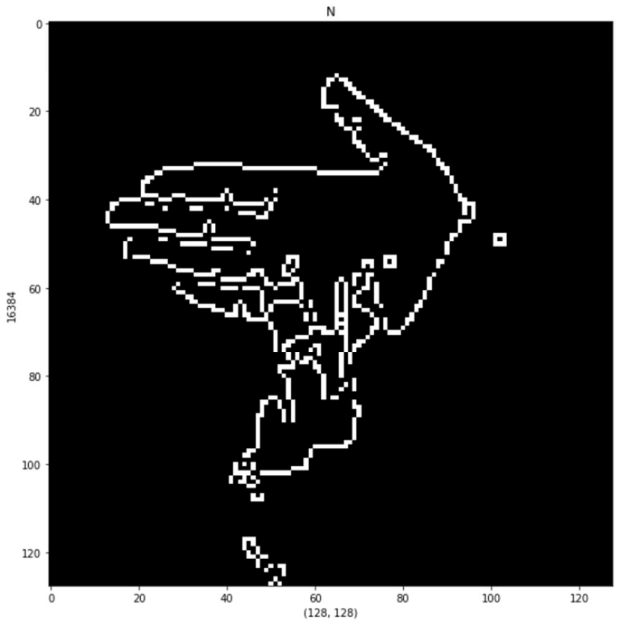
In [44]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = threshold_vision(Main_Sign_Data["JPG"][10000])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][10000])
plt.imshow(Image_Sign,cmap="gray")
```

Out[44]: <matplotlib.image.AxesImage at 0x1849a27bb80>



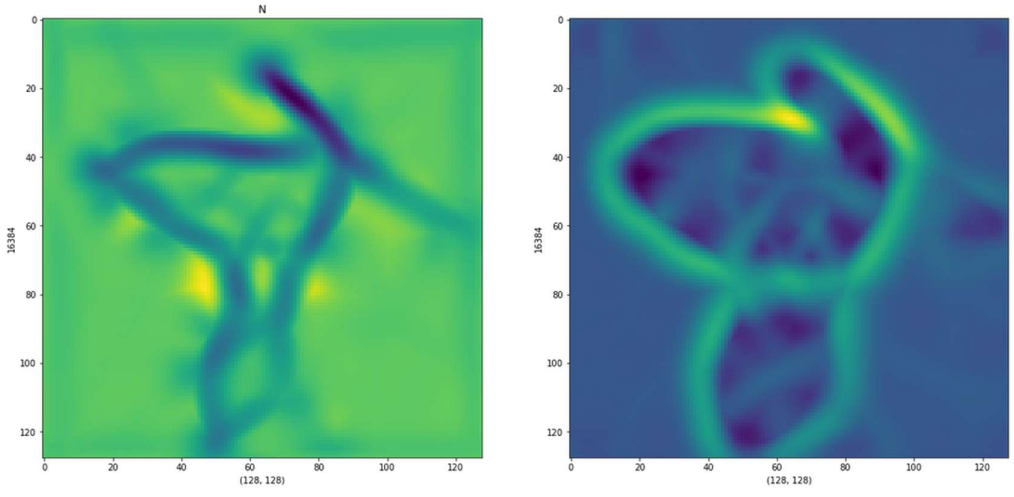
In [45]:
```
figure = plt.figure(figsize=(10,10))

Image_Sign = canny_vision(Main_Sign_Data["JPG"][10000])

plt.xlabel(Image_Sign.shape)
plt.ylabel(Image_Sign.size)
plt.title(Main_Sign_Data["CATEGORY"][10000])
plt.imshow(Image_Sign,cmap="gray")
```

Out[45]:
```
<matplotlib.image.AxesImage at 0x1849a2d8310>
```



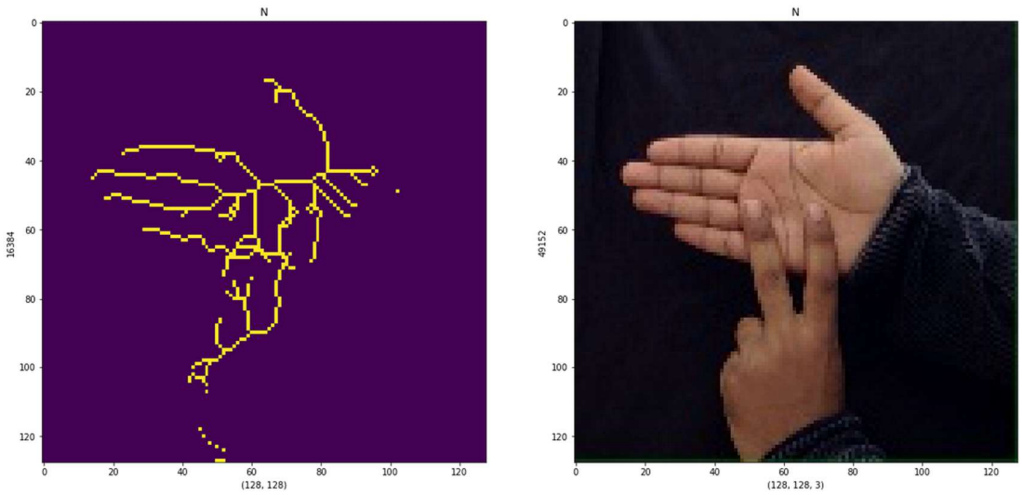
In [46]:
```
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(20,20))

Example_Image = cv2.cvtColor(cv2.imread(Main_Sign_Data["JPG"][10000]),cv2.COLOR_BGR2GRAY)

Hessian_Mat = hessian_matrix(Example_Image,sigma=5,order="rc")
max_S,min_S = hessian_matrix_eigvals(Hessian_Mat)

axis[0].imshow(min_S)
axis[0].set_xlabel(min_S.shape)
axis[0].set_ylabel(min_S.size)
axis[0].set_title(Main_Sign_Data["CATEGORY"][10000])
axis[1].imshow(max_S)
axis[1].set_xlabel(max_S.shape)
axis[1].set_ylabel(max_S.size)
```

Out[46]:
```
Text(0, 0.5, '16384')
```



In [47]:
```
figure,axis = plt.subplots(nrows=1,ncols=2,figsize=(20,20))

Skel_Img = skeleton_morph_vision(Main_Sign_Data["JPG"][10000])
Simple_Img = simple_vision(Main_Sign_Data["JPG"][10000])

axis[0].imshow(Skel_Img)
axis[0].set_xlabel(Skel_Img.shape)
axis[0].set_ylabel(Skel_Img.size)
axis[0].set_title(Main_Sign_Data["CATEGORY"][10000])
axis[1].imshow(Simple_Img)
axis[1].set_xlabel(Simple_Img.shape)
axis[1].set_ylabel(Simple_Img.size)
axis[1].set_title(Main_Sign_Data["CATEGORY"][10000])
```

Out[47]:
```
Text(0.5, 1.0, 'N')
```



SPLITTING DATA

In [48]:
```
X_Train,X_Test = train_test_split(Main_Sign_Data,train_size=0.9,random_state=123,shuffle=True)
```

In [49]:
```
print(X_Train.shape)
print(X_Test.shape)
```

```
(38470, 2)
(4275, 2)
```

In [50]:
```
print(type(X_Train))
print(type(X_Test))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

IMAGE DATA GENERATOR PROCESS

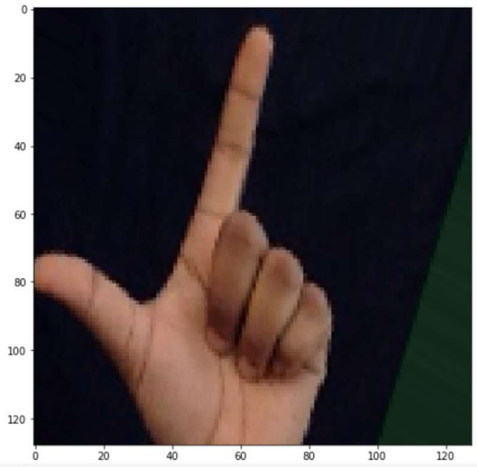
GENERATOR STRUCTURE

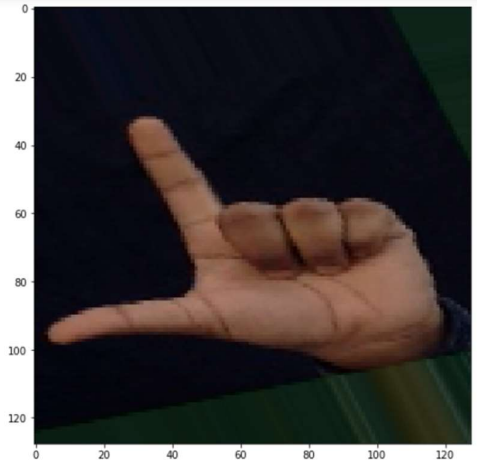
In [51]:
```
Train_IMG_Generator = ImageDataGenerator(rescale=1./255,
                                         zoom_range=0.5,
                                         shear_range=0.5,
                                         brightness_range=[0.6,1.0],
                                         rotation_range=35,
                                         width_shift_range=0.1,
                                         height_shift_range=0.1,
                                         vertical_flip=True,
                                          featurewise_std_normalization=False,
                                          samplewise_center=False,
                                          samplewise_std_normalization=False,
                                         fill_mode="nearest",
                                         validation_split=0.1)
```

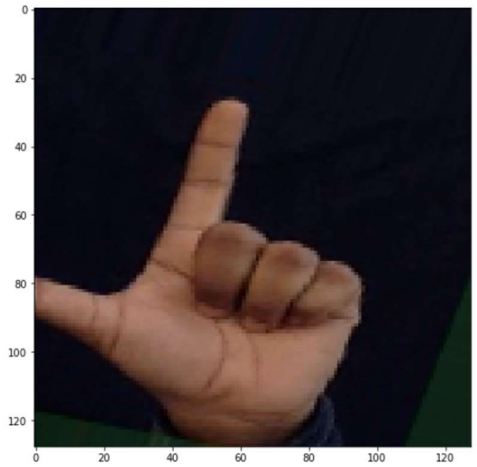
In [52]:
```
Test_IMG_Generator = ImageDataGenerator(rescale=1./255)
```

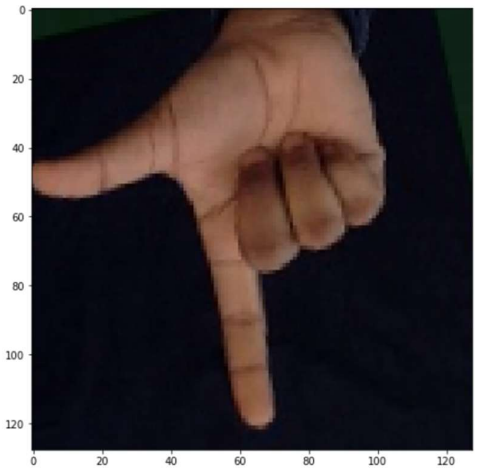
HOW TO LOOK BY GENERATOR

In [53]:
```
Example_Img = simple_vision(X_Train.JPG[3])
Example_Img = Example_Img.reshape((1,) + Example_Img.shape)

i = 0

for batch in Train_IMG_Generator.flow(Example_Img,batch_size=32):

    figure = plt.figure(figsize=(8,8))
    plt.imshow(image.img_to_array(batch[0]))

    i += 1
    if i % 4 == 0:
        break

plt.show()
```









APPLYING

In [54]:
```
Train_Set = Train_IMG_Generator.flow_from_dataframe(dataframe=X_Train,
                                                    x_col="JPG",
                                                    y_col="CATEGORY",
                                                    batch_size=32,
                                                    class_mode="categorical",
                                                    color_mode="grayscale",
                                                    subset="training")
```

```
Found 34623 validated image filenames belonging to 35 classes.
```

In [55]:
```
Validation_Set = Train_IMG_Generator.flow_from_dataframe(dataframe=X_Train,
                                                    x_col="JPG",
                                                    y_col="CATEGORY",
                                                    batch_size=32,
                                                    class_mode="categorical",
                                                    color_mode="grayscale",
                                                    subset="validation")
```

```
Found 3847 validated image filenames belonging to 35 classes.
```

In [56]:
```
Test_Set = Test_IMG_Generator.flow_from_dataframe(dataframe=X_Test,
                                                    x_col="JPG",
                                                    y_col="CATEGORY",
                                                    batch_size=32,
                                                    class_mode="categorical",
                                                    color_mode="grayscale",
                                                    shuffle=False)
```

```
Found 4275 validated image filenames belonging to 35 classes.
```

In [57]:
```
print("TRAIN: ")
print(Train_Set.class_indices)
print(Train_Set.classes[0:5])
print(Train_Set.image_shape)
print("---"*20)
print("VALIDATION: ")
print(Validation_Set.class_indices)
print(Validation_Set.classes[0:5])
print(Validation_Set.image_shape)
print("---"*20)
print("TEST: ")
print(Test_Set.class_indices)
print(Test_Set.classes[0:5])
print(Test_Set.image_shape)
```

```
TRAIN:
{'1': 0, '2': 1, '3': 2, '4': 3, '5': 4, '6': 5, '7': 6, '8': 7, '9': 8, 'A': 9, 'B': 10, 'C': 11, 'D': 12, 'E': 13, 'F': 14, '
G': 15, 'H': 16, 'I': 17, 'J': 18, 'K': 19, 'L': 20, 'M': 21, 'N': 22, 'O': 23, 'P': 24, 'Q': 25, 'R': 26, 'S': 27, 'T': 28, 'U
': 29, 'V': 30, 'W': 31, 'X': 32, 'Y': 33, 'Z': 34}
[12, 26, 10, 6, 13]
(256, 256, 1)
------------------------------------------------------------
```

In [57]:
```
print("TRAIN: ")
print(Train_Set.class_indices)
print(Train_Set.classes[0:5])
print(Train_Set.image_shape)
print("---"*20)
print("VALIDATION: ")
print(Validation_Set.class_indices)
print(Validation_Set.classes[0:5])
print(Validation_Set.image_shape)
print("---"*20)
print("TEST: ")
print(Test_Set.class_indices)
print(Test_Set.classes[0:5])
print(Test_Set.image_shape)
```

```
TRAIN:
{'1': 0, '2': 1, '3': 2, '4': 3, '5': 4, '6': 5, '7': 6, '8': 7, '9': 8, 'A': 9, 'B': 10, 'C': 11, 'D': 12, 'E': 13, 'F': 14, '
G': 15, 'H': 16, 'I': 17, 'J': 18, 'K': 19, 'L': 20, 'M': 21, 'N': 22, 'O': 23, 'P': 24, 'Q': 25, 'R': 26, 'S': 27, 'T': 28, 'U
': 29, 'V': 30, 'W': 31, 'X': 32, 'Y': 33, 'Z': 34}
[12, 26, 10, 6, 13]
(256, 256, 1)
------------------------------------------------------------
VALIDATION:
{'1': 0, '2': 1, '3': 2, '4': 3, '5': 4, '6': 5, '7': 6, '8': 7, '9': 8, 'A': 9, 'B': 10, 'C': 11, 'D': 12, 'E': 13, 'F': 14, '
G': 15, 'H': 16, 'I': 17, 'J': 18, 'K': 19, 'L': 20, 'M': 21, 'N': 22, 'O': 23, 'P': 24, 'Q': 25, 'R': 26, 'S': 27, 'T': 28, 'U
': 29, 'V': 30, 'W': 31, 'X': 32, 'Y': 33, 'Z': 34}
[23, 34, 30, 33, 17]
(256, 256, 1)
------------------------------------------------------------
TEST:
{'1': 0, '2': 1, '3': 2, '4': 3, '5': 4, '6': 5, '7': 6, '8': 7, '9': 8, 'A': 9, 'B': 10, 'C': 11, 'D': 12, 'E': 13, 'F': 14, '
G': 15, 'H': 16, 'I': 17, 'J': 18, 'K': 19, 'L': 20, 'M': 21, 'N': 22, 'O': 23, 'P': 24, 'Q': 25, 'R': 26, 'S': 27, 'T': 28, 'U
': 29, 'V': 30, 'W': 31, 'X': 32, 'Y': 33, 'Z': 34}
[25, 16, 17, 19, 15]
(256, 256, 1)
```

MODEL

PARAMETERS

In [58]:
```
COMPILE_OPTIMIZER = "adam"
COMPILE_LOSS = "categorical_crossentropy"
COMPILE_METRICS = ["accuracy"]
INPUT_DIM = (256,256,1)
OUTPUT_DIM = 35
```

In [59]:
```
Early_Stopper = tf.keras.callbacks.EarlyStopping(monitor="loss",patience=3,mode="min")
Checkpoint_Model = tf.keras.callbacks.ModelCheckpoint(monitor="val_accuracy",
                                                      save_best_only=True,
                                                      save_weights_only=True,
                                                      filepath="./modelcheck")
```

In [60]:
```
Model = Sequential()

Model.add(Conv2D(24,(3,3),activation="relu",input_shape=INPUT_DIM))
Model.add(BatchNormalization())
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Conv2D(64,(3,3),activation="relu",padding="same"))
Model.add(Dropout(0.3))
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Conv2D(64,(3,3),activation="relu",padding="same"))
Model.add(Dropout(0.3))
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Conv2D(128,(3,3),activation="relu",padding="same"))
Model.add(Conv2D(128,(3,3),activation="relu",padding="same"))
Model.add(Dropout(0.3))
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Conv2D(256,(3,3),activation="relu",padding="same"))
Model.add(Dropout(0.3))
Model.add(MaxPooling2D((2,2),strides=2))

Model.add(Flatten())
Model.add(Dense(2352,activation="relu"))
Model.add(Dropout(0.5))
Model.add(Dense(OUTPUT_DIM,activation="softmax"))
```

In [61]:
```
Model.compile(optimizer=COMPILE_OPTIMIZER,loss=COMPILE_LOSS,metrics=COMPILE_METRICS)
```

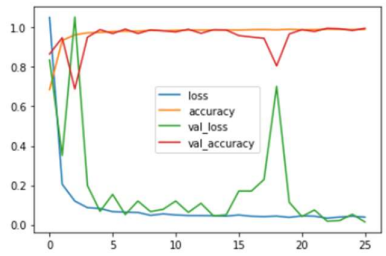
In [49]:
```
CNN_Model = Model.fit(Train_Set,
                      validation_data=Validation_Set,
                      callbacks=[Early_Stopper,Checkpoint_Model],
                      epochs=50)
```

```
Epoch 1/50
1082/1082 [==============================] - 373s 339ms/step - loss: 1.9940 - accuracy: 0.4481 - val_loss: 0.8351 - val_accurac
y: 0.8656
Epoch 2/50
1082/1082 [==============================] - 285s 263ms/step - loss: 0.2570 - accuracy: 0.9177 - val_loss: 0.3511 - val_accurac
y: 0.9483
Epoch 3/50
1082/1082 [==============================] - 287s 266ms/step - loss: 0.1310 - accuracy: 0.9588 - val_loss: 1.0531 - val_accurac
y: 0.6883
Epoch 4/50
1082/1082 [==============================] - 286s 264ms/step - loss: 0.0911 - accuracy: 0.9708 - val_loss: 0.1989 - val_accurac
y: 0.9506
Epoch 5/50
1082/1082 [==============================] - 284s 262ms/step - loss: 0.0960 - accuracy: 0.9717 - val_loss: 0.0677 - val_accurac
y: 0.9896
Epoch 6/50
1082/1082 [==============================] - 282s 261ms/step - loss: 0.0685 - accuracy: 0.9785 - val_loss: 0.1537 - val_accurac
y: 0.9693
Epoch 7/50
1082/1082 [==============================] - 281s 259ms/step - loss: 0.0654 - accuracy: 0.9813 - val_loss: 0.0514 - val_accurac
y: 0.9919
Epoch 8/50
1082/1082 [==============================] - 281s 259ms/step - loss: 0.0624 - accuracy: 0.9811 - val_loss: 0.1200 - val_accurac
y: 0.9696
Epoch 9/50
1082/1082 [==============================] - 280s 259ms/step - loss: 0.0502 - accuracy: 0.9852 - val_loss: 0.0664 - val_accurac
y: 0.9880
Epoch 10/50
1082/1082 [==============================] - 282s 260ms/step - loss: 0.0539 - accuracy: 0.9840 - val_loss: 0.0780 - val_accurac
y: 0.9831
Epoch 11/50
1082/1082 [==============================] - 280s 259ms/step - loss: 0.0464 - accuracy: 0.9855 - val_loss: 0.1201 - val_accurac
y: 0.9776
Epoch 12/50
1082/1082 [==============================] - 283s 262ms/step - loss: 0.0535 - accuracy: 0.9850 - val_loss: 0.0623 - val_accurac
y: 0.9912
Epoch 13/50
1082/1082 [==============================] - 281s 260ms/step - loss: 0.0478 - accuracy: 0.9869 - val_loss: 0.1085 - val_accurac
y: 0.9706
Epoch 14/50
1082/1082 [==============================] - 282s 260ms/step - loss: 0.0495 - accuracy: 0.9869 - val_loss: 0.0446 - val_accurac
y: 0.9888
Epoch 15/50
1082/1082 [==============================] - 282s 260ms/step - loss: 0.0453 - accuracy: 0.9882 - val_loss: 0.0501 - val_accurac
y: 0.9873
Epoch 16/50
1082/1082 [==============================] - 282s 260ms/step - loss: 0.0467 - accuracy: 0.9874 - val_loss: 0.1706 - val_accurac
y: 0.9592
Epoch 17/50
1082/1082 [==============================] - 284s 263ms/step - loss: 0.0367 - accuracy: 0.9904 - val_loss: 0.1705 - val_accurac
y: 0.9519
Epoch 18/50
1082/1082 [==============================] - 282s 261ms/step - loss: 0.0453 - accuracy: 0.9879 - val_loss: 0.2294 - val_accurac
y: 0.9454
Epoch 19/50
1082/1082 [==============================] - 283s 261ms/step - loss: 0.0378 - accuracy: 0.9898 - val_loss: 0.7019 - val_accurac
y: 0.8053
Epoch 20/50
1082/1082 [==============================] - 280s 259ms/step - loss: 0.0387 - accuracy: 0.9901 - val_loss: 0.1134 - val_accurac
y: 0.9675
Epoch 21/50
1082/1082 [==============================] - 282s 260ms/step - loss: 0.0407 - accuracy: 0.9901 - val_loss: 0.0414 - val_accurac
y: 0.9888
Epoch 22/50
1082/1082 [==============================] - 285s 263ms/step - loss: 0.0360 - accuracy: 0.9909 - val_loss: 0.0746 - val_accurac
y: 0.9797
Epoch 23/50
1082/1082 [==============================] - 298s 275ms/step - loss: 0.0356 - accuracy: 0.9907 - val_loss: 0.0180 - val_accurac
y: 0.9961
Epoch 24/50
1082/1082 [==============================] - 344s 318ms/step - loss: 0.0444 - accuracy: 0.9899 - val_loss: 0.0213 - val_accurac
y: 0.9938
Epoch 25/50
1082/1082 [==============================] - 318s 294ms/step - loss: 0.0420 - accuracy: 0.9900 - val_loss: 0.0528 - val_accurac
y: 0.9849
Epoch 26/50
1082/1082 [==============================] - 320s 296ms/step - loss: 0.0352 - accuracy: 0.9909 - val_loss: 0.0141 - val_accurac
y: 0.9961
```

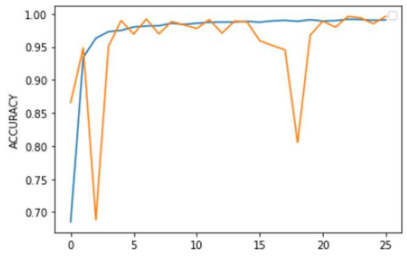
In [50]:
```
Model.save("Prediction_Model.h5")
```

In [51]:
```
Grap_Data = pd.DataFrame(CNN_Model.history)
Grap_Data.plot()
```

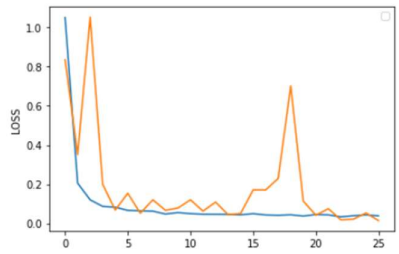
Out[51]: <AxesSubplot:>



In [52]:
```
plt.plot(CNN_Model.history["accuracy"])
plt.plot(CNN_Model.history["val_accuracy"])
plt.ylabel("ACCURACY")
plt.legend()
plt.show()
```



In [53]:
```
plt.plot(CNN_Model.history["loss"])
plt.plot(CNN_Model.history["val_loss"])
plt.ylabel("LOSS")
plt.legend()
plt.show()
```



In [54]:
```
Model_Results = Model.evaluate(Test_Set)
print("LOSS:  " + "%.4f" % Model_Results[0])
print("ACCURACY:  " + "%.2f" % Model_Results[1])
```

```
134/134 [==============================] - 34s 257ms/step - loss: 0.0040 - accuracy: 0.9995
LOSS:  0.0040
ACCURACY:  1.00
```

In [55]:
```
Model_Test_Prediction = Model.predict(Test_Set)
Model_Test_Prediction = Model_Test_Prediction.argmax(axis=-1)
```

In [56]:
```
fig, axes = plt.subplots(nrows=5,
                         ncols=5,
                         figsize=(20, 20),
                        subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(X_Test["JPG"].iloc[i]))
    ax.set_title(f"PREDICTION:{Model_Test_Prediction[i]}")
    ax.set_xlabel(X_Test["CATEGORY"].iloc[i])
plt.tight_layout()
plt.show()
```