**PROJECT REPORT**

**ON**

**INDIAN SIGN LANGUAGE RECOGNITION SYSTEM**

Under the Supervision of

**Dr. Awdhesh Kumar Mishra**



DST-CENTRE FOR INTERDISCIPLINARY MATHEMATICAL SCIENCES (DST-CIMS), INSTITUTE OF SCIENCE,

**BANARAS HINDU UNIVERSITY**

VARANASI - 221005

Submitted by

**KISHAN**

Exam roll no – 21419CAS012.

## ACKNOWLEDGEMENT

Date: 31st MAY,2023

I would like to sincerely thank my supervisor, **Dr. AWDHESH KUMAR MISHRA**, for sharing appropriate strategies and extending unrelenting support to facilitate the completion of my project. His vast knowledge, profound experience, and indefatigable spirit enabled me to complete this project successfully. The efforts of myself in completing this project are highly appreciated.

**KISHAN**

**M.Sc. COMPUTATIONAL SCIENCE & APPLICATIONS**

**Roll No.: 21419CAS012**

**Enrolment No.: 448283**

**Semester :IV**

**CERTIFICATE**

This is to certify that the contents of this dissertation titled “**INDIAN SIGN LANGUAGE RECOGNITION SYSTEM USING PYTHON”,** by KISHAN, submitted to DST-Centre for Interdisciplinary Mathematical Sciences, BHU, is the bona fide work of him under my supervision.

This dissertation forms a part of the M.Sc. Computational Science & Applications course and is compulsory for the completion of the course successfully.

This dissertation work has been found to be satisfactory and hence it is recommended to consider this work for evaluation.

**Signature of the guide**

**(Dr. Awdhesh Kr. Mishra)**

# DECLARATION

I hereby declare that the work which is being presented in the M.Sc. project report entitled **“INDIAN SIGN LANGUAGE RECOGNITION SYSTEM USING PYTHON”,** submitted to DST-CIMS, Banaras Hindu University in partial fulfilment of the M.Sc. Computational Science & Applications course requirement is a record of original work done by me under the guidance of **Dr. Awdhesh Kumar Mishra**, during a period from **February 2023 to May 2023**.

The information and data submitted in this project are authentic to the best of my knowledge and belief. The project has not been submitted to any other university or institution for the award of any degree, diploma or fellowship and has not been published any time before.

**Date:** 31-05-2023 Signature of the student

**Place:** VARANASI (**KISHAN)**

**Contents**

1. **ABSTRACT**
2. **INTRODUCTION**
3. **OBJECTIVE**
4. **METHODOLOGY** 
   1. **Image Classification Techniques**
   2. **Gathering Dataset**
   3. **DATA PREPROCESSING**
   4. **Important Libraries**
   5. **LOADING DATASET**
   6. **Checking the dataset**
   7. **Splitting the dataset**
   8. **MODEL DEFINITION**
   9. **TRAINING THE MODEL**
   10. **Plotting the training results**
   11. **Saving the model**
   12. **Testing the model**

1. **CONCLUSIONS**
2. **PYTHON CODE**

**7 REFERENCES**

## 1.Abstract

Sign languages have become a significant medium of communication for the deaf and dumb people. The project is a way to implement this medium of communication on a computer using image classification techniques based on different categories of images. W have different signs which stand for different words and meanings and to make computer recognize & interpret those signs correctly, deep learning is a very obvious pathway.

Images of different signs have been collected based on the category(label) name and the machine has been trained over the dataset to get insights in to the patterns of data in an order to learn to classify and tell which sign it is.

The world is trying to be more inclusive and thus different strategies are developed to make it more suitable for different people irrespective of their limitations. Hence, the governments are trying to make all the facilities Divyaang friendly. So this dissertation is an attempt to make the computers and interactive platforms friendly for the deaf & dumb.

|  |
| --- |
| 7    **2. INTRODUCTION**  Image classification has become very popular over the last decade and it can be said that it is one of the most evolving sub field of image processing. Different softwares for object detection, image identification and classification have emerged over the last decade. The most obvious plus point of modern day computers has been that it can communicate with the users. So the most pertinent question here is “Why not deaf & dumb also communicate with the computer?”  The challenge before us, before the concept of machine learning, was how to make computer understand the sign language. But with the advent of Machine Learning & Deep Learning , it has become possible to make computer learn anything. In this project, we have made a program which can recognize the Indian Sign Language very comfortably. The thing is very simple and that is –first, collect the dataset, make the dataset suitable to use for the deep learning, the train the model over the dataset and note its performance using various metrics. Definitely, the basic desire is to create a program which can make Indian Sign Language to be recognized by the system. |

### 3. OBJECTIVE

To create a program in Python which can recognize the Indian Sign language indicating the digits from 0 to 9. Besides this, the project’s intermediate goals also include the calculation of accuracy, error in classification, validation loss & validation accuracy. Then the model will be ready to be applied on unseen test data.

# 4. METHODOLOGY

**4.1** **Image Classification Techniques**

Image data is represented as a two-dimensional grid of pixels, be it monochromatic or in color. Accordingly each pixel corresponds to one or multiple numerical values respectively. So far we ignored this rich structure and treated them as vectors of numbers by flattening the images, irrespective of the spatial relation between pixels. This deeply unsatisfying approach was necessary in order to feed the resulting one-dimensional vectors through a fully connected MLP.

A deep learning technique called Convolutional Neural Networks (CNNs) is used for image classification & object detection in images. We can now make these intuitions more concrete by enumerating a few desiderata to guide our design of a neural network architecture suitable for computer vision:--

1. In the earliest layers, our network should respond similarly to the same patch, regardless of where it appears in the image. This principle is called translation invariance (or translation equivariance).

2. The earliest layers of the network should focus on local regions, without any concern for the contents of the image in distant regions. This is the locality principle.

3. As we proceed, deeper layers should be able to capture longer-range features of the image, in a way similar to higher level vision in nature.



Convolution: In the two-dimensional cross-correlation operation, we begin with the convolution window positioned at the upper-left corner of the input tensor and slide it across the input tensor, both from left to right and top to bottom. When the convolution window slides to a certain position, the input subtensor contained in that window and the kernel tensor are multiplied elementwise and the resulting tensor is summed up yielding a single scalar value. This result gives the value of the output tensor at the corresponding location.

Feature Map & Receptive Fields: the convolutional layer output is sometimes called a feature map, as it can be regarded as the learned representations (features) in the spatial dimensions (e.g., width and height) to the subsequent layer. In CNNs, for any element x of some layer, its receptive field refers to all the elements (from all the previous layers) that may affect the calculation of x during the forward propagation.

Padding: Adding extra pixels of filler around the boundary of our input image, thus increasing the effective size of the image. Typically, the values of the extra pixels are set to zero.

Striding: The number of rows and columns traversed per slide are referred to as stride. Sometimes, either for computational efficiency or because we wish to down sample, we move our window more than one element at a time, skipping the intermediate locations. This is helpful if the convolution kernel is large since it captures a large area of the underlying image.

Pooling: Like convolutional layers, pooling operators consist of a fixed-shape window that is slid over all regions in the input according to its stride, computing a single output for each location traversed by the fixed-shape window (sometimes known as the pooling window). However, unlike the cross-correlation computation of the inputs and kernels in the convolutional layer, the pooling layer contains no parameters.

**4.2 Gathering Dataset**

The dataset has been downloaded from the Kaggle. The dataset has already very cleaned and suitable images particularly for this project.

**4.3. DATA PREPROCESSING:**

Initially, all the images in the dataset were of dimension 256 \* 256. For this project, the dimensionality of images were brought down to 64\*64. The colour channels were left unchanged. The pixel values(intensity at a particular location in the image) are in the range 0 to 255 and thus to simplify the calculations, the values have been normalized through dividing the pixel values by 255. The pixel values are now between 0 to 1.

**4.4. IMPORTANT LIBRARIES:**

4.4.1 tensorflow: It is a platform for developing machine learning programs which can be used by beginners as well as experts. It comes with a rich variety of datasets also for beginners.It comes with a lot of features like auto differentiation, Eager Execution, distribute, losses, metrics etc.

4.4.2 matplotlib: It is a kind of visualization tool which helps to present the results and data in a beautiful and comprehensible way so that a lay man can also understand it.

4.4.3 opencv: It is meant for computer vision tasks.

4.4.4 OS: This module helps to build a bridge between user and the operating system so that files can be accessed & saved in to the disks etc.

4.4.5 keras: Keras is a deep learning API written in Python, running on top of tensorflow.

**4.5.** **LOADING DATASET :**

This was achieved using tensorflow keras module called utils. This is a very special module which contains a lot of things to facilitate smooth performance of tasks ranging from converting one type of data to another to automatically generating dataset directly from the directory stored on the disk. The most important component of this module which has been used in my project is “tf.keras.utils.image\_dataset\_from\_directory”. The basic function of this is to generate a tf.data.dataset from image files in a directory.

The most important property of this module is that one can specify batch size, image dimensions & validation split through this module.

Then the loaded dataset from the directory is converted to a numpy array using .as\_numpy\_iterator().

**4.6. Checking the dataset:--**

The dataset has also been checked by displaying the contents in the program. It was done to ensure that the labelling of images has been done correctly. This was achieved using matplotlib library and considering the images in batches. Labels have been displayed in the form of titles of the images.

**4.7. Splitting the dataset:--**

For any further progress, the dataset needs to be divided in to 3 parts—

1. Training Set
2. Validation Set
3. Test Set

70% of the total dataset was used for training , 20% was used for validation and the remaining 10% was for testing the model.

**4.8. MODEL DEFINITION:--**

The convolutional Neural Networks (CNNs) is the basis of this project which is the most dominant deep learning model for images. A sequential model has been created by calling tf.keras.sequential while the optimizer used was SGD(Stochastic Gradient Descent). A total of 21 layers have been used, out of which 14 layers are trainable while the remaining 7 are non-trainable.

Six 2-D convolutional layers

Six batch normalization layers

Three Max pooling layers

Three dropout layers

One flatten layer

Two dense layers in the end

**4.9. TRAINING THE MODEL:--**

The model was fitted to the training data using 5 epochs with the following track—

Epoch 1/5

113/113 [==============================] - 566s 5s/step - loss: 0.4111 - accuracy: 0.9334 - val\_loss: 9.1740 - val\_accuracy: 0.3232

Epoch 2/5

113/113 [==============================] - 439s 4s/step - loss: 0.0459 - accuracy: 1.0000 - val\_loss: 0.0463 - val\_accuracy: 1.0000

Epoch 3/5

113/113 [==============================] - 453s 4s/step - loss: 0.0458 - accuracy: 1.0000 - val\_loss: 0.0459 - val\_accuracy: 1.0000

Epoch 4/5

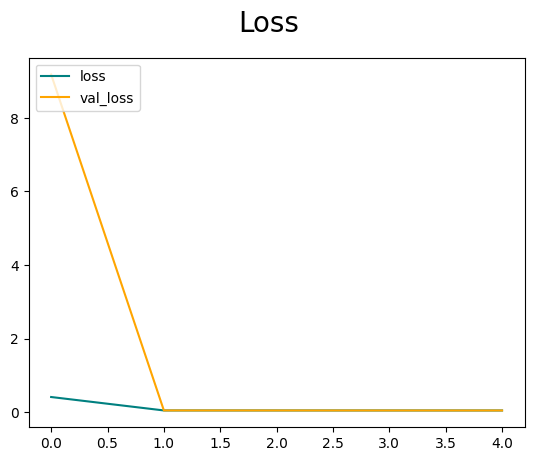
113/113 [==============================] - 452s 4s/step - loss: 0.0455 - accuracy: 1.0000 - val\_loss: 0.0454 - val\_accuracy: 1.0000

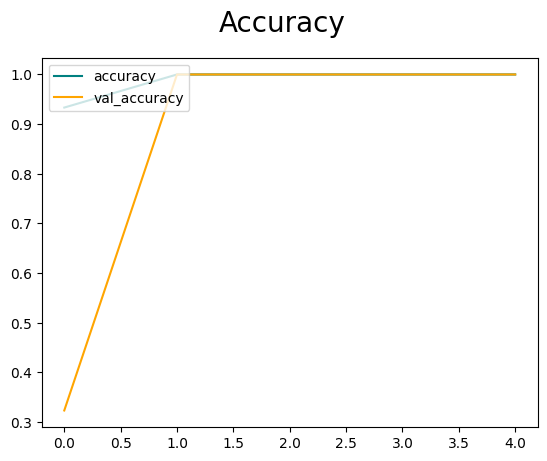
Epoch 5/5

113/113 [==============================] - 454s 4s/step - loss: 0.0453 - accuracy: 1.0000 - val\_loss: 0.0452 - val\_accuracy: 1.0000

**4.10. Plotting the training results:--**

The performance of the model on training data was evaluated using sparse categorical entropy as the loss function with Stochastic Gradient Descent as an optimizer. The visualization of the performance was done using matplotlib . The following are the results of the training—





**4.11 Saving the model:--**

Then the model was saved using the model.save module from the tensorflow.

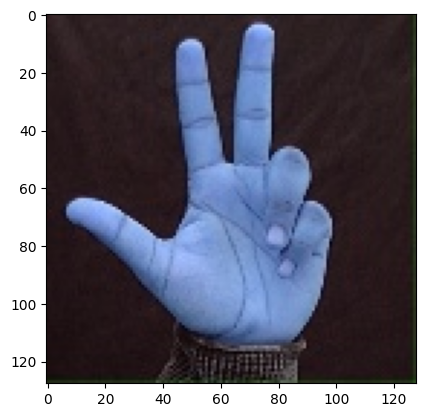
**4.12 Testing the model:--**

Then the data which had been kept isolated for testing was used for testing the model . Several of the images from the training set were chosen randomly and the model was applied on them to predict the class label. The model output the correct labels for most of the images. Before applying the model , the images were brought down to the dimension 64\*64 using the tf.image.resize module. The intensity values of the images were also normalized using np.expand\_dims to make them lie between 0 to 1. The output is the softmax probability values where the maximum value is attached to the most probable class. In this project, the most appropriate class got the highest output value .

img = cv2.imread(r'/content/1174.jpg')

plt.imshow(img)

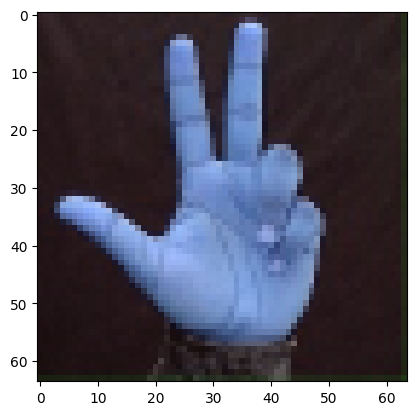
plt.show()



resize = tf.image.resize(img, (64,64))

plt.imshow(resize.numpy().astype(int))

plt.show()



yhat = model.predict(np.expand\_dims(resize/255, 0))

np.argmax(yhat)

8

np.argmin(yhat)

3

**5.CONCLUSION:**

In this project, I have considered nine (9) classes for Indian Sign Language. The classes each of the digits from 0 to 9. Each class had 515 images in the data set, so the dataset set contained a total of 515\*10=5150 images.

Noting the values of loss function, accuracy metric per epoch , I can conclude that the model is working well over the dataset. The validation loss & validation accuracy per epoch is also very satisfactory. Overall, I can say that the model has learnt the intricacies of the dataset very clearly and it is able to generalize well to the unseen data.

**7.PYTHON CODE:**

import os

import tensorflow as tf

import matplotlib.pyplot as plt

import os

import cv2

import imghdr

import numpy as np

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.models import load\_model

from tensorflow.keras.models import Sequential

from tensorflow.keras.preprocessing.image import load\_img , img\_to\_array

from tensorflow.keras.layers import Dense , Dropout , Conv2D , MaxPooling2D, Flatten , BatchNormalization

train\_dir=r'/content/drive/MyDrive/500'

data\_train=tf.keras.utils.image\_dataset\_from\_directory(train\_dir,batch\_size=32,image\_size=(64, 64))

data\_iterator=data\_train.as\_numpy\_iterator()

batch=data\_iterator.next()

len(batch)

2

batch[0].shape

batch[0][0].shape

print(batch[0].min())

print(batch[0].max())

(32, 64, 64, 3)

(64, 64, 3)

0.0

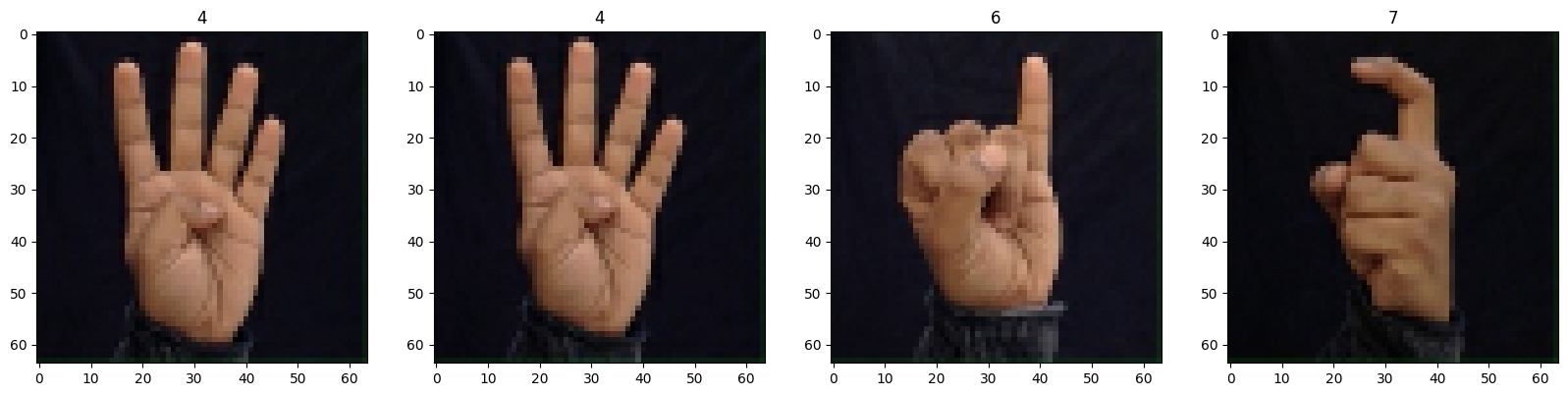
255.0

fig,ax=plt.subplots(ncols=4,figsize=(20,20))

for idx,img in enumerate(batch[0][:4]):

    ax[idx].imshow(img.astype(int))

    ax[idx].title.set\_text(batch[1][idx])



data = data\_train.map(lambda x,y: (x/255, y))

print(data.as\_numpy\_iterator().next()[0].min())

print(data.as\_numpy\_iterator().next()[0].max())

scaled=data.as\_numpy\_iterator().next()

scaled[0].shape

scaled[1].shape

0.0

1.0

(32, 64, 64, 3)

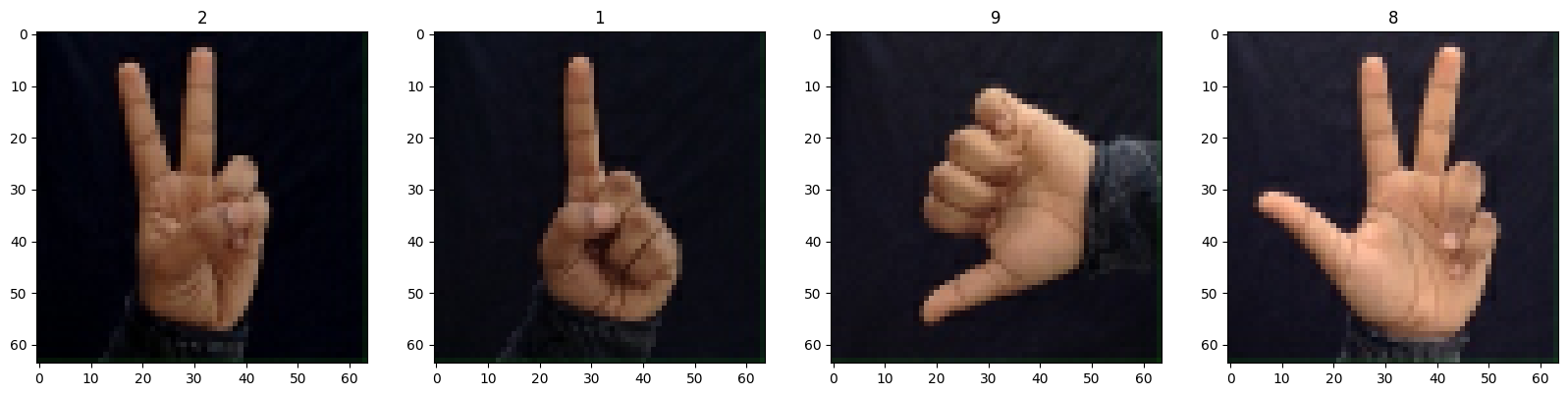
(32,)

fig,ax=plt.subplots(ncols=4,figsize=(20,20))

for idx,img in enumerate(scaled[0][:4]):

    ax[idx].imshow(img)

    ax[idx].title.set\_text(scaled[1][idx])



train\_size=int(len(data)\*.7)

val\_size=int(len(data)\*.2)

test\_size=int(len(data)\*.1)

len(data)

162

train = data.take(train\_size)

val = data.skip(train\_size).take(val\_size)

test = data.skip(train\_size+val\_size).take(test\_size)

train

<\_TakeDataset element\_spec=(TensorSpec(shape=(None, 64, 64, 3), dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32, name=None))>

from tensorflow.keras.models import Sequential

from keras.optimizers import SGD

weight\_decay = 1e-4

model = Sequential([

                    Conv2D(64, (5, 5), activation='relu', padding='same',kernel\_regularizer=tf.keras.regularizers.l2(weight\_decay), input\_shape=(64, 64, 3)),

                    BatchNormalization(),

                    Conv2D(64, (5, 5), activation='relu',kernel\_regularizer=tf.keras.regularizers.l2(weight\_decay), padding='same'),

                    BatchNormalization(),

                    MaxPooling2D((2, 2)),

                    Dropout(0.2),

                    Conv2D(32, (3, 3), activation='relu',kernel\_regularizer=tf.keras.regularizers.l2(weight\_decay), padding='same'),

                    BatchNormalization(),

                    Conv2D(32, (3, 3), activation='relu',kernel\_regularizer=tf.keras.regularizers.l2(weight\_decay), padding='same'),

                    BatchNormalization(),

                    MaxPooling2D((2, 2)),

                    Dropout(0.3),

                    Conv2D(128, (3, 3), activation='relu',kernel\_regularizer=tf.keras.regularizers.l2(weight\_decay), padding='same'),

                    BatchNormalization(),

                    Conv2D(128, (3, 3), activation='relu',kernel\_regularizer=tf.keras.regularizers.l2(weight\_decay), padding='same'),

                    BatchNormalization(),

                    MaxPooling2D((2, 2)),

                    Dropout(0.3),

                    Flatten(),

                    Dense(128, activation='relu'),

                    Dense(10, activation='softmax')

])

opt =    tf.keras.optimizers.SGD(lr=0.001, momentum=0.9)

model.compile(optimizer=opt, loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

model.summary()

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 64, 64, 64) 4864

batch\_normalization (BatchN (None, 64, 64, 64) 256

ormalization)

conv2d\_1 (Conv2D) (None, 64, 64, 64) 102464

batch\_normalization\_1 (Batc (None, 64, 64, 64) 256

hNormalization)

max\_pooling2d (MaxPooling2D) (None, 32, 32, 64) 0

dropout (Dropout) (None, 32, 32, 64) 0

conv2d\_2 (Conv2D) (None, 32, 32, 32) 18464

batch\_normalization\_2 (Batc (None, 32, 32, 32) 128

hNormalization)

conv2d\_3 (Conv2D) (None, 32, 32, 32) 9248

batch\_normalization\_3 (Batc (None, 32, 32, 32) 128

hNormalization)

max\_pooling2d\_1 (MaxPooling (None, 16, 16, 32) 0

2D)

dropout\_1 (Dropout) (None, 16, 16, 32) 0

conv2d\_4 (Conv2D) (None, 16, 16, 128) 36992

batch\_normalization\_4 (Batc (None, 16, 16, 128) 512

hNormalization)

conv2d\_5 (Conv2D) (None, 16, 16, 128) 147584

batch\_normalization\_5 (Batc (None, 16, 16, 128) 512

hNormalization)

max\_pooling2d\_2 (MaxPooling (None, 8, 8, 128) 0

2D)

dropout\_2 (Dropout) (None, 8, 8, 128) 0

flatten (Flatten) (None, 8192) 0

dense (Dense) (None, 128) 1048704

dense\_1 (Dense) (None, 10) 1290

=================================================================

Total params: 1,371,402

Trainable params: 1,370,506

Non-trainable params: 896

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

logdir='ISLlogs'

tensorboard\_callback = tf.keras.callbacks.TensorBoard(log\_dir=logdir)

hist = model.fit(train,epochs=5, validation\_data=val, callbacks=[tensorboard\_callback])

Epoch 1/5

113/113 [==============================] - 566s 5s/step - loss: 0.4111 - accuracy: 0.9334 - val\_loss: 9.1740 - val\_accuracy: 0.3232

Epoch 2/5

113/113 [==============================] - 439s 4s/step - loss: 0.0459 - accuracy: 1.0000 - val\_loss: 0.0463 - val\_accuracy: 1.0000

Epoch 3/5

113/113 [==============================] - 453s 4s/step - loss: 0.0458 - accuracy: 1.0000 - val\_loss: 0.0459 - val\_accuracy: 1.0000

Epoch 4/5

113/113 [==============================] - 452s 4s/step - loss: 0.0455 - accuracy: 1.0000 - val\_loss: 0.0454 - val\_accuracy: 1.0000

Epoch 5/5

113/113 [==============================] - 454s 4s/step - loss: 0.0453 - accuracy: 1.0000 - val\_loss: 0.0452 - val\_accuracy: 1.0000

fig = plt.figure()

plt.plot(hist.history['loss'], color='teal', label='loss')

plt.plot(hist.history['val\_loss'], color='orange', label='val\_loss')

fig.suptitle('Loss', fontsize=20)

plt.legend(loc="upper left")

plt.show()

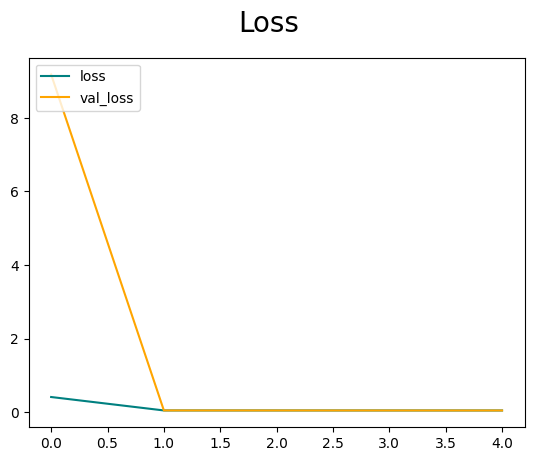


fig = plt.figure()

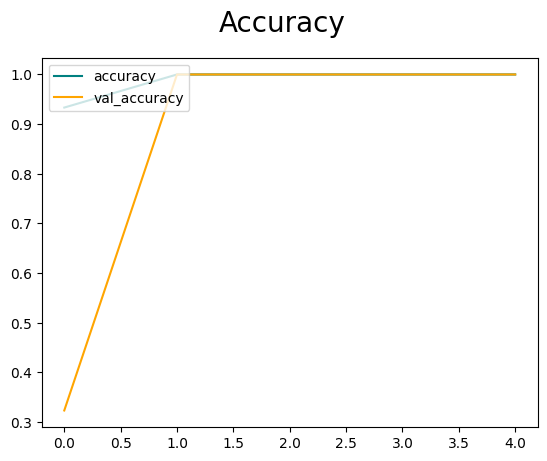
plt.plot(hist.history['accuracy'], color='teal', label='accuracy')

plt.plot(hist.history['val\_accuracy'], color='orange', label='val\_accuracy')

fig.suptitle('Accuracy', fontsize=20)

plt.legend(loc="upper left")

plt.show()



from tensorflow.keras.models import load\_model

model.save(os.path.join('models','5ISLsoft64max515.h5'))

**7.REFERENCES**

* E. Newman, M. Kilmer, L. Horesh, *Image classification using local tensor singular value decompositions* (IEEE, international workshop on computational advances in multi-sensor adaptive processing. IEEE, Willemstad, 2018).
* A.A.M. Al-Saffar, H. Tao, M.A. Talab, *Review of deep convolution neural network in image classification* (International conference on radar, antenna, microwave, electronics, and telecommunications. IEEE, Jakarta, 2018).
* L. Yang, A.M. Maceachren, P. Mitra, et al., Visually-enabled active deep learning for (geo) text and image classification: a review. ISPRS Int. J. Geo-Inf. **7**(2), 65 (2018).
* B. Kieffer, M. Babaie, S. Kalra, et al., *Convolutional neural networks for histopathology image classification: training vs. using pre-trained networks* (International conference on image processing theory. IEEE, Montreal, 2018).
* Newman E, Kilmer M, Horesh L. Image classification using local tensor singular value decompositions IEEE, international workshop on computational advances in multi-sensor adaptive processing. IEEE, 2018.
* S. Roychowdhury, J. Ren, *Non-deep CNN for multi-modal image classification and feature learning: an azure-based model* (IEEE international conference on big data. IEEE, Washington, D.C., 2017).
* Shima Y. Image augmentation for object image classification based on combination of pre-trained CNN and SVM. International Conference on Informatics, Electronics and Vision & 2017, International sSymposium in Computational Medical and Health Technology. 2018
* Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Computation 18(7), 1527–1554 (2006).
* Radha Kanabarkar, Mallikarjun Anandhalli, "Performance Analysis of Convolutional Neural Network for Image Classification", *2022 International Conference on Futuristic Technologies (INCOFT)*, pp.1-4, 2022.
* S. Senthilarasi, S. Kamalakkannan, "The Role of Convolutional Neural Network in Vehicle Detection on Spatial - Temporal Road Traffic Data", *2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS)*, pp.59-66, 2022.
* efri Zulkarnain, Mohammad Rezza Pahlevi, Yustikamasy Astica, Widi Pangestuti, Kusrini, "Fire Detection based on Smoke Image using Convolutional Neural Network (CNN)", *2022 4th International Conference on Cybernetics and Intelligent System (ICORIS)*, pp.1-5, 2022.
* M.A. Hafeez Khan, Prasad S. Giri, J. Angel Arul Jothi, "Detection of Cavities from Oral Images using Convolutional Neural Networks", *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pp.1-6, 2022.
* Jmour, Sahar Maamri, Sehla Loussaief, Afef Abdelkrim, "Traffic sign recognition for controlling intelligent vehicle", *2022 IEEE Information Technologies & Smart Industrial Systems (ITSIS)*, pp.1-6, 2022.