Kammavari Sangham (R) 1952, K.S.Group of Institutions

K. S INSTITUTE OF TECHNOLOGY

No.14, Raghuvanahalli, Kanakapura Road, Bengaluru - 560109

Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi, Accredited by NBA , NAAC & IEI

National Conference on Recent Innovations in Engineering-2022

Department of Computer Science and Engineering

Forest Fire Susceptibility using Neural Network

Track ID : Net-04

Amaravathi M: 1KS18CS002
Chandan Kumar: 1KS18CS016
Likhitha N: 1KS18CS039
Nandini J K: 1KS18CS056

Guided By:
Prof. Kumar.K
Assistant Professor
Dept of CSE, KSIT

Contents

- Introduction
- Literature Survey
- Problem Statement and Objectives
- Technologies / Tools Used
- Methodology Proposed
- Implementation of Modules
- Results and Snapshots
- Conclusion and Future Enhancement
- References

Introduction

The Project is based on image classification , model will detect fire in the image. Where user will provide the image and output will be “Fire”, ”No Fire” and “Smoke”. Architecture is based on CNN(Convolutional Neural Network), Model is trained on approximately 15000 images including of fire, smoke ,and no fire images.

Literature Survey

Sl . No	Base Paper Title	Author	Methodology	Outcomes
1.	"Forest fire image recognition based on convolutional neural network". Published in "Journal of Algorithm & Computational Technology" in the year 2019	Wang Yuanbin, Dang Langfei and Ren Jieying3	They divided Fire identification algorithms into two categories. The first is based on classic image processing, while the second is based on convolutional neural networks	Experiments reveal that the convolutional neural network approach based on adaptive pooling has a greater recognition rate and better performance
2.	"Image fire detection algorithms based on convolutional neural networks" . Published in "ELSEVIER Journal" in the year 2020	Pu Li and Wangda Zhao	Faster-RCNN, R–FCN, SSD, and YOLO v3	The method based on YOLO v3 has an average precision of 83.7 percent, which is greater than the other offered algorithms.

Problem Statement

Modelling and anticipating the incidence of wildfires are essential to minimize these damages and reducing forest fires because they can help with forest fire prevention strategies. As a result developing a model which can suspect the fire flame and smoke in the image was the main challenge.

In order to overcome this challenge we come up with the idea of CNN based model which is very popular approach in the field of image classification. Not only this maintaining the accuracy of the CNN model the one of challenging problem.

We over come all these challenges and build the CNN model, which successfully predicted Fire, Smoke and No Fire images with higher rate of accuracy. Also we managed to minimize the number of false alarm and alerting Fire Authority with SMS as well as siren in case of emergency or once it recognize the fire and smoke in the image.

Goals and Objectives

- Improved accuracy on Data Augmentation to get a better output.
- Make use of the CNN model which has important practical application value for forest fire prevention planning and forest management
- Deep analysis of different phases of images.
- To detect forest fires with reasonable accuracy.

Technologies and Tools Used

- Anaconda
- Jupyter Notebook
- Python 3.9

- Hard Disk : 50gb Minimum
- Operating System : Windows/Linux/Macos

Methodology Proposed

Four Stages of Methodology

- Data Augmenting
- Implementing CNN Architecture
- Training the Architecture
- Testing the Images

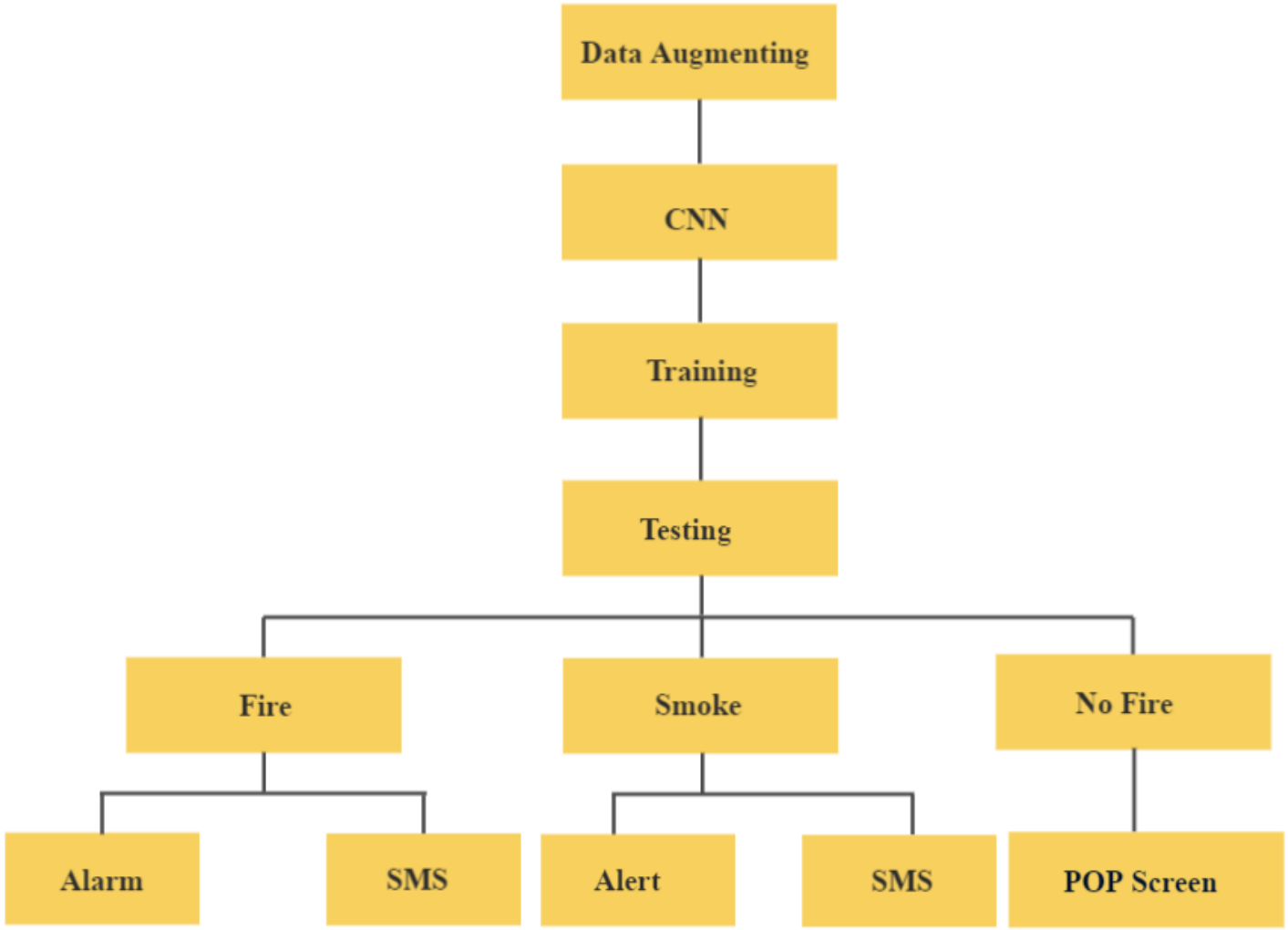


Fig. 1 Work flow of the model

Implementation with codes

Data Augmentation

```
#Rescaling
    training_datagenerator = ImageDataGenerator(rescale=1.0/255,horizontal_flip=True,
    vertical_flip=True, zoom_range=0.2, shear_range=0.2, featurewise_center=True,
    featurewise_std_normalization=True, rotation_range=40,width_shift_range=0.2,
    height_shift_range=0.2,validation_split=0.2)


#Data is being divided into training and validation.
        training =
        training_datagenerator.flow_from_directory(r"C:\Users\chand\OneDrive\Desktop\Proj
        ect_Phase\FireDataset\Training Data", target_size = (256,256), color_mode = ‘rgb’,
        class_mode = 'categorical', batch_size = 16, subset = 'training')
        #validation means to find and optimiz the best model to solve a given problem
        Validation =
        training_datagenerator.flow_from_directory(r"C:\Users\chand\OneDrive\Desktop\Proj
        ect_Phase\FireDataset\Training Data", target_size=(256, 256),color_mode='rgb',
        class_mode='categorical', batch_size=16,subset='validation')
```

Implementation with codes

CNN Architecture

```
#Cnn Architechture or Initializing cnn Architechture
cnn=tf.keras.models.Sequential()

#Adding 1st layer / Input Layer
cnn.add(tf.keras.layers.Conv2D(filters=16, padding='same', kernel_size=3, activation='relu', input_shape=[256,256,3]))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2))

#Adding 2nd layer
cnn.add(tf.keras.layers.Conv2D(filters=32, padding='same', kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2))


#Adding 3rd layer
cnn.add(tf.keras.layers.Conv2D(filters=64, padding='same', kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2))

#Adding 4th layer
cnn.add(tf.keras.layers.Conv2D(filters=128, padding='same', kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2))
```

Implementation with codes

CNN Architecture

```
#Adding 5th layer
cnn.add(tf.keras.layers.Conv2D(filters=256, padding='same', kernel_size=3, activation='relu'))
#Polling Layer
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2))

#falttened layer
cnn.add(tf.keras.layers.Flatten())

#hidden Layer
cnn.add(tf.keras.layers.Dense(units=512,activation='relu'))
cnn.add(tf.keras.layers.Dense(units=512,activation='relu'))
cnn.add(tf.keras.layers.Dense(units=512,activation='relu'))
cnn.add(tf.keras.layers.Dense(units=512,activation='relu'))
cnn.add(tf.keras.layers.Dense(units=512,activation='relu'))

#output Layer
cnn.add(tf.keras.layers.Dense(units=3, activation='softmax'))
```

Implementation with codes

Training the data

```
#after every model to check
        ckeckpoint=tf.keras.callbacks.ModelCheckpoint(r'C:\Users\chand\Desktop\Project_Phase\FireDatase
                t\Training models\model.h5', monitor='val_loss', verbose=0, mode="min",
                save_weights_only=False, save_best_only=True)
        callbacks = ckeckpoint
#Training the CNN Model i.e. Compile and Train
        cnn.compile(optimizer='Adam',loss='binary_crossentropy', metrics=['accuracy’])
        cnn.fit_generator(training, validation_data=validation, epochs=2,
                steps_per_epoch=training.samples//16,  validation_steps=validation.samples//16,
                callbacks = callbacks)
#"steps_per_epoch=  It is used to define how many batches of samples to use in one epoch.
#it is used to declaring one epoch finished and starting the next epoch"
```

Implementation with codes

Testing/Predicting:

```
###to load the trained model and test due to time constraint, and execute only when data is not trained
cnn=load_model(r'C:\Users\chand\Desktop\Project_Phase\FireDataset\Training models\model.h5')

#Loading the image using path
        image_for_testing= r'C:\Users\chand\Desktop\Project_Phase\FireDataset\Training Data\Fire\1.png'
        test_image=image.load_img(image_for_testing,target_size=(256,256))
#Converting the image into array
        test_image = image.img_to_array(test_image)
#Dividing the array value by 255 to reduce the complexity
        test_image=test_image/255
        test_image=np.expand_dims(test_image,axis=0)
#predicting the result
        predict=cnn.predict(test_image)
        result=np.argmax(predict[0], axis=0)
```

Result and Snapshots

```
In [132]: test_image=image.load_img(image_for_testing,target_size=(256,256))

In [133]: test_image
```

Out[133]:

```
In [134]: test_image = image.img_to_array(test_image)
```

Result and Snapshots

```
In [12]:  1  cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 16)      448

 max_pooling2d (MaxPooling2D  (None, 128, 128, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 64, 64, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 32, 32, 128)       73856

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 128)      0
 2D)

 conv2d_4 (Conv2D)           (None, 16, 16, 256)       295168

 max_pooling2d_4 (MaxPooling  (None, 8, 8, 256)        0
 2D)

 flatten (Flatten)           (None, 16384)             0

 dense (Dense)               (None, 512)               8389120

 dense_1 (Dense)             (None, 512)               262656

 dense_2 (Dense)             (None, 512)               262656

 dense_3 (Dense)             (None, 512)               262656

 dense_4 (Dense)             (None, 512)               262656

 dense_5 (Dense)             (None, 3)                 1539
```

Result and Snapshots

In [7]:
```
#after every model to check
ckeckpoint=tf.keras.callbacks.ModelCheckpoint(r'C:\Users\chand\Desktop\Project_Phase\FireDataset\Training models\model.h5',
                                              monitor='val_loss',
                                              verbose=0,
                                              mode="min",
                                              save_weights_only=False,
                                              save_best_only=True)
callbacks = ckeckpoint
```

In [14]:
```
#Training the CNN Model i.e. Compile and Train
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

cnn.compile(optimizer='Adam',loss='binary_crossentropy', metrics=['accuracy'])

cnn.fit(training, validation_data=validation, epochs=5,
                  steps_per_epoch=training.samples//16,
                  validation_steps=validation.samples//16,
                  callbacks = callbacks)
#"steps_per_epoch=  It is used to define how many batches of samples to use in one epoch.
#It is used to declaring one epoch finished and starting the next epoch"
```

```
st by calling `.fit(numpy_data)`.
  warnings.warn('This ImageDataGenerator specifies '

Epoch 1/5
704/704 [==============================] - 1229s 2s/step - loss: 0.1996 - accuracy: 0.8810 - val_loss: 0.1359 - val_accuracy:
0.9106
Epoch 2/5
704/704 [==============================] - 939s 1s/step - loss: 0.1160 - accuracy: 0.9299 - val_loss: 0.1028 - val_accuracy:
0.9462
Epoch 3/5
704/704 [==============================] - 1030s 1s/step - loss: 0.1008 - accuracy: 0.9387 - val_loss: 0.1169 - val_accuracy:
0.9385
Epoch 4/5
704/704 [==============================] - 979s 1s/step - loss: 0.0806 - accuracy: 0.9566 - val_loss: 0.0839 - val_accuracy:
0.9582
Epoch 5/5
704/704 [==============================] - 973s 1s/step - loss: 0.0702 - accuracy: 0.9624 - val_loss: 0.0846 - val_accuracy:
0.9547
```

Out[14]: <keras.callbacks.History at 0x1e2a4a15850>

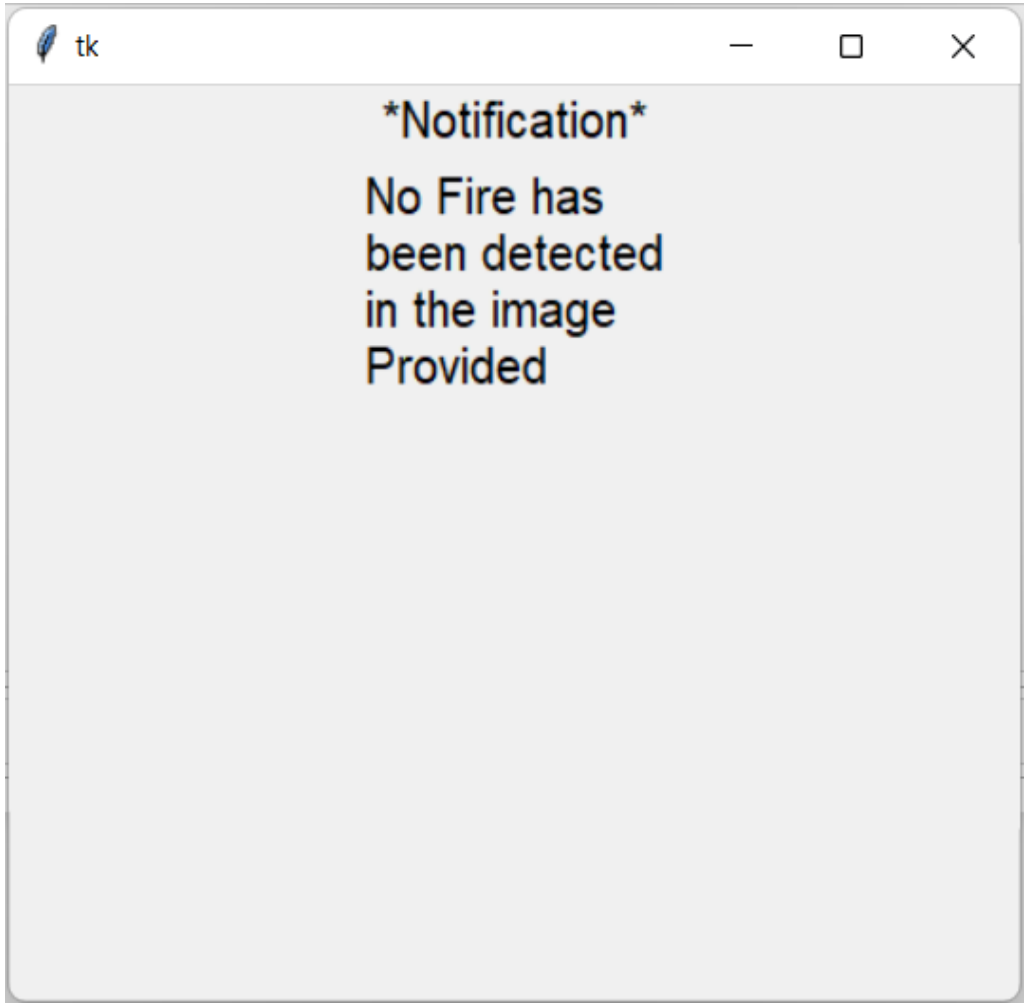
Result and Snapshots

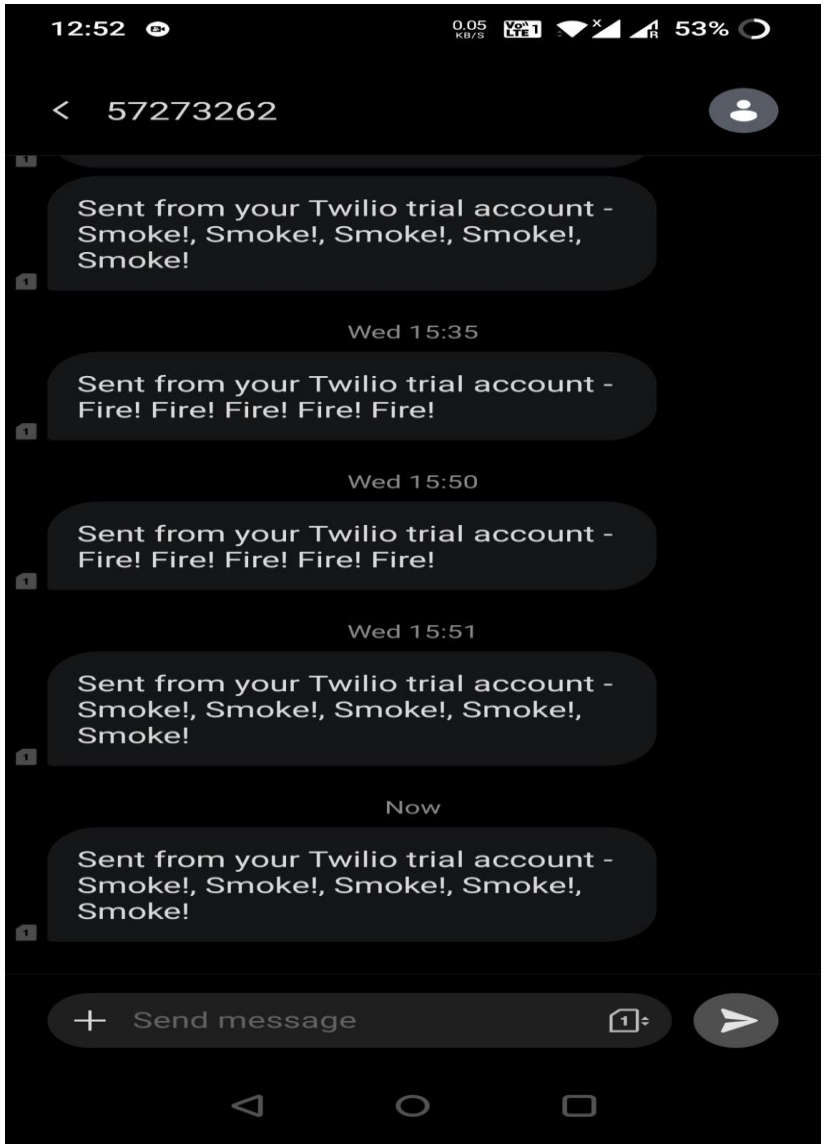
```
test_image=test_image/255
```

```
test_image
```

```
array([[[0.25490198, 0.22745098, 0.24705882],
        [0.25490198, 0.22745098, 0.24313726],
        [0.2627451 , 0.22352941, 0.23921569],
        ...,
        [0.33333334, 0.23921569, 0.21568628],
        [0.33333334, 0.23921569, 0.20784314],
        [0.34117648, 0.23921569, 0.2       ]],

       [[0.2509804 , 0.22352941, 0.23921569],
        [0.25490198, 0.22745098, 0.23921569],
        [0.2627451 , 0.22745098, 0.23137255],
        ...,
        [0.3372549 , 0.23921569, 0.20784314],
        [0.3372549 , 0.23921569, 0.2       ],
        [0.34117648, 0.23921569, 0.19607843]],

       [[0.2509804 , 0.22352941, 0.23921569],
        [0.25490198, 0.22745098, 0.23921569],
        [0.25882354, 0.22352941, 0.22352941],
        ...,
        [0.34117648, 0.23921569, 0.20784314],
        [0.34117648, 0.23921569, 0.2       ],
        [0.34117648, 0.23921569, 0.19215687]],

       ...,

       [[0.10196079, 0.07058824, 0.05490196],
        [0.10588235, 0.07450981, 0.05882353],
        [0.09803922, 0.0627451 , 0.05098039],
        ...,
        [0.09019608, 0.05098039, 0.03137255],
        [0.09019608, 0.05098039, 0.03137255],
        [0.09803922, 0.04705882, 0.03137255]],

       [[0.10196079, 0.07058824, 0.05490196],
        [0.10196079, 0.07058824, 0.05490196],
        [0.09803922, 0.06666667, 0.05098039],
        ...,
        [0.10588235, 0.0627451 , 0.04313726],
        [0.07843138, 0.03529412, 0.01568628],
        [0.10980392, 0.05882353, 0.04313726]],
```

Result and Snapshots





Conclusion and Future Enhancements

- Successfully implemented CNN model with five-five layers of convolution and polling respectively.
- Successfully tested the various scenario of “Fire”, “Smoke” and “No Fire” images.
- As a part of future enhancement Fog detection and training can be done with upcoming tools and technologies with better accuracy.
- As a part of future enhancement this model can be combine with IoT tools to cope up with the problem of fire in the forest at beginning stage.

References

Journal Papers:

[1] Pu Li, Wangda Zhao, “Image fire detection algorithms based on convolutional neural networks”, in ELSEVIER Journal, in the year 2020 .

[2] Wang Yuanbin, Dang Langfei and Ren Jieying3, “Forest fire image recognition based on convolutional neural network”, in “Journal of Algorithm & Computational Technology”, in the year 2019

Book:

“Automate the Boring Stuff with Python: Practical Programming for Total Beginners”, Al Sweigart, William Pollock, published: 14 April 2015

Website:

https://keras.io/api/

https://www.tensorflow.org/tutorials


Thank You