



CS340 – Artificial Intelligence – Project Instructions

Fire Detection



Group project



Contents

INTRODUCTION	3
IDEA AND AIMS	3
MAIN METHODS OF PROJECT	4
MAIN SECTION	9
CONCLUSION	12
REFERENCE	13
APPENDIX	13

INTRODUCTION

Fire accidents pose a great danger to the country in terms of destroying industries and densely populated places. Everyone knows the extent of the danger caused by fires and their ability to spread very quickly. There are many factors that cause these fires, including the lack of training in dealing with fires, as well as the poor storage of dangerous raw materials. One of the closest examples the world has witnessed is what happened in Beirut on August 4, 2020, was the huge fire that accompanied an explosion specifically in the port of Beirut. Early detection of fire accidents can save countless lives, in addition to saving property from damage and resulting in high financial losses.

When we look at traditional electronic fire detection systems, we see that they have many defects. Images in detecting fires played a major role in alerting the discovery of fire, it is based on computational analysis of the image. However, there was a lack of accuracy and delay in detection. developments have allowed the use of new algorithms to detect fires faster, more accurate and much better than previously to detect fires with images based on CNN models for advanced object detection, which are We will use it in our model.

IDEA AND AIMS

Our idea is to use a fire detection system that uses a convolutional neural network (CNN) that automatically detects a fire without manual intervention. The CNN method is one of the neural networks that are mostly used for image recognition and classification.

As an essential step, we have to train the classifier by deep learning and OpenCV to do the job well and distinguish images that contain the fire from images that do not. In this model, we have created an ImageDataGenerator for InceptionV3 for training. We have added two training classes: 1800 images for training and 200 images for validation.

This report shows a custom InceptionV3 model for fire detection. Considering the fair fire detection accuracy of the CNN model, we discuss the main methods for work in our project and show screenshots of our output.

Where the aims are to detect fires, distinguish images of fires from others, as well as detect distant fires.

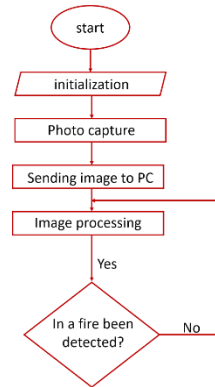
Here is a simplified sample flowchart showing how the system works:

First, it uploads capturing image from PC folders or files.

Second, the captured image is processed.

Third, the captured image is checked.

Fourth, if a fire is detected, it will quickly send an alarm mail to the administrator.

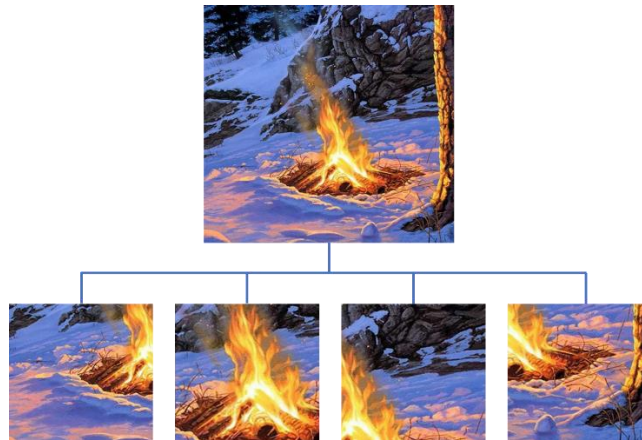


MAIN METHODS OF PROJECT

We used a CNN architecture that enables early detection of fires through images.

- * Use `!wget` in order to download files from URL.
- * Use `!unzip` in order to decompress.
- * We did import the necessary packages .
- * The path will be set to the folder containing all the images, for both training and validation.
- * We created an `ImageDataGenerator()` for our custom InceptionV3. It contains 1800 images for training and 200 images for validation. In order to make the training more accurate we used data augmentation techniques. Keras ImageDataGenerator will enable augmentation of your images in real time from training and this will result in random transformations in each training image and we will have a powerful model.

Look at the picture that will explain the idea he will make of the picture:



If we notice that the main image above is the original and the images below are the images that he produced for further training.

As this is the technique used to produce many different converted copies of the same original images. The versions will all be different in other ways depending on the optimization

techniques we'll use in code, and we'll bring a new perspective on object capture in real life.

- We used `1./255` in the parameter `rescale` to multiply every pixel in the preprocessing, need this before training neural network
- We inverted existing images, zoomed them in to add more training to the data and avoid over-fitting. used a `zoom range=0.15` and we flipped the image horizontally in order to get better results.
- In `fill mode` we used the default value “nearest” in order to replace the empty area with the nearest pixel value.

* `flow_from_directory()` method , it will read images directly from the manual as well as augment them while learning the neural network model on the training data. The method expects that images belonging to different classes are present in different folders but are inside the same folder.

- We will choose the `size 224X224`, as it will resize all images regardless of the original size of the image in the train guide.
- `shuffle` the data, is default: `True`, used for split the data randomly.
- `class_mode` for classes to predict . we will Set `"categorical"` to 2-D one-hot encoded labels.
- We will set `batch size = 128` which is the number of images to extract from the folder for each batch. This is in `training`, but in `validation` it will be `14`.

* We use `Inception v3` class from ‘`tensorflow.keras.applications`’ library is a convolutional neural network for assisting in image analysis and object detection. Then inside the code we use `weights='imagenet'` because The models used in the `imagenet` classification competitions are measured against each other for performance. Therefore, it provides a "standard" measure for how good a model is for image classification. So many often use imagenet weights.

* Also, we import `image` class from `keras.preprocessing` which is the data preprocessing and data augmentation module of the Keras deep learning library. It provides utilities for working with image data, text data, and sequence data.

* We work to import `Model` class from `keras.models` to Create a model in functional way by specifying both input and output layer. In machine learning, a model is a function with

learnable parameters that maps an input to an output. The optimal parameters are obtained by training the model on data. A well-trained model will provide an accurate mapping from the input to the desired output. And `model.predict(x)` function using to pass the input vector through the model and returns the output tensor for each datapoint.

*We try to import many layers class from `keras.layers` which are:

- `input_tensor` method is `layers.Input()` to use as image input for the model and useful for sharing inputs between multiple different networks. And include `shape=(224, 224, 3)` is also the shape of the input for keras
- `base_model` method to create the base pre-trained model, used model is `InceptionV3()`. And This function returns a Keras image classification model, optionally loaded with weights pre-trained on ImageNet. It has three arguments. Firstly, `input_tensor` is an optional Keras tensor, as we mentioned before. Secondly, weights such as random initialization, and imagenet are default. Third, `include_top` determines the fully-connected layer at the top if be `true`(this is the default state), if be `false` means has the last layer of the network. So we used weights is imagenet and included the last layer of the network
- `base_model.output` is for retrieving the output tensor of a layer. But applicable if the layer has exactly one output and is connected to one incoming layer
- `GlobalAveragePooling2D()` is global average pooling designed to replace fully connected layers in classical CNNs, `channels_last` corresponds to inputs with shape (batch, height, width, channels) while `channels_first` corresponds to inputs with shape (batch, channels, height, width) and It defaults to the `image_data_format` value found in your Keras config file. It applies average pooling on the spatial dimensions until each spatial dimension is one, so the output of the model will be a 2D tensor, which will be applied to the output of the last convolutional layer, and leaves other dimensions unchanged. In this case values are not kept as they are averaged. In this case values are not kept as they are averaged. For example a tensor (samples, 11, 19, 1) would be output as (samples, 1, 1, 1)

- **Dense() layer class**, which is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output. Here its returned a list of two values: the **kernel matrix(2048)** and the bias **vector(activation='relu')** and can be used to set the weights of another Dense layer. it is another non-linear activation function that has gained popularity in the deep learning domain. **'relu'** stands for Rectified Linear Unit which is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. Also, we use **activation='softmax'** which is a mathematical function that converts a vector of numbers into a vector of probabilities
- **Dropout()** helps prevent overfitting. And it is randomly set input units to 0 with a frequency of rate at each step during training time. Inputs not set to 0 are **scaled up by 1/(1 - rate)** such that the sum over all inputs is unchanged, and It only applies when training part and will set to true value. We add a dropout rate of 0.2 Dropout.

* for layer in base_model.layers: **layer.trainable = False.**

trainable API in detail, which underlies most transfer learning & fine-tuning workflows. So we set **trainable = False** on a model or on any layer that has sublayers, all children's layers become non-trainable as well. And vice versa if it is true

* **model.compile()** method we need to configure the learning process before training a model. It receives three arguments. **Firstly**, an optimizer such as **rmsprop** or **degrade**. **Secondly**, a loss function is an objective that the model will try to **minimize**. It can be the string identifier of an existing loss function. **Third**, is a list of metrics. For any classification problem, you will want to set this to **metrics=['accuracy']**. A metric could be the string identifier of an existing metric (**only accuracy is supported at this point**) or a custom metric function .

* **class myCallback(tf.keras.callbacks.Callback)**

Often, when training we want to stop training once the training accuracy reaches a certain desired threshold. Thus, we can achieve what we want and avoid wastage of resources (**time and computation power**). By this method, we are creating a new class by extending **tf.keras.callbacks.Callback** and implement a callback to stop training. And a callback is a tool to customize the behavior of a Keras model during training, evaluation, or inference .

* **def on_epoch_end(self, epoch, logs={ }):**

This function is called at the end of an epoch during training. And invoked at the end of each epoch. Next, we are fetching the value of accuracy at the end of that epoch, and if it is greater than our threshold, we are setting the `stop_training` of the model to True .

- * `callbacks = myCallback()`: Instantiate an object of `myCallback` class.

- * `history = model.fit()`

Every time you call `model.fit()`, it returns a `keras.callbacks.History` object whose history attribute contains a dictionary. The keys to the dictionary are a loss for training, `val_loss` for validation loss, and any other metrics that you might have set while compiling .

- * We use `SGD` as an optimizer with a learning rate of `0.0001` and a momentum of `0.9`.

Learning rate that takes no arguments and returns the actual value to use. Defaults to `0.01`.

Momentum that accelerates gradient descent in the relevant direction and dampens oscillations. Defaults to `0`.

- * Also, we work to result plots and store it in the notebook document by using `%matplotlib inline` which sets the backend of matplotlib to the 'inline' backend: With this backend, the output of plotting commands is displayed inline within frontends directly below the code cell that produced it.

- * And we Import `matplotlib.pyplot` because it offers a collection of functions in the popular visualization package Matplotlib. Its functions manipulate elements of a figure, such as creating a figure, creating a plotting area, plotting lines and adding plot labels.

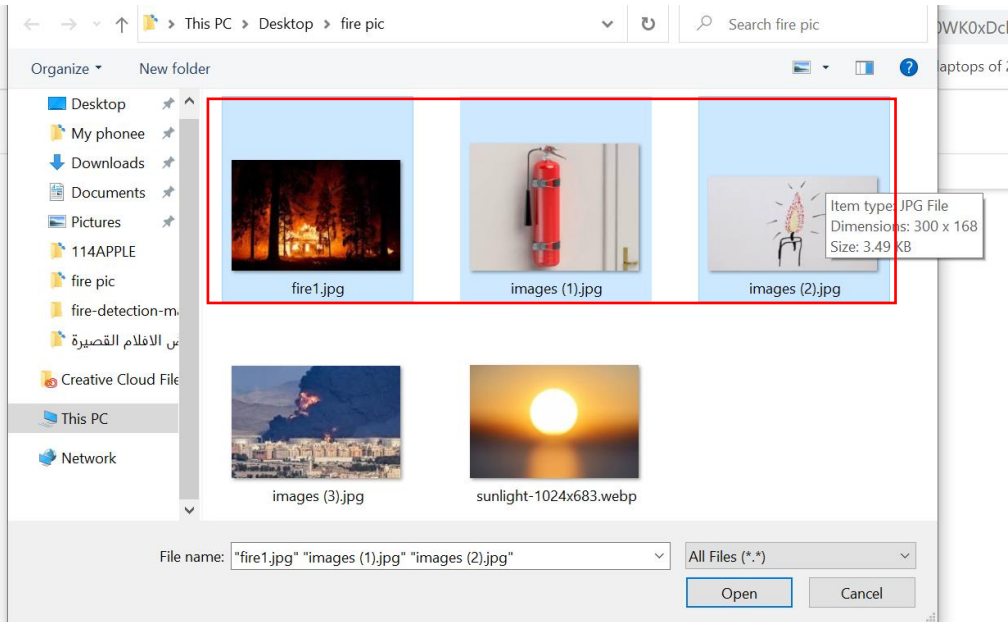
- * `import numpy as np` that's let us import `numpy` package to bring the NumPy library into our environment. It gives numPy the alias of `np`. This allows you to use NumPy functions by simply typing `np`. `expand_dims()` function is used to expand the shape of an array. Insert a new axis that will appear at the axis position in the expanded array shape. `Argmax` function to print maximum value in the array.

- * `google.colab import files` we already install Drive into the Colab meaning that setting up the google drive account as a virtual drive so that we can access the resources of the drive just like a local hard drive. And here we use `files` package to uplope files that already install on google colab.

- * Again we use `keras.preprocessing import image` to work with images also to converts the image to a NumPy array and reports the path and shape.

MAIN SECTION

Final output:



First run

Output explanation: we try to test the system by uploading 3 images from already downloading image on the PC, only one image of them contains a real forest fire.

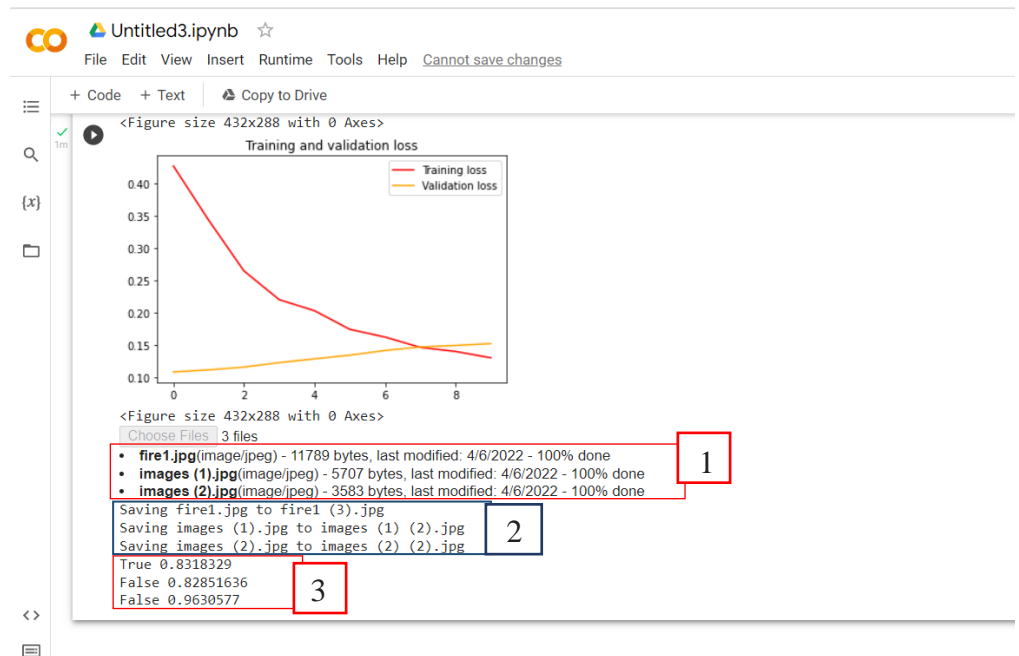
Output explanation:

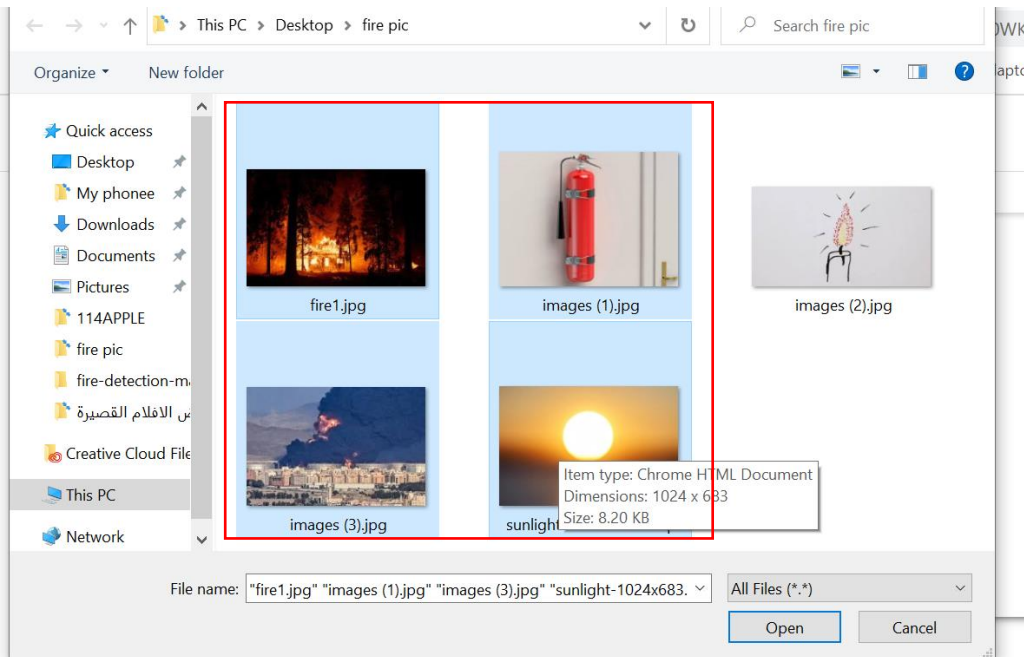
1- uploading done for 3 images.

2- Saving all images on drive.

3- The prediction result was organized as the order of uploading images. All images prediction was

correct and the real forest fire was the only true one. Note that the number near to the result is the predict result percentage.





Second run

Output explanation: we try to test the system by uploading 4 images from already downloading image on the PC, two images contain a real forest fire and the other was a city fire and smoke.

Output explanation:

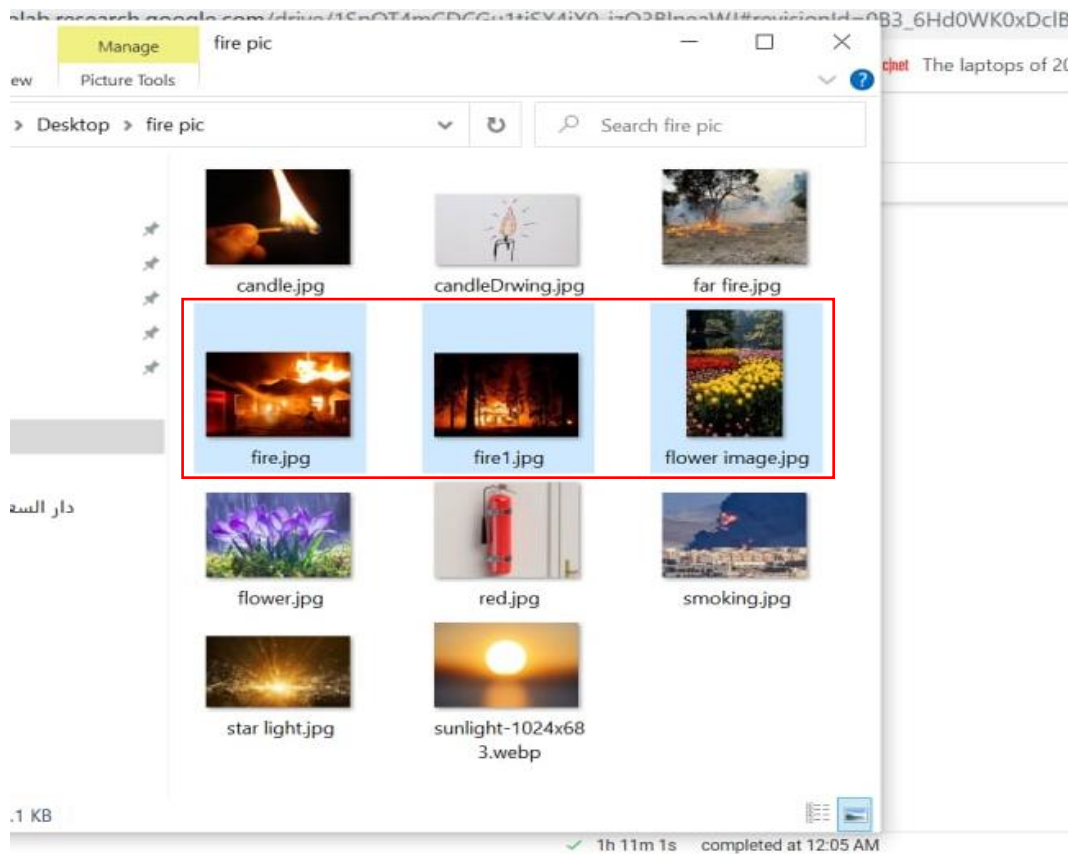
1- uploading done for 4 images.

2- Saving all images on drive.

3- The prediction result was organized as the order of uploading images. All images prediction was correct but the

sun light was confusing the system though it fire. The real forest fire and the city fire and smoke should be the only two. Note that the number near to the result is the predict result percentage.





Third run

Output explanation:

we try to test the system by uploading 3 images from already downloading image on the PC, first two images contain a real forest fire and the last one was flowers garden.

Output explanation:

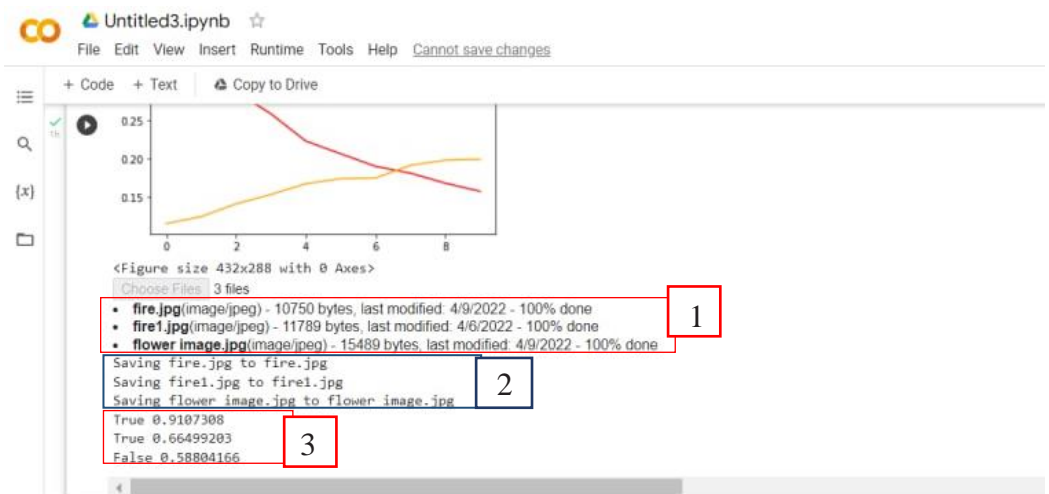
1- uploading done for 3 images.

2- Saving all images on drive.

3- The prediction result was

organized as the order of uploading images. All images prediction was

correct and flowers garden image was the only the false one. Note that the number near to the result is the predict result percentage.



CONCLUSION








The systems came with different methods for identifying the target, distinguishing and controlling movement. They have a high computational ability to identify the fire, which enables the detection of fires quickly and early. In this case, it is very useful for emergency relief programs, in order to avoid disasters, The fires caused huge human and economic losses.

In our model, we proposed a very effective method of early detection of fires based on well-defined CNNs.

We integrated deep features into our system, and we proved that it can identify fires with high accuracy in different environments, whether indoors or outdoors, whether in forests or in cities, as well as recognize and identify remote fires that appear in the image, and all of that was in very early stages.

Among its features is that it works independently without human intervention.

REFERENCE

-  <https://keras.io/api/applications/>
-  <https://github.com/>
-  <https://www.tensorflow.org/>
-  <https://www.w3schools.com/python/>
-  <https://stackoverflow.com/>
-  <https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/>
-  <https://www.pluralsight.com/guides/image-classification-using-tensorflow>

APPENDIX

Dataset file attached

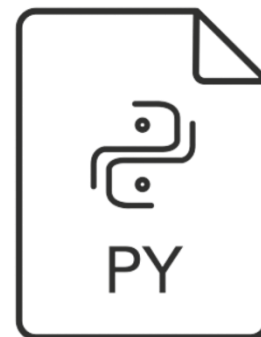
Just click on the image to open



Dataset

Complete source code attached

Just click on the image to open



Or source code in colab

Just click on the image to open

In Colab

To download files from URL

```
!wget https://github.com/DeepQuestAI/Fire-Smoke-Dataset/releases/download/v1/FIRE-  
SMOKE-DATASET.zip
```

To decompress

```
!unzip FIRE-SMOKE-DATASET.zip
```

```
# import the necessary packages
```

```
import shutil
```

```
shutil.rmtree('/content/FIRE-SMOKE-DATASET/Test/Smoke')
```

```
shutil.rmtree('/content/FIRE-SMOKE-DATASET/Train/Smoke')
```

```
# import the necessary packages
```

```
import tensorflow as tf
```

```
import keras_preprocessing
```

```
from keras_preprocessing import image
```

```
from keras_preprocessing.image import ImageDataGenerator
```

```
import shutil
```

#directory

```
#path to the folder that contains all the images.
```

```
TRAINING_DIR = "/content/FIRE-SMOKE-DATASET/Train"
```

#This is the technique used to produce many different converted copies of the same original images.

#All versions will be different in other respects depending on the enhancement techniques we will use below.

#We will provide a new perspective to capture the object in real life.

```
training_datagen = ImageDataGenerator(rescale=1./255, #the parameter rescale to multiply
every pixel in the preprocessing 1./255 ,need this before training neural network
```

#To existing images are flipped, zoomed to add more training on the data and avoid over-fitting.

#used Zoom

```
zoom_range=0.15,# zoom.
```

#flipping

#To get a better result, we use image flipping, which is a great magnification technique.

#It makes sense to use it with a lot of different objects.

horizontal_flip=True, # horizontal flip.

#use the fill_mode argument.

#the default value used is “nearest” which simply replaces the empty area with the nearest pixel values.

fill_mode='nearest')

#directory

#path to the folder that contains all the images.

VALIDATION_DIR = "/content/FIRE-SMOKE-DATASET/Test"

validation_datagen = ImageDataGenerator(rescale = 1./255)#the parameter rescale to multiply every pixel in the preprocessing 1./255 ,need this before training neural network

ImageDataGenerator flow_from_directory

to read the images directly from the directory and augment them while the neural network model is learning on the training data.

The method expects that images belonging to different classes are present in different folders but are inside the same folder.

train_generator = training_datagen.flow_from_directory(

TRAINING_DIR,

#resize to this size

target_size=(224,224), # To a resizes all the images to the size of 224X224, irrespective of the original size of the image present in train directory.

#shuffle the data

#default: True

#split the data randomly

shuffle = True,

classes to predict

```

class_mode='categorical', #Set categorical to 2-D one-hot encoded labels.
# number of images to extract from folder for every batch
batch_size = 128 #Size of the batches of data.
)
# ImageDataGenerator flow_from_directory
validation_generator = validation_datagen.flow_from_directory(
    VALIDATION_DIR,
    target_size=(224,224), # To a resizes all the images to the size of 224X224, irrespective of
the original size of the image present in validation directory.
    # classes to predict
    class_mode='categorical',#Set categorical to 2-D one-hot encoded labels.
    #shuffle the data
    #default: True
    #split the data randomly
    shuffle = True,
    # number of images to extract from folder for every batch
    batch_size= 14 #Size of the batches of data.
)

#libraries call
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Input, Dropout
# use this code to classify input images that have class labels inside ImageNet
# shape should use for the input for keras
input_tensor = Input(shape=(224, 224, 3))
# create the base pre-trained model
#'imagenet' subpackage provides a variety of pre-trained state-of-the-art models which is
trained on ImageNet dataset

```



```

base_model = InceptionV3(input_tensor=input_tensor, weights='imagenet',
include_top=False)

# add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
# let's add a fully-connected layer, for CNN, ReLU is treated as a standard activation function
x = Dense(2048, activation='relu')(x)
# Add a dropout rate of 0.25. Dropout is a technique where randomly selected neurons are
ignored during training.
x = Dropout(0.25)(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)
# Dense layer does the below operation on the input and return the output, result is the
output and it will be passed into the next layer.
# logistic layer -- let's say we have 2 classes
# Softmax is a mathematical function that converts a vector of numbers into a vector of
probabilities
predictions = Dense(2, activation='softmax')(x)
# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)

# first: train only the top layers (which were randomly initialized)
# freeze all complex InceptionV3 layers
for layer in base_model.layers:
    layer.trainable = False

# compile the model using selected loss function, optimizer and metrics, RMSProp is
designed to accelerate the optimization process
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])
# Implement callback function to stop training.
class myCallback(tf.keras.callbacks.Callback):

```

```

#End epoch {} of training
def on_epoch_end(self, epoch, logs={}):
    if(logs.get('val_loss')<=0.1099 and logs.get('loss')<=0.1099):
        print("\n\n Reached The Destination!")

        self.model.stop_training = True
# create a callback object
callbacks = myCallback()

# Fit model on training data
#pass some validation for
    # monitoring validation loss and metrics
    # at the end of each epoch
history = model.fit(
    train_generator,
    steps_per_epoch = 14,
    epochs = 20,
    validation_data = validation_generator,
    validation_steps = 14,
    callbacks=[callbacks]
)

# Unfreeze some layers in our model
for layer in model.layers[:249]:
    layer.trainable = False
for layer in model.layers[249:]:
    layer.trainable = True
#import package stochastic gradient descent(SGD)
from tensorflow.keras.optimizers import SGD
# Train model with more trainable layers

```

```
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9), loss='categorical_crossentropy',
metrics=['acc'])
```

```
#Implement callback function to stop training.
```

```
class myCallback(tf.keras.callbacks.Callback):
```

```
#End epoch { } of training
```

```
def on_epoch_end(self, epoch, logs={}):
```

```
    if(logs.get('val_loss')<=0.1099 and logs.get('loss')<=0.1099):
```

```
        print("\n\n Reached The Destination!")
```

```
        self.model.stop_training = True
```

```
# create a callback object
```

```
callbacks = myCallback()
```

```
# Fit model on training data
```

```
#pass some validation for
```

```
    # monitoring validation loss and metrics
```

```
    # at the end of each epoch
```

```
history = model.fit(
```

```
    train_generator,
```

```
    steps_per_epoch = 14,
```

```
    epochs = 10,
```

```
    validation_data = validation_generator,
```

```
    validation_steps = 14,
```

```
    callbacks=[callbacks]
```

```
)
```

```
#Number of layers after calling the model
```

```
print(len(base_model.layers))
```

```
# PLOT LOSS AND ACCURACY
```

%matplotlib inline sets the backend of matplotlib to the 'inline' backend: With this backend, the output of plotting commands is displayed inline within frontends directly below the code cell that produced it. The resulting plots will then also be stored in the notebook document

%matplotlib inline

import matplotlib.pyplot as plt

#Retrieve a list of list results on training and test data

sets for each training epoch

acc = history.history['acc']

val_acc = history.history['val_acc']

loss = history.history['loss']

val_loss = history.history['val_loss']

Get number of epochs

epochs = range(len(acc))

#Plot training and validation accuracy per epoch, name lable and color

plt.plot(epochs, acc, 'g', label='Training accuracy')

plt.plot(epochs, val_acc, 'b', label='Validation accuracy')

plt.title('Training and validation accuracy')

#Used to Place a legend on the axes also,Loc in legend() is used to specify the location of the legend, default value of loc is loc="best" (upper left)

plt.legend(loc=0)

#Show figure for training and validation accuracy

plt.figure()

plt.show()

Plot training and validation loss per epoch,name lable and color

plt.plot(epochs, loss, 'r', label='Training loss')

plt.plot(epochs, val_loss, 'orange', label='Validation loss')

plt.title('Training and validation loss')

#Used to Place a legend on the axes also,Loc in legend() is used to specify the location of the legend, default value of loc is loc="best" (upper left)

plt.legend(loc=0)

#Show figure for training and validation loss

```
plt.figure()
plt.show()
```

```
#predicting any random image
```

```
import numpy as np #To allows for a user to upload a image, Means it will prompt you to
select a file. Click on "Choose Files" then select and upload the file.
```

```
from google.colab import files #To upload from local drive
```

```
from keras.preprocessing import image #Use this library to converting an image with the
Keras API
```

```
#upload test images
```

```
uploaded = files.upload()
```

```
for fn in uploaded.keys():
```

```
    path = '/content/' + fn
```

```
# load_img function for loading an image from file as a PIL image object. PIL let saving
many different image file formats.
```

```
    img = image.load_img(path, target_size=(224, 224))
```

```
#convert to numpy array
```

```
    x = image.img_to_array(img)
```

```
#Function used to expand the shape of an array.
```

```
    x = np.expand_dims(x, axis=0) /255
```

```
    classes = model.predict(x)
```

```
# Indices of Max element
```

```
    print(np.argmax(classes[0])==0, max(classes[0]))
```