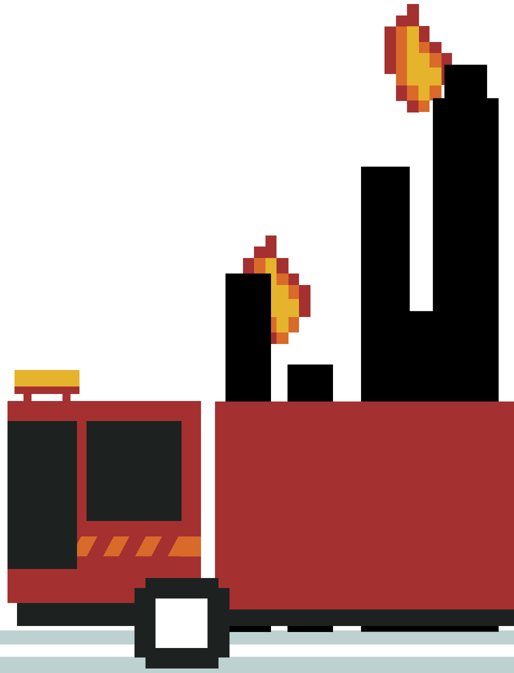
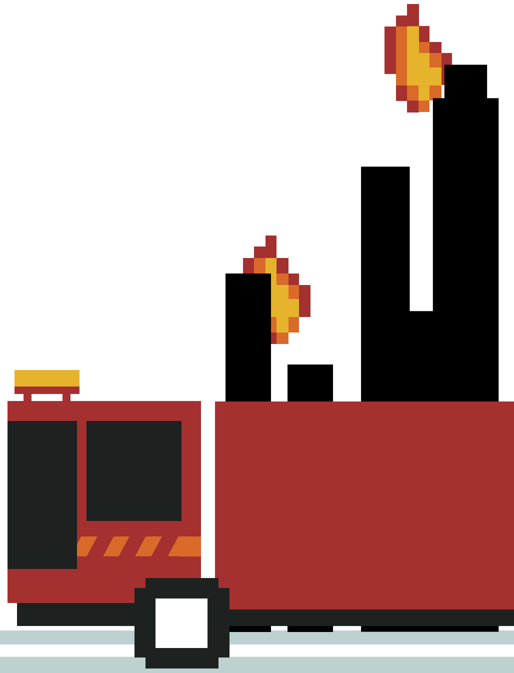
Kingdom of Saudi Arabia
Al-Imam Muhammad Ibn Saud Islamic University
College of Shariah and Islamic Studies in Ahsa'a
Department of Computer Science and Informati
Computer Organization and Assembly Language

CS340 – Artificial Intelligence – Project Instructions

Fire Detection



Contents:

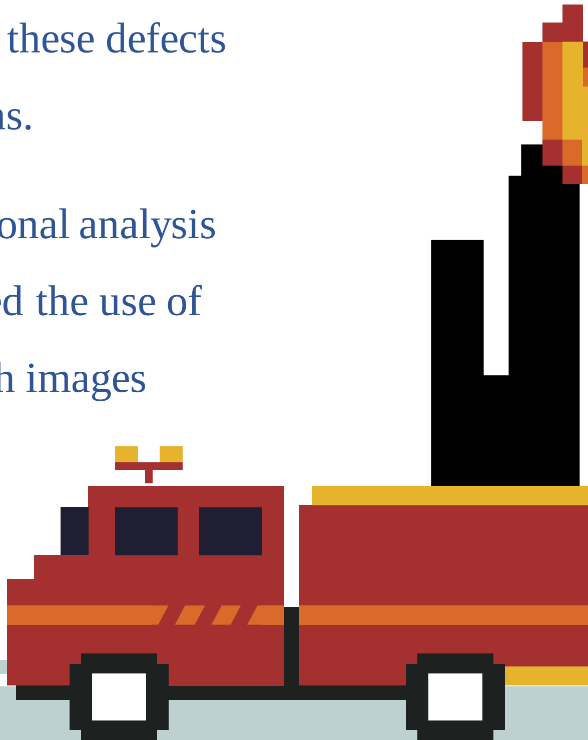


Introduction:

Fire accidents pose a great danger to the country in terms of destroying industries and densely populated places. 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 . Click here (▶)

When we look at traditional electronic fire detection systems, we see that they have many defects, and these defects require periodic maintenance to ensure that they work because they are always exposed to malfunctions.

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:

Our idea is to use a fire detection system that uses a convolutional neural network (CNN) that automatically detects fire. The CNN method is a neural network that is mostly used for image recognition and classification.
As an essential step, we have to train the classifier with Deep Learning and OpenCV to do the job well and mark the images that contain fire. In our template, we've added two training classes: 1800 images for training and 200 images for validation.

Aims:

- Fire detection
- Detecting distant fires
- Distinguishing from being a fire or not

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

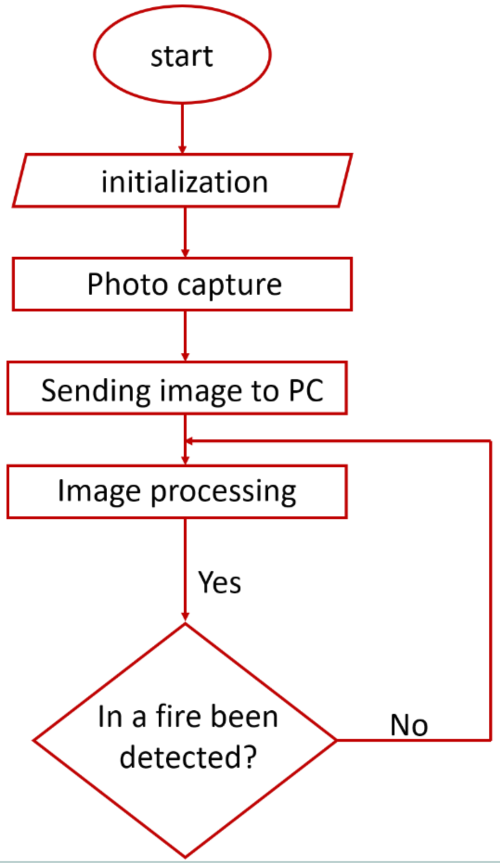
First, it takes the photo through the webcam.

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.

!wget

To download files from URL

!unzip

To decompress

Main methods of project :

ImageDataGenerator ()

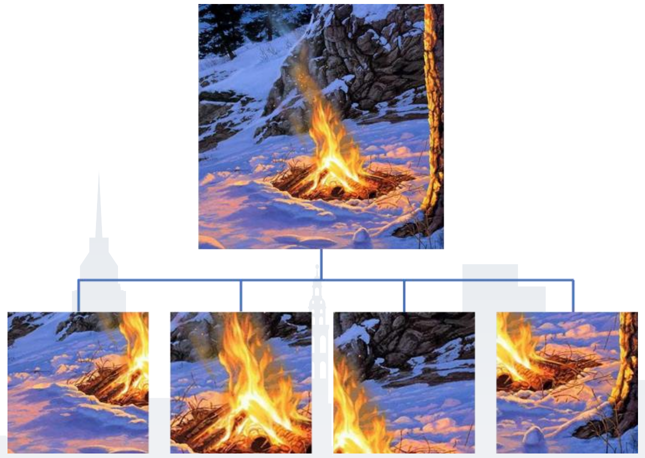
We created an ImageDataGenerator( ) for our custom InceptionV3. It contains 1800 images for training and 200 images for validation. In order to make the training more accurate we used data augmentation techniques.

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.

As this is the technique used to produce many different converted copies of the same original images.

The versions will be different in other ways depending on the optimization techniques we used in the code which are:

* Rescale = 1./255
* zoom range= 0.15
* Flipped (horizontal_flip= true)
* Fill mode = “nearest”



Main methods of project :

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

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

directory

We set the path to the folder containing all the images, for both training and validation.

flow_from_directory()

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.

target_size= (224,224)

shuffle = True

class_mode='categorical'

batch_size = (128) , (14)

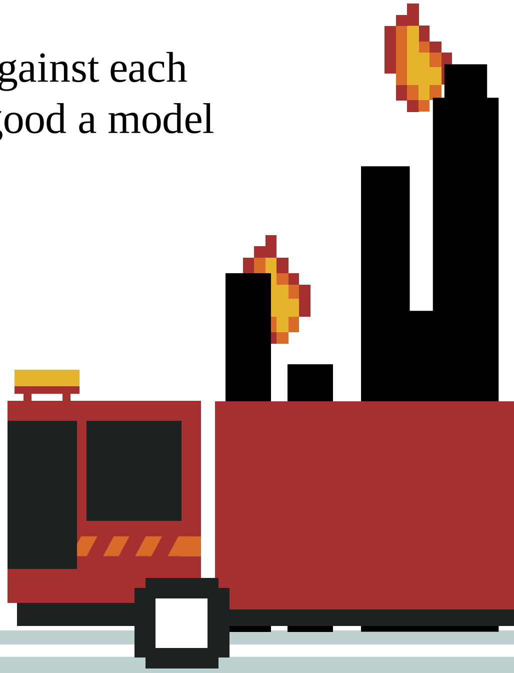
Main methods of project :

Inception v3

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’

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.



Main methods of project :

image

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.

Model

We import Model class from keras.models to Create to 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. Also, it helps us to use:

- model.predict(x) function inside the code to passe the input vector through the model
and returns the output tensor for each datapoint.

Main methods of project :

keras.layers

- input_tensor method is layers.Input() to use as image input for the model.
- base_model method to create the base pre-trained model.
- base_model.output is for retrieving the output tensor of a layer.
- GlobalAveragePooling2D() is global average pooling designed to replace fully connected layers in classical CNNs, channels_last with channels_first .

Main methods of project :

keras.layers

- 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 it’s 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. '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. By set input unit to be 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.

Main methods of project :

keras.layers

- 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.

Main methods of project :

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

A callback is a tool to customize the behavior of a Keras model during training, evaluation, or inference

Main methods of project :

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

This function is called at the end of an epoch during training. When we want to stop training once the training accuracy reaches a certain desired threshold. 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

Main methods of project :

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

Ensures that logs are created and stored. 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

Main methods of project :

```
for layer in model.layers[:249]:
       layer.trainable = False
for layer in model.layers[249:]:
   layer.trainable = True
```

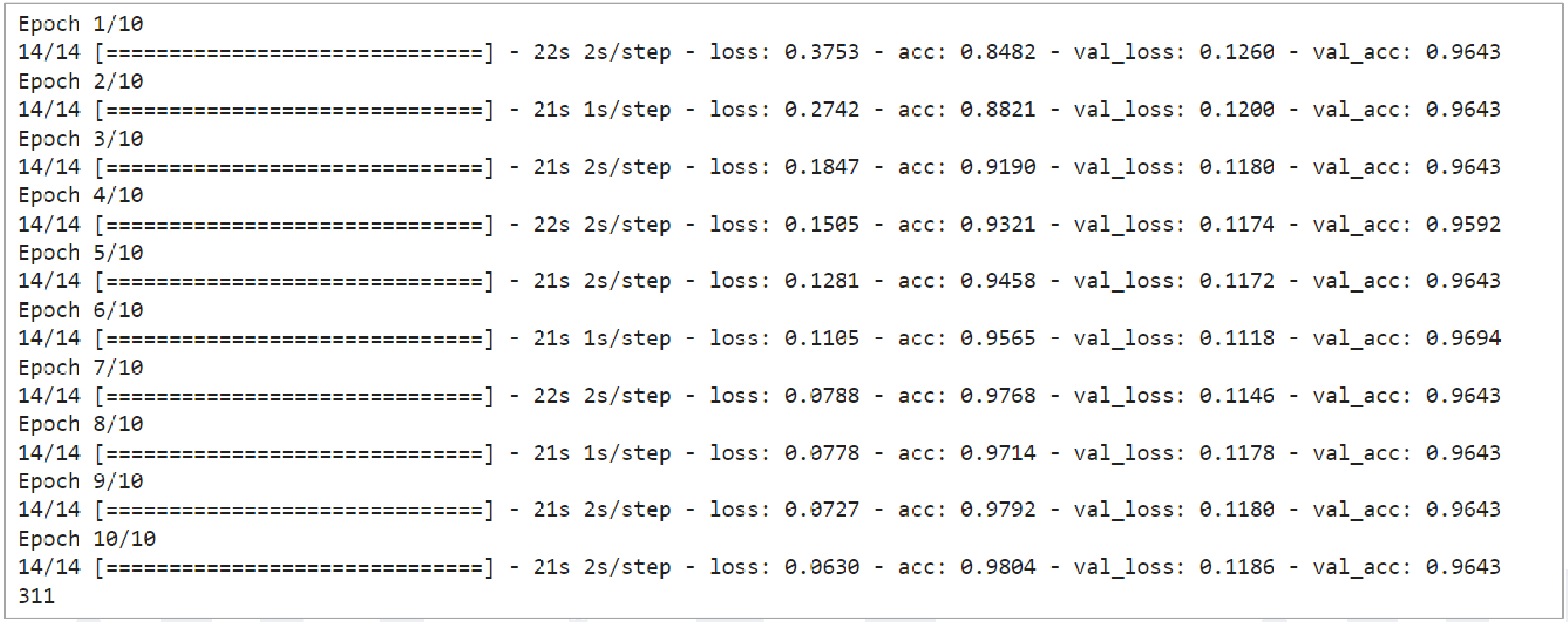
After training our top layers for 20 epochs, we will freeze the first 249 layers of the models and train the rest i.e the top 2 inception blocks.

```
from tensorflow.keras.optimizers import SGD
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9),
loss='categorical_crossentropy', metrics=['acc’])


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

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

Main methods of project :

```
#Number of layers after calling the model
print(len(base_model.layers))
```

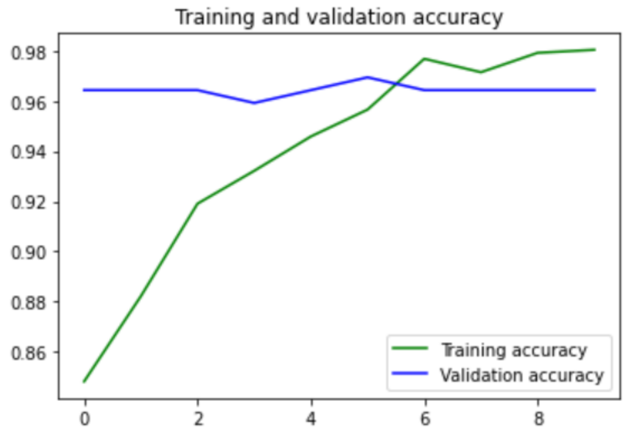


```
Epoch 1/10
14/14 [==============================] - 22s 2s/step - loss: 0.3753 - acc: 0.8482 - val_loss: 0.1260 - val_acc: 0.9643
Epoch 2/10
14/14 [==============================] - 21s 1s/step - loss: 0.2742 - acc: 0.8821 - val_loss: 0.1200 - val_acc: 0.9643
Epoch 3/10
14/14 [==============================] - 21s 2s/step - loss: 0.1847 - acc: 0.9190 - val_loss: 0.1180 - val_acc: 0.9643
Epoch 4/10
14/14 [==============================] - 22s 2s/step - loss: 0.1505 - acc: 0.9321 - val_loss: 0.1174 - val_acc: 0.9592
Epoch 5/10
14/14 [==============================] - 21s 2s/step - loss: 0.1281 - acc: 0.9458 - val_loss: 0.1172 - val_acc: 0.9643
Epoch 6/10
14/14 [==============================] - 21s 1s/step - loss: 0.1105 - acc: 0.9565 - val_loss: 0.1118 - val_acc: 0.9694
Epoch 7/10
14/14 [==============================] - 22s 2s/step - loss: 0.0788 - acc: 0.9768 - val_loss: 0.1146 - val_acc: 0.9643
Epoch 8/10
14/14 [==============================] - 21s 1s/step - loss: 0.0778 - acc: 0.9714 - val_loss: 0.1178 - val_acc: 0.9643
Epoch 9/10
14/14 [==============================] - 21s 2s/step - loss: 0.0727 - acc: 0.9792 - val_loss: 0.1180 - val_acc: 0.9643
Epoch 10/10
14/14 [==============================] - 21s 2s/step - loss: 0.0630 - acc: 0.9804 - val_loss: 0.1186 - val_acc: 0.9643
311
```

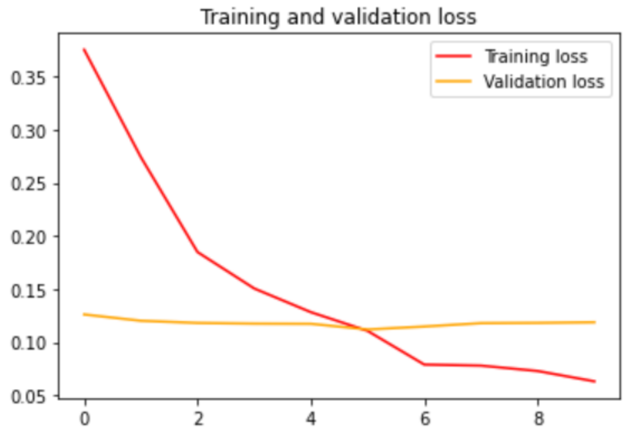
Main methods of project :

```
%matplotlib inline
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss’]

epochs = range(len(acc))

plt.plot(epochs, acc, 'g', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy’)

plt.legend(loc=0)
plt.figure()
plt.show()

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'orange', label='Validation loss')
plt.title('Training and validation loss')

plt.legend(loc=0)
plt.figure()
plt.show()
```

Sets the backend of matplotlib to the 'inline' backend



```
<Figure size 432x288 with 0 Axes>
```

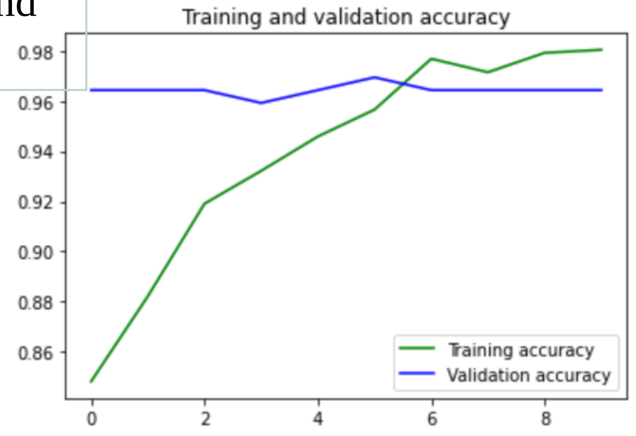


```
<Figure size 432x288 with 0 Axes>
```

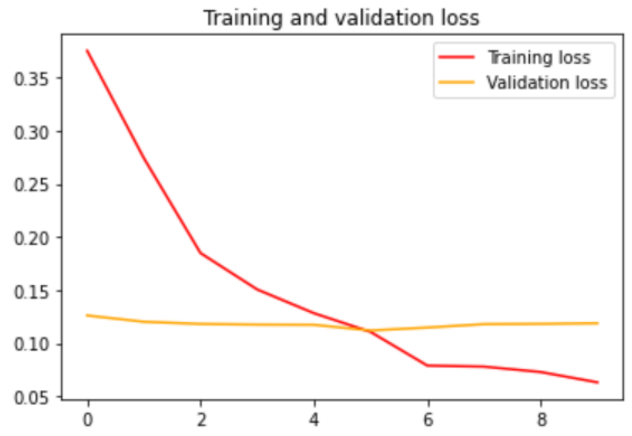
Main methods of project :

```
%matplotlib inline
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss’]

epochs = range(len(acc))

plt.plot(epochs, acc, 'g', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy’)

plt.legend(loc=0)
plt.figure()
plt.show()

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'orange', label='Validation loss')
plt.title('Training and validation loss')

plt.legend(loc=0)
plt.figure()
plt.show()
```

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.



```
<Figure size 432x288 with 0 Axes>
```

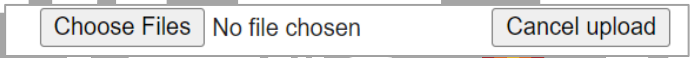


```
<Figure size 432x288 with 0 Axes>
```

Main methods of project :

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()
for fn in uploaded.keys():
  path = '/content/' + fn
  img = image.load_img(path, target_size=(224, 224))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0) /255
  classes = model.predict(x)
  print(np.argmax(classes[0])==0, max(classes[0]))
```

To allows for a user to upload file and a image. Means 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.

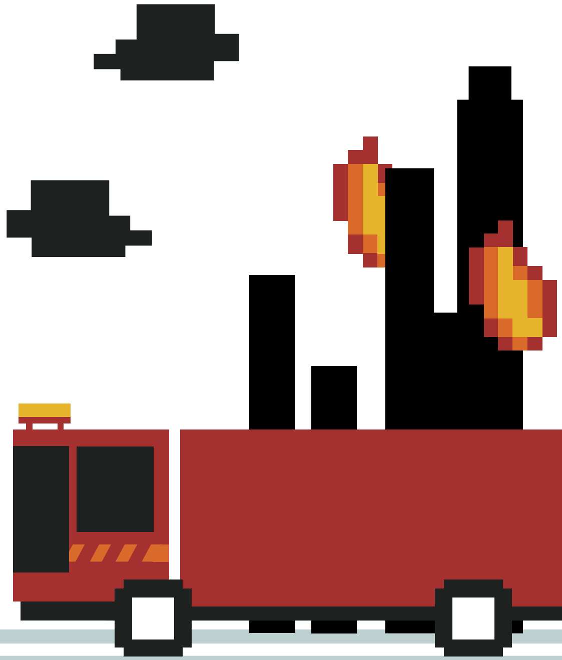


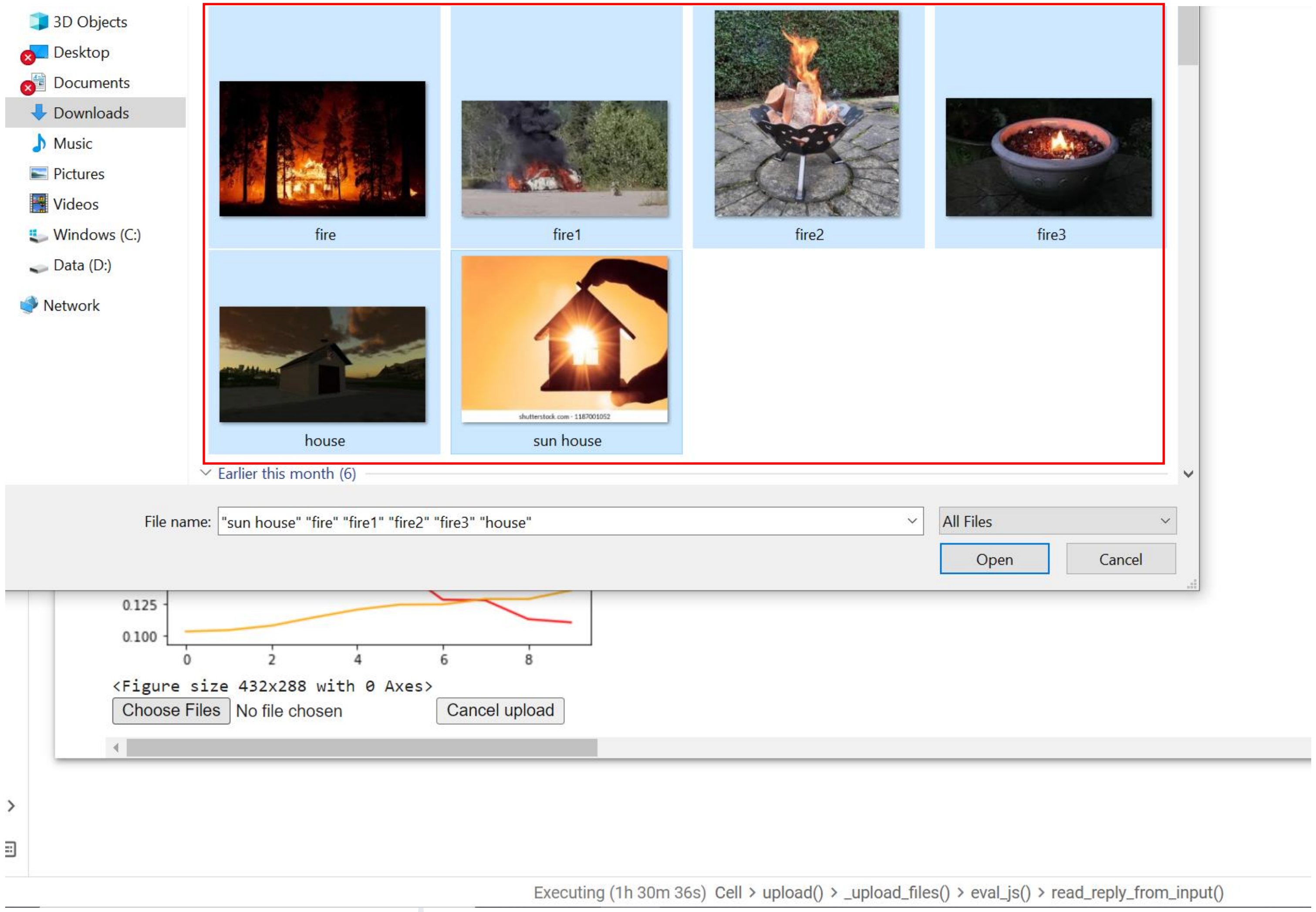
Main methods of project :

```
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()
for fn in uploaded.keys():
  path = '/content/' + fn
  img = image.load_img(path, target_size=(224, 224))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0) /255
  classes = model.predict(x)
  print(np.argmax(classes[0])==0, max(classes[0]))
```

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.

Final output





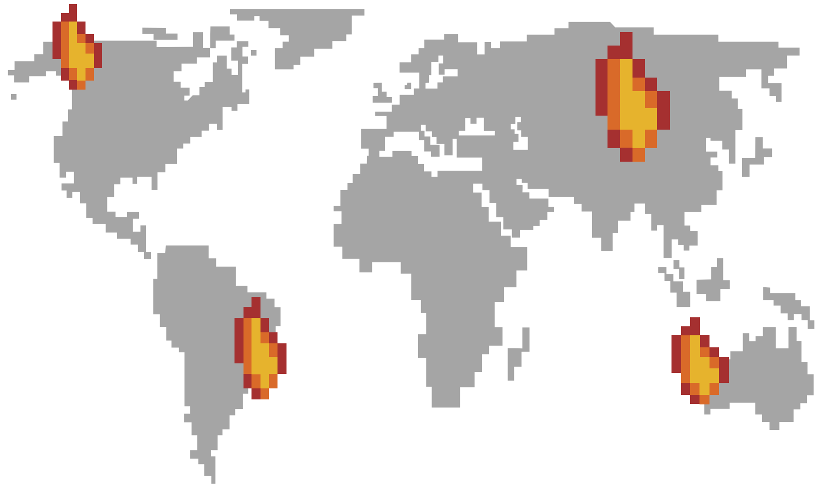
File name: "sun house" "fire" "fire1" "fire2" "fire3" "house"

```
<Figure size 432x288 with 0 Axes>
```

Executing (1h 30m 36s) Cell > upload() > _upload_files() > eval_js() > read_reply_from_input()

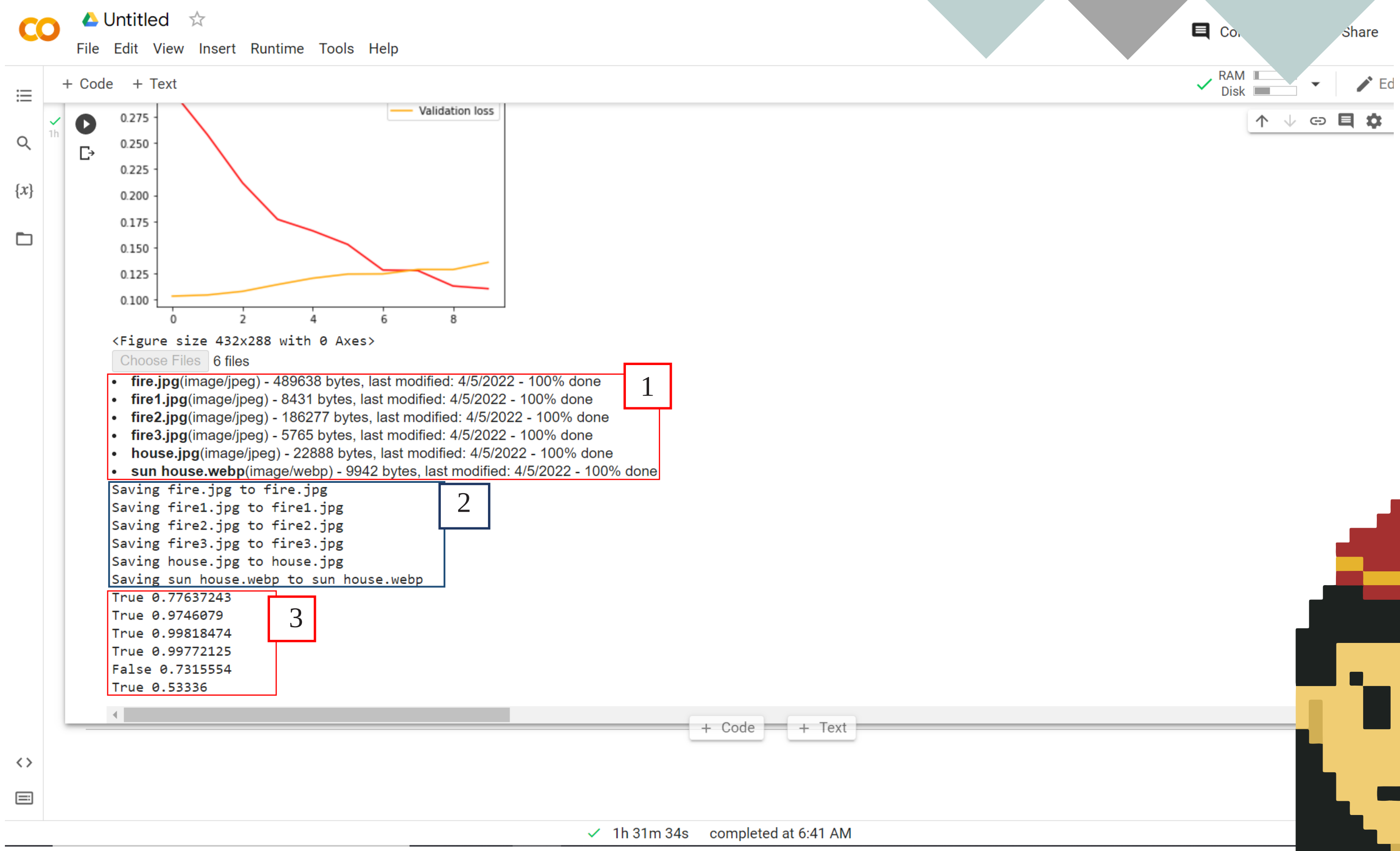
Output explanation:

We try to test the system by uploading 6 images from already downloading image on the PC. Four images contains fire, only one image of them contains house(not contain fire) and another one contain sun and house but gives error result; It ambiguous as for compiler because contain have sun.



Output explanation:

1- uploading done for 6 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 image 'sun house' was confusing the system though it fire. So just this image prediction was incorrect . Note that the number near to the result is the predict result percentage.



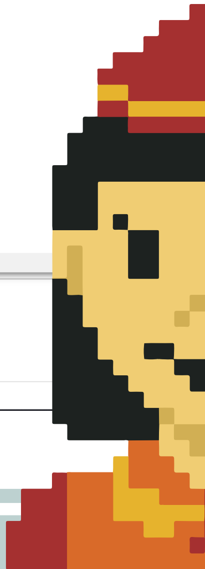
```
<Figure size 432x288 with 0 Axes>
```

Choose Files 6 files

- **fire.jpg**(image/jpeg) - 489638 bytes, last modified: 4/5/2022 - 100% done
- **fire1.jpg**(image/jpeg) - 8431 bytes, last modified: 4/5/2022 - 100% done
- **fire2.jpg**(image/jpeg) - 186277 bytes, last modified: 4/5/2022 - 100% done
- **fire3.jpg**(image/jpeg) - 5765 bytes, last modified: 4/5/2022 - 100% done
- **house.jpg**(image/jpeg) - 22888 bytes, last modified: 4/5/2022 - 100% done
- **sun house.webp**(image/webp) - 9942 bytes, last modified: 4/5/2022 - 100% done

```
Saving fire.jpg to fire.jpg
Saving fire1.jpg to fire1.jpg
Saving fire2.jpg to fire2.jpg
Saving fire3.jpg to fire3.jpg
Saving house.jpg to house.jpg
Saving sun house.webp to sun house.webp
True 0.77637243
True 0.9746079
True 0.99818474
True 0.99772125
False 0.7315554
True 0.53336
```

✓ 1h 31m 34s completed at 6:41 AM



Conclusion :

Our system came with different methods for identifying the target, distinguishing and controlling movement. It has a high computational ability to identify the fire, which enables the detection of fires quickly and early.
We try to collect as more as we can 1800 images for training and 200 images for validation. In order to make the training more accurate we used data augmentation techniques.

In this satiation , it is very useful for emergency alarm , in order to avoid disasters.
The fires caused huge human and economic damages.
We did our best to let the system identify fires with high accuracy in different environments, whether indoors or outdoors.

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

Look at this picture:

before

after

Indeed, this explosion was in the port of Beirut, which caused great human and economic losses.

Back