MACHINE LEARNING AND DEEP LEARNING

EEPE – 34

FIRE DETECTION

GROUP - E

TEAM MEMBERS:

1. Pranav Kumar A V - 107119091
2. Muhilan R - 107119073
3. Prethivan B - 107119093
4. Dinesh Kumar P – 107119087

Introduction:

Fire is a quintessential part of life on earth and is one of the very basic elements of survival. But at the same time fire can also be dangerous and deadly when not handled with care. These are the instances that lead to fire accidents.

According to NCRB, an average of 35 Indians die from fire accidents every day. and according to the statistics from U.S. Fire Administration 14.8 billion dollors is lost due to fire accidents in U.S in 2020.

When we take the place of occurance of fire accidents, it can happen anywhere ranging from residential, commercial places, Factories, mines, vechicles etc.. Forest fire has become a huge problem for many contries these days. As fire accidents can happen this often and in these many places, prevention becomes a essential thing.

AIM:

Our deep learning project 'Fire Detection' aims at early detection of fire during fire accidents. Early detection of fire during fire accidents plays a vital role in alearting people early and extinguishing fire at initial stages.

Thus our project aims at preventing both life and property loss during fire accidents by early fire detection. Our project 'Fire Detection' can be applicable for both indoor and outdoor fire accident detection.

Overview:

Our Project 'Fire Detection' works with the help of image recognition technique.

We are processing the image recognition with CNN (Convolution Neural Networks) which we learned in Deep learning.

Theory:

Convolutional neural network(CNN) is one the main algorithms to perform image recognition and classification. Face recognition, object detection, scene labeling, etc are the application area of convolutional neural network. CNN takes an image as input, which is classified and it process it under a particular category. CNN recognizes an image as an array of pixel and depending on the resolution of the image, it will see it as h*w*d, where h = height, w = width, d = dimension. For example, CNN recognizes an RGB image as 6 * 6 * 3 array of the matrix. In CNN, an input image goes through the sequence of convolutional layers with pooling, padding, fully connected layers, filters, etc. After that we appy softmax function to classify it as 0 and 1.

Convolutional Layer:

It is the first layer to extract the important features from the input image. By learning image features using small square of input data and preserves the relationship between the pixels. It is the mathematical operation, where it takes two matrices as input, image matrix and kernel or filter matrix. Convolution of an image with different filters enables us to perform operations like blur, sharpen, edge detection etc.

Strides:

It is the number of pixel we have to shift over input image. When the stride is equaled to 1, then we move the filters to 1 pixel at a time.

Padding:

It is one the most important process in convolutional neural network. While applying filters, CNN covers the middle region more than once, but the corners are convered only once. It means we have more information of the middle pixels. Disadvantages of this are, we will lose information of the corners of the image and the output will be shrinking. So to overcome this we add additional layers of zeros to the corners of the image, it is called padding.

Pooling:

It is the important process in pre-processing of an image. It reduces the number of parameters when the image is too large. It can be called as downscaling of the image obtained from previous layers and similiar to shrinking of an image to reduce its pixel density. Spatial pooling is also called downsampling or subsampling, which reduces the dimensionality of each map but retains the important information. There are the following types of spatial pooling:

1. **Max Pooling:**

   Max pooling is a sample-based discretization process. Its main objective is to downscale an input representation, reducing its dimensionality and allowing for the assumption to be made about features contained in the sub-region binned. Max pooling is done by applying a max filter to non-overlapping sub-regions of the initial representation.

2. **Average Pooling:**

   Down-scaling will perform through average pooling by dividing the input into rectangular pooling regions and computing the average values of each region.

3. **Sum Pooling:**

   The sub-region for sum pooling are set exactly the same as for max-pooling but instead of using the max function, we use sum.

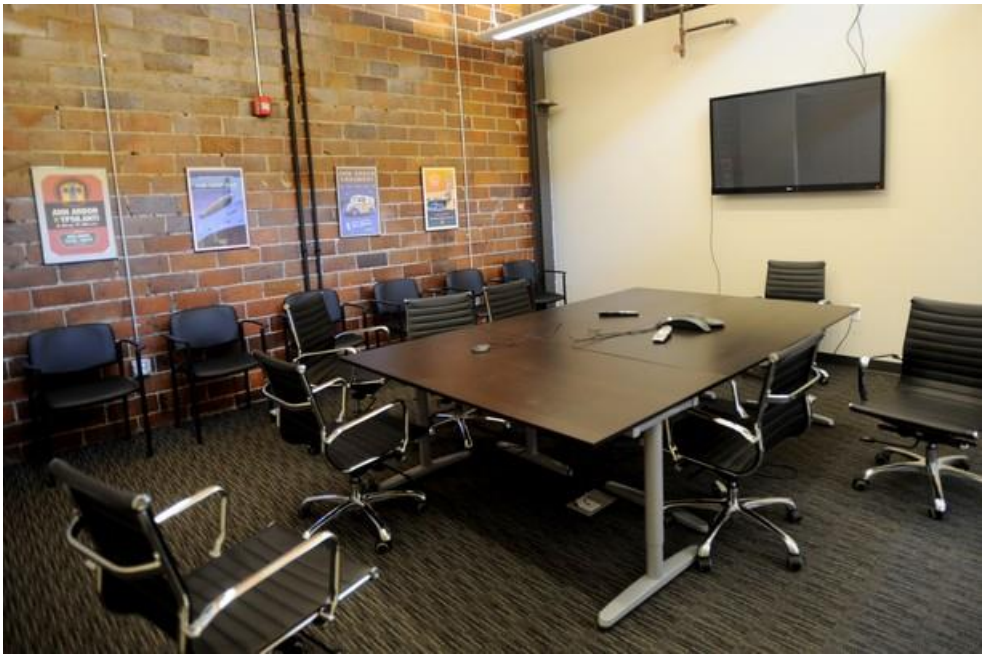
Fully Connected Layers:

The fully connected layer is a layer in which the input from the other layers will be flattened into a vector and sent. It will transform the output into the desired number of classes by the network. For example, the feature map matrix will be converted into the vector such as x1, x2, x3... xn with the help of fully connected layers. We will combine features to create a model and apply the activation function such as softmax or sigmoid to classify the outputs into a particular category.

Data:

We used images as data for this project. We totally had 2000 images as data, in which 1400 set of images are for training and another 600 set of images for testing. In those 1400 images, 700 fire and 700 no fire, and in 600, 300 are fire and 300 are no fire.

Sample images:



Code :

```
# Importing all the required libraries
import os
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
from keras.models import model_from_json
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

# Importing the file from my google drive
!gdown --id 1hWZmR9Wkbm5_DijpzUp8mDmCn1nK-qfi

# Unzipping the Imported file
!unzip Fire_Detection_Dataset.zip

TRAINING_DIR = "/content/Fire_Detection_Dataset/Train"
VALIDATION_DIR = "/content/Fire_Detection_Dataset/Test"

# Calculating the length of Training and Validation dir
print(len(os.listdir(TRAINING_DIR+"/Fire")))
print(len(os.listdir(TRAINING_DIR+"/No_Fire")))
print(len(os.listdir(VALIDATION_DIR+"/Fire")))
print(len(os.listdir(VALIDATION_DIR+"/No_Fire")))

# Creating the model
model = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=(224, 224, 3)),
tf.keras.layers.MaxPooling2D(pool_size = (3,3), strides=(2,2)),
tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size = (2,2), strides=(2,2)),
tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size = (2,2), strides=(2,2)),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size = (2,2), strides=(2,2)),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size = (2,2), strides=(2,2)),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(256, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(2, activation='softmax')])
model.compile(loss='categorical_crossentropy',
optimizer=Adam(lr=0.0001),                    # Using Adam optimization
metrics=['acc'])

model.summary()                    # Printing the summary of the model

# Data Augmentation
training_datagen = ImageDataGenerator(rescale = 1./255,
                                      horizontal_flip=True,
                                      rotation_range=30,
                                      height_shift_range=0.2,
                                      fill_mode='nearest')
```

```
validation_datagen = ImageDataGenerator(rescale = 1./255)
train_generator = training_datagen.flow_from_directory(TRAINING_DIR,
                                  target_size=(224,224),
                                  class_mode='categorical',
                                  batch_size = 64)
validation_generator = validation_datagen.flow_from_directory(
                                    VALIDATION_DIR,
                                    target_size=(224,224),
                                    class_mode='categorical',
                                    batch_size= 16)

# Creating checkpoint to save the best model
checkpoint_path = "training_1/cp.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Creating Callback funtion and saving only the best model
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_path,
    save_weights_only=True,
    monitor='val_acc',
    mode='max',
    save_best_only=True,
    verbose=1)

# Training the model
history = model.fit(
train_generator,
steps_per_epoch = 15,
epochs = 25,
validation_data = validation_generator,
validation_steps = 15,
callbacks=[model_checkpoint_callback])

#Plotting the graphs
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, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()

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

plt.show()

# Importing Images from Laptop to detect the image by our model
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

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

```
# predicting images
  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)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)
  print(classes)
  if classes[0][0]>0.5:
    print(fn + " Fire")
  else:
    print(fn + " No Fire")

# Saving the weights in our Laptop
model.save("Fire_Detection_Weights.h5")
print("Saved model to disk")
```

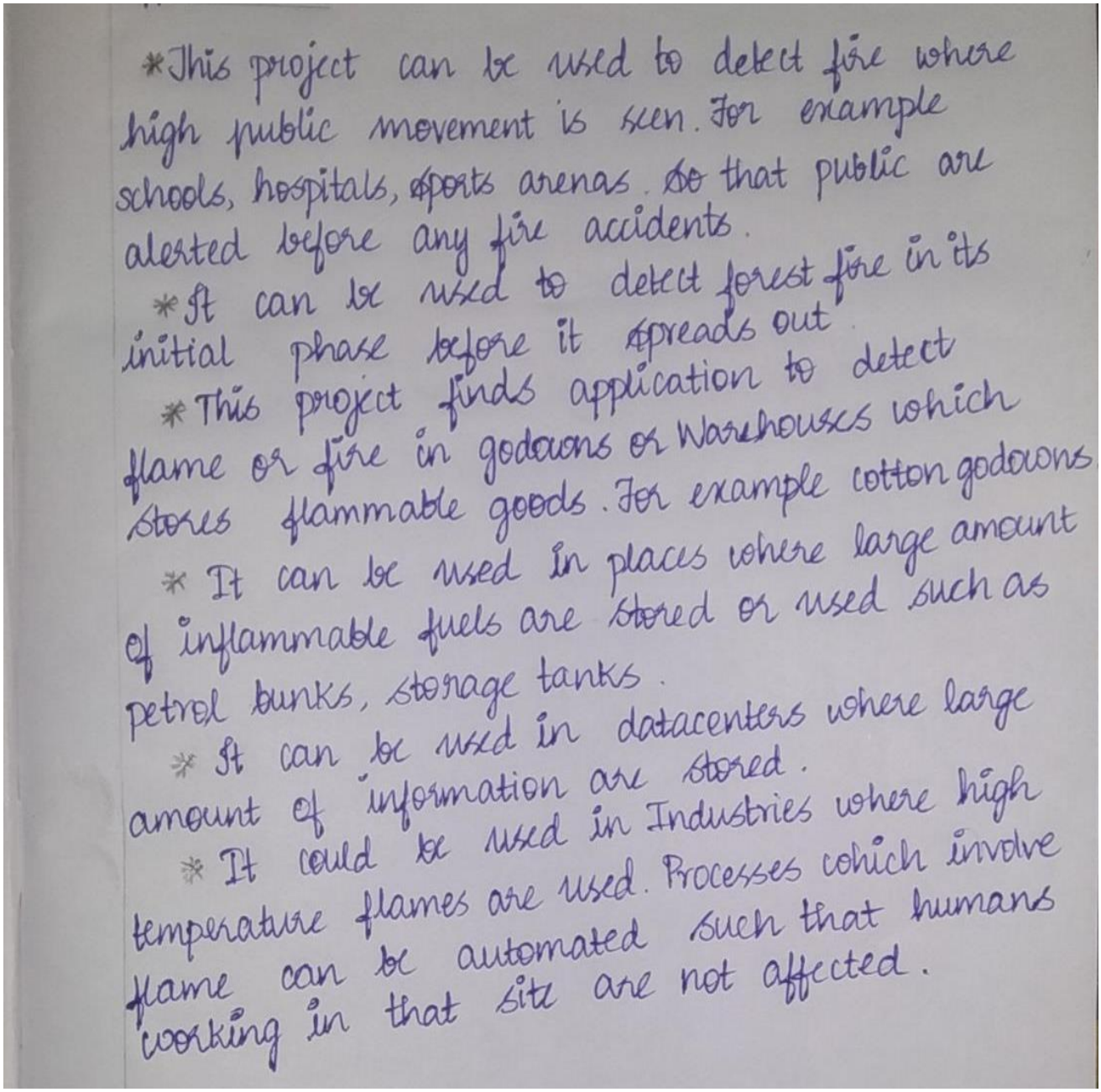
Loading our model's trained weight to classify image as fire or no fire:

```
# Importing all the required libraries
import os
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
from keras.models import load_model


# Importing the model weights from my google drive
!gdown --id 18doFyrFhfZHbSLGsSS2Ly6FahjfkR0mO


# Creating the model
model = tf.keras.models.Sequential([
tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=(224, 224, 3)),
tf.keras.layers.MaxPooling2D(pool_size = (3,3), strides=(2,2)),
tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size = (2,2), strides=(2,2)),
tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size = (2,2), strides=(2,2)),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size = (2,2), strides=(2,2)),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D(pool_size = (2,2), strides=(2,2)),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(256, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(2, activation='softmax')])
model.compile(loss='categorical_crossentropy',
optimizer=Adam(lr=0.0001),                    # Using Adam optimization
metrics=['acc'])
```

```
# Load the weights
model = load_model('Fire_Detection_Weights.h5')


# Importing Images from Laptop to detect the image by our model
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  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)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)
  print(classes)
  if classes[0][0]>0.5:
    print(fn + " Fire")
  else:
    print(fn + " No Fire")
```

Applications:

Thus, we can see that the project has many applications. Providing further developments would make the project much more useful in practical applications. This project can be developed in such a way that the applications given below can be implemented successfully with high efficiency.



Conclusion:

Developed to detect fires promptly as they grow, fire detection systems are an extremely valuable fire protection tool for any establishment. Early detection is crucial in the protection of first responders and other emergency response personnel as they combat fires.

With fire detection systems, appropriately loss can also be significantly reduced. Downtime for business operations can be decreased. Fire control efforts can get started while the fire is not massive yet.

Fires can spread very quickly. Before you know it, a small flame grows into a massive blaze in minutes. Thus, commercial establishments, even residential properties, should have a proper emergency plan, including incorporating fire safety and protection protocol.
While fires can generally spread quickly, some other fires start because of heavy periods of dormant fires. In such cases, highly sensitive fire detection systems must be in place.

By detecting a fire quickly and accurately (i.e., by not sacrificing speed or causing false alarms) and providing early warning notification, a fire-detection system can limit the emission of toxic products created by combustion, as well as global-warming gases produced by the fire itself.
However, when you look at the bigger picture, nothing is more important than securing the safety of public, Goods during an emergency. Reduction of economic and human losses are given higher priority.

By that way, this project would be extremely helpful in fire detection and could be well processed to detect fire at larger extent.