COMP 7745 – Machine learning

Final Project

Fire and Gun Violence based Anomaly Detection System Using Deep Neural Networks

Meghana Bathula- U00866246

Navya Dontham- U00874304

Fall 2022

1. Introduction:

The main idea of our project is to create a system that monitors surveillance data of an area and sends alerts in case a fire or gun is detected. Usually, CCTV footages record the neighborhood while a person watches over them. It takes a lot of manpower to do the surveillance of every area in the country. The data released from CDC shows that there were around 49,000 people who died due to gunshot violence and 338,000 fires reported in the United States in the year 2021. We are using YOLO (You Only Look Once) method to do object detection which uses convolutional neural network.

2. Methodology:

I. Preprocessing:

The dataset was combined and prepared from multiple sources which contains around 4000 images of fire:

data: this folder contains two sub folders with fire and no fire

- fire: this folder contains fire images
- no fire: this folder contains no fi images.

The images were normalized by dividing the RGB values by 255 which is the max RGB value – minimum RGB value to normalize the pixel values between 0 and 1 to ensures that each input pixel has a similar data distribution. Data augmentation techniques such as horizontal flip, zoom range (0.2) were also applied to increase the number of images in dataset.

II. Model Building:

The model was constructed using keras. It uses transfer learning technique in which a The Darknet53 architecture which were loaded from keras library using pretrained weight of image net, was treated as a layer and was fine-tuned when coding. All layers in the Darknet53 were fixed except the last 4 layers which were retrained during the training process. A flatten layer with RELU activation and a 0.5 dropout layer were added into the model to fit the new classes of the problem. Sigmoid activation was applied for classification layer.The models were compiled with Adam optimizer and binary cross entropy loss function. Adam optimizer was set with learning rate of 0.0001. A default "accuracy" metric was also used for evaluated during training and testing.

III. <u>Training and validation:</u>

For training, a batch size of 16 and 50 epochs were chosen. Model Checkpoint callback which saved model weights if the training accuracy of the model had increased from the previous epoch was used to help during the training process. The model was trained using multiclass classification with Guns are represented by label 0 and Fire by label 1

After model is built run the yolo python program by using the below commands:

python yolo.py --webcam True – for testing webcam or cctv footages

python yolo.py --play_video True --video_path videos/firesample.mp4 – for testing videos

python yolo.py --image True --image_path images/fire_1.jpg – for testing images

3. Model description:

You Only Look Once algorithm (YOLO):

YOLO algorithm is known for its accuracy and processing speed. It uses one of the best neural network architectures which is based on regression. In one run it can predict the classes and bounding boxes. This is mostly used in various applications to detect traffic signals, parking meters and animals. YOLO algorithm firstly takes an input image of shape. Then, it passes this image to convolution neural network which returns a dimensional output. This is passed to get the final output using flattening, dropout, and other methods.

4. Experiment and Results

a. Database:

The model is first trained with the database containing fire and no fire images. Then we test the model using a custom dataset has been created as there was no dataset for images of guns from a CCTV perspective. Therefore, 6-gun images were collected with various angles, many of which are CCTV images of humans with a gun. Along with this we have 5 fire images that are taken from google. Our dataset also includes 1 video containing gun and fire to test the performance of our model on video.

b. Training and testing logs:

Model Building

In [1]:
```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as BK
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyStopping
```

In [2]:
```
# dimensions of our images.
img_width, img_height = 200, 200

train_data_path = 'data/train'
validation_data_path = 'data/validation'
nb_train_samples = 1914
nb_validation_samples = 182
batch_size = 16
epochs = 50


if BK.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)
```

In [3]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('sigmoid'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

In [4]:

```
train_data_gen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)


test_data_gen = ImageDataGenerator(rescale=1. / 255)

train_generator = train_data_gen.flow_from_directory(
    train_data_path,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary')

validation_generator = test_data_gen.flow_from_directory(
    validation_data_path,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='binary')

checkpoint = ModelCheckpoint("first1_cp.h5", monitor='val_acc', verbose=1, save_best_only=True, save_weights_only=False, mode

history = model.fit(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size)

model.save_weights('gun_fire.h5')
```

```
Found 1914 images belonging to 2 classes.
Found 182 images belonging to 2 classes.
Epoch 1/50
119/119 [==============================] - 91s 750ms/step - loss: 0.6266 - accuracy: 0.6628 - val_loss: 0.4277 - val_ac
curacy: 0.8125
Epoch 2/50
119/119 [==============================] - 62s 522ms/step - loss: 0.4028 - accuracy: 0.8246 - val_loss: 0.3204 - val_ac
curacy: 0.9148
Epoch 3/50
119/119 [==============================] - 63s 532ms/step - loss: 0.3499 - accuracy: 0.8641 - val_loss: 0.2789 - val_ac
curacy: 0.9148
Epoch 4/50
119/119 [==============================] - 61s 508ms/step - loss: 0.3373 - accuracy: 0.8530 - val_loss: 0.3069 - val_ac
curacy: 0.8523
Epoch 5/50
119/119 [==============================] - 61s 507ms/step - loss: 0.3260 - accuracy: 0.8593 - val_loss: 0.3020 - val_ac
curacy: 0.9091
Epoch 6/50
119/119 [==============================] - 61s 510ms/step - loss: 0.3219 - accuracy: 0.8656 - val_loss: 0.3022 - val_ac
```

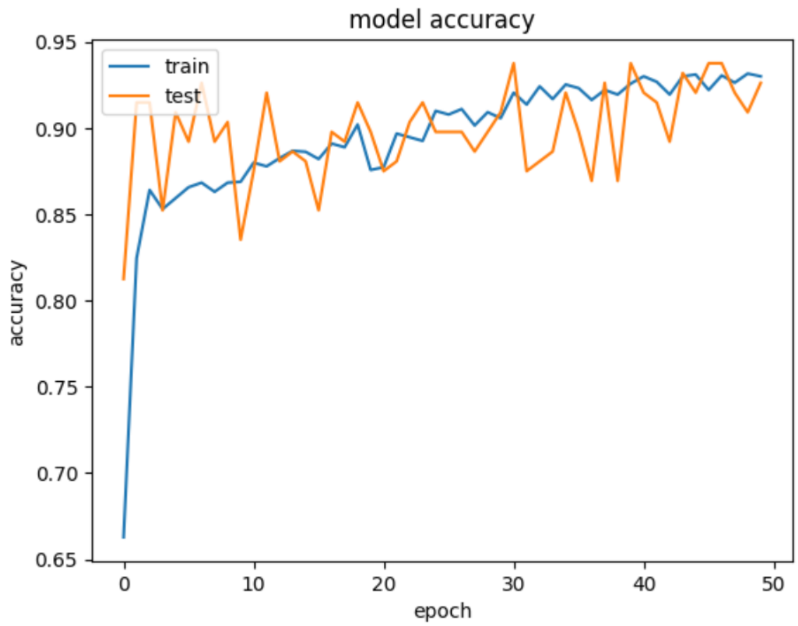
In [5]:

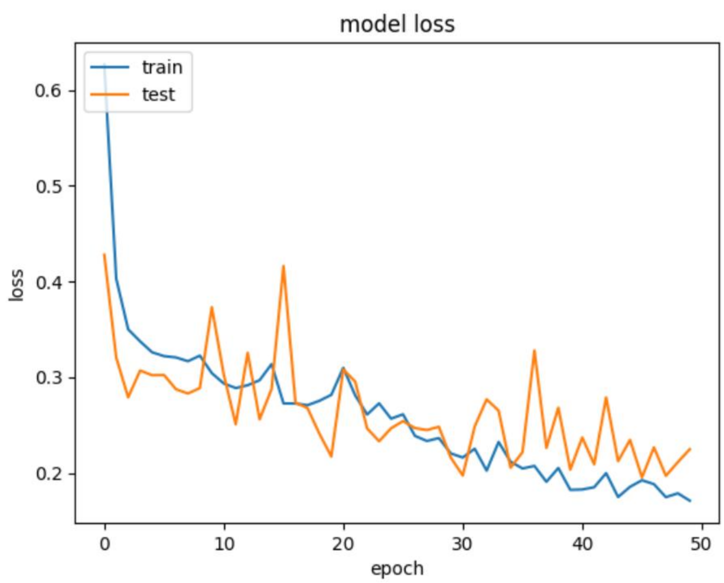
```
model.save('firstimplementation.h5')
```

In [7]:

```
import matplotlib.pyplot as plt
%matplotlib inline
# list all data in present in history
print(history.history.keys())
# get accuracy graph
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# get loss graph
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```


model accuracy
train
test
accuracy
epoch
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
10
20
30
40
50


model loss
train
test
loss
epoch
0.6
0.5
0.4
0.3
0.2
0
10
20
30
40
50

c. Discussion and comparison:

The model became stable after about 22 epochs. On training data, the model accuracy reached 94.16%, loss was ~0.156. On validation data, the accuracy reached ~89.77%, loss was 0.22. The model showed stable results

5. Conclusion

In this project, a real-time frame-based efficient fire and gun detection deep learning model has been presented with a accuracy of 94.16%.

6. References

- T. Celik, H. Demirel, H. Ozkaramanli and M. Uyguroglu, "Fire Detection in Video Sequences Using Statistical Color Model," 2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings, Toulouse, 2006, pp. II-II.
-] B. U. Toreyin, Y. Dedeoglu and A. E. Cetin, "Flame detection in video using hidden Markov models," IEEE International Conference on Image Processing 2005, Genova, 2005, pp. II-1230.

- Z. Li, S. Nadon and J. Cihlar “Satellite-based detection of Canadian boreal forest fires: Development and application of the algorithm,” International Journal of Remote Sensing, vol. 21, no.16, pp. 3057-3069, 2000.
- T. J. Lynham, C. W. Dull and A. Singh, "Requirements for space-based observations in fire management: a report by the Wildland Fire Hazard Team, Committee on Earth Observation Satellites (CEOS) Disaster Management Support Group (DMSG)," IEEE International Geoscience and Remote Sensing Symposium, Toronto, Ontario, Canada, 2002, pp. 762-764 vol.2
- M. T. Basu, R. Karthik, J. Mahitha, and V. L. Reddy, “IoT based forest fire detection system,” International Journal of Engineering & Technology, vol. 7, no. 2.7, p. 124, 2018.
- T.Celik, H.Demirel, H.Ozkaramanli, “Fire and Smoke Detection without Sensors: Image Processing Based Approach,” Proceedings of 15th European Signal Processing Conference, Poland, September 3-7, 2007.