14/08/2024

ACCIDENT DETECTION USING DEEP LEARNING

Team Member:

Haya Almalki

Jehan Almutairi

Hanan Mohammed

INTRODUCTION

Traffic congestion is one of the problems in cities, and with the advancement of technology, it has become easy to find solutions that help to eliminate this problem. In this mini project, we focus on detecting one of the most common causes of traffic jams, which is accidents. We created deep learning models that can detect the accident through images. We implemented Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN). We chose these two models because they are the most common models in deep learning. We evaluated the performance of these models, compared the accuracy results of the two models, and then improved the best model. Our project aims to improve traffic flow by using advanced technologies. In the next sections, we will explain the implementation in detail.

DATASET

We uploaded the dataset from Kaggle called (**Accident Detection From CCTV Footage**), which contains images taken from YouTube videos that show accidents.

It contains three folders of images as follows:

1- Training folder (791 images) ~80%.
2- Validation folder (98 images) ~10%.
3- Testing folder (100 images) ~10%.

The dataset has two labels: accidents and non-accidents. The next is the sample of the dataset:

DATA PREPROCESSING

First, we are using ImageDataGenertor(), which is a class containing methods for preprocessing and augmenting images, and we implemented the following:

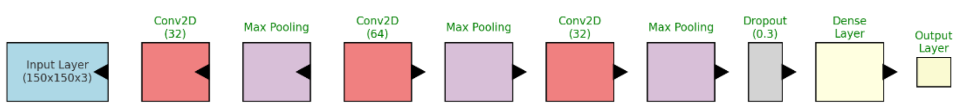
- ❑ Normalization by dividing by 255.
- ❑ Rotation images.
- ❑ Image resizing to 150x150 pixels.

```
datagen = ImageDataGenerator(rescale=1./255, rotation_range=20)
```

```
# Here is for Load and
train_generator = datagen.flow_from_directory('/content/accident-detection-from-cctv-footage/data/train', target_size=(150,150), batch_size=32, class_mode='binary')
validation_generator = datagen.flow_from_directory('/content/accident-detection-from-cctv-footage/data/val', target_size=(150,150), batch_size=32, class_mode='binary')
testing_generator = datagen.flow_from_directory('/content/accident-detection-from-cctv-footage/data/test', target_size=(150,150), batch_size=32, class_mode='binary')
```

METHOD:

First, we uploaded the dataset and preprocessed it. Next, we build ANN and CNN models to compare their performances. Then, we take the better model to improve it by adding layers and implementing methods to enhance its accuracy. The next shows the architecture of the improved CNN model that we used in this project.



ANN MODEL

We created the ANN model using the following code:

```
# Here is the define the model
ANN_model = Sequential()

# Here is the input layer with flatten the image to 1D
ANN_model.add(Flatten(input_shape=(150, 150, 3)))

# Here is the hidden layers
ANN_model.add(Dense(units=128, activation='relu'))
ANN_model.add(Dense(units=64, activation='relu'))

# Here is the output layer
ANN_model.add(Dense(units=1, activation='sigmoid'))
```

Then, we compile the model by determining the optimizer, loss, and metrics.

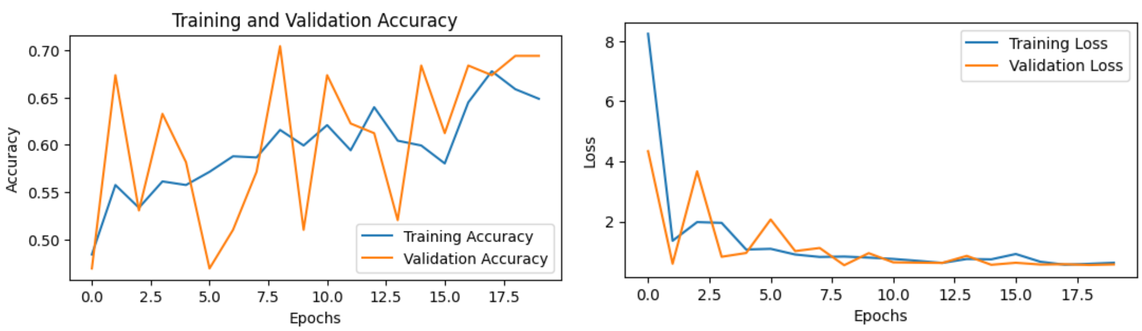
```
[16] CNN_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

After that, we train the model as the following:

```
ANN_history = ANN_model.fit(train_generator, epochs=20, validation_data=validation_generator)
```

```
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `Py
  self._warn_if_super_not_called()
25/25 ━━━━━━━━━━━━━━━━━━━━ 36s 1s/step - accuracy: 0.4953 - loss: 9.9755 - val_accuracy: 0.4694 - val_loss: 4.3404
Epoch 2/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 22s 522ms/step - accuracy: 0.5349 - loss: 2.0185 - val_accuracy: 0.6735 - val_loss: 0.6021
Epoch 3/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 21s 521ms/step - accuracy: 0.5522 - loss: 1.2754 - val_accuracy: 0.5306 - val_loss: 3.6714
Epoch 4/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 580ms/step - accuracy: 0.5461 - loss: 2.5818 - val_accuracy: 0.6327 - val_loss: 0.8329
Epoch 5/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 519ms/step - accuracy: 0.5614 - loss: 1.2172 - val_accuracy: 0.5816 - val_loss: 0.9589
Epoch 6/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 535ms/step - accuracy: 0.6021 - loss: 0.9977 - val_accuracy: 0.4694 - val_loss: 2.0709
Epoch 7/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 18s 523ms/step - accuracy: 0.5919 - loss: 1.0966 - val_accuracy: 0.5102 - val_loss: 1.0219
Epoch 8/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 19s 515ms/step - accuracy: 0.5915 - loss: 0.8057 - val_accuracy: 0.5714 - val_loss: 1.1248
Epoch 9/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 21s 508ms/step - accuracy: 0.5923 - loss: 0.9917 - val_accuracy: 0.7041 - val_loss: 0.5528
Epoch 10/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 21s 559ms/step - accuracy: 0.6531 - loss: 0.6742 - val_accuracy: 0.5102 - val_loss: 0.9528
Epoch 11/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 531ms/step - accuracy: 0.6062 - loss: 0.8202 - val_accuracy: 0.6735 - val_loss: 0.6481
Epoch 12/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 17s 514ms/step - accuracy: 0.5994 - loss: 0.7260 - val_accuracy: 0.6224 - val_loss: 0.6381
Epoch 13/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 520ms/step - accuracy: 0.6255 - loss: 0.6286 - val_accuracy: 0.6122 - val_loss: 0.6288
Epoch 14/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 504ms/step - accuracy: 0.5857 - loss: 0.8350 - val_accuracy: 0.5204 - val_loss: 0.8634
Epoch 15/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 507ms/step - accuracy: 0.6283 - loss: 0.6655 - val_accuracy: 0.6837 - val_loss: 0.5666
Epoch 16/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 16s 459ms/step - accuracy: 0.6002 - loss: 0.8898 - val_accuracy: 0.6122 - val_loss: 0.6340
Epoch 17/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 512ms/step - accuracy: 0.6458 - loss: 0.6887 - val_accuracy: 0.6837 - val_loss: 0.5757
Epoch 18/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 21s 486ms/step - accuracy: 0.6864 - loss: 0.5669 - val_accuracy: 0.6735 - val_loss: 0.5850
Epoch 19/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 20s 527ms/step - accuracy: 0.6763 - loss: 0.5778 - val_accuracy: 0.6939 - val_loss: 0.5606
Epoch 20/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 15s 487ms/step - accuracy: 0.6762 - loss: 0.5917 - val_accuracy: 0.6939 - val_loss: 0.5792
```

We got the result of accuracy and loss for training and validation accuracy as shown in the next chart.



CNN MODEL

We created the CNN model using the following code:

```
[15] # Build the model
     CNN_model = Sequential([
         Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
         MaxPooling2D(2, 2),
         Conv2D(64, (3, 3), activation='relu'),
         MaxPooling2D(2, 2),
         Flatten(),
         Dense(128, activation='relu'),
         Dense(1, activation='sigmoid')
     ])
     CNN_model.summary()
```

Then, we compile the model by determining the optimizer, loss, and metrics.

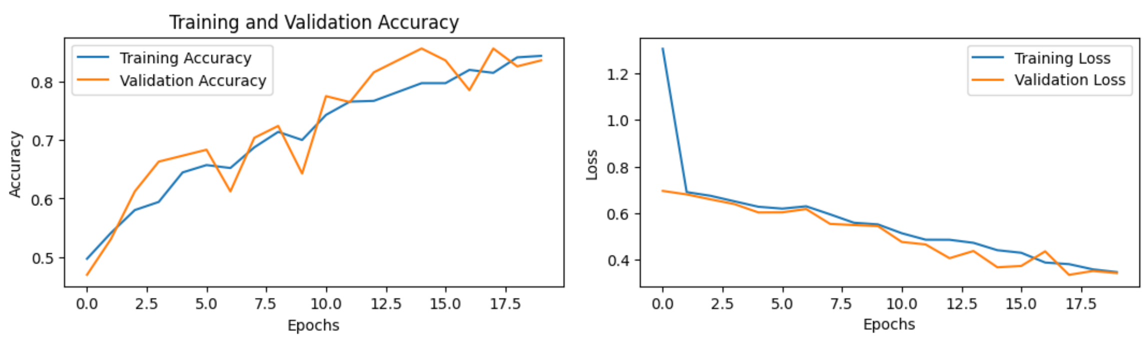
```
[16] CNN_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

After that, we train the model as the following:

```
[17] CNN_history = CNN_model.fit(train_generator, epochs=20, validation_data=validation_generator)

Epoch 1/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 48s 2s/step - accuracy: 0.4862 - loss: 2.1731 - val_accuracy: 0.4694 - val_loss: 0.6961
Epoch 2/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.5054 - loss: 0.6942 - val_accuracy: 0.5306 - val_loss: 0.6811
Epoch 3/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.5710 - loss: 0.6811 - val_accuracy: 0.6122 - val_loss: 0.6599
Epoch 4/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 42s 1s/step - accuracy: 0.6103 - loss: 0.6489 - val_accuracy: 0.6633 - val_loss: 0.6396
Epoch 5/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.6452 - loss: 0.6260 - val_accuracy: 0.6735 - val_loss: 0.6038
Epoch 6/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.6502 - loss: 0.6235 - val_accuracy: 0.6837 - val_loss: 0.6042
Epoch 7/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.6684 - loss: 0.6150 - val_accuracy: 0.6122 - val_loss: 0.6182
Epoch 8/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 83s 1s/step - accuracy: 0.6878 - loss: 0.6138 - val_accuracy: 0.7041 - val_loss: 0.5547
Epoch 9/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.7164 - loss: 0.5508 - val_accuracy: 0.7245 - val_loss: 0.5497
Epoch 10/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.7287 - loss: 0.5274 - val_accuracy: 0.6429 - val_loss: 0.5454
Epoch 11/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.7416 - loss: 0.5155 - val_accuracy: 0.7755 - val_loss: 0.4773
Epoch 12/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.7724 - loss: 0.4911 - val_accuracy: 0.7653 - val_loss: 0.4665
Epoch 13/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 81s 1s/step - accuracy: 0.7655 - loss: 0.4898 - val_accuracy: 0.8163 - val_loss: 0.4075
Epoch 14/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.7929 - loss: 0.4649 - val_accuracy: 0.8367 - val_loss: 0.4384
Epoch 15/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.8094 - loss: 0.4411 - val_accuracy: 0.8571 - val_loss: 0.3686
Epoch 16/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.8086 - loss: 0.4158 - val_accuracy: 0.8367 - val_loss: 0.3743
Epoch 17/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.8217 - loss: 0.4012 - val_accuracy: 0.7857 - val_loss: 0.4370
Epoch 18/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 39s 1s/step - accuracy: 0.8060 - loss: 0.3924 - val_accuracy: 0.8571 - val_loss: 0.3362
Epoch 19/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.8374 - loss: 0.3553 - val_accuracy: 0.8265 - val_loss: 0.3529
Epoch 20/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.8634 - loss: 0.3336 - val_accuracy: 0.8367 - val_loss: 0.3438
```

We got the results of accuracy and loss for training and validation are shown in the next chart.

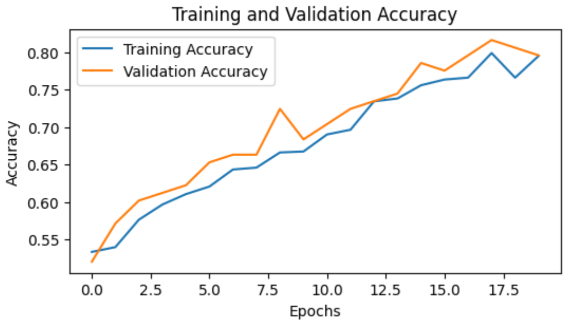


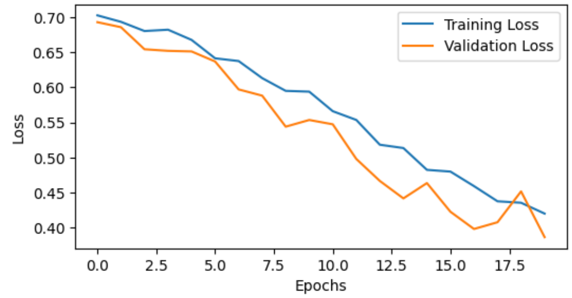
RESULT

The accuracy of the ANN model is (66%), and the accuracy of the CNN is (81%). We tried to improve the accuracy of the CNN model by adding more layers and Dropout(0.3) using the following code:

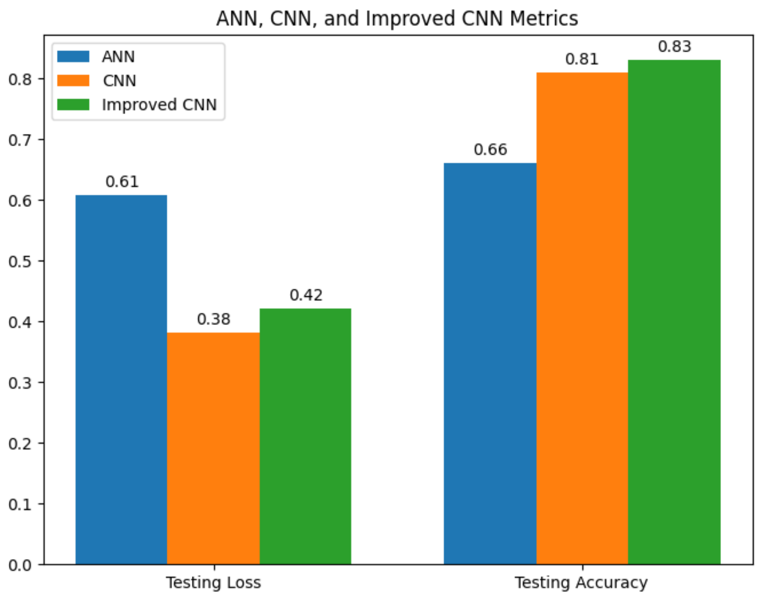
```
] # Build the model
  CNN_model_improvment = Sequential([
      Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
      MaxPooling2D(2, 2),
      Conv2D(64, (3, 3), activation='relu'),
      MaxPooling2D(2, 2),
      Conv2D(32, (3, 3), activation='relu'),
      MaxPooling2D(2, 2),
      Flatten(),
      Dense(128, activation='relu'),
      Dropout(0.3),
      Dense(1, activation='sigmoid')
  ])
  CNN_model_improvment.summary()
```

We got the results of accuracy and loss for training and validation are shown in the next chart.

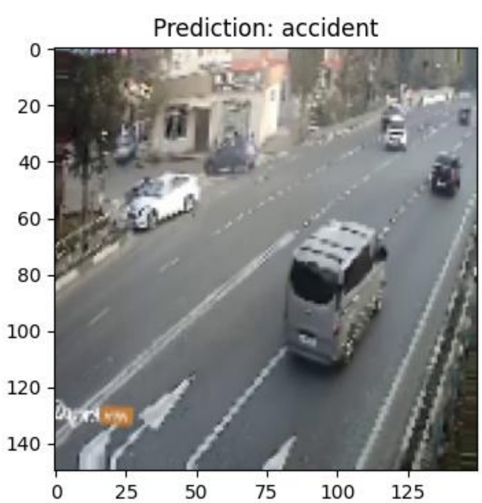




The improved model accuracy is (83%). The next chart shows the comparison of the accuracy of the three models.



We gave the improvement model images to make predictions, and the following is the result of one image.



Accuracy_Val (CNN IM)	Accuracy_Test (CNN IM)	Accuracy_Val (CNN)	Accuracy_Test (CNN)	Accuracy_Test (ANN)	Accuracy_Val (ANN)	
0.98	0.93	0.88	0.93	0.75	0.74	Run 1
0.85	0.81	0.79	0.81	0.66	0.69	Run 2
0.8	0.83	0.83	0.81	0.69	0.7	Run 3
0.876666667	0.856666667	0.833333333	0.85	0.7	0.71	AVG

The following table shows the average of accuracies when running the code three times:

CONCLUSION

In conclusion, using deep learning techniques provides a strong alternative to traditional methods. Deep learning models can effectively detect traffic accidents from images. It can be used in traffic systems, leading to reduce the waiting time for Najm to arrive and report when accidents happen. In the future, we will seek to train the model in the best way and improve its accuracy to achieve optimal results. We will try improving accuracy by adding extra data augmentation, adding more layers to the model, and using larger datasets. Also, we will implement advanced models like VGG-16, and desnet50.

Group Task:

Member	Tasks
Haya Almalki	Coding, and report.
Jehan Almutairi	Coding, and presentation.
Hanan Mohammed	Coding, and creating notebook and layout.