CMSC 691 Computer Vision, Spring 2019.

Nipun Ramagiri, nipunr1@umbc.edu
Ashish Kumar, akumar5@umbc.edu

PROJECT:
Real or Fake: Live Face Detection
Project Report

Introduction

Proposal

Biometric recognition systems are continuing to be integrated as a part of security features across various fields to identify individuals. The subset of Facial recognition systems is becoming very popular due to the latest advancements in computer vision and machine learning, especially deep learning, but at the same time they are being attacked by spoofing a person's identity to trick the system for access that should be denied.

A spoof attack, a type of presentation attack, is the use of an artificial replica of a biometric used to circumvent a system. "Liveness detection" is a method used to recognize a presentation attack. In our case these attacks may be in the form of a printed image on paper or on a screen, a video of the person used to imitate a liveness etc.

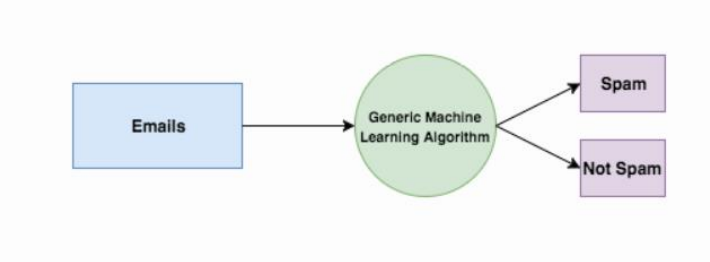
We propose to construct a liveness detector using a Deep Neural Network with convolutional layers by treating facial detection as a binary classification problem, i.e. a real or fake face being shown to our system so that we can mitigate such simple spoofs.

Brief Introduction to Machine Learning and Deep Learning

Arthur Samuel said, "Machine Learning is the ability to learn without being explicitly programmed." It is a subset of AI that can be classified into three ways of being implemented: Supervised, Unsupervised and Reinforcement Learning. In our case we use Supervised Learning, so we'll be describing it in further detail.

In supervised learning, the system tries to learn from the previous examples that are given. (On the other hand, in unsupervised learning, the system attempts to find the patterns directly from the example given.)

Speaking mathematically, supervised learning is where you have both input variables (x) and output variables(Y) and can use an algorithm to derive the mapping function from the input to the output. The mapping function is expressed as Y = f(X).



Supervised learning problems can be further divided into two parts, namely classification, and regression.
Classification: A classification problem is when the output variable is a category or a group, such as "black" or "white" or "spam" and "no spam".
Regression: A regression problem is when the output variable is a real value, such as "Rupees" or "height."

For our project we use the Supervised Learning as a Classification Problem. This makes it simple to train and provide a dataset of real and fake images for a model we design to run and learn to differentiate between them.

Good resources on how neural networks work and how CNN are more relevant to be used in conjunctions with images are as follows:

- https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/
- https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

These helped us a lot to formulate our code and understand the theory of our implementation.

Related Work

- BioID® Face Liveness Detection: A company working on Reliable Anti-Spoofing against Biometric Fraud. (https://www.bioid.com/liveness-detection/)

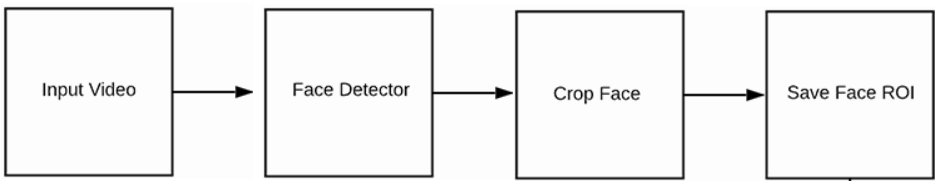
- Face Liveness Detection https://rose.ntu.edu.sg/research/Object%20Search/Pages/Face-Liveness-Detection.asp
- Adrian Rosenbrock (https://www.pyimagesearch.com/2019/03/11/liveness-detection-with-opencv/)
- AN OVERVIEW OF FACE LIVENESS DETECTION (https://arxiv.org/pdf/1405.2227.pdf)

The above points are a few of the main works that are closely related to what we are trying to achieve and derived our inspiration from.

Method

For our implementation we use the python programming language along with its libraries like numpy, sklearn, tensorflow, keras for machine learning and array calculations; OpenCV for image processing.

First to train our model we need data to run it on, so in order to do that we need to build a dataset containing real and fake set of face images. The below diagram breaks down the workflow of such operation to be done whose code is written in *gather_examples.py* of the project code directory.



We created videos of ourselves by recording them on mobiles and webcams for the real videos, the fake ones were made by making recordings of those on our webcams. In addition to these we also collected a few videos from Youtube to diversify our dataset a bit.

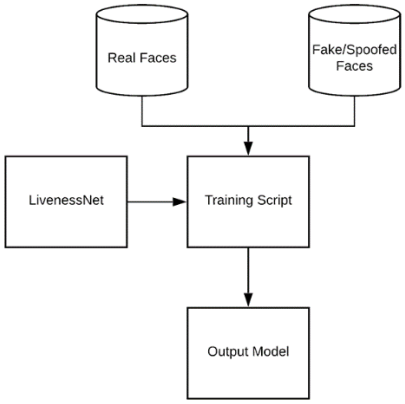
The command to do so is > *python gather_examples.py --input videos/fake --output dataset/fake --detector face_detector --skip 15*

Here the 'input' parameter is the directory of videos containing the fake/real videos, our code loops through all the videos in that directory performing the workflow as shown in the previous image. The face detection is done using OpenCVs own deep learning face detector, with the necessary files present in our code directory. The 'output' parameter is the directory where our images are saved. 'skip' is a parameter used to describe how many frames it should skip.

Next, its time to train our data on the models we define, that shall be described a bit more in the next sections. The command to train our data is as follows:

python train_liveness.py --dataset dataset --model liveness1.model --le le.pickle --plot plot1.png

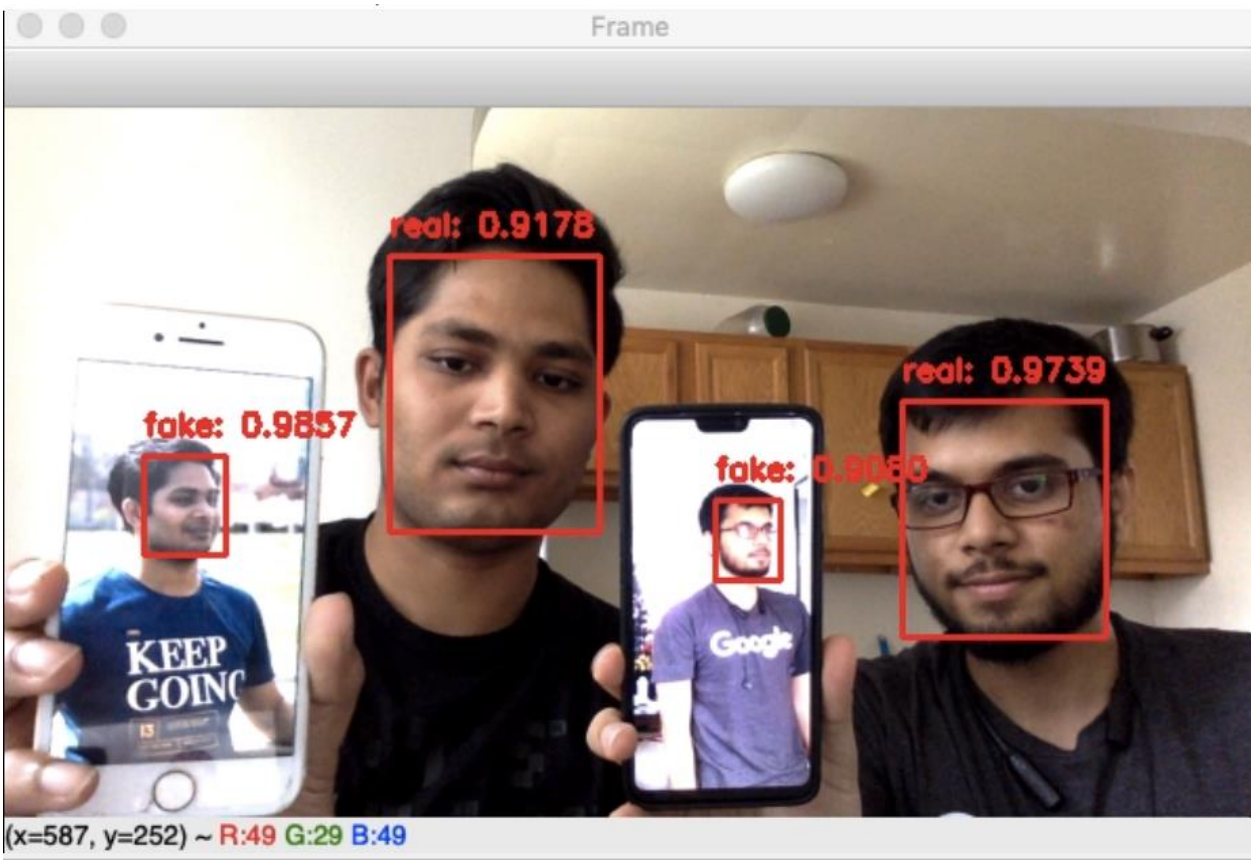
We pickle the trained model to be read later on and also plot its accuracy and loss for training and test data, all done with the sklearn library of Python. Below image shows the workflow to obtain our trained model.



Finally, we need to test our trained model, this is done by feeding it a live stream of video from our webcam, mimicking to a certain degree actual face detection system. The command for that is as follows:

python liveness_demo.py --model liveness1.model --le le.pickle --detector face_detector

All these commands are to be run within the code directory of our submission. The model parameter specifies which model to use, before the live feed is run through our model it first has to extract the faces obtained in those frames which are then fed into the network, that is why we also need to se OpenCVs face detection model in conjunction with our own one.



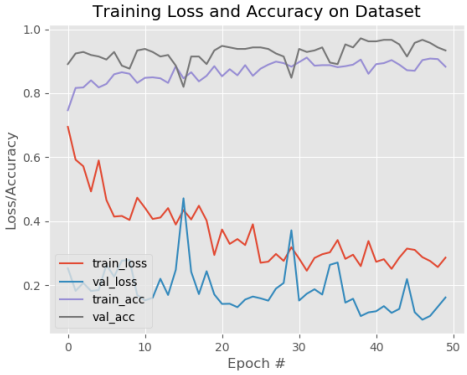
Here is a screenshot of a live demo on our most successful model, our submission also contains the video from which this screenshot has been taken.

Experiments

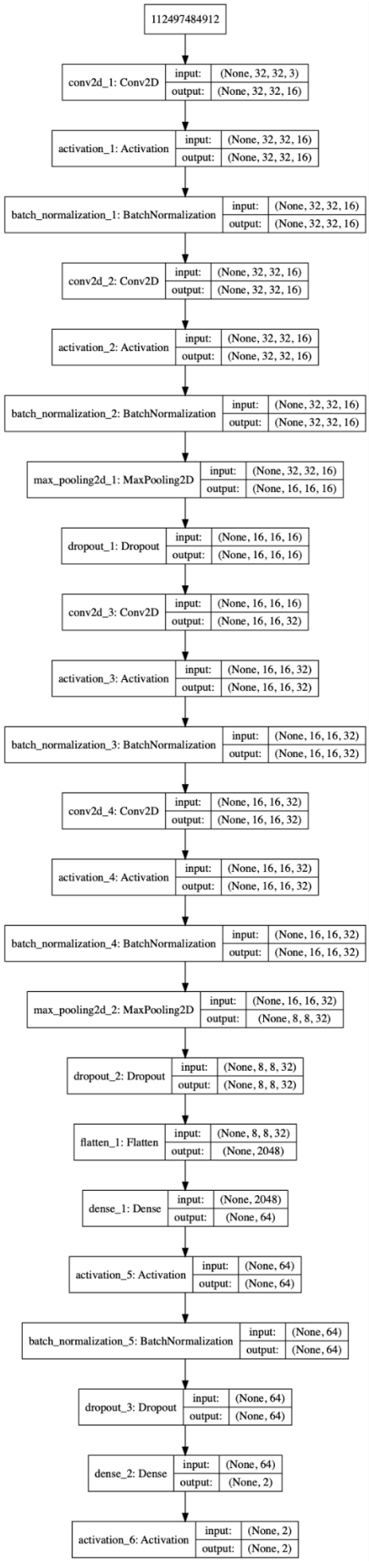
We ran experiments on three models, the most successful one has been presented in the video while the other two are described here.

The first model, which shows more accuracy has 3 convolutional layers with batch normalization and dropout layers to maintain relevancy of features obtained and not loose the hyperparameter values to something very negligible.

Its model and evaluations are as follows:

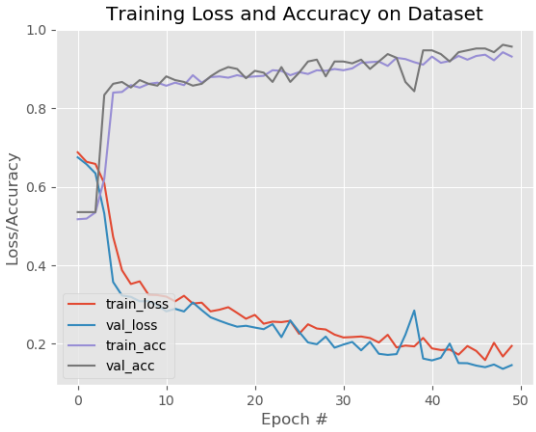



```
[INFO] evaluating network...
              precision    recall  f1-score   support

        fake       1.00      0.86      0.92        98
        real       0.89      1.00      0.94       113

   micro avg       0.93      0.93      0.93       211
   macro avg       0.94      0.93      0.93       211
weighted avg       0.94      0.93      0.93       211

[INFO] serializing network to 'liveness1.model'...
```


112497484912
conv2d_1: Conv2D
input: (None, 32, 32, 3)
output: (None, 32, 32, 16)
activation_1: Activation
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)
batch_normalization_1: BatchNormalization
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)
conv2d_2: Conv2D
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)
activation_2: Activation
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)
batch_normalization_2: BatchNormalization
input: (None, 32, 32, 16)
output: (None, 32, 32, 16)
max_pooling2d_1: MaxPooling2D
input: (None, 32, 32, 16)
output: (None, 16, 16, 16)
dropout_1: Dropout
input: (None, 16, 16, 16)
output: (None, 16, 16, 16)
conv2d_3: Conv2D
input: (None, 16, 16, 16)
output: (None, 16, 16, 32)
activation_3: Activation
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
batch_normalization_3: BatchNormalization
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
conv2d_4: Conv2D
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
activation_4: Activation
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
batch_normalization_4: BatchNormalization
input: (None, 16, 16, 32)
output: (None, 16, 16, 32)
max_pooling2d_2: MaxPooling2D
input: (None, 16, 16, 32)
output: (None, 8, 8, 32)
dropout_2: Dropout
input: (None, 8, 8, 32)
output: (None, 8, 8, 32)
flatten_1: Flatten
input: (None, 8, 8, 32)
output: (None, 2048)
dense_1: Dense
input: (None, 2048)
output: (None, 64)
activation_5: Activation
input: (None, 64)
output: (None, 64)
batch_normalization_5: BatchNormalization
input: (None, 64)
output: (None, 64)
dropout_3: Dropout
input: (None, 64)
output: (None, 64)
dense_2: Dense
input: (None, 64)
output: (None, 2)
activation_6: Activation
input: (None, 2)
output: (None, 2)

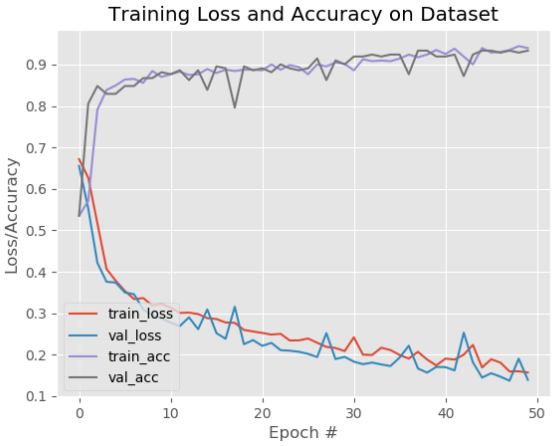
Model 2:

```
class LivenessNetCNN1:
    @staticmethod
    def build(width, height, depth, classes):
        # initialize the model along with the input shape to be
        # "channels last" and the channels dimension itself
        model = Sequential()
        inputShape = (height, width, depth)
        chanDim = -1

        # if we are using "channels first", update the input shape
        # and channels dimension
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)
            chanDim = 1

        # first CONV => RELU => CONV => RELU => POOL layer set
        model.add(Conv2D(16, (3, 3), padding="same",
            input_shape=inputShape))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(16, (3, 3), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        # second CONV => RELU => CONV => RELU => POOL layer set
        model.add(Conv2D(32, (3, 3), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(32, (3, 3), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        # first (and only) set of FC => RELU layers
        model.add(Flatten())
        model.add(Dense(64))
        model.add(Activation("relu"))
        model.add(Dropout(0.5))

        # softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # saving model structure as png
        # plot_model(model, to_file='model_plot2.png', show_shapes=True, show_layer_names=True)

        # return the constructed network architecture
        return model
```




```
[INFO] evaluating network...
              precision    recall  f1-score   support

        fake       0.88      0.90      0.89        98
        real       0.91      0.89      0.90       113

   micro avg       0.90      0.90      0.90       211
   macro avg       0.89      0.90      0.90       211
weighted avg       0.90      0.90      0.90       211

[INFO] serializing network to 'liveness2.model'...
```

Model 3:




```
[INFO] evaluating network...
              precision    recall  f1-score   support

        fake       0.96      0.91      0.93        98
        real       0.92      0.96      0.94       113

   micro avg       0.94      0.94      0.94       211
   macro avg       0.94      0.94      0.94       211
weighted avg       0.94      0.94      0.94       211

[INFO] serializing network to 'liveness2.model'...
```

```python
class LivenessNetCNN2:
    @staticmethod
    def build(width, height, depth, classes):
        # initialize the model along with the input shape to be
        # "channels last" and the channels dimension itself
        model = Sequential()
        inputShape = (height, width, depth)
        chanDim = -1

        # if we are using "channels first", update the input shape
        # and channels dimension
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)
            chanDim = 1

        # first CONV => RELU => CONV => RELU => POOL layer set
        model.add(Conv2D(16, (5, 5), padding="same",
            input_shape=inputShape))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(16, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        # second CONV => RELU => CONV => RELU => POOL layer set
        model.add(Conv2D(32, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(32, (3, 3), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        # first (and only) set of FC => RELU layers
        model.add(Flatten())
        model.add(Dense(64))
        model.add(Activation("relu"))
        model.add(Dropout(0.5))

        # softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # saving model structure as png
        # plot_model(model, to_file='model_plot3.png', show_shapes=True, show_layer_names=True)

        # return the constructed network architecture
        return model
```

Conclusion

We have decided that the first model is the best since its seems the most generalized one based on the evaluation plot, while the other two models tend to overfit, which is mainly due to lack of convolution and batch normalization.

Finally, to arrive at a conclusion, we can say that the advantage of treating this problem as a binary classification one is that our model can be fine tuned to get good results, but as a disadvantage, we need larger and diverse data of faces as a dataset to make it work in the real world and avoid over fitting.

Additional heuristics like Optical Flow Analysis, Texture Analysis, 3D Face contour Analysis can b used in conjunction with this to provide rich features, which may require less training and give higher accuracies, since no single method gives a complete solution.