Deepfake detection challenge - a research journal

Francesco Stablum

January 27, 2020

1 Motivation

If there is something that I need to refine in my education it's probably machine learning applied on 3D rasters and temporal data. The deepfake detection challenge seems to be the perfect opportunity to dive deep in methods that I haven't tried before. Specifically, analysis of video data may employ techniques from both domains as videos can be interpreted as 3D grids, as well as a sequence of datapoints that is subject to causal evolution (time). The first interpretation would suggest using convolutional neural network with a 3D kernel; the latter suggests recurrent neural networks and LSTMs.

2 23/01/2020 - first attempts with a 3D convolutional neural network

2.1 The dataset

The dataset comprises 400 videos that have about 300 frames each; sometimes 299 - with 3 colors channels. The resolution is 1920x1080. Sometimes transposed (vertical smartphone videos).

Even if the datapoints are effectively quite similar, they still require some operations to make them have the same dimensionality. Moreover, the sheer size of the uncompressed video suggests some preliminary scaling down of the input data.

2.2 Tackling the huge dimensionality of the datapoints

Every datapoint, loaded trough the library `skvideo`, has an uncompressed size of 1.8 GB. In these conditions it seems an obvious choice to load one datapoint at once. In order to have faster epochs, a slice of 15 frames is selected from each datapoint scaling down the datapoint dimensionality to 93 MB. The starting frame number of the frame slice will be chosen randomly. This will allow all the frames of a video to be eventually considered trough all the epochs.

2.3 Structure of the first model

The convolutional layers have been set with ELU activation functions. The reason for this is to avoid dead units that emerge with ReLU activation functions caused by 0 gradients. For the dense layers the sigmoid activation function has been chosen. Different activation functions could be also be investigated in the future, such as ReLU and tanh, respectively for the convolutional and dense layers. Also, not much tought has been given to regularization. But, since the network seem to not fit on the training set, that's a problem for later.

The amount of parameters for this model is 22648.

2.4 Considerations on using kernels that operate trough time

What I noticed by watching the deepfake videos from the dataset is that some of the videos were showing very scattered application of the face alteration. Such abrupt changes are one of the first applications of convolutions for image processing, such as edge detectors. Such characteristic would be detected here w.r.t. time.

For future reference, exclusively time-based convolution layers might be used in a model - i.e. convolutional layers with kernel sizes that are 1 in both spatial axes but arbitrary in the time dimension.

Also, factorized convolution could be used in a future model, as described in [2].

2.5 Network output

Since the output of the network is a 2-node softmax, the first node is being used to learn a probability quantity of the input datapoint of being a deepfake. The second node, since it's softmax, will always output 1 minus the output of the first node.

The target vectors are hence $(1, 0)$ for a deepfake and $(0, 1)$ for a real video.

2.6 Training

Since training takes a considerable amount of time (about one hour for a training set sweep), the number of training epochs is chosen to be as small as 100. SGD has been chosen as optimization method. Adam/Nadam/Amsgrad/ND-Adam might be considered as future improvement.

2.7 Evaluation measures

Currently two poorly-named (by myself) evaluation measures are being employed: unquantized average error, and quantized average error. Considering uniquely the value of the first output node, an unquantized error is the difference between the expected value - 1 for deepfake, 0 for real - and the emitted probability from the network. The unquantized average error is simply the average value across the dataset.

The quantized error is, conversely, a rounding to the nearest integer of the unquantized error. In a typical classification model a decision has to be made on how to use the output of a neural network. Hence, the integer-rounding of the output represents the decision on interpreting the binary classification model output either of one or the other category. Hence, the quantized error is either 1 or 0 whenever that decision was correct or wrong. The quantized average error is hence the average of the individual quantized errors after a sweep of a validation/testing set.

In order to have a better overview on how the model is performing with regard to the class unbalance problem, ratios of correct and wrong predictions disaggregated per class are being kept.

2.8 Class unbalance problem

The first model is not performing well. The output category is always the most frequent one, which is the deepfake videos, amounting to 323, while the real videos are only 77. This class unbalance problem could be tackled by submitting more frequently the real videos to the training. The ratio of fake/real is 4.2, hence one possible future improvement could be to submit about 4 different slices of the real video to the fitting function for every slice of fake videos.

2.9 Future improvements

- localize the face and follow it to remove useless background

- log number of parameters
- log average loss
- factorized convolutions
- bottleneck layers
- skip connections
- different update methods
- LSTMs/RNN
- checkpoints to resume training

Figure 1: First model: a 3d-convolutional neural network

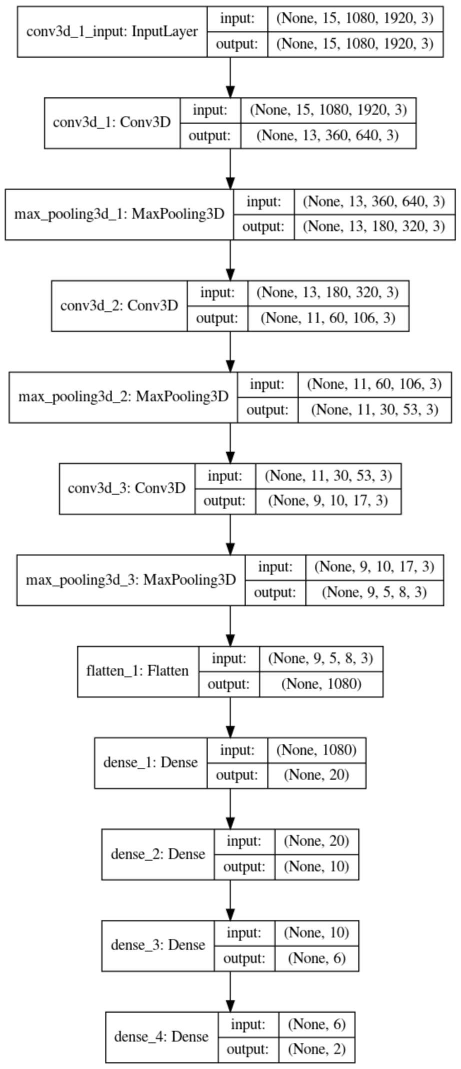

conv3d_1_input: InputLayer
input: (None, 15, 1080, 1920, 3)
output: (None, 15, 1080, 1920, 3)
conv3d_1: Conv3D
input: (None, 15, 1080, 1920, 3)
output: (None, 13, 360, 640, 3)
max_pooling3d_1: MaxPooling3D
input: (None, 13, 360, 640, 3)
output: (None, 13, 180, 320, 3)
conv3d_2: Conv3D
input: (None, 13, 180, 320, 3)
output: (None, 11, 60, 106, 3)
max_pooling3d_2: MaxPooling3D
input: (None, 11, 60, 106, 3)
output: (None, 11, 30, 53, 3)
conv3d_3: Conv3D
input: (None, 11, 30, 53, 3)
output: (None, 9, 10, 17, 3)
max_pooling3d_3: MaxPooling3D
input: (None, 9, 10, 17, 3)
output: (None, 9, 5, 8, 3)
flatten_1: Flatten
input: (None, 9, 5, 8, 3)
output: (None, 1080)
dense_1: Dense
input: (None, 1080)
output: (None, 20)
dense_2: Dense
input: (None, 20)
output: (None, 10)
dense_3: Dense
input: (None, 10)
output: (None, 6)
dense_4: Dense
input: (None, 6)
output: (None, 2)

Figure 2: First model: parameter count for each layer

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv3d_1 (Conv3D)            (None, 13, 360, 640, 3)   246
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 13, 180, 320, 3)   0
_________________________________________________________________
conv3d_2 (Conv3D)            (None, 11, 60, 106, 3)    246
_________________________________________________________________
max_pooling3d_2 (MaxPooling3 (None, 11, 30, 53, 3)     0
_________________________________________________________________
conv3d_3 (Conv3D)            (None, 9, 10, 17, 3)      246
_________________________________________________________________
max_pooling3d_3 (MaxPooling3 (None, 9, 5, 8, 3)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1080)              0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                21620
_________________________________________________________________
dense_2 (Dense)              (None, 10)                210
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 66
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 14
=================================================================
Total params: 22,648
Trainable params: 22,648
Non-trainable params: 0
_________________________________________________________________
```

3 25/01/2020 : second model

3.1 Decomposed kernels

It's well known that a symmetric matrix can be decomposed in two vector-sized kernels. Subsequent convolution with the two (in the case of 2D images) separated kernels result into the same convolution as the original kernel. An example is shown in [3].

In theory it's possible to learn symmetric matrices (or tensors) by separation in vectors, each for every dimension.

This is going to be the focus of the second model: every 3D convolution is replaced by three 3D convolutions that have kernel dimension set at 1 except for the dimension in which the convolution is going to operate.

This way, every "block" of convolutions is going to have 30 parameters instead of 246.

The 'stride' subsampling that was characteristic of every convolutional layer in the first model is being set only at the third layer in each block of "decomposed" convolutional layers.

3.2 Excessive computation of separated convolution layers

Unfortunately this network, even if it has a lower number of parameters, its still very computationally expensive. This is possibly due to the triplication of sweeps on data that has the input dimensionality.

To tackle this problem, a subsampling maxpool layer has been added as input layer to have a preliminary dimensionality reduction. This follows an observation of genetic algorithm-evolved convolutional neural networks that were optimized also by running time. Many of the best-fit genotypes selected by the genetic algorithm were putting a maxpool layer on the input with suprising results also in accuracy of the network.

The resulting network has a much reduced amount of trainable parameters: only 1660.

3.3 Excessive loading and pre-processing times

Moreover, loading a single video in tensor form takes about 7 seconds. This is not different even using a ramdisk, where all the videos are stored in memory instead of on the hard drive.

This issue requires a different strategy. Under the current model configuration, just 15 frames are being used over all 300 frames of each video. Considering the high loading time of a single video this is quite a waste. A better approach could be to load and process multiple 15-frames slices after a video has been loaded.

This will cause a stronger "learning intensity" per epoch, as multiple calls to the fit function will happen for every video.

Memory limits need to be taken into consideration with these changes. A 15-frames uncompressed slice requires 93.3 MB. A conservative memory allocation can be 2GB, hence about 20 slices can be loaded from a video.

Such changes come very handy also to tackle the class unbalance problem, which can be dealt by submitting 5 slices for every fake video and 21 slices for every real video, which will amount to the previously reported 4.2 ratio of fake videos / real videos.

By submitting 5 slices for every fake video and 21 slices for every real one, one “new” epoch will have processed 3232. Hence, it would be equivalent as 8.08 epochs in the first model.

To keep a fair comparison with the first model, the amount of videos loaded at every epoch will be one eight, i.e. 50 videos.

3.4 Changes in activation functions and optimizing method

The fully connected layers have been added the tanh activation function. The co-domain of the function between -1 and +1 has possibly some advantages. Vanishing gradient might still be an issue, hence ELU should be considered in the future.

Adam has been used as optimization method, due to using momentum.

3.5 A note on the platforms used

MLFlow it’s a python library employed to track all the experiments, artifact archive, parameter and metric logging. It provides a convenient user interface to navigate plots of evolutions of the metrics trough time.

Floyd Hub is a remote service that provides easy deployment of runs on powerful hardware such as nodes with NVidia K80 GPUs. It’s a rental, so it comes with a price.

3.6 Second model - an additional larger variant

As a way to make proper use of the newly-rented GPUs from floydhub, a larger version of the aforementioned model has been trained. Among the differences: the number of convolutional blocks has been increased from 3 to 7. This was made possible by the removal of the strides in the ending layer of each convolutional block. This reduced the dimensionality reduction allowing more convolutional blocks (and max pool layers). Moreover, the full initial dimensionality has been given to the convolutional blocks by removing the initial maxpool layer. Also, the first dense layer has been increased from 20 to 50. These changes resulted in a much higher trainable parameter count: $47,075$

3.7 Performances of the second model and the larger variant

Both models failed again to even hint at the learning some differences between the two categories, having mean accuracy oscillating around 0.5, basically a coin toss.

3.8 Reflecting on mistakes and future directions

First of all, in the second model the advice given by [2] of *not* using separated convolutions on the first layers was mistakenly not considered. Moreover, having separated convolutions does not mean reduced computation by the triplication of the number of sweeps. Especially when the kernel size is kept as low as 3. It may make more sense if the kernel size would be much higher.

Another huge issue is that 90% of the video is useless for the task of deepfake detection. It may be more useful to identify a smaller window where the face is located and concentrate the learning only on that sub-tensor. That would result in faster learning times,hence more dataset sweeps (epochs) and inferior signal/noise ratio. A library such as [1] can be used for the task.

Also, every fit call is currently processing one single datapoint, minibatch learning needs to be seriously considered to take full advantage of the GPU.

Industry practices suggest using an incresing number of filters as the network gets deeper.

Figure 3: Second model: separated convolutions

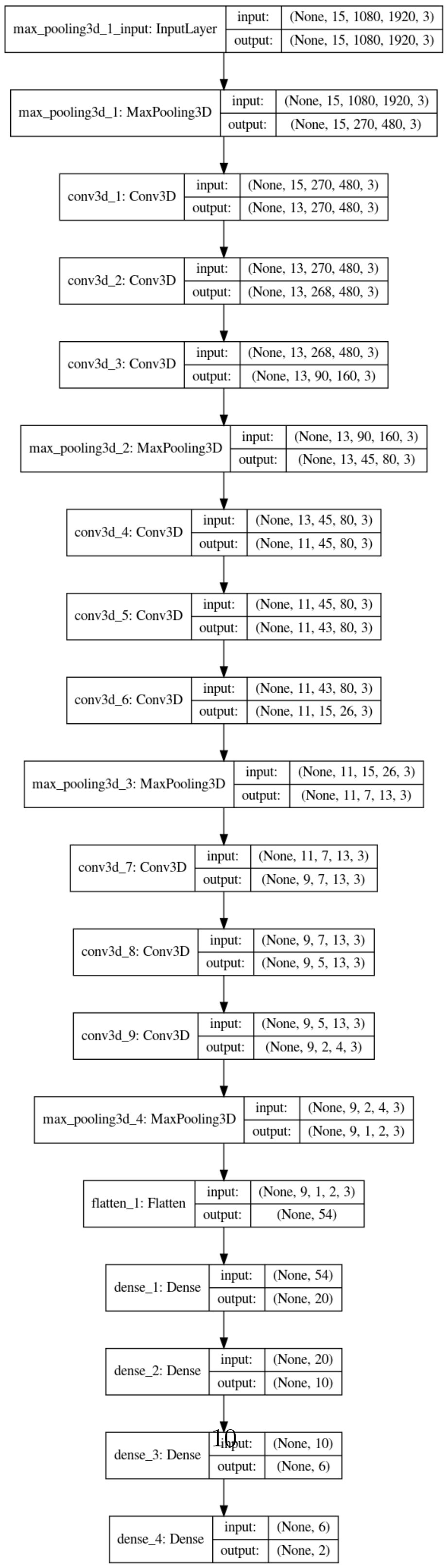

max_pooling3d_1_input: InputLayer
input: (None, 15, 1080, 1920, 3)
output: (None, 15, 1080, 1920, 3)
max_pooling3d_1: MaxPooling3D
input: (None, 15, 1080, 1920, 3)
output: (None, 15, 270, 480, 3)
conv3d_1: Conv3D
input: (None, 15, 270, 480, 3)
output: (None, 13, 270, 480, 3)
conv3d_2: Conv3D
input: (None, 13, 270, 480, 3)
output: (None, 13, 268, 480, 3)
conv3d_3: Conv3D
input: (None, 13, 268, 480, 3)
output: (None, 13, 90, 160, 3)
max_pooling3d_2: MaxPooling3D
input: (None, 13, 90, 160, 3)
output: (None, 13, 45, 80, 3)
conv3d_4: Conv3D
input: (None, 13, 45, 80, 3)
output: (None, 11, 45, 80, 3)
conv3d_5: Conv3D
input: (None, 11, 45, 80, 3)
output: (None, 11, 43, 80, 3)
conv3d_6: Conv3D
input: (None, 11, 43, 80, 3)
output: (None, 11, 15, 26, 3)
max_pooling3d_3: MaxPooling3D
input: (None, 11, 15, 26, 3)
output: (None, 11, 7, 13, 3)
conv3d_7: Conv3D
input: (None, 11, 7, 13, 3)
output: (None, 9, 7, 13, 3)
conv3d_8: Conv3D
input: (None, 9, 7, 13, 3)
output: (None, 9, 5, 13, 3)
conv3d_9: Conv3D
input: (None, 9, 5, 13, 3)
output: (None, 9, 2, 4, 3)
max_pooling3d_4: MaxPooling3D
input: (None, 9, 2, 4, 3)
output: (None, 9, 1, 2, 3)
flatten_1: Flatten
input: (None, 9, 1, 2, 3)
output: (None, 54)
dense_1: Dense
input: (None, 54)
output: (None, 20)
dense_2: Dense
input: (None, 20)
output: (None, 10)
dense_3: Dense
input: (None, 10)
output: (None, 6)
dense_4: Dense
input: (None, 6)
output: (None, 2)

Figure 4: Second model: parameter count for each layer

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
max_pooling3d_1 (MaxPooling3 (None, 15, 270, 480, 3)   0
_________________________________________________________________
conv3d_1 (Conv3D)            (None, 13, 270, 480, 3)   30
_________________________________________________________________
conv3d_2 (Conv3D)            (None, 13, 268, 480, 3)   30
_________________________________________________________________
conv3d_3 (Conv3D)            (None, 13, 90, 160, 3)    30
_________________________________________________________________
max_pooling3d_2 (MaxPooling3 (None, 13, 45, 80, 3)     0
_________________________________________________________________
conv3d_4 (Conv3D)            (None, 11, 45, 80, 3)     30
_________________________________________________________________
conv3d_5 (Conv3D)            (None, 11, 43, 80, 3)     30
_________________________________________________________________
conv3d_6 (Conv3D)            (None, 11, 15, 26, 3)     30
_________________________________________________________________
max_pooling3d_3 (MaxPooling3 (None, 11, 7, 13, 3)      0
_________________________________________________________________
conv3d_7 (Conv3D)            (None, 9, 7, 13, 3)       30
_________________________________________________________________
conv3d_8 (Conv3D)            (None, 9, 5, 13, 3)       30
_________________________________________________________________
conv3d_9 (Conv3D)            (None, 9, 2, 4, 3)        30
_________________________________________________________________
max_pooling3d_4 (MaxPooling3 (None, 9, 1, 2, 3)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 54)                0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                1100
_________________________________________________________________
dense_2 (Dense)              (None, 10)                210
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 66
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 14
=================================================================
Total params: 1,660
Trainable params: 1,660
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Second model: separated convolutions - larger variant

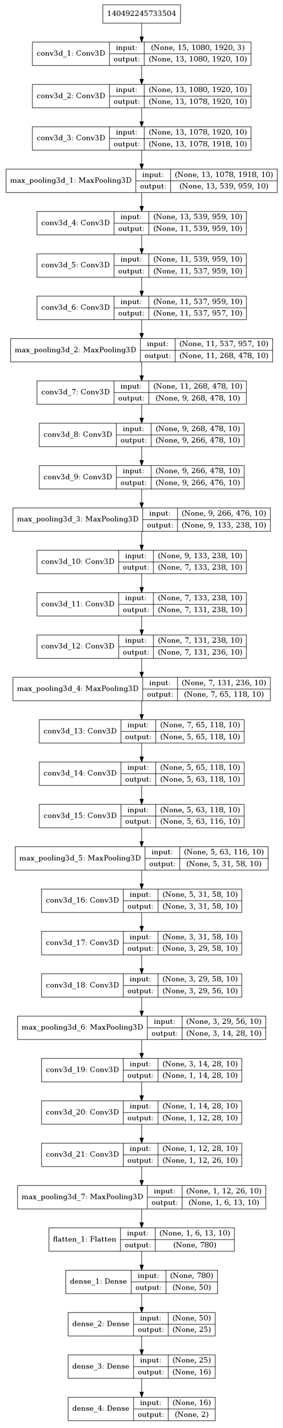
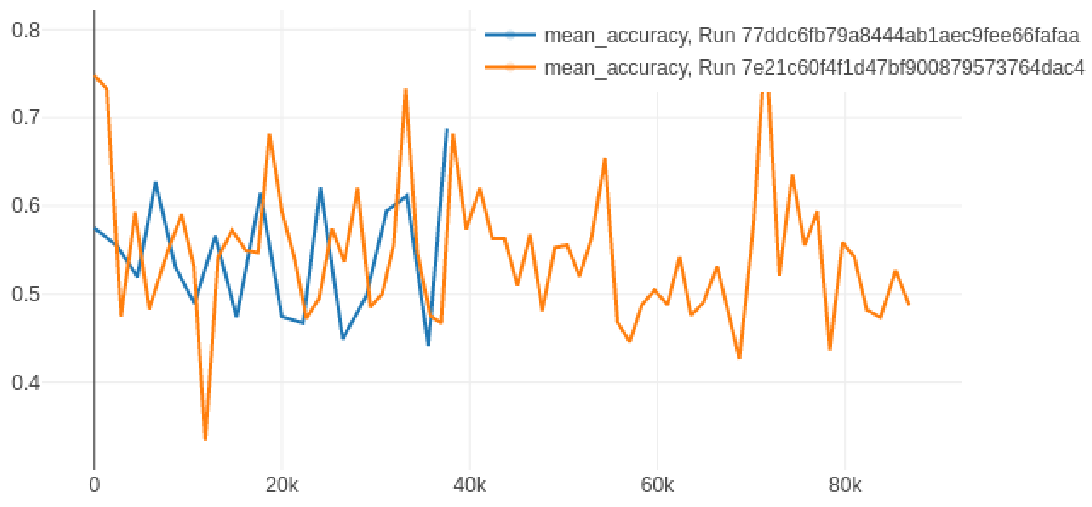

140492245733504
conv3d_1: Conv3D input: (None, 15, 1080, 1920, 3) output: (None, 13, 1080, 1920, 10)
conv3d_2: Conv3D input: (None, 13, 1080, 1920, 10) output: (None, 13, 1078, 1920, 10)
conv3d_3: Conv3D input: (None, 13, 1078, 1920, 10) output: (None, 13, 1078, 1918, 10)
max_pooling3d_1: MaxPooling3D input: (None, 13, 1078, 1918, 10) output: (None, 13, 539, 959, 10)
conv3d_4: Conv3D input: (None, 13, 539, 959, 10) output: (None, 11, 539, 959, 10)
conv3d_5: Conv3D input: (None, 11, 539, 959, 10) output: (None, 11, 537, 959, 10)
conv3d_6: Conv3D input: (None, 11, 537, 959, 10) output: (None, 11, 537, 957, 10)
max_pooling3d_2: MaxPooling3D input: (None, 11, 537, 957, 10) output: (None, 11, 268, 478, 10)
conv3d_7: Conv3D input: (None, 11, 268, 478, 10) output: (None, 9, 268, 478, 10)
conv3d_8: Conv3D input: (None, 9, 268, 478, 10) output: (None, 9, 266, 478, 10)
conv3d_9: Conv3D input: (None, 9, 266, 478, 10) output: (None, 9, 266, 476, 10)
max_pooling3d_3: MaxPooling3D input: (None, 9, 266, 476, 10) output: (None, 9, 133, 238, 10)
conv3d_10: Conv3D input: (None, 9, 133, 238, 10) output: (None, 7, 133, 238, 10)
conv3d_11: Conv3D input: (None, 7, 133, 238, 10) output: (None, 7, 131, 238, 10)
conv3d_12: Conv3D input: (None, 7, 131, 238, 10) output: (None, 7, 131, 236, 10)
max_pooling3d_4: MaxPooling3D input: (None, 7, 131, 236, 10) output: (None, 7, 65, 118, 10)
conv3d_13: Conv3D input: (None, 7, 65, 118, 10) output: (None, 5, 65, 118, 10)
conv3d_14: Conv3D input: (None, 5, 65, 118, 10) output: (None, 5, 63, 118, 10)
conv3d_15: Conv3D input: (None, 5, 63, 118, 10) output: (None, 5, 63, 116, 10)
max_pooling3d_5: MaxPooling3D input: (None, 5, 63, 116, 10) output: (None, 5, 31, 58, 10)
conv3d_16: Conv3D input: (None, 5, 31, 58, 10) output: (None, 3, 31, 58, 10)
conv3d_17: Conv3D input: (None, 3, 31, 58, 10) output: (None, 3, 29, 58, 10)
conv3d_18: Conv3D input: (None, 3, 29, 58, 10) output: (None, 3, 29, 56, 10)
max_pooling3d_6: MaxPooling3D input: (None, 3, 29, 56, 10) output: (None, 3, 14, 28, 10)
conv3d_19: Conv3D input: (None, 3, 14, 28, 10) output: (None, 1, 14, 28, 10)
conv3d_20: Conv3D input: (None, 1, 14, 28, 10) output: (None, 1, 12, 28, 10)
conv3d_21: Conv3D input: (None, 1, 12, 28, 10) output: (None, 1, 12, 26, 10)
max_pooling3d_7: MaxPooling3D input: (None, 1, 12, 26, 10) output: (None, 1, 6, 13, 10)
flatten_1: Flatten input: (None, 1, 6, 13, 10) output: (None, 780)
dense_1: Dense input: (None, 780) output: (None, 50)
dense_2: Dense input: (None, 50) output: (None, 25)
dense_3: Dense input: (None, 25) output: (None, 16)
dense_4: Dense input: (None, 16) output: (None, 2)

Figure 6: Second model - larger variant: parameter count for each layer

```
-----------------------------------------------------------------
Layer (type)                 Output Shape              Param #
=================================================================
conv3d_1 (Conv3D)            (None, 13, 1080, 1920, 10 100
-----------------------------------------------------------------
conv3d_2 (Conv3D)            (None, 13, 1078, 1920, 10 310
-----------------------------------------------------------------
conv3d_3 (Conv3D)            (None, 13, 1078, 1918, 10 310
-----------------------------------------------------------------
max_pooling3d_1 (MaxPooling3 (None, 13, 539, 959, 10)  0
-----------------------------------------------------------------
conv3d_4 (Conv3D)            (None, 11, 539, 959, 10)  310
-----------------------------------------------------------------
conv3d_5 (Conv3D)            (None, 11, 537, 959, 10)  310
-----------------------------------------------------------------
conv3d_6 (Conv3D)            (None, 11, 537, 957, 10)  310
-----------------------------------------------------------------
max_pooling3d_2 (MaxPooling3 (None, 11, 268, 478, 10)  0
-----------------------------------------------------------------
conv3d_7 (Conv3D)            (None, 9, 268, 478, 10)   310
-----------------------------------------------------------------
conv3d_8 (Conv3D)            (None, 9, 266, 478, 10)   310
-----------------------------------------------------------------
conv3d_9 (Conv3D)            (None, 9, 266, 476, 10)   310
-----------------------------------------------------------------
max_pooling3d_3 (MaxPooling3 (None, 9, 133, 238, 10)   0
-----------------------------------------------------------------
conv3d_10 (Conv3D)           (None, 7, 133, 238, 10)   310
-----------------------------------------------------------------
conv3d_11 (Conv3D)           (None, 7, 131, 238, 10)   310
-----------------------------------------------------------------
conv3d_12 (Conv3D)           (None, 7, 131, 236, 10)   310
-----------------------------------------------------------------
max_pooling3d_4 (MaxPooling3 (None, 7, 65, 118, 10)    0
-----------------------------------------------------------------
conv3d_13 (Conv3D)           (None, 5, 65, 118, 10)    310
-----------------------------------------------------------------
conv3d_14 (Conv3D)           (None, 5, 63, 118, 10)    310
-----------------------------------------------------------------
conv3d_15 (Conv3D)           (None, 5, 63, 116, 10)    310
-----------------------------------------------------------------
max_pooling3d_5 (MaxPooling3 (None, 5, 31, 58, 10)     0
-----------------------------------------------------------------
conv3d_16 (Conv3D)           (None, 3, 31, 58, 10)     310
-----------------------------------------------------------------
conv3d_17 (Conv3D)           (None, 3, 29, 58, 10)     310
-----------------------------------------------------------------
conv3d_18 (Conv3D)           (None, 3, 29, 56, 10)     310
-----------------------------------------------------------------
max_pooling3d_6 (MaxPooling3 (None, 3, 14, 28, 10)     0
-----------------------------------------------------------------
conv3d_19 (Conv3D)           (None, 1, 14, 28, 10)     310
-----------------------------------------------------------------
conv3d_20 (Conv3D)           (None, 1, 12, 28, 10)     310
-----------------------------------------------------------------
conv3d_21 (Conv3D)           (None, 1, 12, 26, 10)     310
-----------------------------------------------------------------
max_pooling3d_7 (MaxPooling3 (None, 1, 6, 13, 10)      0
-----------------------------------------------------------------
flatten_1 (Flatten)          (None, 780)               0
-----------------------------------------------------------------
dense_1 (Dense)              (None, 50)                39050
-----------------------------------------------------------------
dense_2 (Dense)              (None, 25)                1275
-----------------------------------------------------------------
dense_3 (Dense)              (None, 16)                416
-----------------------------------------------------------------
dense_4 (Dense)              (None, 2)                 34
=================================================================
Total params: 47,075
Trainable params: 47,075
Non-trainable params: 0
-----------------------------------------------------------------
```

Figure 7: Second model: accuracy - vanilla and larger variant



References

[1] The world's simplest facial recognition api for python and the command line. https://github.com/ageitgey/face_recognition.

[2] Alex Burlacu. Speeding up convolutional neural networks. https://towardsdatascience.com/speeding-up-convolutional-neural-networks-240beac5e30f, 2018.

[3] Wikipedia. Sobel operator. https://en.wikipedia.org/wiki/Sobel_operator.