DEEP-FAKE DETECTOR

by GLEM

DATA ANALYTICS AND ARTIFICIAL INTELLIGENCE [EM1405]

Ca' Foscari University of Venice

Ca' Foscari University of Venice

ABOUT THE PROJECT

The aim is to create a model that can tell whether a video is real or generated by AI.

The model is based on a cooperation of CNN and LSTM.

Ca' Foscari University of Venice

AGENDA

01. IMAGE CLASSIFICATION

Ca' Foscari University of Venice

The very first dataset is an image one.

The aim is to build a CNN which can classify fake/AI generated images from real ones.

It is divided into three folders, each of them containing 50% of fake and 50% of real images of 256x256 pixels.

```python
os.listdir(data_dir)
```

✓ 0.0s Python

```
['Test', 'Train', 'Validation']
```

```python
print(len( os.listdir((os.path.join(data_dir,'Train','Real')))))
print(len( os.listdir((os.path.join(data_dir,'Train','Fake')))))
```

✓ 0.5s Python

```
70001
70001
```

01. IMAGE CLASSIFICATION

Before building the models, we first created objects that can be fed to the networks.

We generated iterators made up of NumPy arrays that store features and corresponding class labels of each image.

These iterators facilitate easier access to the images, allowing batch-by-batch processing.

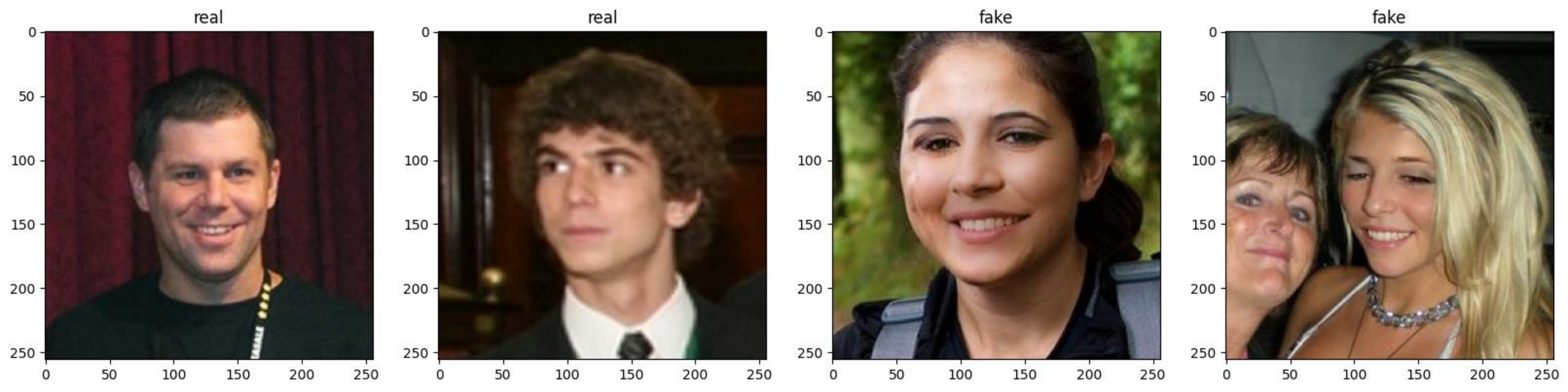
```python
train_direct = "/content/Train"
train_data = tf.keras.utils.image_dataset_from_directory(train_direct)

data_iterator = train_data.as_numpy_iterator()

batch = data_iterator.next()

print(len(batch))
print(batch[0].shape)
print(batch[1])
```

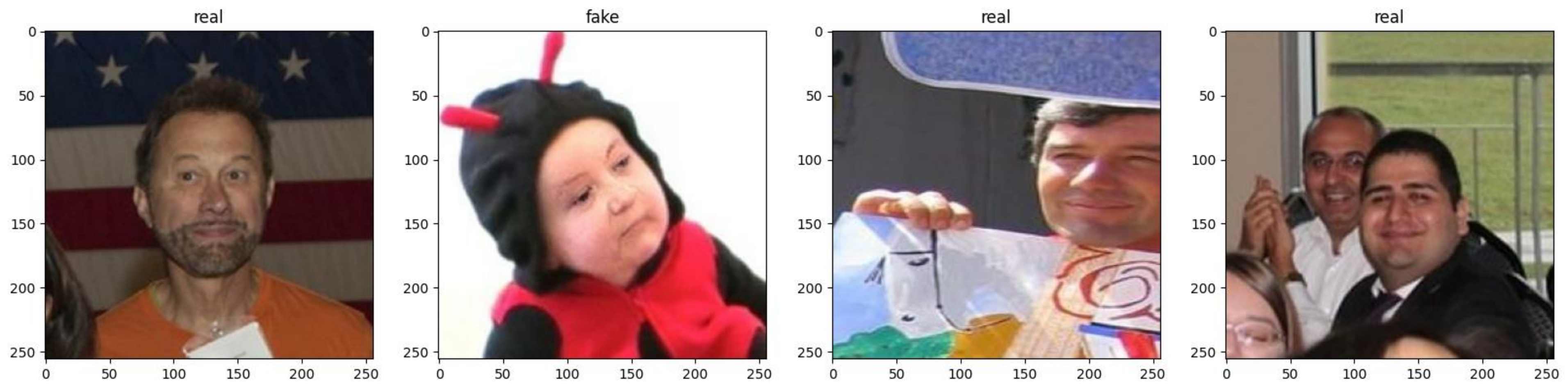
Python

```
Found 140002 files belonging to 2 classes.
2
(32, 256, 256, 3)
[0 0 1 1 0 0 1 0 1 0 0 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 0 0 0 1 1 1]
```

01. IMAGE CLASSIFICATION

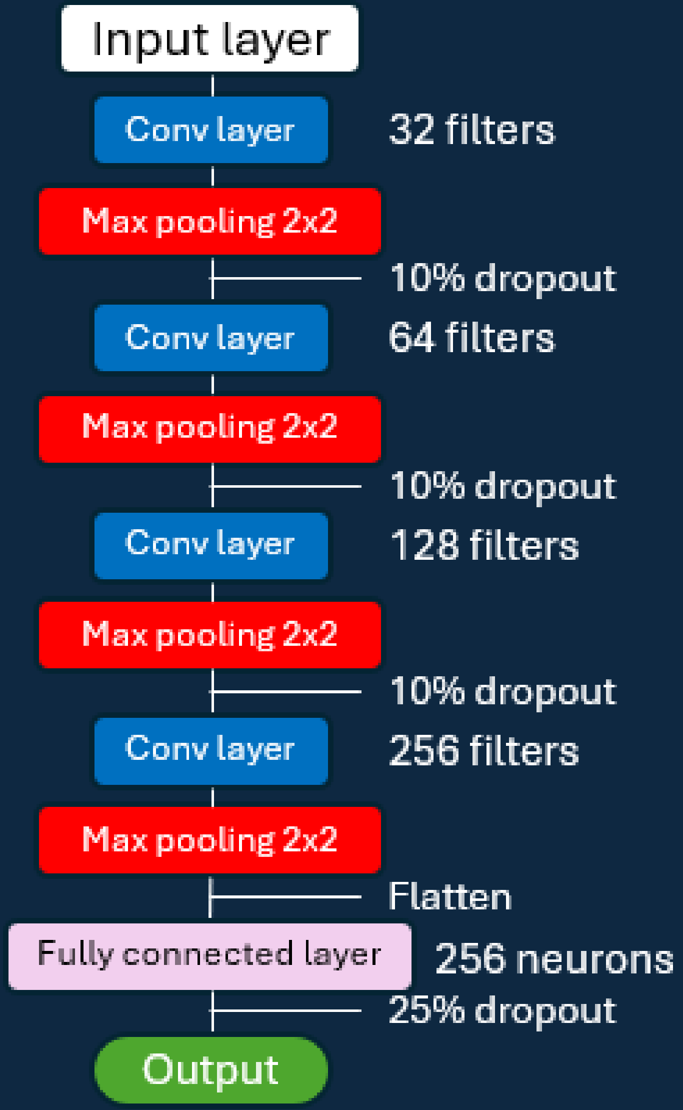
Ca' Foscari University of Venice





Ca' Foscari University of Venice

02. CNN



Once retrieved our dataset, we trained the very first CNN.

After some attempts, the accuracy of our model increased to the 96%, which seemed to be the best achievable by it.

On the left, a representation of our network

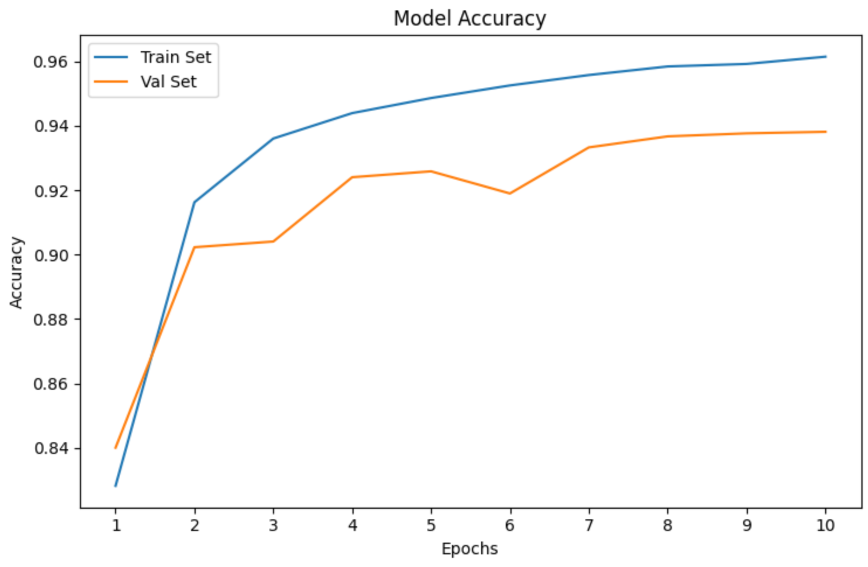
Ca' Foscari University of Venice

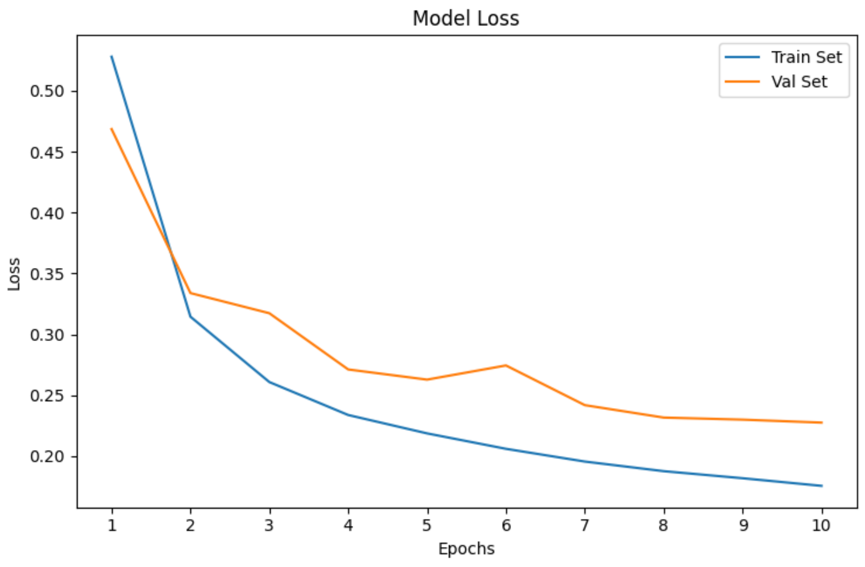
02. CNN

```
Epoch 1/10
4376/4376 ━━━━━━━━━━━━━━━━━━━━ 354s 77ms/step - accuracy: 0.7584 - loss: 0.6772 - val_accuracy: 0.8197 - val_loss: 0.4993
Epoch 2/10
4376/4376 ━━━━━━━━━━━━━━━━━━━━ 352s 74ms/step - accuracy: 0.9041 - loss: 0.3373 - val_accuracy: 0.8942 - val_loss: 0.3444
Epoch 3/10
4376/4376 ━━━━━━━━━━━━━━━━━━━━ 383s 74ms/step - accuracy: 0.9306 - loss: 0.2711 - val_accuracy: 0.9148 - val_loss: 0.2963
```

● ● ●

```
Epoch 8/10
4376/4376 ━━━━━━━━━━━━━━━━━━━━ 323s 74ms/step - accuracy: 0.9556 - loss: 0.1951 - val_accuracy: 0.9322 - val_loss: 0.2457
Epoch 9/10
4376/4376 ━━━━━━━━━━━━━━━━━━━━ 324s 74ms/step - accuracy: 0.9571 - loss: 0.1897 - val_accuracy: 0.9317 - val_loss: 0.2464
Epoch 10/10
4376/4376 ━━━━━━━━━━━━━━━━━━━━ 401s 78ms/step - accuracy: 0.9590 - loss: 0.1836 - val_accuracy: 0.9319 - val_loss: 0.2447
```





Ca' Foscari University of Venice

02. CNN

```
341/341 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - accuracy: 0.9119 - loss: 0.2106
Test Loss: 0.3689965009689331
Test Accuracy: 0.8395231366157532
```

Even if the accuracy is quite good for a first approach, the loss value up here indicates that misclassification still occurs, as shown by the results on the right.

This also explains the values shown in the classification report and in the confusion matrix.

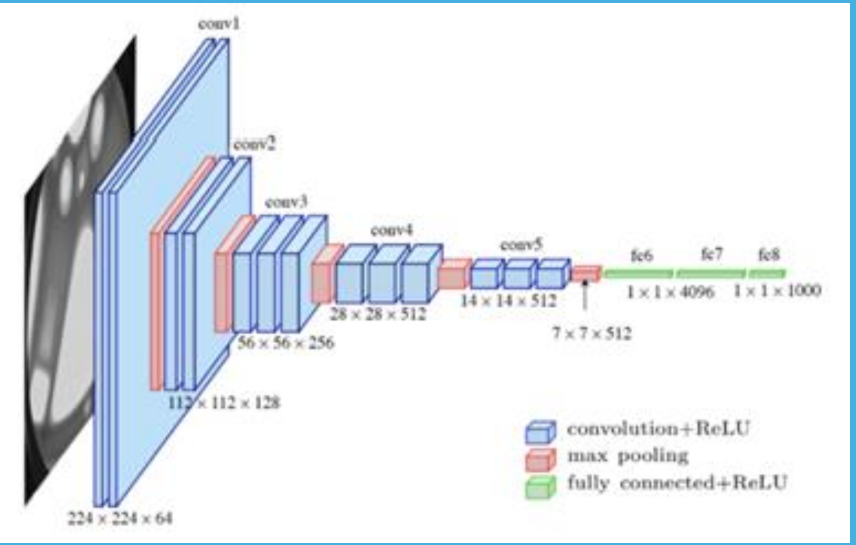
```
Sample Predictions and True Labels:
True Label: 1, Predicted: [1]
True Label: 1, Predicted: [1]
True Label: 1, Predicted: [0]
True Label: 1, Predicted: [1]
True Label: 0, Predicted: [0]
True Label: 0, Predicted: [1]
True Label: 0, Predicted: [0]
True Label: 1, Predicted: [0]
True Label: 0, Predicted: [1]
True Label: 0, Predicted: [1]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.93      0.87      5492
           1       0.92      0.77      0.84      5413

    accuracy                           0.86     10905
   macro avg       0.86      0.85      0.85     10905
weighted avg       0.86      0.86      0.85     10905

Confusion Matrix:
[[5132  360]
 [1218 4195]]
```

02. CNN

But it was just the beginning of our tries.

As a matter of fact, to see how far we could have gone, we decided to use our dataset to an already made model: InceptionV3.



We obtained a 99% of accuracy but encountered few problems along the way.

Training InceptionV3 is time-consuming and demands a certain amount of computational power.

For this reason, we used ColabPro which guarantees a more powerful GPU.

02. CNN

```
Total params: 23,903,010 (91.18 MB)

Trainable params: 23,868,578 (91.05 MB)

Non-trainable params: 34,432 (134.50 KB)
```

InceptionV3 is a model with 23mln trainable parameters.

We encountered a significant issue: each training epoch, even with the most powerful GPU available on ColabPro, took around 10 minutes. This resulted in over 1.5 hours for just one training session.

```
4376/4376 ━━━━━━━━━━━━━━━━━━━━ 701s 142ms/step - accuracy: 0.9436 - loss: 0.1487 - val_accuracy: 0.9607 - val_loss: 0.1020
```

Test evaluation: the test accuracy is nearly10% higher.

```
341/341 ━━━━━━━━━━━━━━━━━━━━ 43s 79ms/step - accuracy: 0.9046 - loss: 0.4563
Test Loss: 0.4639233350753784
Test Accuracy: 0.9033470749855042
```

03. VIDEO CLASSIFICATION

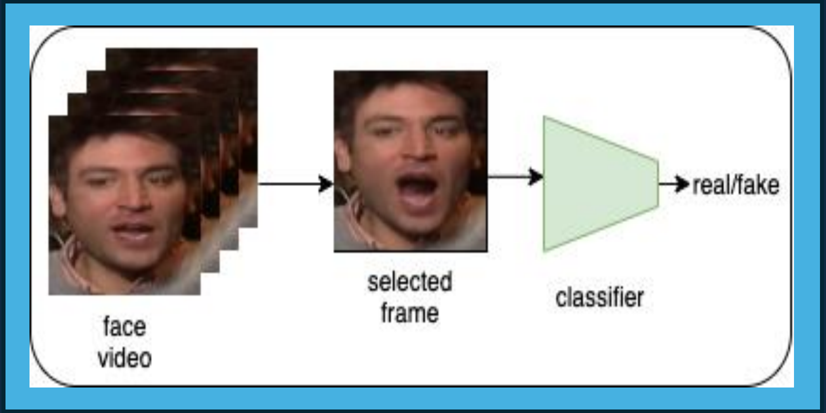
Ca' Foscari University of Venice

But let's cut to the chase and see which results we can achieve by classifying videos.

The dataset is divided into three folders, each of them containing 50%-50% of real-fake videos of 256x256 pixels.

Powerful machines could have worked with any dimension videos, but ours not! :)

For this reason, we downscaled them to 64x64.



03. VIDEO CLASSIFICATION

Ca' Foscari University of Venice

Dimensionality has been a though problem.

Objects storing all the data required for training were too heavy for the RAM to handle.

Data generators extracting data "on-the-fly" is lighter on the RAM, but it's slow during runtime.

To obtain good results in a "small" amount of time, we pre-process data beforehand.

```python
class VideoDataGenerator(Sequence):
    def __init__(self, data_dir, subset, batch_size=BATCH_SIZE):
        self.data_dir = data_dir
        self.subset = subset
        self.batch_size = batch_size
        self.classes = ['real', 'fake']
        self.videos = self._get_video_paths()
        self.on_epoch_end()

    def _get_video_paths(self):
        videos = []
        subset_dir = os.path.join(self.data_dir, self.subset)
        for class_name in self.classes:
            class_dir = os.path.join(subset_dir, class_name)
            for video_name in os.listdir(class_dir):
                if video_name.endswith('.npy'):
                    videos.append((os.path.join(class_dir, video_name), self.classes.index(class_name)))
        return videos

    def __len__(self):
        return len(self.videos) // self.batch_size

    def __getitem__(self, idx):
        batch_videos = self.videos[idx * self.batch_size:(idx + 1) * self.batch_size]
        batch_frames = []
        batch_labels = []

        for video_path, label in batch_videos:
            frames = np.load(video_path)
            batch_frames.append(frames)
            batch_labels.append(label)

        return np.array(batch_frames), np.array(batch_labels)

    def on_epoch_end(self):
        np.random.shuffle(self.videos)


train_generator = VideoDataGenerator(PROCESSED_DIR, "Train")
val_generator = VideoDataGenerator(PROCESSED_DIR, "Val")
test_generator = VideoDataGenerator(PROCESSED_DIR, "Test")
```

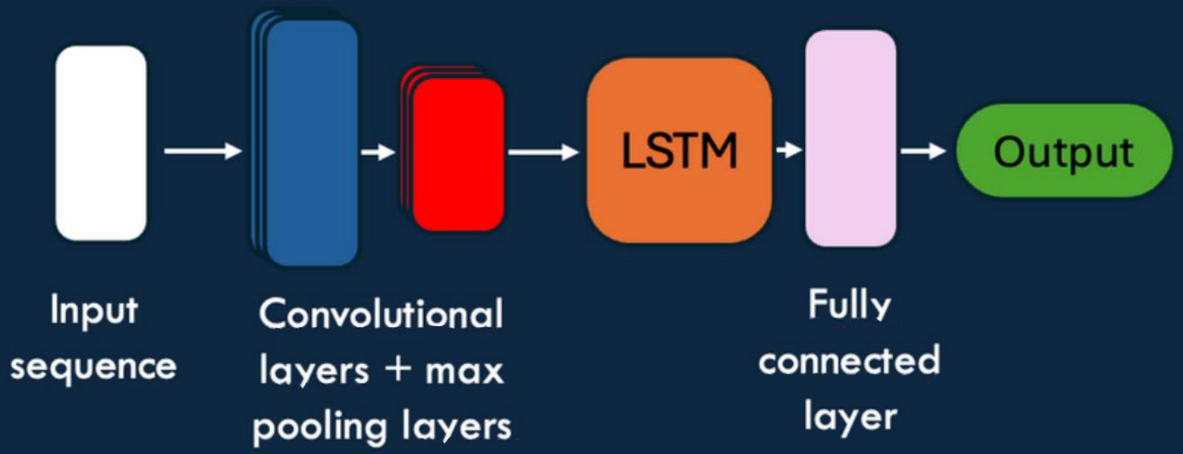
03. VIDEO CLASSIFICATION

Ca' Foscari University of Venice

Now that videos are involved, it's time to introduce something new!

First of all, we decided to make our CNN even simpler, getting better results surprisingly!

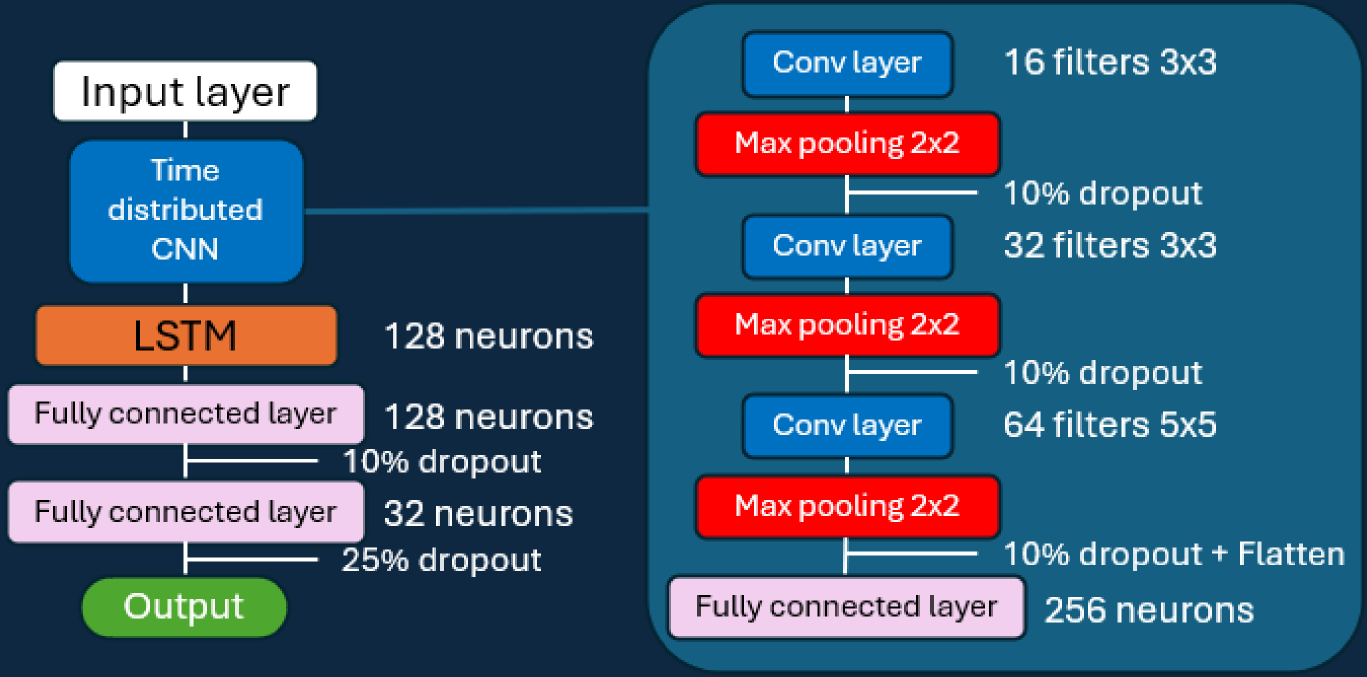
Once ultimated our CNN, we introduced to it a TimeDistributed layer to feed all the frames from each video, followed by an LSTM approach to extract features from the whole video.



03. VIDEO CLASSIFICATION

Initially we were led to overfitting due to an overly complex model. But once simplified it, the model performed well.

Here you can see a more detailed representation of the model that led us to the best results.



03. VIDEO CLASSIFICATION

Ca' Foscari University of Venice

```
437/437 ━━━━━━━━━━━━━━━━━━━━ 302s 652ms/step - accuracy: 0.5198 - loss: 0.7172 - val_accuracy: 0.6324 - val_loss: 0.6407
Epoch 2/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 178s 405ms/step - accuracy: 0.6967 - loss: 0.5843 - val_accuracy: 0.8659 - val_loss: 0.3252
Epoch 3/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 169s 384ms/step - accuracy: 0.8778 - loss: 0.3046 - val_accuracy: 0.9278 - val_loss: 0.1959
Epoch 4/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 172s 392ms/step - accuracy: 0.9371 - loss: 0.1756 - val_accuracy: 0.9425 - val_loss: 0.1718
Epoch 5/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 171s 389ms/step - accuracy: 0.9563 - loss: 0.1280 - val_accuracy: 0.9694 - val_loss: 0.1038
Epoch 6/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 170s 387ms/step - accuracy: 0.9658 - loss: 0.0992 - val_accuracy: 0.9748 - val_loss: 0.0816
Epoch 7/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 168s 382ms/step - accuracy: 0.9751 - loss: 0.0770 - val_accuracy: 0.9765 - val_loss: 0.0884
Epoch 8/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 167s 378ms/step - accuracy: 0.9817 - loss: 0.0660 - val_accuracy: 0.9802 - val_loss: 0.0706
Epoch 9/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 170s 387ms/step - accuracy: 0.9814 - loss: 0.0654 - val_accuracy: 0.9755 - val_loss: 0.0910
Epoch 10/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 174s 396ms/step - accuracy: 0.9847 - loss: 0.0550 - val_accuracy: 0.9688 - val_loss: 0.1074
Epoch 11/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 166s 376ms/step - accuracy: 0.9879 - loss: 0.0516 - val_accuracy: 0.9758 - val_loss: 0.0852
Epoch 12/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 166s 378ms/step - accuracy: 0.9877 - loss: 0.0501 - val_accuracy: 0.9745 - val_loss: 0.0978
Epoch 13/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 165s 375ms/step - accuracy: 0.9862 - loss: 0.0495 - val_accuracy: 0.9842 - val_loss: 0.0597
...
Epoch 39/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 165s 374ms/step - accuracy: 0.9974 - loss: 0.0233 - val_accuracy: 0.9899 - val_loss: 0.0539
Epoch 40/40
437/437 ━━━━━━━━━━━━━━━━━━━━ 165s 376ms/step - accuracy: 0.9967 - loss: 0.0274 - val_accuracy: 0.9849 - val_loss: 0.0818
```

```
93/93 ━━━━━━━━━━━━━━━━━━━━ 17s 186ms/step - accuracy: 0.9892 - loss: 0.0693
Test Loss: 0.08829152584075928
Test Accuracy: 0.9872311949729919
```

Here, some results!

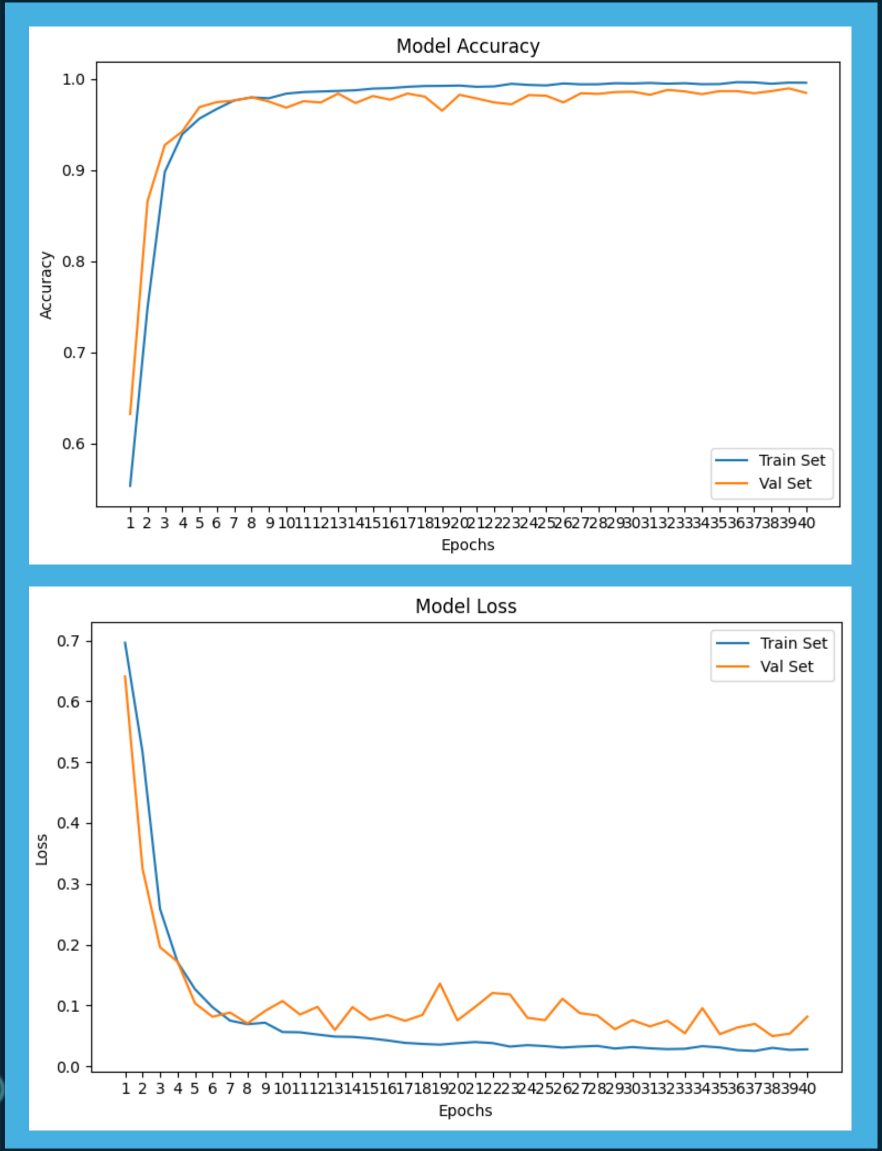
You can see how fast the model reaches the elbow – look at the fall of the loss value!

In the last epochs, the accuracy on the training set nearly reaches 100%, with the validation set accuracy showing similarly high values. This indicates a strong model performance.

Proof of this, is the test set accuracy!

03. VIDEO CLASSIFICATION

Ca' Foscari University of Venice



On the left, a representation of what we were saying previously.

Below, the confusion matrix and the summary table provide a more detailed breakdown of the test results.

```
Confusion Matrix:
[[1452   33]
 [   4 1487]]
Classification Report:
              precision    recall  f1-score   support

        fake       1.00      0.98      0.99      1485
        real       0.98      1.00      0.99      1491

    accuracy                           0.99      2976
   macro avg       0.99      0.99      0.99      2976
weighted avg       0.99      0.99      0.99      2976
```

04. A DIFFERENT APPROACH: PYTORCH

So far, all our models were built using Tensorflow and Keras.

PyTorch is a different deep learning framework presenting few differences, primarily in its dynamic nature.

On the other hand, we get better results using the more static Tensorflow and Keras.

For the sake of knowledge, we replicated using PyTorch what already done.

Let’s have a look!

Ca' Foscari University of Venice

04. A DIFFERENT APPROACH: PYTORCH

PyTorch lacks a predefined TimeDistributed class.

The first challenge was to implement it ourselves.

```python
# defining the time distributed class
# returns an embedding for each temporal slice of the input

class TimeDistributed(nn.Module):
    '''
    This object takes a layer or a sequence of layers and applies it to each individual temporal slice of an input.
    It returns the embedding produced by the given layers for each temporal slice for each data point in the batch
    '''
    def __init__(self,layer):
        super(TimeDistributed,self).__init__()
        self.layer = layer

    def forward(self, x):
        batch_size, t_slices , *slice_dims = x.shape
        x = x.reshape(batch_size*t_slices, *slice_dims)
        y = self.layer(x)
        output_batches, *embedding_shape = y.shape
        assert output_batches == batch_size * t_slices, f"Wrong Number of batches in the output: the layers inside TimeDistributed should output
        {batch_size * t_slices} batches, but got {output_batches} instead"
        y = y.reshape((batch_size, t_slices, *embedding_shape))
        return y
```

The TimeDistributed layer processes multidimensional temporal data by applying the same network in parallel to each slice of the sequence.

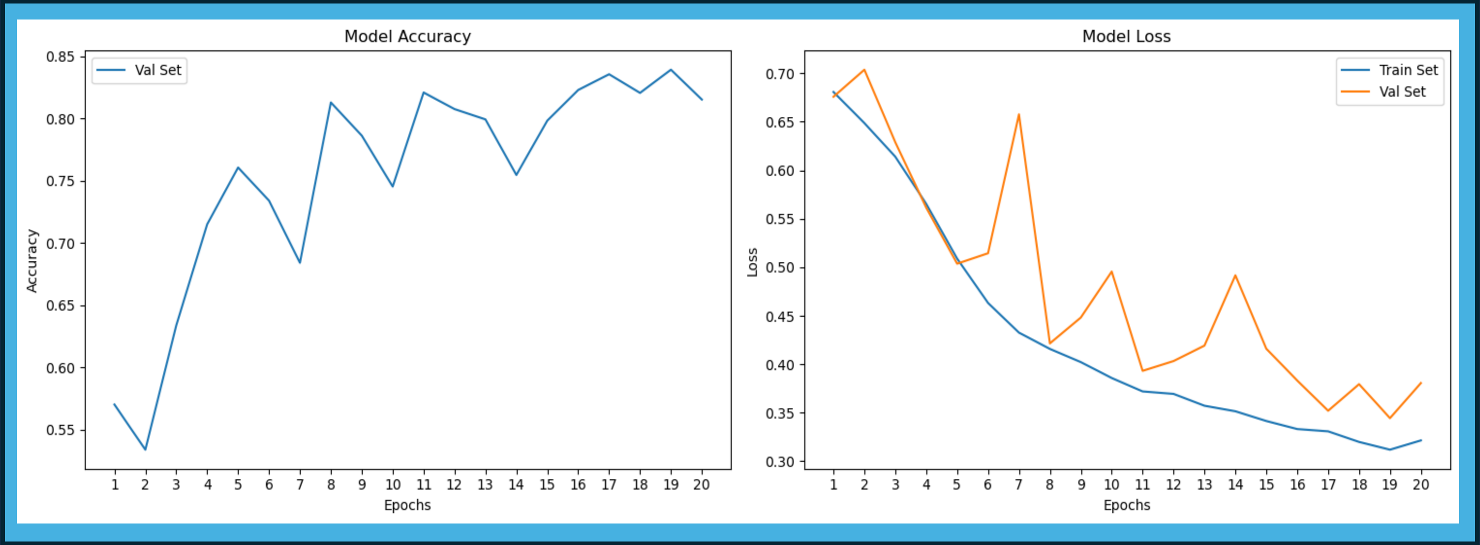
The output are then reshaped to preserve the original temporal data.

Ca' Foscari University of Venice

04. A DIFFERENT APPROACH: PYTORCH

```
Validation accuracy: 0.8153333333333334
Area under the curve of validation set: 0.9533848888888888
Confusion matrix on validation set with threshold 0.5
 [[ 985  515]
 [  39 1461]]
Classification Report on validation set, t = 0.5
              precision    recall  f1-score   support

         0.0       0.96      0.66      0.78      1500
         1.0       0.74      0.97      0.84      1500

    accuracy                           0.82      3000
   macro avg       0.85      0.82      0.81      3000
weighted avg       0.85      0.82      0.81      3000
```

Firstly, we trained a CNN on the image dataset, then “froze” it and placed it into a TimeDistributed layer, feeding it into an LSTM similar to our successful TensorFlow model.



The LSTM parameters were trained using the preprocessed videos.

Ca' Foscari University of Venice

04. A DIFFERENT APPROACH: PYTORCH

```
number of parameter:  552865
t_dist_cnn.layer.0.weight: torch.float32, params: 432
t_dist_cnn.layer.0.bias: torch.float32, params: 16
t_dist_cnn.layer.4.weight: torch.float32, params: 4608
t_dist_cnn.layer.4.bias: torch.float32, params: 32
t_dist_cnn.layer.8.weight: torch.float32, params: 51200
t_dist_cnn.layer.8.bias: torch.float32, params: 64
t_dist_cnn.layer.13.weight: torch.float32, params: 409600
t_dist_cnn.layer.13.bias: torch.float32, params: 256
rnn.weight_ih_l0: torch.float32, params: 65536
rnn.weight_hh_l0: torch.float32, params: 16384
rnn.bias_ih_l0: torch.float32, params: 256
rnn.bias_hh_l0: torch.float32, params: 256
sequence.0.weight: torch.float32, params: 4096
sequence.0.bias: torch.float32, params: 64
sequence.3.weight: torch.float32, params: 64
sequence.3.bias: torch.float32, params: 1
```

Though capable of detecting deepfakes, the previous method delivered results clearly inferior to our TensorFlow implementation.

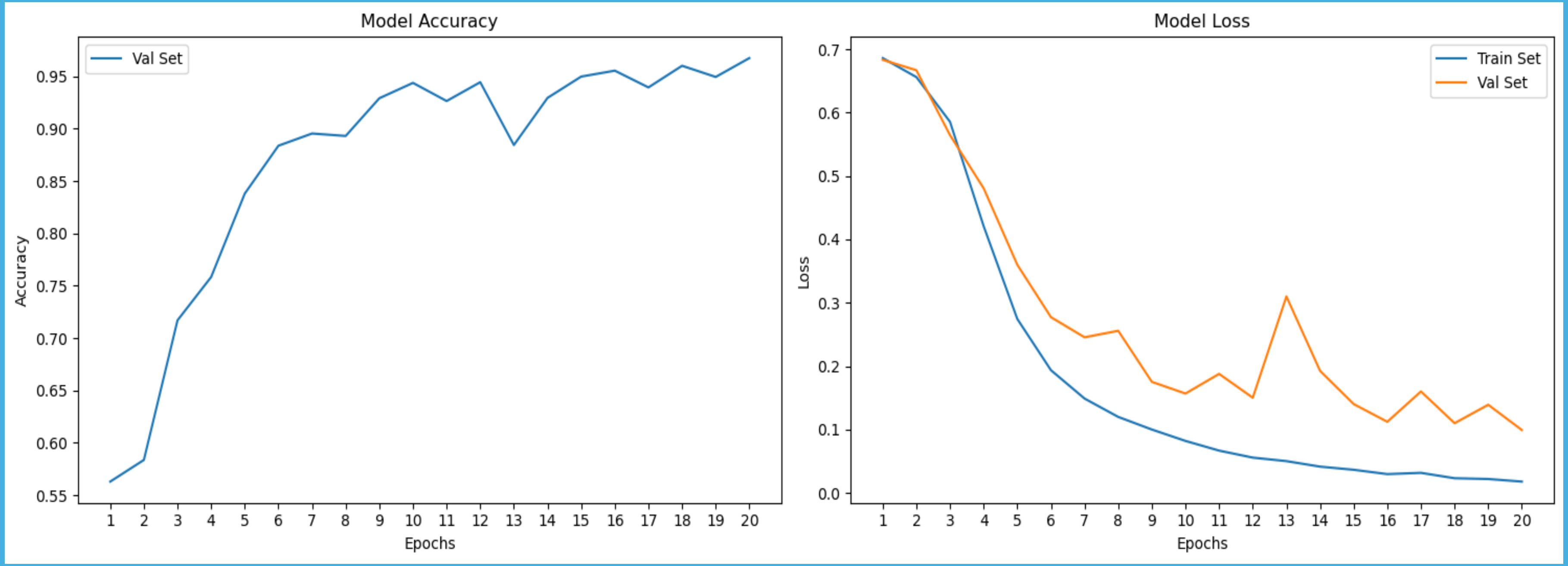
We replicated our first approach: training the CNN and the LSTM together on the vide dataset.

```
----------------------------------------|EPOCH 1|----------------------------------------
Training loop: 438it [02:15,  3.24it/s]
Train Loss: 0.68595 | Progress: [438/438]
Validation Loop: 100%|██████████| 94/94 [00:27<00:00,  3.41it/s]
Validation Accuracy 56.30% | Validation Loss 0.683177 | Train Loss 0.68595
```

```
----------------------------------------|EPOCH 19|----------------------------------------
Training loop: 438it [00:57,  7.58it/s]
Train Loss: 0.02200 | Progress: [438/438]
Validation Loop: 100%|██████████| 94/94 [00:10<00:00,  8.63it/s]
Validation Accuracy 94.93% | Validation Loss 0.139116 | Train Loss 0.02200
----------------------------------------|EPOCH 20|----------------------------------------
Training loop: 438it [00:58,  7.44it/s]
Train Loss: 0.01792 | Progress: [438/438]
Validation Loop: 100%|██████████| 94/94 [00:10<00:00,  8.89it/s]
Validation Accuracy 96.73% | Validation Loss 0.099268 | Train Loss 0.01792
```

We noticed a slight drop in accuracy compared to TensorFlow, likely due to differences in optimizer tuning.

04. A DIFFERENT APPROACH: PYTORCH



Training the CNN simultaneously with the LSTM yielded better results compared to using a pretrained CNN. However, both approaches are effective at detecting deepfake videos.

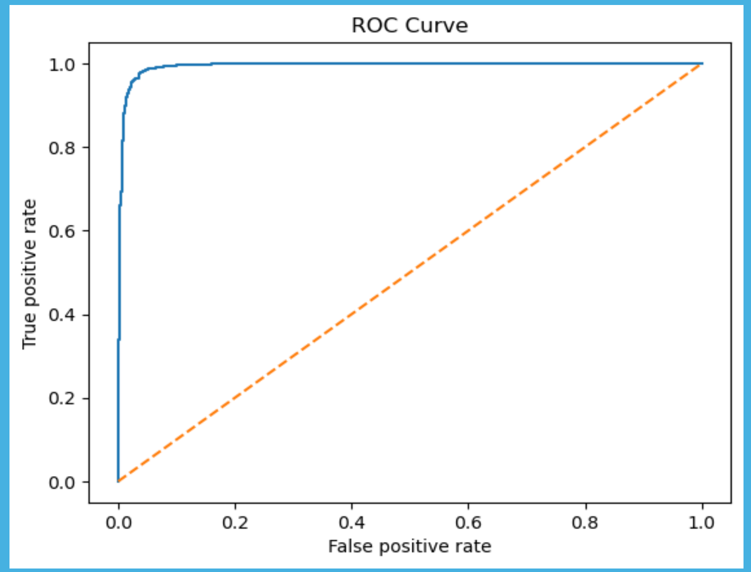
Ca’ Foscari University of Venice

04. A DIFFERENT APPROACH: PYTORCH

This model demonstrates nearly identical performance to the model we developed using TensorFlow.

The high AUC score highlights its strong ability to discriminate between the classes.

```
Validation accuracy: 0.9673333333333334
Area under the curve of validation set: 0.9938391111111111
Confusion matrix on validation set with threshold 0.5
 [[1422   78]
 [  20 1480]]
Classification Report on validation set, t = 0.5
              precision    recall  f1-score   support

         0.0       0.99      0.95      0.97      1500
         1.0       0.95      0.99      0.97      1500

    accuracy                           0.97      3000
   macro avg       0.97      0.97      0.97      3000
weighted avg       0.97      0.97      0.97      3000
```



THANK YOU FOR YOUR ATTENTION!
from GLEM

Eleonora Marcassa	900571
Giuseppe Massidda	901029
Lorenzo Muscillo	1000916
Manuel Trevisan	886937