Lambton College

Big Data Analytics

BDM 3035 - Big Data Capstone Project

PhD. Meysam Effati

Mississauga, Ontario

July 16th, 2024

Deep Face Detection Model – Milestone Report # 4

- Brayan Leonardo Gil Guevara (C0902422)
- Eduardo Williams (C0896405)
- Marzieh Mohammadi Kokaneh (C089839)
- Rohit Kumar (C0895100)
- Saurabh Laltaprasad Gangwar (C0894380)

Contents

Figures

Tables

Introduction

The face recognition industry has been crucial for security purposes over the years, however, once COVID hit in late 2019, more requirements for this technology have been raised.

This document explores the use of deep learning for facial recognition. The Idea comes from creating a local dataset having different photos of people on a team and training the model to detect the identity of each person. We will discuss how to collect and prepare data, as well as develop models for this combined task. The target for this project is security companies for letting people to entrance the office or not.

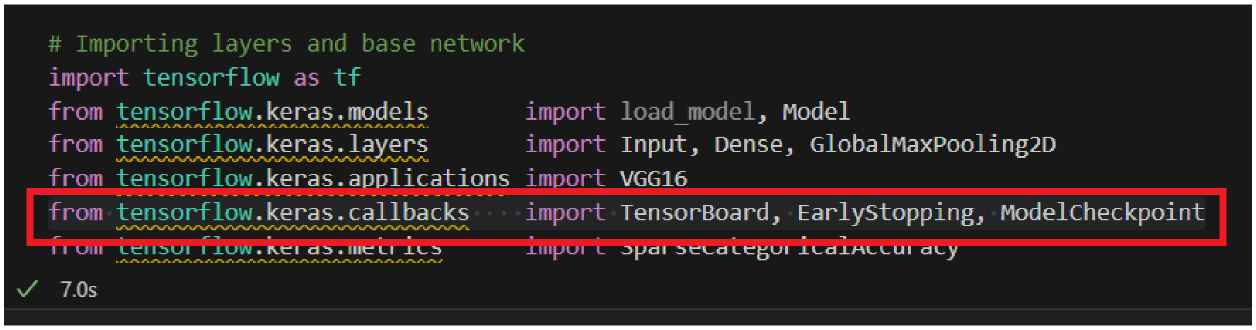
In this delivery we are approaching in topics such as: model fitting, callbacks and metrics evaluations, as well as several tests done in the test dataset.

Progress Report

In previous reports we detailed how data (pictures) was collected and labeled. Also, how we performed the augmentation process for having a heterogeneous dataset. Additionally, we reported about the model definition and the model fitting customized class, as well as classification and regression loss calculations methods. In this delivery we are going to deep in the model fitting process, callbacks and metrics evaluations in train, validation and test datasets.

Callbacks

Callbacks are objects which can modify the model fitting process according to their set parameters. In this way we can extend the range of adjustments we can apply to this stage for controlling and getting better results. In order to use callbacks, we have to import them from Tensorflow library, exactly from **tensorflow.keras.callbacks** package.



```
# Importing layers and base network
import tensorflow as tf
from tensorflow.keras.models       import load_model, Model
from tensorflow.keras.layers       import Input, Dense, GlobalMaxPooling2D
from tensorflow.keras.applications import VGG16
from tensorflow.keras.callbacks    import TensorBoard, EarlyStopping, ModelCheckpoint
from tensorflow.keras.metrics      import SparseCategoricalAccuracy
✓ 7.0s
```

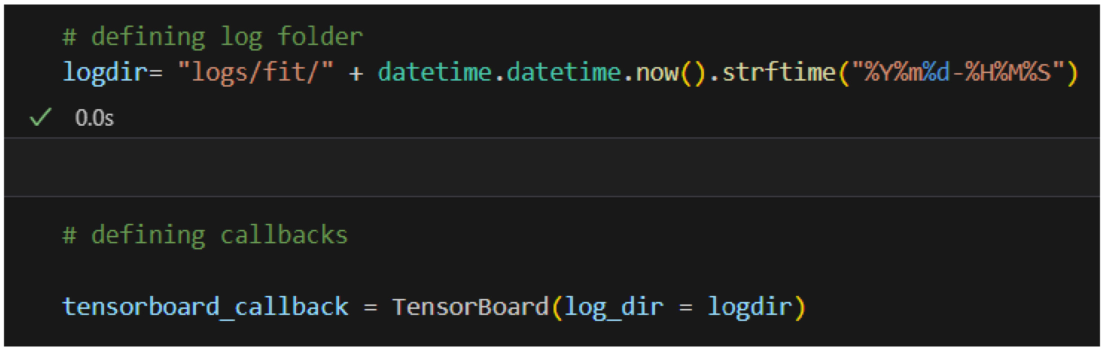
Figure 1 Callbacks import

TensorBoard

TensorBoard is a library designed to work with tensors. The goal of TensorBoard is to log metrics while model fitting so as to show them later through several kinds of visualizations and dashboards.

According to tensorflow.org “TensorBoard is a tool for providing the measurements and visualizations needed during the machine learning workflow. It enables tracking experiment metrics like loss and accuracy, visualizing the model graph, projecting embeddings to a lower dimensional space, and much more.”

We are not using TensorBoard for generating visualizations in this project, however we are using it for logging the model fitting process. Therefore, we are using it as shown in Figure 2.



```
# defining log folder
logdir= "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
```

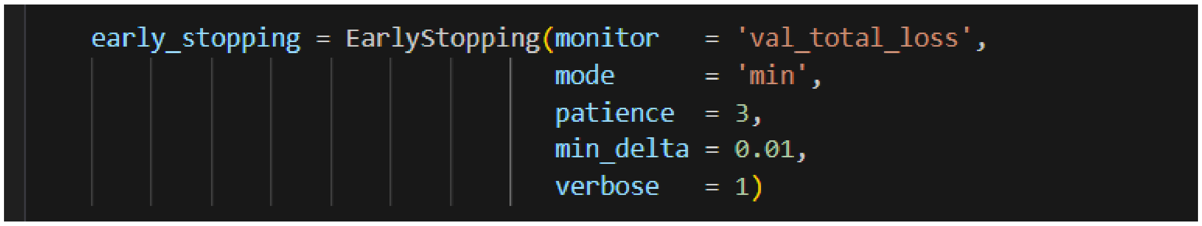
```
# defining callbacks

tensorboard_callback = TensorBoard(log_dir = logdir)
```

Figure 2 TensorBoard definition

In the previous code, we are just defining a path where files logs are going to save. The name path includes a dynamic part which is made up by the date and time of the process. Then, we assigned that file path to the TensorBoard callback through the **log_dir** parameter.

EarlyStopping

EarlyStopping is another very helpful callback, whereby we can stop the fitting process if metrics are not getting better. The main objective is to avoid overfitting, but additionally it has other benefits, for example saving time and processing resources.



```
early_stopping = EarlyStopping(monitor   = 'val_total_loss',
                               mode      = 'min',
                               patience  = 3,
                               min_delta = 0.01,
                               verbose   = 1)
```

Figure 3 EarlyStopping definition

EarlyStopping implementation is through some parameters detailed in the next table:

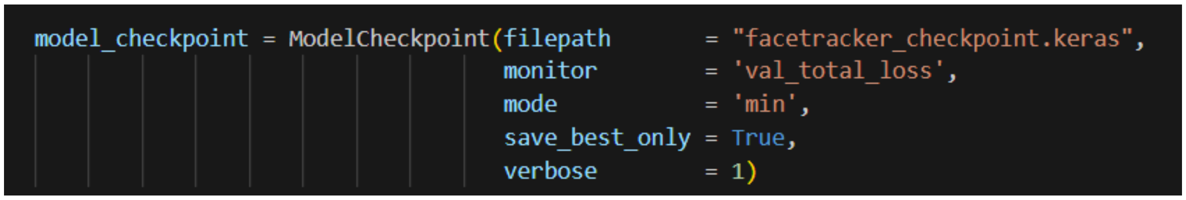
Parameter	Description
monitor	This parameter specifies which metric is going to be monitored. In this case we are monitoring total loss (class loss + regress loss) for validation dataset.
mode	Through mode we can specify if increased or decreased of the monitored metric has to be considered. In order words, if it has to take the metric when it is in the minimum or maximum value.
patience	It indicates how many epochs EarlyStopping has to wait to consider that the monitored metric is not getting better.
min_delta	Minimum change in the monitored quantity to qualify as an improvement.
verbose	It just shows the step status in screen

Table 1 EarlyStopping parameters

ModelCheckpoint

ModelCheckpoint is another callback object which goes hand in hand with EarlyStopping. ModelCheckpoint focuses on monitoring metrics in order to compare them and stablish which are the best. In addition, it saves models with these metrics so that we can recover them later for saving the final model.

ModelCheckpoint is a better alternative than using **restore_best_weights** parameter in EarlyStopping object, since in some cases, when EarlyStopping does not stop the training and the process runs until the last epoch, EarlyStopping will not restore the best fit and your model will end up with the weights from the last epoch, not with the best.



```
model_checkpoint = ModelCheckpoint(filepath       = "facetracker_checkpoint.keras",
                                   monitor        = 'val_total_loss',
                                   mode           = 'min',
                                   save_best_only = True,
                                   verbose        = 1)
```

Figure 4 ModelCheckpoint definition

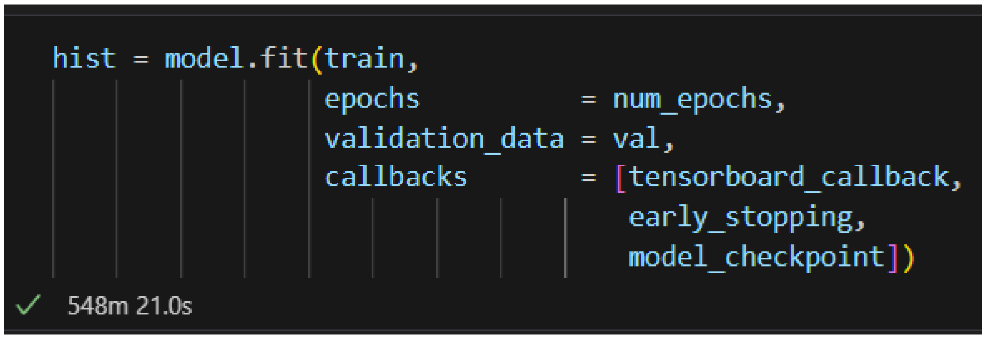
ModelCheckpoint has to be in tune with EarlyStopping in terms of the monitored metric, it means, monitor and mode parameters should have the same values. Table 2 contains ModelCheckpoint parameters details:

Parameter	Description
monitor	This parameter specifies which metric is going to be monitored. In this case we are monitoring total loss (class loss + regress loss) for validation dataset.
mode	Through mode we can specify if increased or decreased of the monitored metric has to be considered. In order words, if it has to take the metric when it is in the minimum or maximum value.
filepath	Model filename for saving
save_best_only	It just will save when the model is considered the best
verbose	It just shows the step status in screen

Table 2 ModelCheckpoint parameters

Model fitting

Model fitting process was covered by the 3rd milestone report, however we improved this process by adding callbacks to it.



```
hist = model.fit(train,
                 epochs          = num_epochs,
                 validation_data = val,
                 callbacks       = [tensorboard_callback,
                                    early_stopping,
                                    model_checkpoint])
```

✓ 548m 21.0s

Figure 5 Model fitting definition

Use of callbacks, especially EarlyStopping, allowed us to save time and processing resources, since we started running 50 epochs which reached to take up to more than 24 hours. Nowadays, the model fitting process takes about 5 hours with a dataset made up of 24,000 pictures between train and validation.

Parameter	Description
x, y	Input data for training
epochs	Number of epochs to train the model. An epoch is an iteration over the entire x and y provided dataset.
validation_data	Validation dataset
callbacks	List of callback objects

Table 3 Model fitting parameters

Metrics

Next table shows metrics results for train and validation datasets after last epoch:

Metric	Train dataset	Validation dataset
Accuracy	100.0000%	100.0000%
Classification loss	0.0040%	0.0897%
Regression loss	0.1500%	3.6200%
Total loss	0.1540%	3.7097%

Table 4 Train and validation datasets metrics

As we mentioned, we did not use TensorBoard for visualizations. Instead, we are drawing basic line charts for showing the model fitting performance. It means, classification and regression loss, and the accuracy history.

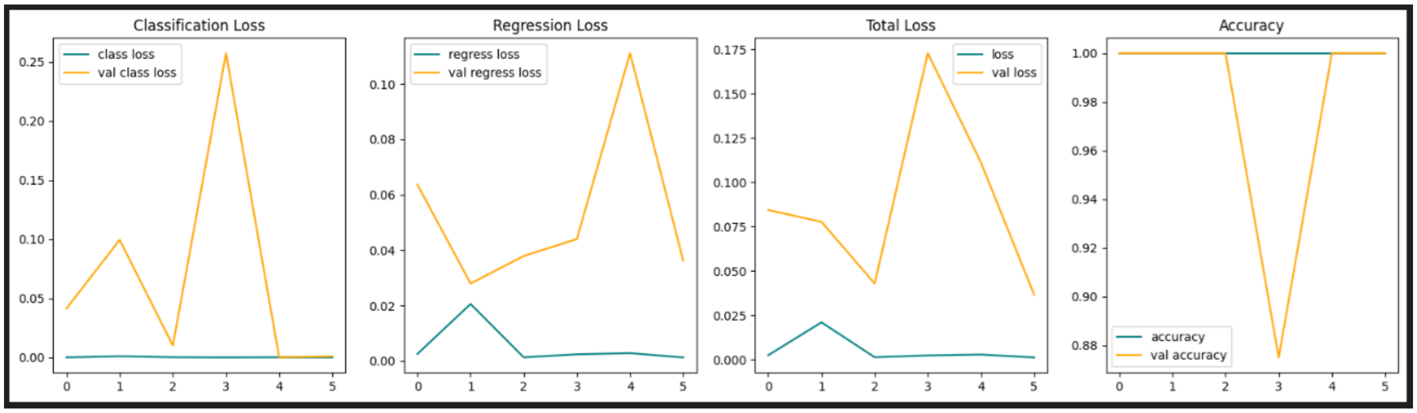


Figure 6 Model fitting performance history

In addition to these metrics, we use one additional metric called Intersection over Union on test dataset.

Intersection over Union

Intersection over Union (IoU) is used to measure the accuracy of object detection by comparing the ground-truth bounding box to the predicted bounding box. Therefore, we defined a function for this calculation.

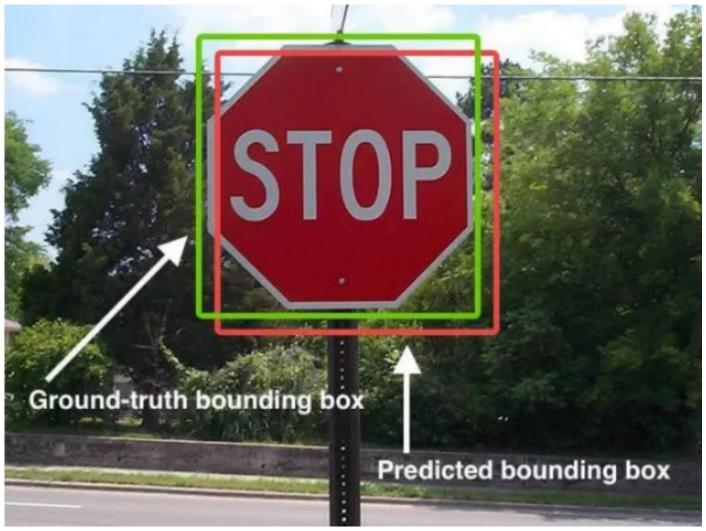


Figure 7 Intersection over Union

Source: (pyimagesearch)

Since test dataset contained 4,000 pictures, we defined an iterator for assessing the accuracy and the Intersection over Union for each batch as well as for the entire dataset. Next table shows obtained metrics on test dataset:

Metric	Value
Accuracy	96.53%
Intersection over Union	89.94%

Table 5 Test dataset metrics

Additionally, we graphed some pictures from test dataset, marking ground-truth bounding box (green) and predicted bounding box (red).

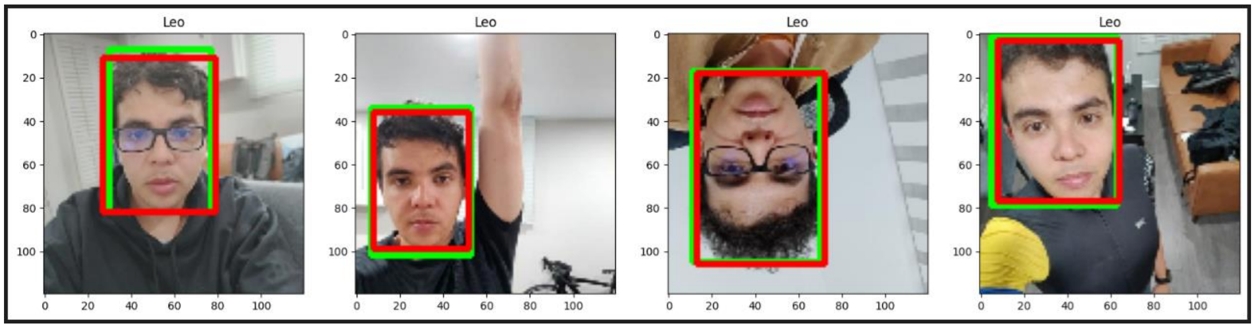


Figure 8 Ground-truth and predicted bounding box

Next Steps

Once the model was fitted and metrics evaluated over the test dataset were good, next steps are related to model building, saving and testing with unseen pictures. Additionally, we have to work in final deliveries such as: demo, presentation and documentation.

Faced Challenges

We faced several challenges, specifically defining and testing parameters values for both callbacks and fitting procedures. Since each change involved re-fit the model and it took a lot of time to get results.

Lessons Learned

We have learnt different helpful techniques in this stage. We have learnt a little about **Tensorboard**, whose purpose is to log metrics during training in order to show them later through dashboards and other kinds of visualizations. Also, we have learnt about **EarlyStopping** which is a wonderful and very useful technique to avoid the necessity of performing all defined epochs when metrics are not getting better anymore. In addition, we have learnt about **ModelCheckpoint** which identifies the best metrics and automatically saves the model with those metrics, from which we can recover weights later in order to save the definitive model. We learnt about **restore_best_weights** parameter in Early Stopping and the difference between using it and using ModelCheckpoint. Finally, we have learnt about **Intersection over Union** evaluation, which is a reliable metric to assess the accuracy between ground-truth and predicted bounding box.

Conclusion

As we move forward, we acquire new knowledge and understanding about machine learning and deep learning too. Callbacks are helpful objects which allows us to expand a lot of possibilities to customize the model fitting process. In addition, Intersection over Union give us a better idea of labeling predictions accuracy that we can even represent graphically. We are looking forward to face and complete next steps so as to get the final product.

References

pyimagesearch. (n.d.). *Intersection over Union (IoU) for object detection*. Retrieved from pyimagesearch: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

tensorflow.org. (n.d.). *Get started with TensorBoard*. Retrieved from Tensorflow: https://www.tensorflow.org/tensorboard/get_started