HOOPSTER

Granata Annalaura 1999328
Matrullo Zoe 1985908
Rodriguez Dionigi 2006057
Schulz Kilian 1986842

The idea

Every morning our group member Kilian goes to the park in order to make 20 free throws. He rarely manages to properly count how many attempts it takes him, since his mathematical capabilities are already atrocious, and definitely don't improve while shooting basketballs. Knowing this struggle during a coffee break right after AI Lab class and we had an idea:

Why not help Kilian improve his subpar free throw skills and have an exciting and interesting project for AI Lab at the same time?

Our goal was to make an app, tracking made and missed basketball shots in such an elegant and efficient way, that it could be run even on Kilian's rather budget friendly Xiaomi phone.

A quick google search revealed, that we would be the first ones ever to manage something like this, but with a lack of understanding of how hard this was going to turn out and the arrogance of a young computer science student, we didn't think much of it and set out to do exactly that:

A **lightweight mobile application, that counts makes and misses on free throws** and can be run on any phone that doesn't burst into flames when loading Angry Birds.

Overview of the implementation

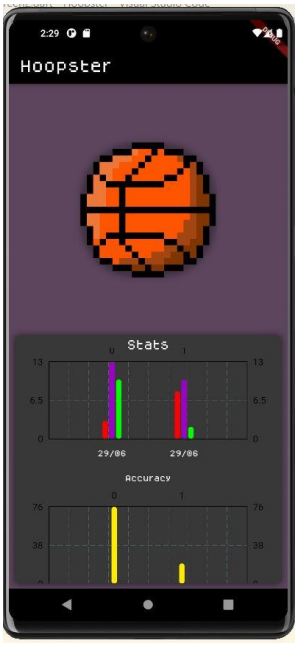
The app is build upon the following major components:

- **An object tracking model** that **recognizes** the **ball and the hoop**
- **detection** of successful **shots** and **misses**
- **a recording screen** to track your shooting session
- **A user-friendly graphical user interface**
- A **statistics section** to track shooting progress over time

Our purpose is providing these elements in a **light-weight offline app**, in order to be supported by mobile devices.

The mobile app

Before explaining the functioning of *Hoopster*'s computer vision approach, let's briefly go through the UI.
The mobile app is built using the Flutter framework, as it allows running ìboth on iOS and android.
It presents 2 main screens:



- a homescreen through which you can check your progress with the statistics
- a recording screen, the core of the app, to start a shooting session

A real life use of *Hoopster* requires a live recording of the shooting session, which is followed by instant feedback.
That is why we purposefully decided to build our project within a **mobile application**.
This allows to make the most of *Hoopster* with just a phone and a basketball, making the utilization frictionless for the user while simultaneously making our job harder (which we love as computer science students).

The challenge and our experiments

The **main challenge** is providing shot recognition while finding a good trade-off between speed and accuracy, through a model that is lightweight enough to run offline on a phone.
Given the magnitude of the challenge ahead of us, we experimented with various methods, in the quest for the best solution. Here we are going to mention only a few of them, mainly the ones that developed furthest.

During the creation of **Hoopster**, we followed two main approaches:

Approach A - ball recognition and shot detection with parabolic motion

The first step to determine whether a shot was attempted, is to identify a **parabolic motion in the movement of the ball** when throwing it towards the basket.
Our function, using three lists of numbers representing the time, x position, and y position of the object, **calculates the coefficients of a second-degree polynomial curve** that best fits the object's x and y position data, using the "parabolicFunc()" function to evaluate the curve for each time interval.
The function then calculates the **residuals** between the object's position data and the values predicted by the parabolic curve, and calculates the **sum of squared residuals**. It also calculates the **total sum of squares and the coefficient of determination (R-squared)**.
Finally, the function determines if the object followed a parabolic motion by **comparing the coefficient** of determination with a threshold value (0.95) and returns a boolean.
Now, we can finally determine whether a **shot was successful or not**. The function “ShotChecker” gets two lists as input representing the **coordinates of the vertices** of a circle (the **basket**) and of a quadrilateral (the **ball**).
After calculating the areas of the two figures, we compare them to determine the bigger one: if the **intersection** is bigger or equal to 95% of the bigger area, then the function returns True, False otherwise, consequently updating the respective counter.*
Here follow some of our experiments that follow this approach.

*We want to specify that the idea for intersection-based approach for shot validation was inspired by the successful results reported by the following paper: *Xu-Bo Fu, Shao-Long Yue, De-Yun Pan. Camera-based Basketball Scoring Detection Using Convolutional Neural Network. International Journal of Automation and Computing, vol. 18, no. 2, pp.266-276, 2021. https://doi.org/10.1007/s11633-020-1259-7.*

- ***Tensorflow model trained with azure and cloud computing.*** The latter was fundamental in order to handle long training time and ensure the recognition of ball and hoop. This model was a good start as it was precise and had a good recall but we decided to move forward with other methods as it was not mobile friendly. It had a training accuracy of 97 percent and a training recall of 91 percent. Its implementation was hindered by Flutter only accepting TFlite models, which we explored in the following.

- ***TFlite model (2 versions).*** We attempted a second model with the purpose of running on a mobile device.

Our first tflite model was also optimized using the "YOLO" algorithm, which I promise we did not just make up. The algorithm is perfect for our purposes, since a classical CNN algorithm grows exponentially with each pixel and thus gets very computation heavy. A YOLO, or "You only look once" algorithm on the other hand divides the image into a grid of boxes, and treats each bounding box with different weights as its own space where a basketball or hoop could be located. Here we managed slightly better values with a training accuracy of 85 percent and a training recall of 70 percent. The low recall percentage might be due to the fact that the ball can often "disappear" in the background and seems to be hard to find even to the human eye, from a single frame.

This training image, where the black ball basically disappears within the black jacket of the gentleman in the background is a great example of why the model has problems with recall.

Additionally, the slight improvement in speed obtained was not yet enough to fulfill its purpose. Unfortunately gravity is too fast for our model to catch up. Despite the results, we did not give up and attempted a second TFlite model, which presented more accuracy with respect to the previous versions. We again did not reach a good trade off between speed and accuracy as this attempt revealed itself slow, especially on mobile.

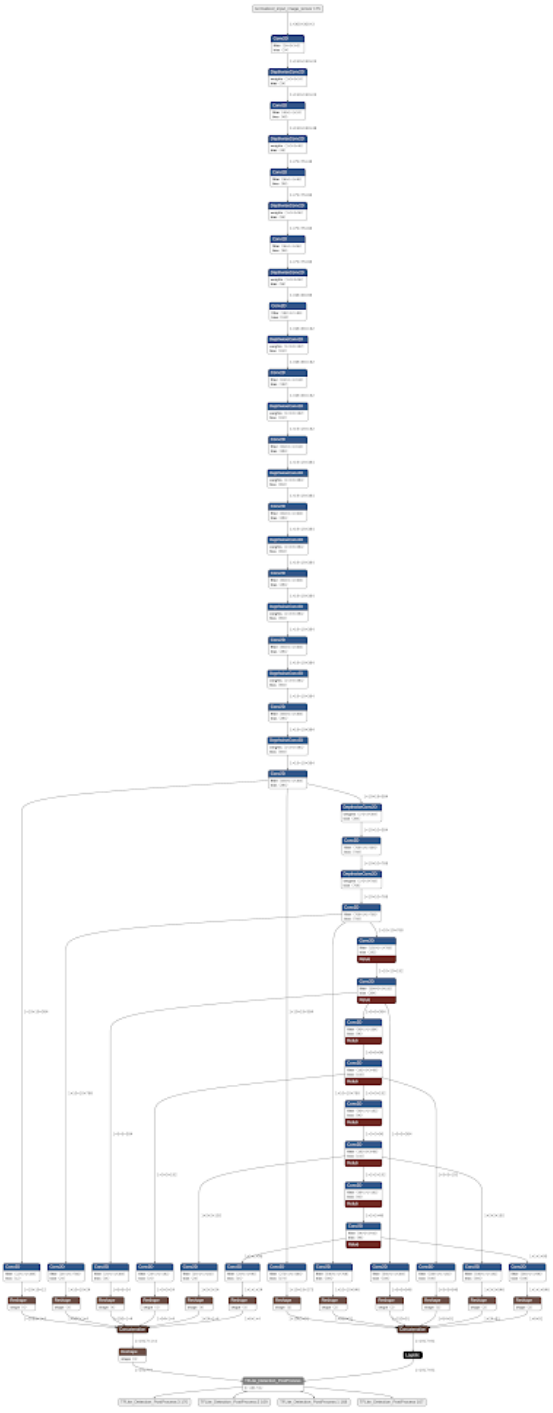
Approach B - ML classifier to hits and misses

At this point we decided to change our approach. Instead of working on recognizing the ball and therefore evaluating the shot, we focused directly on the shot and classifying its outcome through ML techniques, leaving momentarily aside the mobile app aspect.
Here follow our efforts following this approach.

- ***SKlearn model and first attempt with a simple Tensorflow NN.*** We first tried this new approach by training an SKlearn model using a series of labeled videos of successful and unsuccessful shots. As SKlearn is not optimized for videos, we opted to recreate this same idea but with Tensorflow. A simple Tensorflow NN was not suitable either since we learned that it's not optimal for sequential data like our videos.
- ***Tensorflow CNN in 3d.*** Even though it was a great accomplishment towards the goal of our project, the flaw of this experiment is the accuracy. After several tries, the best results we achieved where first a 58% and then a 64% of accuracy.

```
1/1 [==============================] - ETA: 0s - loss: 0.9331 - accuracy: 0.6452
1/1 [==============================] - 2s 2s/step - loss: 0.9331 - accuracy: 0.6452
C:\Python311\Lib\site-packages\keras\src\engine\training.py:3000: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format,
g. `model.save('my_model.keras')`.
  saving_api.save_model(
Accuracy: 0.6451612710952759

[Done] exited with code=0 in 1434.027 seconds
```

A representation of our multilayer CNN

Statistics

At the end of each playing session, on the app's home screen, will be visualized the **statistical data**.

Each object session saves and gets the data with the functions "saveAll()" and "getAll()" from the local storage by using the library "shared_preferences". Data will be later used by the class "statsObject" to build the graphs "shots" and "accuracy".

- **Shots:** histogram that, for each day in which the session happens, shows 3 bars representing: missed shots (in red), total number of tried shots (in violet) and successful shots (in green).

- **Accuracy:** histogram that, for each day in which the session happens, shows a bar representing the percentage of accuracy of your shot according to how many have entered the basket and how many not.

Based on our research, no app currently does what we envisioned for Hoopster. More specifically, all the mobile applications with this same purpose run all their computations online (e.g. the most successful app of this kind, Homecourt, requires online connection or allows for offline recording of sessions but provides their analysis only once given internet access).
To our knowledge, no one so far has managed to create a project of this kind that runs fully locally, and now we understand why.
We shot for the stars, driven by passion and creativity, which led us to face many challenges. Even though we did not completely manage to complete what we initially set out to do, we are happy to have persevered with our idea.
The harder things became, the more this project pushed us to experiment, thus learning and expanding our knowledge. We still decided to give it a shot and present the not perfect product of months of work because we still value the experience despite the final product. We are proud of each experiment of this project, especially the best-working ones. We hope that this report gives an overview of Hoopster's challenges and conveys how much effort and hard work we put in it.

Thank you for your attention.