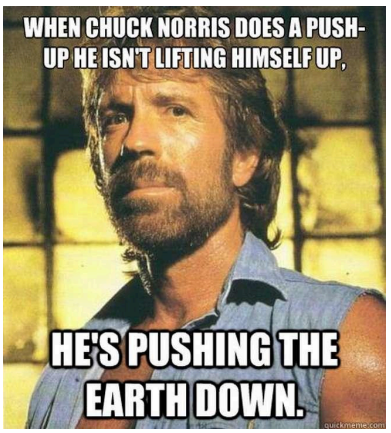
Advanced Computer Vision:
Partie 2 : Mode Projet - Jour 1

Mise en situation

Votre équipe a été contactée pour répondre à un appel d'offre d'une salle de sport 4.0 : **Better-Faster-Stronger.ai**. Celle-ci souhaite développer des systèmes de vision par ordinateurs pour proposer à ses adhérents un suivi personnalisé à domicile. Dans un premier temps, cette société souhaite développer une preuve de concept (POC), et si cela se montre satisfaisant de nouvelles missions vous seront proposées !

Voici le POC qu'ils vous demandent de réaliser:



"Nous aimerions étudier la possibilité d'automatiser le comptage du nombre de pompes que font nos clients à partir du flux vidéo de leur webcam."

Better-Faster-Stronger.ai

Modalités

- Travail en groupes de 3.
- Gestion de projet en s'inspirant des méthodes agiles.

Objectifs de la journée

- **Trouver un nom pour votre groupe et un nom pour le projet**.
- **Mettre en place un dépôt git** pour collaborer entre vous et livrer le code final au client. Le nom du dépôt devra être une composition du nom de votre groupe et de celui du projet (vous pouvez utiliser des acronymes) . Plus de détails sur la manière de procéder sont donnés dans la section "Recommandation" ci-dessous.

- **Explorer la base de code déjà existante** qui a habilement été stockée sur un notebook par vos prédécesseurs. Cette base vous guidera dans la mise en place d'un algorithme de détection/classification de poses, et de comptage sur un flux vidéo. Prenez le temps de comprendre le fonctionnement et l'importance de chaque fonction proposée.
- **Constituer une base de données annotées pour « entraîner » l'algorithme** avec quelques images de vous faisant des pompes.
- **Préparer une vidéo** démontrant la faisabilité d'un tel projet.
- **Optionnel J1 :** une démo live + un repo git structuré sans notebook.

Jalons jour #1

- 9h : constitution des groupes
- 9h15 : début de la première phase du projet
- 16h : livrable client (démo)
- 17h : concours de pompe

Recommandations

Prenez le temps de parcourir le code fourni et de comprendre les points principaux (nous pourrons prévoir des moments de remédiation) :

- la détection de pose avec la librairie Mediapipe
- l'algorithme de classification k-nn
- la normalisation et les embeddings adaptés pour votre cas d'usage
- l'importance de la constitution d'un dataset « propre »

Optionnel en 3 jours, mais pour votre culture

Collaboration et versionnage de votre code sur Github

Il vous est recommandé de mettre en place dès maintenant un repo github de groupe pour versionner le code de votre projet. Voici un rappel sur les étapes à suivre pour limiter les potentiels accros que vous pourrez rencontrer pour cette étape:

1. simplifiez-vous la vie en mettant en place un tunnel ssh entre votre poste de travail et le votre dépôt github qui vous permettra de ne pas avoir à renseigner votre identifiant/mot de passe à chaque push.
   Si vous avez besoin de vous rafraîchir la mémoire sur la manière de procéder, n'hésitez pas à vous référer au kit élève du cas d'étude #3 (projet end-to-end Docker), dans la partie I, section 1.1 *"Mise en place de l'environnement de développement"*.

2. Choisissez une personne de votre groupe qui sera le propriétaire du dépôt. Son rôle est de créer le projet et de s'assurer de sa bonne tenue (acceptations des pull requests, résolution de merge conflicts etc.). Il invitera les autres à collaborer sur le projet.

A partir de là deux méthodes pour collaborer sur git:

à la "open source"	**en mode entreprise**
Vision globale : Chaque collaborateur est indépendant, il créé un fork du projet : c'est un nouveau répo git, qui peut être en local, sur gitlab, sur github ou tout autre hébergeur. Il fait ses modifications, peut les publier indépendamment du projet source. Il peut proposer ses modifications au projet source (upstream) via des "*pull requests*".	*Vision globale* : Tout le monde travaille sur un seul et même répo git. On fonctionne avec des branches, certaines seront protégées et seul certains utilisateurs privilégiés auront le droit d'y intervenir (main, production). Les autres développeurs proposent leurs features / fix via des "*merge requests*" .
Une personne du groupe (maintainer) sera désigné pour créer le répo et sera garant de la qualité et de l'utilisabilité des branches protégées.	
Les autres membres du groupe devront créer un fork du projet. C'est ce fork qui sera ensuite cloné sur votre machine. N'oubliez pas d'ajouter un pointeur (un 'remote') vers le projet original (celui qui a été "forked"). Ce remote est par convention appelé "upstream": `git remote add upstream "git@github.com:<propriétaire>/<dépôt>.git"`	Le maintainer invite les autres à collaborer au projet en tant que "developper" (la terminologie peut changer selon les hébergeurs).
Gardez en tête que la branche "`main`" du projet ne doit jamais être bugée, le code qu'elle contient doit toujours pouvoir s'exécuter *(sauf bug non anticipé qui nécessitera un "hot-fix")*. Au départ du projet cette branche ne contient généralement que le fichier Readme.md, ne le négligez pas trop, c'est la description du projet sur laquelle on tombe en premier quand on parcourt votre répo git.	
Le développement se fait sur une branche "`develop`" ou une branche "`feature`-#" avec un nom évocateur de ladite feature qui est en train d'être développée. Cette branche pourra contenir du code buggé (même si ce n'est pas recommandé). C'est sur cette branche que vous allez créer vos	

nouvelles features qui, lorsqu'elles seront stabilisées, seront incorporées à la branche "`main`" du dépôt au travers d'un merge (ou d'un rebase/merge).

Avant de proposer de merger votre branche dans une branche protégée, assurez vous d'avoir bien rapatrié les dernières modifications de cette branche dans la vôtre et d'avoir résolu les éventuels conflits.

Exemple:
1 - vous avez créé votre branch 'topic' quand 'master était au commit D, puis fait les commits A, B, C. Vous pensez être prêt à merge dans 'master'

```
      A---B---C topic
     /
    D  master
```

2- Sauf que la branche master a avancé pendant ce temps là (d'autres développements en cours)

```
# pour mettre 'master' à jour
git checkout master
git pull
```

```
      A---B---C topic
     /
    D---E---F---G master
```

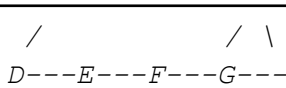
3- Vous devez donc vous assurer que vos modifications sont compatibles avec l'état courant de master

```
#on se remet sur la branche 'topic'
git checkout topic
# et on merge les dernières avancées de 'master'
git merge master
```

```
    A---B---C---H topic
   /           /
  D---E---F---G master
```

4- Tout se passe bien, votre branche topic fonctionne bien avec les dernières avancées de master et vos propres développements. Vous pouvez maintenant créer une "merge request" topic -> master. Ou bien directement merge dans master si vous êtes le maintainer du repo.

```
    A---B---C---H topic
```

	
Les étapes précédentes doivent être complétées par une mise a jour de votre répo par les avancées du répo upstream: `git pull upstream`	
Vous pourrez alors ouvrir une “pull request” (souvent Github vous le propose déjà par défaut) permettant de soumettre votre travail au mainteneur du dépôt upstream.	

Pour plus de détails sur le flot de travail avec git et fork :
https://medium.com/singlestone/a-git-workflow-using-rebase-1b1210de83e5