**Informations personnelles**

Nom : Sow date : 16/03/25

Prénom : Cheikh Mansour SOW

**Rapport détaillé sur le Fine-Tuning du modèle de Deep Learning**

**MobileNetV2**

**Introduction**

L'objectif de ce projet est d'utiliser un modèle pré-entraîné, tel que **MobileNetV2**, pour effectuer un fine-tuning sur un dataset (**PedroSampaio/fruits-360**). Le but est de créer un modèle capable de classifier des images de fruits. Le fine-tuning permet d’adapter un modèle préexistant à un nouveau problème spécifique, tout en utilisant les connaissances apprises sur un autre dataset (ImageNet).

1. **Préparation du Dataset**

**1.Chargement du dataset et exploration**

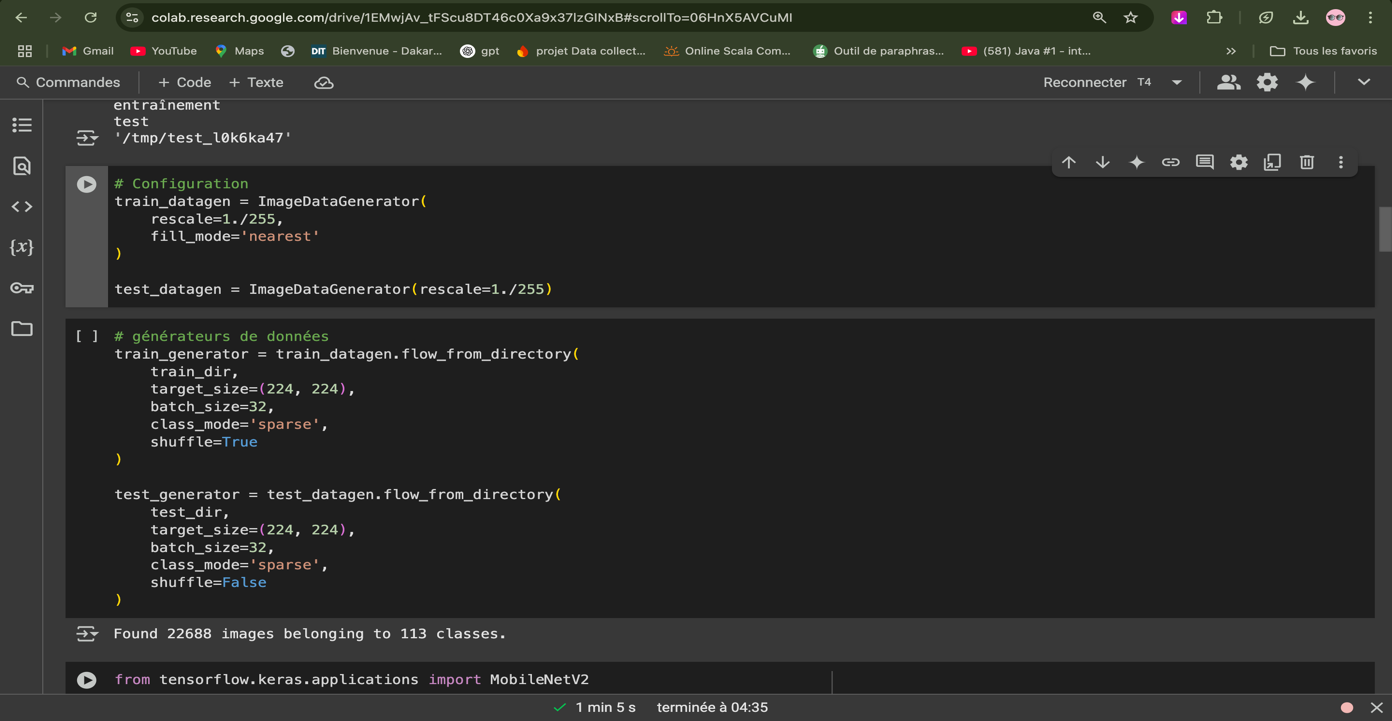
Tout d'abord, nous avons chargé le dataset **PedroSampaio/fruits-360** en utilisant la fonction load\_ dataset de la bibliothèque **datasets**. Ce dataset contient des images de fruits avec environ 67000 images en entrainement et environ 22000 images en test.

**2. Création des dossiers d'images**

Nous avons créé des dossiers pour chaque classe (fruit) afin de stocker les images et les organiser de manière structurée pour l’entraînement et le test. Cela nous a permis de préparer les images dans les dossiers nécessaires pour les générateurs de données.

**3. Générateurs de Données**

Nous avons utilisé **ImageDataGenerator** pour préparer les images et appliquer des transformations comme la mise à l'échelle (rescaling) pour les images d’entraînement et de test.

****

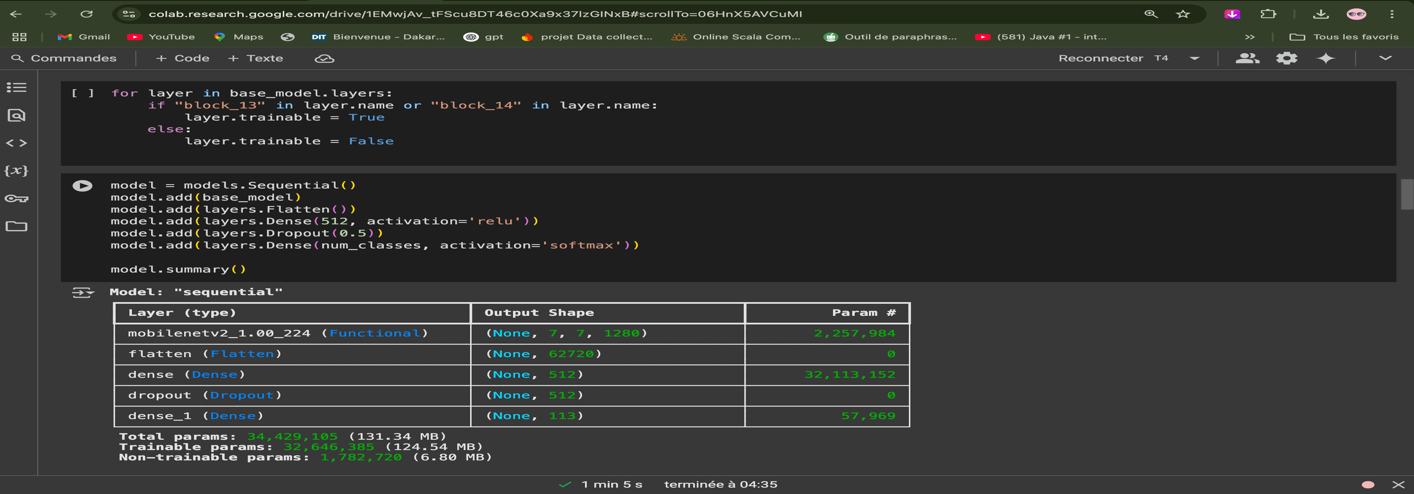
1. **Chargement et Préparation du Modèle**
2. **Utilisation du modèle pré-entrainé**

Nous avons utilisé MobileNetV2, un modèle de deep learning pré-entraîné sur ImageNet. Ce modèle est idéal car il est déjà formé à reconnaître des objets généraux. Cependant, nous devons l'adapter pour notre tâche spécifique de classification des fruits.

1. **Congélation des couches et ajout de couches personnalisées**

Lors du fine-tuning, il est important de ne pas réentraîner toutes les couches du modèle. Nous avons donc choisi de ne rendre entraînables que certaines couches proches de la sortie du modèle, ici les couches **block\_13** et **block\_14**. Cela permet de conserver les caractéristiques générales apprises par le modèle tout en l’adaptant à notre jeu de données spécifique.

Sans oublie l’ajout plusieurs couches personnalisées pour adapter le modèle aux fruits. Cela inclut une couche **Dense** (fully connected) et une couche de **Dropout** pour réduire le sur-apprentissage.



1. **Compilation et Entraînement du Modèle**
2. **Compilation du modèle et les Callbacks**

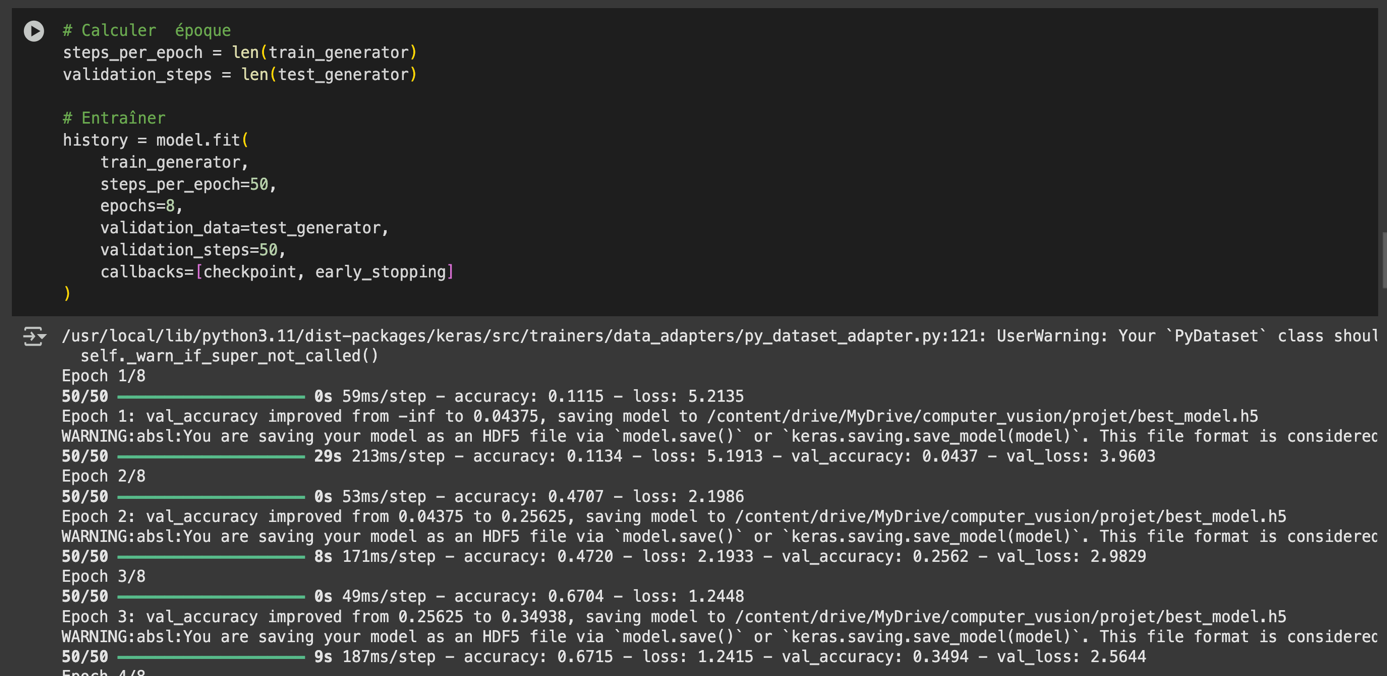
Nous avons compilé le modèle en utilisant l'optimiseur **Adam** avec un taux d'apprentissage bas (5e-5) pour effectuer une mise à jour des poids efficace. Pour ne pas mettre en danger notre model.

Pour éviter le sur-apprentissage et enregistrer le meilleur modèle pendant l’entraînement, nous avons défini deux **callbacks** :

* **ModelCheckpoint** : pour enregistrer le modèle chaque fois que la précision de validation s'améliore.
* **EarlyStopping** : pour arrêter l'entraînement si la précision de validation ne s'améliore pas après un certain nombre d'époques.

1. **Entraînement**

Nous avons ensuite entraîné le modèle pendant 8 époques, le choix du nombre d’époques a été définie du fait qu’on ne disposer plus de gpu dans colab, en ajustant le nombre d'étapes par époque et d'étapes de validation en fonction des générateurs de données.



1. Évaluation

* Matrice de Confusion

Après l'entraînement, nous avons évalué les performances du modèle à l'aide de la **matrice de confusion**, qui permet de visualiser les erreurs de classification.

Nous avons ensuite affiché la matrice de confusion sous forme de carte thermique pour mieux comprendre les erreurs du modèle.

* Rapport de Classification

Enfin, nous avons généré un rapport de classification détaillant la précision, le rappel, et le score F1 pour chaque classe de fruits.

Le modèle final a été sauvegardé pour pouvoir être utilisé ultérieurement dans des applications ou des API.

* Sauvegarde du modèle et la création de l’API Rest

Le modèle final a été sauvegardé pour pouvoir être utilisé ultérieurement dans le code pour l’API.

L'API qu’on a créé avec **Fastapi** dans ce projet permet d'intégrer le modèle de classification d'images dans une application web. Elle sert à recevoir des requêtes, telles que l'uploade d'images par un utilisateur, puis à traiter ces images avec le modèle préalablement entraîné pour obtenir des prédictions. Cela rend le modèle accessible à distance, via des appels HTTP, facilitant son utilisation dans des applications en ligne sans avoir besoin de réentraîner ou redéployer constamment le modèle. L'API permet de rendre le modèle interactif et d'effectuer des prédictions en temps réel.

Les Résultats :

Les résultats montrent une amélioration progressive de l'exactitude et de la perte pendant les 8 epochs

* **Accuracy d'entraînement** : On observe une augmentation régulière de l'accuracy d'entraînement, qui atteint 93,94% à la fin de l'entraînement.
* **Validation** : L'accuracy de validation augmente également, passant de 4,37% à 59,69% pendant les premiers 7 epochs, avant de se stabiliser à 52,50% à l'epoch finale.
* **Perte (Loss)** : La perte d'entraînement diminue de manière significative, atteignant une valeur basse (0,20), indiquant que le modèle devient plus précis dans ses prédictions.
* **F1-Score** : Le F1-score montre une variabilité selon les classes. Certaines classes (comme "Banana", "Grape Blue", "Tomato") ont des scores F1 élevés, tandis que d'autres (comme "Pomegranate", "Potato Red", "Mango Red") présentent des scores très faibles. Cela peut indiquer que le modèle a des difficultés à classifier correctement certaines catégories.

**Conclusion**

Le fine-tuning d'un modèle pré-entraîné comme **MobileNetV2** permet de gagner du temps et d'utiliser un modèle puissant tout en l'adaptant à un nouveau jeu de données spécifique. Grâce à cette approche, nous avons pu créer un modèle de classification d'images de fruits qui offre de bonnes performances après un entraînement relativement court.