Pneumonia FAQ

Quels types de réponses je donne aux questions :

A quoi sa vise

pourquoi c'est fait de cette manière

Pourquoi on l'a pas fait autrement

Pourquoi cet enchainement-là

1) Comment séparer les différentes classes en différents folder ?

- D'abord on unzip le fichier contenant les images
- Je lui demande parcourir toutes les images
- Je lui demande de supprimer toutes les images qui ne se trouvent pas dans les fichiers train, test ou val.
- En fonction du nom présent dans le fichier, je l'envoie dans une classe qui lui correspond.
- Par exemple, si il trouve "NORMAL", alors je l'envoie dans le fichier interim/normal. Pareil ppur "virus" et "bacteria".

2) Est-ce qu'on ajoute des données?

Non, justement on réduit. Après

```
SMALL_TRAIN_SPLIT = 0.20
SMALL_VAL_SPLIT = 0.15
```

```
# Extract a sample
small_train_size = int(num_train_img * SMALL_TRAIN_SPLIT)
small_val_size = int(num_val_img * SMALL_VAL_SPLIT)
```

et

```
test_split = 0.2
val_split = 0.25
```

validation_split=test_split,

val_size = int(num_test_img * val_split)

3) Est-ce qu'on ajoute des poids ?

```
self.false_positives = self.add_weight(name='fn',
                                       initializer='zeros')
        self.actual_negatives = self.add_weight(name='total',
                                        initializer='zeros')
```

En utilisant self.add_weight(...), le code crée une variable de poids appelée "false_positives" avec une valeur initiale de zéro. Cette variable de poids peut ensuite être utilisée dans la couche ou le modèle personnalisé à diverses fins, telles que le suivi et la mise à jour du nombre de faux positifs au cours de la formation ou de l'évaluation.

Question : Pourquoi le faire?

L'ajout de poids entraînables est un aspect fondamental de la construction de réseaux neuronaux et de modèles d'apprentissage profond. Les poids entraînables sont essentiels car ils permettent au modèle d'apprendre et de s'adapter aux données par le biais du processus d'optimisation.

Voici quelques raisons pour lesquelles l'ajout de poids est crucial :

1. **Paramètres apprenables** : Les poids entraînables sont des paramètres qui peuvent être appris et qui permettent au modèle de capturer et de représenter des modèles et des relations dans les données. En ajustant les poids pendant le processus de formation, le modèle peut optimiser ses performances et faire des prédictions précises.
2. **Flexibilité du modèle** : Les poids entraînables permettent de modéliser avec souplesse différentes fonctions et relations entre les entrées et les sorties. Les poids agissent comme des coefficients appris à partir des données, ce qui permet au modèle de s'adapter et de se généraliser à de nouveaux exemples.
3. **Capacité du modèle** : Le nombre de poids entraînables détermine la capacité ou la complexité du modèle. En ajustant les poids, le modèle peut augmenter ou diminuer sa capacité à traiter des tâches plus complexes ou plus simples, respectivement.
4. **Optimisation par descente de gradient** : Les poids entraînables sont essentiels pour les algorithmes d'optimisation tels que la descente de gradient. Pendant l'apprentissage, l'algorithme calcule les gradients par rapport aux poids, indiquant comment les poids doivent être mis à jour pour minimiser la fonction de perte et améliorer les performances du modèle.

En résumé, l'ajout de poids entraînables est un élément fondamental des modèles d'apprentissage profond. Ces poids permettent au modèle d'apprendre à partir des données, de s'adapter à différentes tâches et d'optimiser ses performances grâce à des algorithmes d'optimisation basés sur le gradient. Sans poids entraînables, le modèle n'aurait pas la capacité d'apprendre et de généraliser efficacement.

4) Quel est le déséquilibre qui compte? Celui entre val/train et test ? Ou virus/bacterie/normale?

5) A quel moment on se rend compte que c'est déséquilibré?

6) Qu'est-ce qu'il y a dans processed?

6.1) Il y a le résultat des analyses de train, test et val :

```
train_path = str(zoidbergManager.data_dir / 'processed' / f'train_{img_height}x{img_width}_{channels}_ds.tfrecord')
val_path = str(zoidbergManager.data_dir / 'processed' / f'val_{img_height}x{img_width}_{channels}_ds.tfrecord')
test_path = str(zoidbergManager.data_dir / 'processed' / f'test_{img_height}x{img_width}_{channels}_ds.tfrecord')
```

6.2) Qu'on a ensuite sauvergardé en le transformant en en tfrecord:

```
save_image_dataset_to_tfrecord(train_ds, train_path)
save_image_dataset_to_tfrecord(val_ds, val_path)
save_image_dataset_to_tfrecord(test_ds, test_path)
```

6.3) Comment a-t-on eu ces valeurs?

Nous utilisons la fonction image_dataset_from_directory pour séparer le train et le test et les charger dans les datasets.

```
train_ds, full_test_ds = keras.utils.image_dataset_from_directory(
    directory=Path(zoidbergManager.data_dir / "interim" / "full"),
    label_mode="categorical",
    color_mode=channels,
    batch_size=None,
    image_size=(img_height, img_width),
    seed=42,
    validation_split=test_split,
    subset="both"
)
```

6.4) Qu'est-ce que ça contient?

- On fetch toutes les images ('full') dans les folders interim/fulls.
- On assigne un label à chacun des dossiers en fonction de leur catégories ('bacteria', 'normal', 'virus').
- batch_size = None indique que les données ne doivent pas être regroupées. En d'autres termes, chaque image du jeu de données sera traitée comme un lot distinct.il peut être utile dans les scénarios où vous souhaitez travailler avec l'ensemble des données en une seule fois, sans les diviser en lots plus petits. C'est généralement le cas lorsqu'on dispose d'un petit ensemble de données.
- le paramètre seed est utilisé pour définir le seed aléatoire pour le mélange et d'autres opérations aléatoires effectuées au cours du processus de création du jeu de données. En définissant une valeur de semence spécifique, vous pouvez contrôler la séquence des nombres aléatoires générés, ce qui rend les résultats déterministes et reproductibles.
- Dans la fonction image_dataset_from_directory, le paramètre validation_split est utilisé pour spécifier la fraction des données qui doit être utilisée pour la validation.

6.5) Pourquoi est-ce qu'on a ensuite besoin de les compter si on les a déjà séparé? Est-ce essentiel d'utiliser count_img_by_class?

Non ce n'est pas essentiel, c'est pour mieux visualiser et comprendre ce qui se passe. Cependant, ce n'est pas une étape qui va être utilisé pour le traitement des données.

Compter le nombre d'images dans chaque classe après avoir divisé l'ensemble de données peut s'avérer utile pour plusieurs raisons :

1. Déséquilibre entre les classes : Il permet d'identifier s'il existe un déséquilibre significatif dans la distribution des images entre les différentes classes.
2. Compréhension de l'ensemble des données : Elle fournit une vue d'ensemble de l'ensemble de données et de la répartition des images entre les différentes classes.

Dans l'ensemble, le comptage des images dans chaque classe fournit des informations précieuses pour l'analyse des données, le développement de modèles et le suivi du processus de formation.

```
print("In validation dataset, there are :")
for class_name, num_img in val_img_by_classes.items():
    print(f"  - {num_img} files for class {class_name}")
print("\nIn test dataset, there are :")
for class_name, num_img in test_img_by_classes.items():
    print(f"  - {num_img} files for class {class_name}")
```

```
print("\nIn training dataset, there are :")
for class_name, num_img in train_img_by_classes.items():
    print(f"  - {num_img} files for class {class_name}")
print("\nIn full test dataset, there are :")
for class_name, num_img in full_test_img_by_classes.items():
    print(f"  - {num_img} files for class {class_name}")
```

6.6) Pourquoi on a isolé full_test_ds et num_train_img ?

6.6.1) Qu'est-ce que full_test_ds?

Le `full_test_ds` est un jeu de données qui contient l'ensemble des images du test. Le but de la création du `full_test_ds` est d'avoir une représentation complète des données de test qui peut être utilisée à des fins d'évaluation ou d'inférence.

Le jeu de données `full_test_ds` peut être utilisé pour évaluer la performance d'un modèle entraîné sur l'ensemble du jeu de test, calculer des mesures telles que l'exactitude, la précision, le rappel ou le score F1, et générer des prédictions pour les images de test. Il fournit une vue d'ensemble des performances du modèle sur des données inédites et permet d'évaluer la généralisation et l'efficacité du modèle entraîné.

6.6.2) Comment est-il utilisé? A quoi vise-t-il?

`full_test_ds` est utilisé dans cet ordre-là:

- num_test_img = full_test_ds.reduce(0, lambda x, _: x + 1).numpy()
- val_size = int(num_test_img * val_split)
- val_ds = full_test_ds.take(val_size)
- test_ds = full_test_ds.skip(val_size)

L'utilisation de `full_test_ds` dans cet exemple vise à diviser le jeu de données de test en deux parties distinctes : un ensemble de validation (`val_ds`) et un ensemble de test (`test_ds`). Cette division permet d'évaluer les performances du modèle sur des données de test indépendantes.

Voici comment fonctionne le processus :

1. Tout d'abord, `num_test_img` est calculé en utilisant la méthode `reduce` qui compte le nombre total d'images dans `full_test_ds`.
2. Ensuite, `val_size` est calculé en multipliant `num_test_img` par la proportion définie par `val_split`. Cela détermine la taille de l'ensemble de validation.
3. L'ensemble de validation (`val_ds`) est créé en utilisant la méthode `take` sur `full_test_ds`, en spécifiant `val_size` pour indiquer le nombre d'images à inclure dans l'ensemble de validation.
4. L'ensemble de test (`test_ds`) est créé en utilisant la méthode `skip` sur `full_test_ds`, en spécifiant `val_size` pour indiquer le nombre d'images à ignorer. Cela garantit que les images incluses dans `test_ds` sont celles qui n'ont pas été incluses dans l'ensemble de validation.

En résumé, cette approche permet de diviser le jeu de données de test en un ensemble de validation et un ensemble de test de manière contrôlée, en utilisant `full_test_ds` comme point de départ. Cela facilite l'évaluation des performances du modèle sur des données de test indépendantes et la sélection des meilleurs hyperparamètres en utilisant l'ensemble de validation.

6.6.3) Pourquoi on utilise val_size comme paramètre?

L'utilisation de `val_size` comme paramètre dans `val_ds` et `test_ds` est une manière de diviser le jeu de données de test en deux sous-ensembles distincts, à savoir l'ensemble de validation et l'ensemble de test.

Lorsque nous créons `val_ds` à l'aide de la méthode `take` sur `full_test_ds`, nous spécifions `val_size` comme nombre d'images à inclure dans cet ensemble de validation. Cela garantit que `val_ds` contient exactement `val_size` images, extraites de `full_test_ds`.

Ensuite, nous créons `test_ds` à l'aide de la méthode `skip` sur `full_test_ds`, en spécifiant `val_size` comme nombre d'images à ignorer. Cela signifie que `test_ds` contiendra toutes les images restantes de `full_test_ds` qui n'ont pas été incluses dans `val_ds`.

En résumé, `val_size` est utilisé comme paramètre dans `val_ds` et `test_ds` pour séparer correctement les images du jeu de données de test en deux ensembles distincts, en utilisant une taille prédéfinie pour l'ensemble de validation. Cela permet d'avoir des ensembles de validation et de test mutuellement exclusifs et représentatifs du jeu de données d'origine.

6.6.4) Quel est le lien entre train_ds, val_ds , test_ds et processed?

- On met séparément toutes les données de train_ds, val_ds , test_ds dans des fichiers qui se trouvent dans un dossier processed.
- Maintenant que tout est regroupé dans un fichier, on peut tout traiter avec un fichier.

7) Pourquoi on load les imaes de dataset avec load_image_dataset_from_tfrecord() ?

7.1) qu'est-ce que load_image_dataset_from_tfrecord()

Il permet dans l'ordre de:

- Parser les images.
- Shuffle les données
- Batcher les données
- Prefetch les données

7.2) A quoi ça sert de prefetch les données?

La mise en cache préalable des données (prefetching) est une technique utilisée pour optimiser les performances lors du traitement des données. Cela permet d'optimiser l'utilisation des ressources système, en réduisant le temps d'attente lors du chargement des données.

Voici quelques avantages de l'utilisation de la mise en cache préalable des données :

1. Amélioration des performances : Précharger les données permet au processeur de continuer à effectuer des calculs sur les données précédemment préchargées, tandis que les nouvelles données sont en cours de chargement depuis le stockage, réduisant ainsi le temps d'attente global.
2. Utilisation efficace des ressources : La mise en cache préalable des données permet d'optimiser l'utilisation des ressources système, car le processeur est utilisé de manière plus efficace en effectuant des calculs et des opérations de prétraitement sur les données déjà chargées pendant que les nouvelles données sont en cours de chargement.
3. Équilibrage de la charge : Lorsque vous travaillez avec des pipelines de données en parallèle, la mise en cache préalable des données permet de répartir équitablement la charge de travail entre le chargement des données et les opérations de calcul, en évitant les goulots d'étranglement potentiels.

En résumé, la mise en cache préalable des données (prefetching) est utile pour optimiser les performances et l'utilisation des ressources lors du traitement des données, en réduisant les temps d'attente et en équilibrant la charge de travail entre le chargement des données et les opérations de calcul.

7.3) A quoi sert la ligne image_val_1, label_val_1 = next(val_ds.take(1).as_numpy_iterator()) ?

La ligne de code `image_val_1, label_val_1 = next(val_ds.take(1).as_numpy_iterator())` extrait un seul exemple (image et label) du jeu de données de validation (`val_ds`).

Voici ce que cela fait en détail :

1. `val_ds.take(1)` : Cette partie de la ligne de code crée un nouveau jeu de données en extrayant les premières données d'un autre jeu de données. Dans ce cas, nous extrayons seulement 1 élément du jeu de données de validation (`val_ds`).
2. `as_numpy_iterator()` : Cette méthode convertit le jeu de données en un itérable de numpy, ce qui nous permet d'itérer sur les éléments du jeu de données sous forme de tableaux numpy.
3. `next(...)` : La fonction `next(...)` est utilisée pour obtenir le prochain élément de l'itérateur. Dans ce cas, nous obtenons le premier élément du jeu de données de validation.
4. `image_val_1, label_val_1 = ...` : Cette ligne de code assigne les valeurs de l'image et de l'étiquette du premier exemple extrait du jeu de données de validation aux variables `image_val_1` et `label_val_1`, respectivement.

En résumé, la ligne de code extrait un exemple unique (image et label) du jeu de données de validation pour une utilisation ultérieure dans le code.

7.4) Quelle est la différence entre label_val_1 et label_load_val_1 ?

7.4.1) loaded_val_ds

c'est le résultat de val_ds stocké dans val_path qui est chargé puis traité pour l'entrainement par load_image_dataset_from_tfrecord qui est utilisé par loaded_val_ds.

val_path = str(zoidbergManager.data_dir / 'processed' / f'val_{img_height}x{img_width}_{channels}_ds.tfrecord')

loaded_val_ds = load_image_dataset_from_tfrecord(val_path)

7.4.2) val_ds

C'est un dataset vierge qu'on a juste extrait de full_test. C'est juste qu'il contient les images de validations, mais sans valeur ajoutée.

val_ds = full_test_ds.take(val_size)

- val_size = int(num_test_img * val_split)

L'utilisation de `full_test_ds` dans cet exemple vise à diviser le jeu de données de test en deux parties distinctes : un ensemble de validation (`val_ds`).

1. L'ensemble de validation (`val_ds`) est créé en utilisant la méthode `take` sur `full_test_ds`, en spécifiant `val_size` pour indiquer le nombre d'images à inclure dans l'ensemble de validation.

7.5) Comment on passe de label_val_1 à label_load_val_1 ? A quoi ça sert?

En suivant le chemin détaillé dans loaded path. Le but est de montrer la différence entre l'image avant traitement (val_ds) et l'image après traitement (loaded_val_ds). C'est fait pour les distinguer.

On extrait une instance de chacun d'entre eux (ici, le premier) :

image_val_1, label_val_1 = next(val_ds.take(1).as_numpy_iterator())
image_load_val_1, label_load_val_1 = next(loaded_val_ds.take(1).as_numpy_iterator())

puis on ledeux différentes images + tritre l'une à côté de l'autre :

I -

```
plt.imshow(image_val_1.astype('int64'))
plt.title(f'image before tfrecord : {train_ds.class_names[np.nonzero(label_val_1)[0][0]]}')
```

II-

```
plt.imshow(image_load_val_1)
plt.title(f'image after tfrecord : {train_ds.class_names[np.nonzero(label_load_val_1)[0][0]]}')
```

7.6) Pourquoi on fait tout ça?

Parce que le processus consiste à transformer un jpeg en numpy table puis en binaire puis en jpeg. Cela peut entraîner des erreurs, tant il y a d'étapes. Donc on pour vérifier qu'il n'y a pas d'erreurs, par exemple un inversement des contrastes, on souhaite visualier les images avant et après traitements.

7.7) A quel moment on le fait?

8) Comment le modèle identifie et distingue un FP d'un FN.

9) Comment entraîne-t-ton un modèle?

9.1) Comment fonctionne un entraînement de modèle?

9.1.1) On appelle le modèle qu'on souhaite à travers la librairie de keras

```
base_efficientnetb0 = tf.keras.applications.EfficientNetV2B0(weights='imagenet', input_shape=(224,224,3), include_top=False)
```

Ici c'est EfficientNetV2B0

9.1.2) On appelle le modèle qu'on souhaite à travers la librairie de keras

```
def make_efficientnetb0():
    base_efficientnetb0 = tf.keras.applications.EfficientNetV2B0(weights='imagenet', input_shape=(224,224,3), include_top=False)
    for layer in base_efficientnetb0.layers:
        layer.trainable = False
```

Dans l'extrait de code donné, `layer.trainable = True` est utilisé pour mettre la propriété "trainable" de chaque couche du modèle `base_efficientnetb0` à False.

Lorsqu'une couche est entraînable (donc c'est à True), cela signifie que ses poids peuvent être mis à jour pendant le processus d'entraînement.

Par défaut, `layer.trainable = False`, . En effet, la plupart des modèles pré-entraînés ont leurs couches réglées sur non-trainable, ce qui signifie que leurs poids sont gelés et ne sont pas mis à jour pendant l'entraînement.

En réglant `layer.trainable = True`, on permet aux poids des couches du modèle `base_efficientnetb0` d'être entraînés et mis à jour pendant le processus d'entraînement suivant. Ceci est utile lorsque vous voulez affiner un modèle pré-entraîné sur un nouveau jeu de données ou une nouvelle tâche, où vous voulez que le modèle apprenne des données et ajuste ses poids en fonction du problème spécifique que vous êtes en train de résoudre.

9.1.3) En quoi consiste tf.keras.Sequential et qu'est-ce qui est à l'intérieur?

Dans l'extrait de code donné, `efficientnetb0` est un modèle séquentiel construit à l'aide de l'API séquentielle Keras. Il définit une architecture de réseau neuronal en utilisant le modèle EfficientNetB0 comme base, avec des couches supplémentaires ajoutées pour un traitement et une classification plus poussés.

Voici une ventilation des couches du modèle `efficientnetb0` :

1. InputLayer : Spécifie la forme d'entrée du modèle.
2. Resizing : Redimensionne les images d'entrée à une taille spécifique (224x224) à l'aide d'une interpolation bilinéaire.

3. base_efficientnetb0 : Le modèle EfficientNetB0 de base, qui est un réseau neuronal convolutionnel pré-entraîné. Il sert d'extracteur de caractéristiques pour les couches suivantes.
4. GlobalAveragePooling2D : effectue une mise en commun de la moyenne globale, qui réduit les dimensions spatiales des cartes de caractéristiques et fournit une représentation globale des caractéristiques.
5. Dense : Une couche entièrement connectée avec 1024 unités et une activation ReLU, qui introduit une non-linéarité et une capacité d'apprentissage supplémentaire dans le modèle.
6. Dense : Une autre couche entièrement connectée avec 512 unités et une activation ReLU.
7. Dense : La couche de sortie avec 3 unités et une activation softmax, qui produit la distribution de probabilité finale sur les trois classes.

Dans l'ensemble, cette architecture adapte le modèle EfficientNetB0 à la tâche spécifique de classification d'images, en ajoutant quelques couches supplémentaires pour le traitement des caractéristiques et une couche de sortie dense pour la classification.

9.1.4) On compile

```
efficientnetb0.compile(optimizer=keras.optimizers.Adam(learning_rate=LEARNING_RATE),
                  loss='categorical_crossentropy',
                  metrics=evaluation.get_training_metrics()
                 )
    return efficientnetb0
```

La méthode `compile()` est appelée sur le modèle `efficientnetb0`. Cette fonction configure le modèle pour la formation en spécifiant l'optimiseur, la fonction de perte et les métriques à utiliser pendant la formation.

Voici une décomposition des arguments passés à la méthode `compile()` :

- `optimizer` : L'optimiseur détermine comment le modèle sera mis à jour sur la base des gradients calculés. Dans ce cas, l'optimiseur Adam est utilisé avec un taux d'apprentissage spécifié (`LEARNING_RATE`). Le taux d'apprentissage détermine la taille du pas à chaque itération pendant l'optimisation.
- `loss` : La fonction de perte est utilisée pour calculer l'écart entre la sortie prédite du modèle et la sortie réelle. L'entropie croisée catégorielle est une fonction de perte couramment utilisée pour les problèmes de classification multi-classes, où les étiquettes de sortie sont codées à un seul instant.
- `metrics` : L'argument metrics spécifie les métriques d'évaluation à utiliser pendant l'apprentissage. La fonction `get_training_metrics()` est appelée pour récupérer une liste de métriques d'apprentissage. Ces métriques sont utilisées pour surveiller les performances du modèle pendant l'apprentissage, telles que l'exactitude, la précision, le rappel, etc.

Après avoir compilé le modèle, la fonction le renvoie. Le modèle est maintenant prêt à être entraîné sur les données d'entraînement à l'aide de l'optimiseur, de la fonction de perte et des métriques spécifiés.

9.1.5) On défini le scope

```
with strategy.scope():
    efficientnetb0 = make_efficientnetb0()
```

`with strategy.scope():` est un gestionnaire de contexte fourni par TensorFlow pour spécifier la portée dans laquelle un modèle ou un ensemble d'opérations sera exécuté.

Lors de l'utilisation de l'apprentissage distribué avec TensorFlow, il est courant d'utiliser plusieurs appareils ou machines pour accélérer le processus d'apprentissage. L'objet `strategy` représente la stratégie distribuée utilisée, comme

`tf.distribute.MirroredStrategy` ou `tf.distribute.experimental.TPUStrategy`, qui permet l'entraînement sur plusieurs GPU ou TPU.

En utilisant `with strategy.scope():`, tout modèle ou opération défini dans ce contexte sera automatiquement placé dans le périmètre distribué approprié. Cela garantit que le modèle et les opérations sont exécutés correctement sur plusieurs appareils ou machines.

9.1.6) On entraîne le modèle

Après avoir configuré le modèle, on peut commencer de l'entraîner. Pour cela nous allons utiliser la méthode train_model() que nous avons écrite la dernière fois.

```
if TRAIN_EFFICIENTNETB0:
    efficientnetb0_history, efficientnetb0_time = train_model(efficientnetb0, save=True)
```

Mais de quoi est-il composé?

```
def train_model(model, save=False):
    start_time = time.time()
    history = model.fit(small_train_ds,
                        validation_data=small_val_ds,
                        epochs=EPOCHS,
                        class_weight=dic_class_weights,
                        callbacks=[checkpoint_cb(model)],
                        )
    training_time = time.time() - start_time

    if save:
        save_history(model, history, training_time)

    return history, training_time
```

Déconstruisons le modèle :

9.1.6.1) model.fit()

`model.fit()` est une méthode de TensorFlow qui est utilisée pour entraîner un modèle sur un ensemble de données séléctionné. C'est une API de haut niveau pour l'entraînement des modèles qui simplifie le processus de configuration et d'exécution de la boucle d'entraînement.

```


La méthode `model.fit()` prend plusieurs arguments, y compris les données d'apprentissage (`x` et `y`), le nombre d'époques
 d'apprentissage, et diverses options pour configurer le processus d'apprentissage.

Pendant l'apprentissage, `model.fit()` itère sur les données d'apprentissage pour le nombre d'époques spécifié, en mettant à
 jour les poids du modèle en fonction de l'optimiseur et de la fonction de perte choisis. Il effectue des passes avant et arr
ière, calcule les gradients et met à jour les paramètres du modèle pour minimiser la perte.

La méthode prend également en charge les options pour les données de validation, les rappels pour personnaliser le comporteme
nt de l'apprentissage, le brassage des données, les poids de classe pour les ensembles de données déséquilibrés, et plus enco
re.

Une fois l'apprentissage terminé, `model.fit()` renvoie un objet `History` qui contient des informations sur l'historique de
 l'apprentissage, telles que les valeurs de perte et de métriques à chaque époque.

Voici un exemple d'utilisation de `model.fit()` :

history = model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_val, y_val))
```

Comme on peut le voir, nous avons repris ce modèle

```
history = model.fit(small_train_ds,
                        validation_data=small_val_ds,
                        epochs=EPOCHS,
                        class_weight=dic_class_weights,
                        callbacks=[checkpoint_cb(model)],
                        )
```

Cependant, il est quelque peu différent :

I - Points communs :

- validation_data
- Epochs
- x_train qui est ici small_train_ds

II - Différences:

Nous y avons ajouté certains optionnalités:

- class_weight=dic_class_weights,
- callbacks=[checkpoint_cb(model)],

Pourquoi?

1. class_weight

```
class_weights = class_weight.compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weights
dic_class_weights = {}
for idx, weight in enumerate(class_weights):
    dic_class_weights[idx] = weight
    print(f'class {class_names[idx]} => weight : {weight:2f}')
```

A quoi sert class_weights ?

`compute_class_weight` est utile dans le contexte des tâches de classification déséquilibrées où la distribution des classes dans les données d'apprentissage est inégale. Dans de tels scénarios, les algorithmes standards d'apprentissage automatique peuvent avoir du mal à apprendre efficacement à partir des données déséquilibrées et finissent par être biaisés en faveur de la classe majoritaire.

La fonction `compute_class_weight` aide à résoudre ce problème en calculant les poids de classe qui peuvent être utilisés pour donner plus d'importance à la (aux) classe(s) minoritaire(s) pendant l'apprentissage du modèle. Ces poids de classe sont calculés sur la base de la fréquence des classes dans les données d'apprentissage. Plus le poids attribué à une classe est élevé, plus le modèle accordera d'importance à la prédiction correcte des instances de cette classe.

De quoi est-ce constitué?

class_weight

- class_weight → spécifie la stratégie de calcul des poids des classes. La valeur "balanced" indique que les pondérations des classes seront déterminées de manière à donner plus d'importance à la classe sous-représentée (minorité) et moins d'importance à la classe surreprésentée (majorité).
- classes → est un tableau ou une liste contenant les étiquettes de classe uniques dans l'ensemble de données.

Ici, on utilise np.unique() parce que les labels de classe uniques sont obtenues à l'aide de np.unique(y_train).

- y → est le tableau ou la liste des étiquettes de classe correspondant aux instances d'apprentissage.

C’est basé sur sur les labels d'apprentissage y_train, que voici :

```
y_train_iterator = train_ds.map(lambda x, y: y).as_numpy_iterator()
y_train = []
for one_vector in y_train_iterator:
    y_train.append(one_vector)
y_train = np.argmax(y_train, axis=1)
```

2- callbacks

train_model prend en compte deux paramètres:

model

save=false

A quoi correspond model?

Il est utilisé comme paramètre dans checkpoint_cb(). C’est un callback qui sauvegarde un modèle séléctionné à chaque époque (sauvegarde uniquement du meilleur poids).

```
def checkpoint_cb(model):
    checkpoint_dir = zoidbergManager.model_dir / 'checkpoints'
    checkpoint_filepath = checkpoint_dir / f'ckpt_smallds_{model.name}.h5'
    ckpt_cb = keras.callbacks.ModelCheckpoint(
        filepath=checkpoint_filepath,
        monitor='val_CKS',
        mode='max',
        save_best_only=True
    )
    return ckpt_cb
```

Conclusion :

AInsi nous pouvons conclure pourquoi nous nous avons rajouer en option callback et class_weight. En plus d’entrainer notre modèle, nous souhaitons également sauvergarder un modèle à chaque époque et réajuster les poids.

9.1.3) On configure les layers dans le modèle

```
def make_efficientnetb0():
    base_efficientnetb0 = tf.keras.applications.EfficientNetV2B0(weights='imagenet', input_shape=(224,224,3), include_top=False)
    for layer in base_efficientnetb0.layers:
        layer.trainable = True

    efficientnetb0 = tf.keras.Sequential([
        keras.layers.InputLayer(input_shape=(512,512,3), name='input'),
        keras.layers.Resizing(224, 224, interpolation="bilinear", name='resize'),
        base_efficientnetb0,
        keras.layers.GlobalAveragePooling2D(name='avg_pool'),
        keras.layers.Dense(1024, activation='relu', name='fully_conn1'),
        keras.layers.Dense(512, activation='relu', name='fully_conn2'),
        keras.layers.Dense(3, activation='softmax', name='out_softmax'),
    ], name = 'efficientnetb0')

    efficientnetb0.compile(optimizer=keras.optimizers.Adam(learning_rate=LEARNING_RATE),
                  loss='categorical_crossentropy',
                  metrics=evaluation.get_training_metrics()
                 )
    return efficientnetb0

with strategy.scope():
    efficientnetb0 = make_efficientnetb0()
```

```
if TRAIN_EFFICIENTNETB0:
    efficientnetb0_history, efficientnetb0_time = train_model(efficientnetb0, save=True)
```

9.2) Comment la méthode d'entraînement fonctionne?

```
def train_model(model, save=False):
    start_time = time.time()
    history = model.fit(small_train_ds,
                        validation_data=small_val_ds,
                        epochs=EPOCHS,
                        class_weight=dic_class_weights,
                        callbacks=[checkpoint_cb(model)],
                        )
    training_time = time.time() - start_time

    if save:
        save_history(model, history, training_time)

    return history, training_time
```

5) To do.