Rapport de projet

Penne Alexandre, Lattard Grégoire

M1 Technologie de l'internet
IA pour système cyber physique

Choix du dataset

Pour le dataset, il nous fallait quelque chose d'assez conséquent pour être pertinent mais également obtenable simplement depuis un lien afin d'être utilisable directement dans google Colab sans avoir à donner directement un lourd dossier de Dataset en local, pour ce faire, nous avons opté pour un dataset obtenable sur Kaggle car il est aisé de l'importer sur google colab par un simple lien

Plus spécifiquement, nous avons choisi un dataset de reconnaissance de fruits car il possède un nombre d'image conséquent et liste 15 classes, ce qui est intéressant pour l'exercice de classification

Le dataset étant déjà suffisamment grand, il n'est pas nécessaire d'ajouter de la data augmentation pour minimiser l'overfitting

Nous avons malgré tout garder dans le notebook nos essais de variations des images pour nous preparer a l'utilisation de cette option si le besoin devait apparaitre

Parametres du modele

```
[ ]  #creation du modele dynamique (dropout pour ajouter une couche de dropout, batch_norm pour ajouter la couche de BatchNormalization)
     def create_model(nb_couches, dropout=False,batch_norm=False):
       model = tf.keras.models.Sequential()
       #normalization
       model.add(tf.keras.layers.Input(shape=(IMG_H, IMG_W, 3))) #redimensionne l'image dans la taille souhaitée
       model.add(tf.keras.layers.Rescaling(1./255)) #reduit les valeurs des pixels a la fourchette 0,1


       for i in range(nb_couches):
         model.add(tf.keras.layers.Conv2D((32*(i+1)),3,activation="relu"))
         if(batch_norm):
           model.add(tf.keras.layers.BatchNormalization())
         model.add(tf.keras.layers.MaxPooling2D())

       model.add(tf.keras.layers.Flatten())
       model.add(tf.keras.layers.Dense(128,activation='relu'))
       if(dropout):
         model.add(tf.keras.layers.Dropout(0.5))
       model.add(tf.keras.layers.Dense(len(class_names),activation='softmax'))

       return model
```

1 - Couche de Normalisation

```
#normalization
model.add(tf.keras.layers.Input(shape=(IMG_H, IMG_W, 3))) #redimensionne l'image dans la taille souhaitée
model.add(tf.keras.layers.Rescaling(1./255)) #reduit les valeurs des pixels a la fourchette 0,1
```

-La couche de Rescaling a pour but de réduire les valeurs des pixels a une fourchette de 0 a 1 afin de réduire la plage de données utilisées pour pouvoir optimiser le modèle. De plus, les fonctions d'activations utilisées plus tard tel que le softmax ont besoin d'avoir des valeurs comprises dans cette plage

-La couche Input a pour but de normaliser la taille de l'image en une valeur standardisée, Ici, on la réduit à la taille définie par IMG_H et IMG_W
Qui sont

```
[ ]  IMG_H = 100
     IMG_W = 100
     data_dir = Path("fruits")
```

Au départ, nous avions opté pour 180*180 ce qui ne nous posait pas de problemes particulier car nous avons pu avoir accès aux TPU de google colab, mais après avoir réalisé à quel point leur accès était limité, nous avons vite été contraint de réduire la taille de l'image pour éviter à notre runtime de planter à cause d'une utilisation de RAM trop élevée. 100*100 fut un bon compromis entre fidélité de base de l'image et utilisation des ressources

2 - Couches de convolution2D, MaxPooling et Flatten

```
for i in range(nb_couches):
  model.add(tf.keras.layers.Conv2D((32*(i+1)),3,activation="relu"))
  if(batch_norm):
    model.add(tf.keras.layers.BatchNormalization())
  model.add(tf.keras.layers.MaxPooling2D())

model.add(tf.keras.layers.Flatten())
```

-Conv2D est la couche de convolution habituelle de la reconnaissance d'image pour pouvoir reconnaître les "features" des images. Dans le contexte de notre application, elle est indispensable. Le paramètre du nombre de filtres est en dynamique avec i pour pouvoir évoluer avec la profondeur des couches car on veut qu'avec

la profondeur, le modèle cherche des caractéristiques plus abstraites et complexes.

-MaxPooling2D est la couche de pooling indispensable pour pouvoir réduire la dimension de notre images et features afin de réduire les paramètres pour accélérer les calculs mais aussi et surtout permettre de généraliser les features pour éviter l'over fitting

-Flatten : Les opérations de convolution et pooling ont pour résultat une matrice de plusieurs dimensions pour pouvoir utiliser les filtres sur l'image, cependant, pour pouvoir continuer, il nous faut un vecteur unidimensionnel. Flatten nous permet de réduire la matrice en ce vecteur souhaité pour préparer l'utilisation de nos entièrement connectées

3 - Couches Denses

```
model.add(tf.keras.layers.Dense(128,activation='relu'))
if(dropout):
  model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(len(class_names),activation='softmax'))
```

Les couches denses sont les couches entièrement connectées de notre modèle, on passe par une étape préliminaire de 128 résultats pour commencer à préparer le modèle à prendre en compte la complexité, puis la couche dense finale a pour but de tout reconnecter sur un nombre de neurones résultats finaux égal au nombre de classes de notre dataset afin d'interpréter nos résultats de prédictions

4 - Couches Dropout et BatchNormalisation

-La couche Dropout permet de désactiver des neurones de manière aléatoire durant l'entraînement, ce qui a pour but de forcer le réseau a mieux généraliser les données.

-La couche BatchNormalisation a également pour but d’aider à généraliser les données

Ces couches apparaissent de manière optionnelle pour pouvoir tester plus aisément leurs impacts à travers les tests qui suivront.

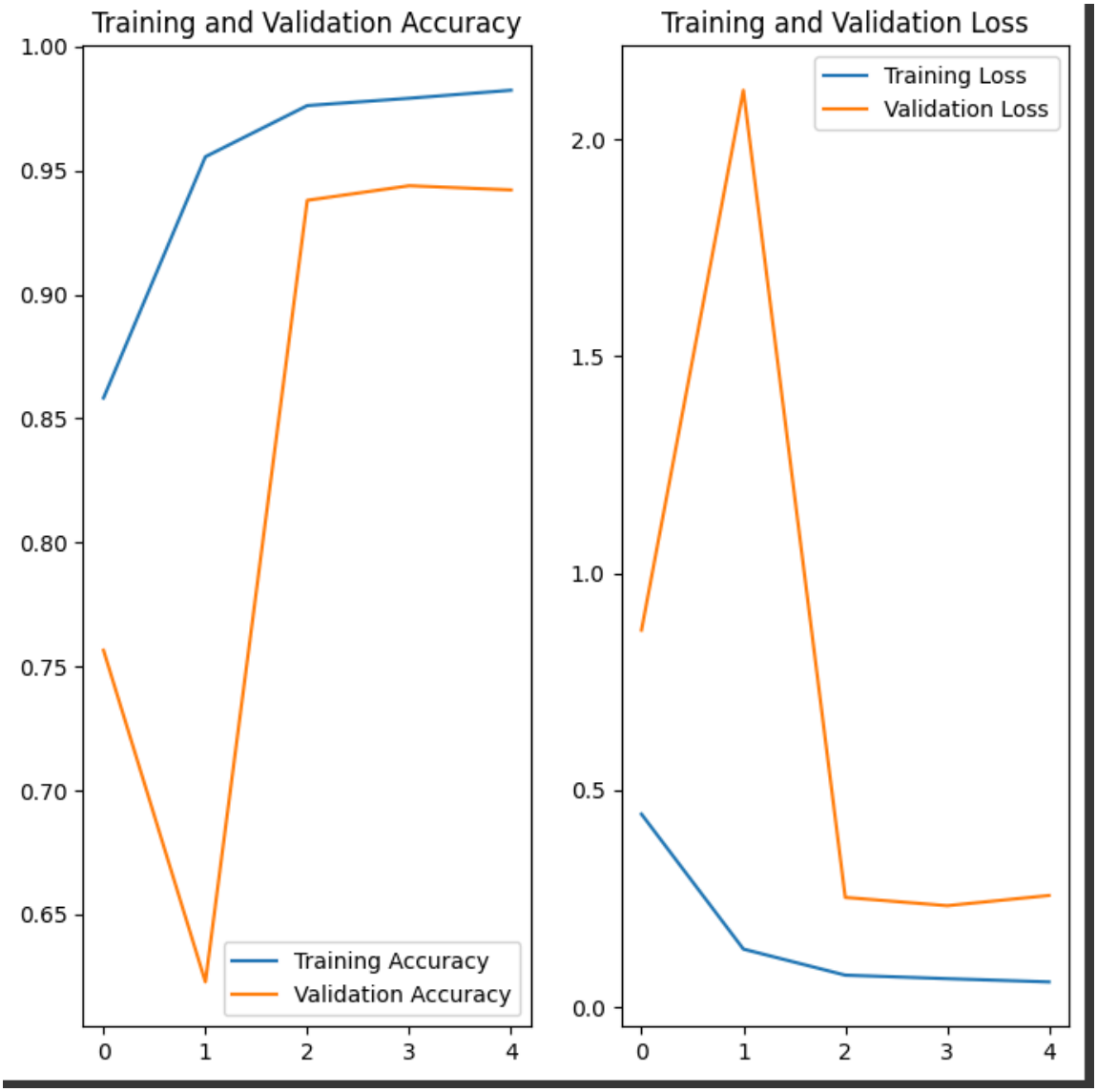
Tests des differentes variantes

Ici, nous avons réalisé 8 tests différents en ajustant les paramètres de notre modèle pour vérifier l’impact de ces paramètres, le nombre de couches a été testé a 3 et a 5 pour voir la différence d’efficacité avec la profondeur du réseau. 5 fut le maximum de couche utilisable pour notre modèle car allez plus profond n’est tout simplement pas possible avec la réduction de l’image

A - 3 couches sans dropout sans BatchNormalisation

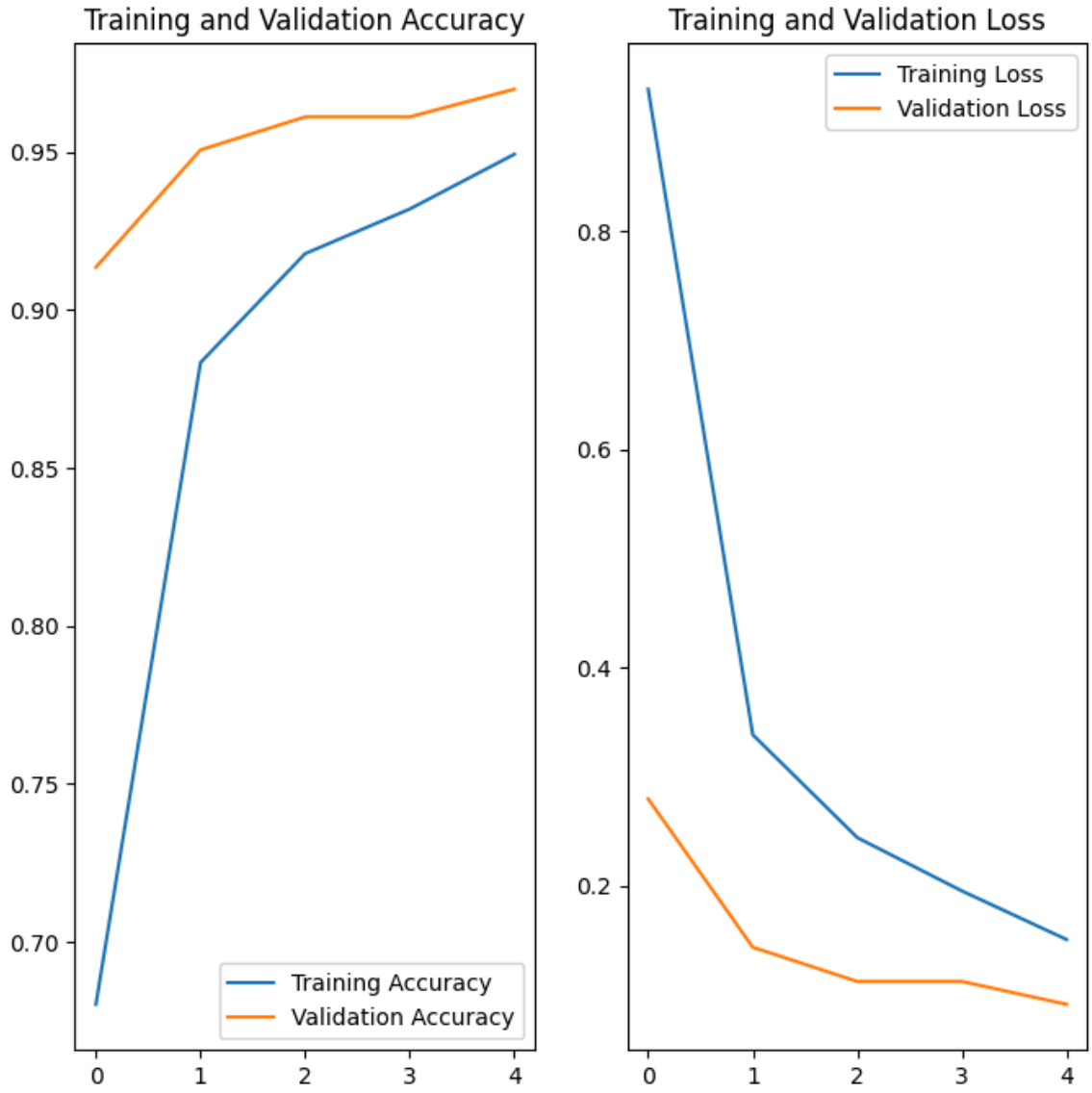

```
loss: 0.0576 - accuracy: 0.9807 - val_loss: 0.3885
- val_accuracy: 0.9013
```

B - 3 couches sans dropout avec BatchNormalisation



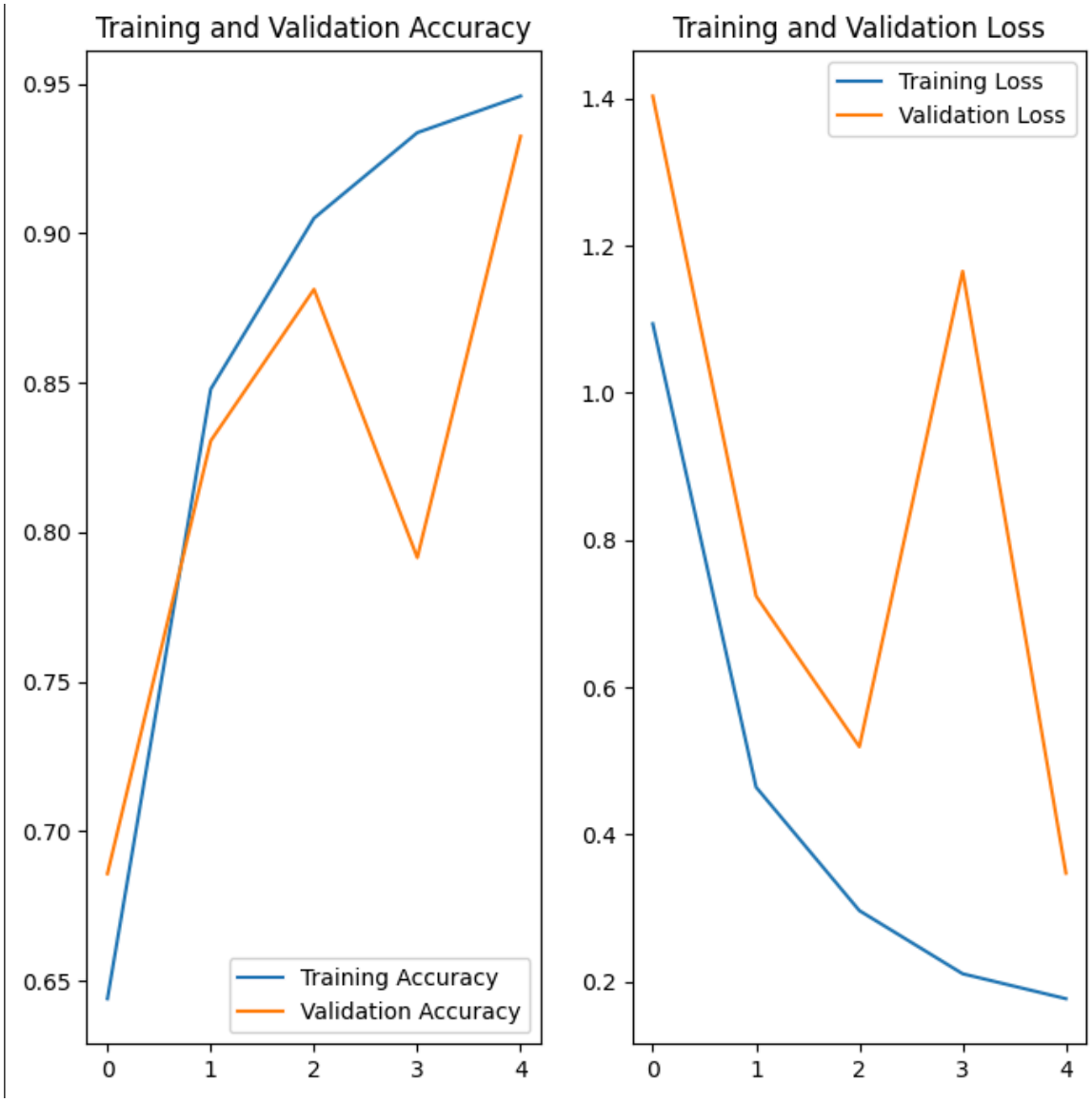

```
loss: 0.0576 - accuracy: 0.9824 - val_loss: 0.2569
- val_accuracy: 0.9422
```

C - 3 couches avec dropout sans BatchNormalisation




```
loss: 0.1507 - accuracy: 0.9492 - val_loss: 0.0912
- val_accuracy: 0.9698
```

D - 3 couches avec dropout avec BatchNormalisation




```
loss: 0.1770 - accuracy: 0.9459 - val_loss: 0.3476
- val_accuracy: 0.9325
```

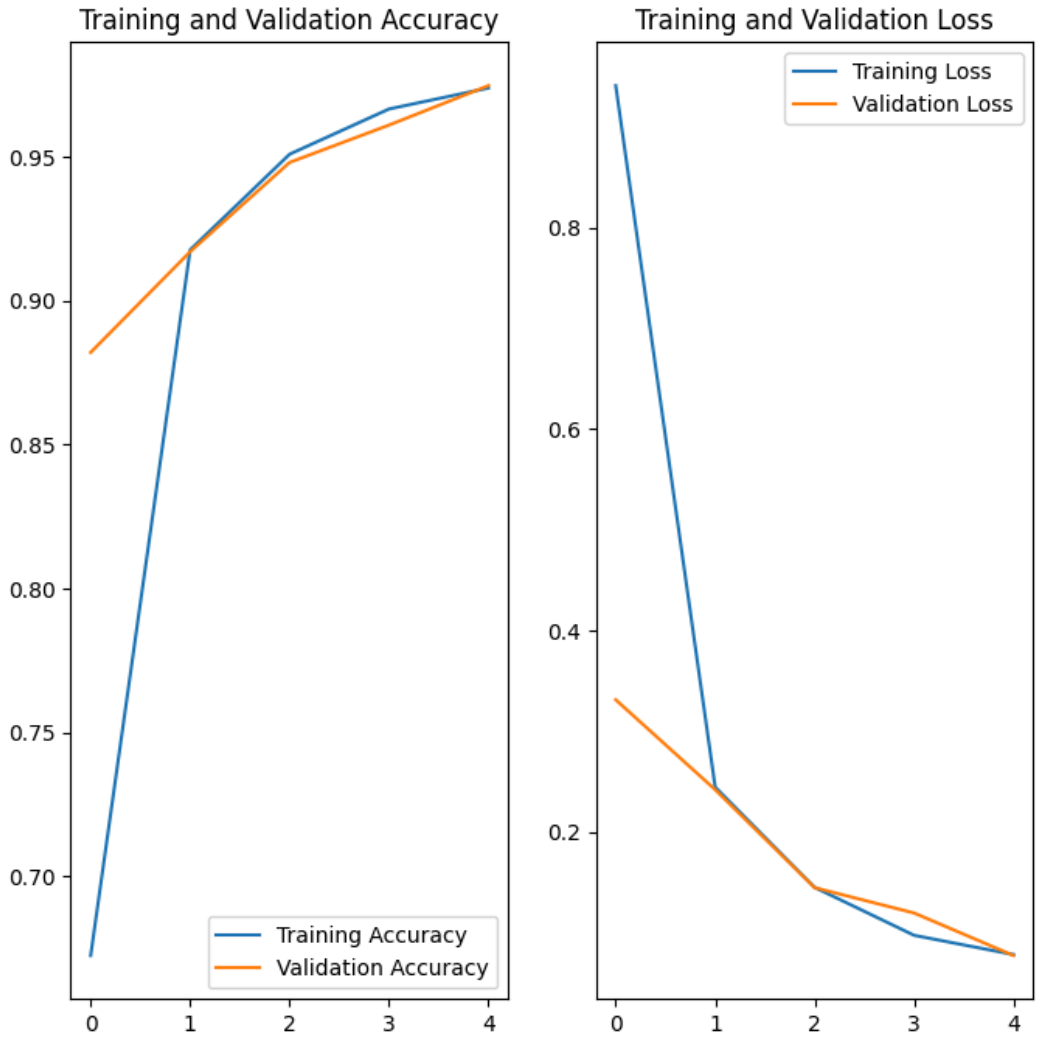
Sans Dropout et Sans BatchNormalisation: bonne performance avec une accuracy de 0.9807, mais une validation accuracy relativement faible (0.9013) et une validation loss élevée (0.3885)

Avec BatchNormalisation: L'ajout de BatchNormalisation améliore significativement la validation accuracy (0.9422) et réduit la validation loss (0.2569), tout en maintenant une accuracy similaire (0.9824).

Avec Dropout: L'ajout de Dropout sans BatchNormalisation augmente la validation accuracy à 0.9698, avec une validation loss beaucoup plus faible (0.0912), bien que la accuracy soit légèrement réduite (0.9492).

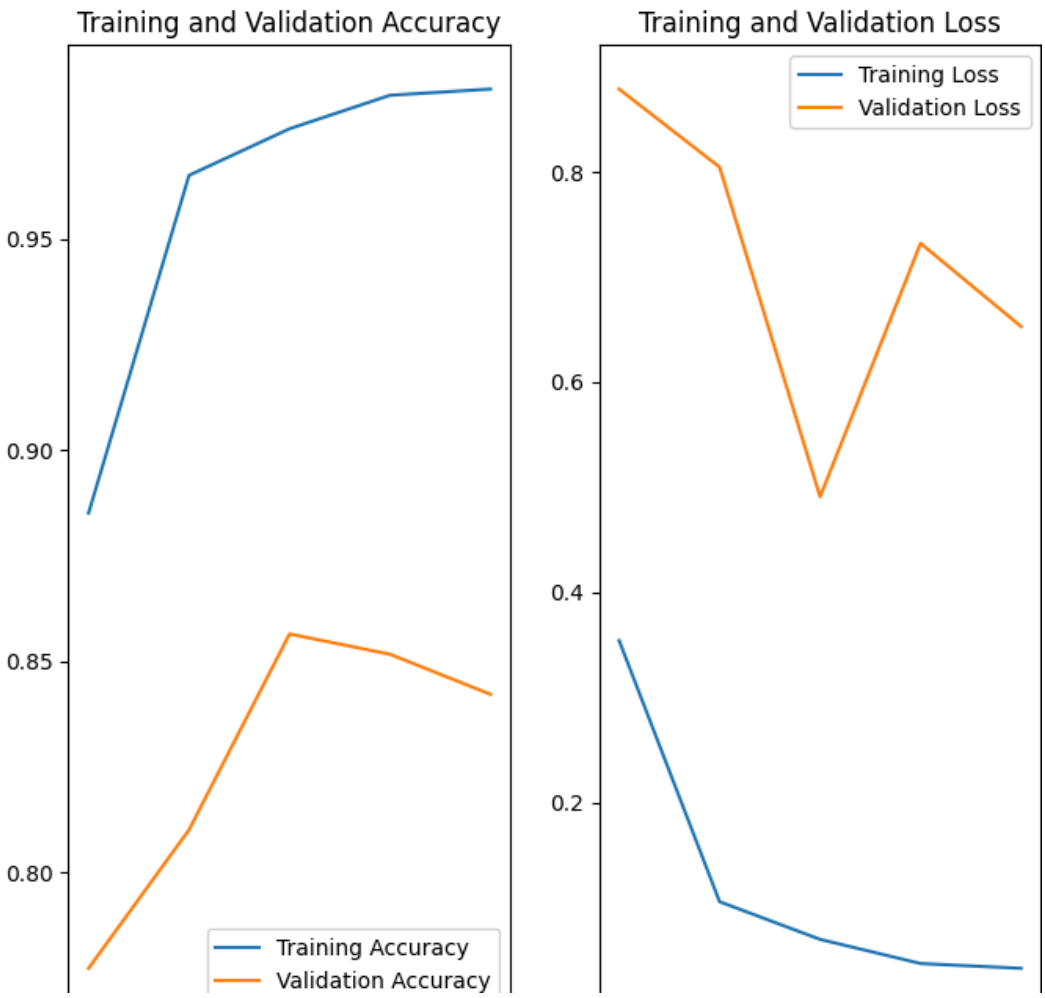
Avec Dropout et BatchNormalisation: Combiner les deux techniques ne donne pas les meilleurs résultats pour 3 couches, avec une validation accuracy de 0.9325 et une validation loss de 0.3476.

E - 5 couches, sans dropout, sans batchNormalization



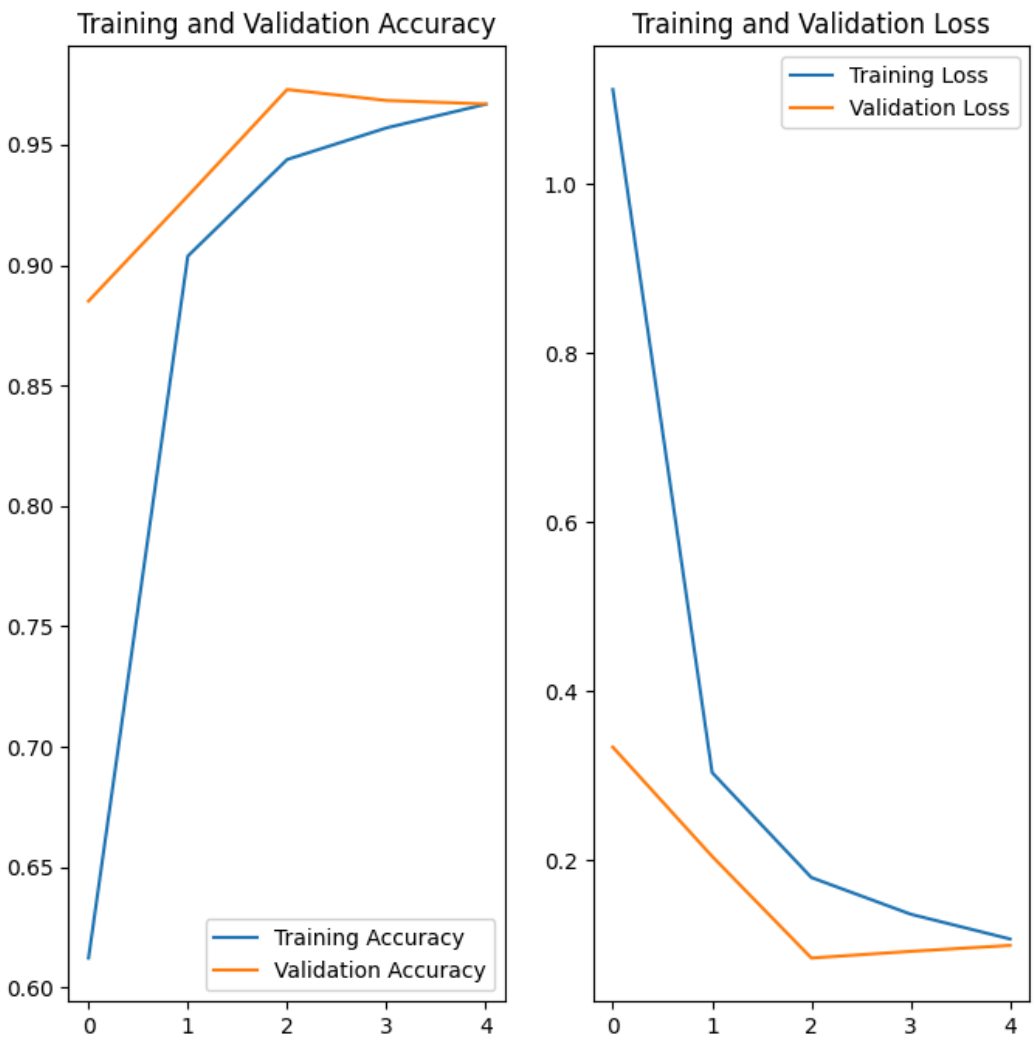

```
loss: 0.0783 - accuracy: 0.9738 - val_loss: 0.0774
- val_accuracy: 0.9746
```

F - 5 couches sans dropout, avec batchNormalization




```
loss: 0.0425 - accuracy: 0.9855 - val_loss: 0.6529
- val_accuracy: 0.8421
```

G - 5 couches, avec dropout, sans batchNormalization




```
loss: 0.1062 - accuracy: 0.9668 - val_loss: 0.0986
- val_accuracy: 0.9670
```

H - 5 couches, avec dropout, avec batchNormalization


```
loss: 0.0683 - accuracy: 0.9784 - val_loss: 0.6855
- val_accuracy: 0.8409
```

Sans Dropout et Sans BatchNormalisation: Le modèle a une bonne performance globale avec une validation accuracy de 0.9746 et une validation loss de 0.0774.

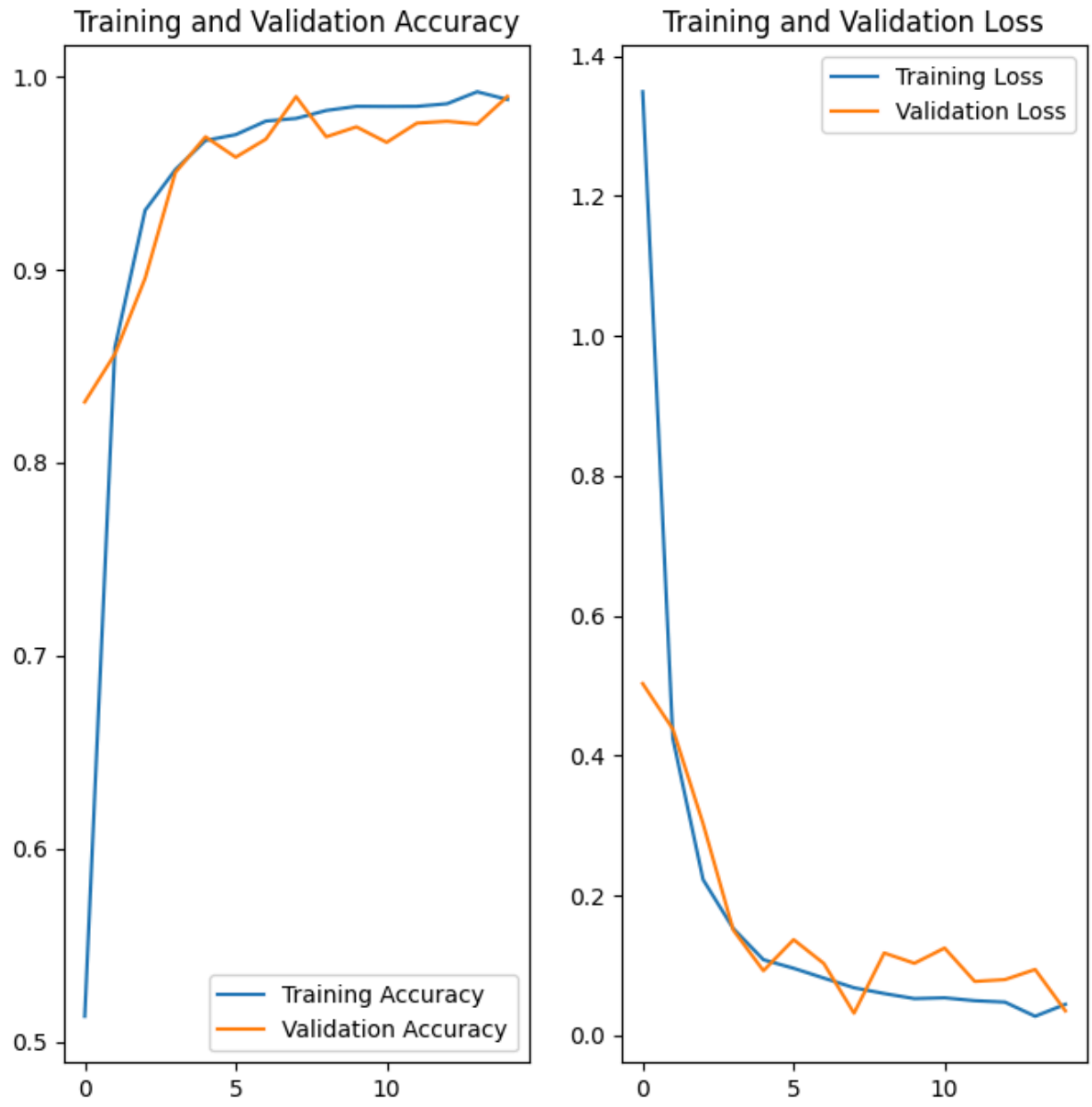
Avec BatchNormalisation: Bien que l'accuracy (0.9855) soit élevée, la validation loss est significativement plus élevée (0.6529) et la validation accuracy est plus basse (0.8421).

Avec Dropout: L'ajout de Dropout sans BatchNormalisation maintient une bonne validation accuracy (0.9670) et une validation loss raisonnable (0.0986).

Avec Dropout et BatchNormalisation: Le modèle présente une validation accuracy basse (0.8409) et une validation loss élevée (0.6855).

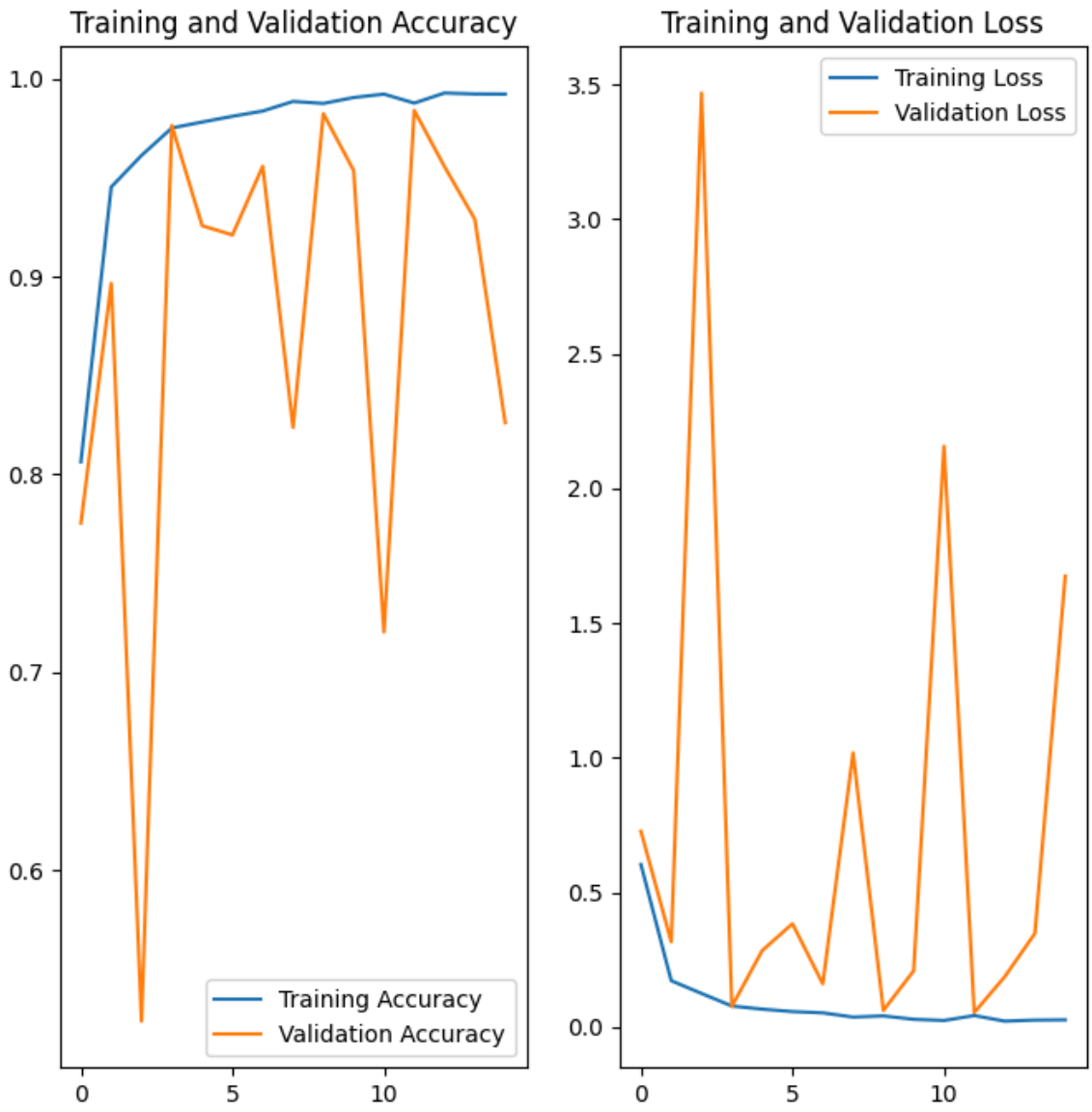
Les Comparaisons de ces tests montrent que le reseau avec le plus de couches est le plus efficace, cependant, l’efficacité de l’utilisation du dropout reste encore floue et la trop grande variance de resultats sur l’utilisation de BatchNormalisation font que nous avons decidé de retester ces parametres sur 5 couches de convolution avec plus d’epoch pour s’assurer de nos resultats

ExA - 15 epochs avec dropout



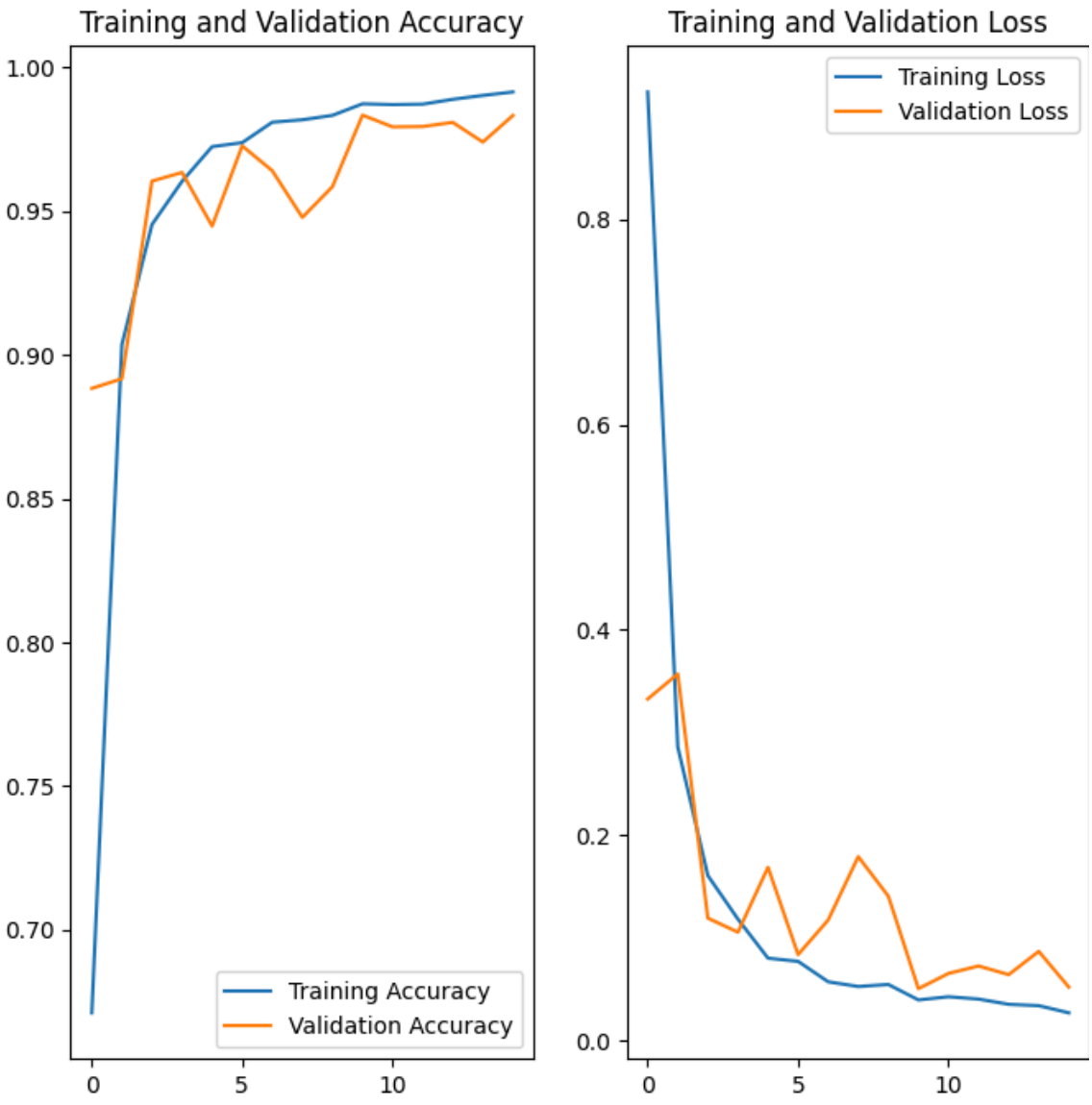
loss: 0.0444 - accuracy: 0.9881 - val_loss: 0.0351 - val_accuracy: 0.9897

ExB - 15 epochs avec normalisation et dropout




```
loss: 0.0270 - accuracy: 0.9924 - val_loss: 1.6741
- val_accuracy: 0.8262
```

ExC - 15 epochs ni l un ni l autre




```
loss: 0.0267 - accuracy: 0.9915 - val_loss: 0.0518
- val_accuracy: 0.9833
```

Avec Dropout: Le modèle affiche une excellente validation accuracy (0.9897) et une validation loss très faible (0.0351).

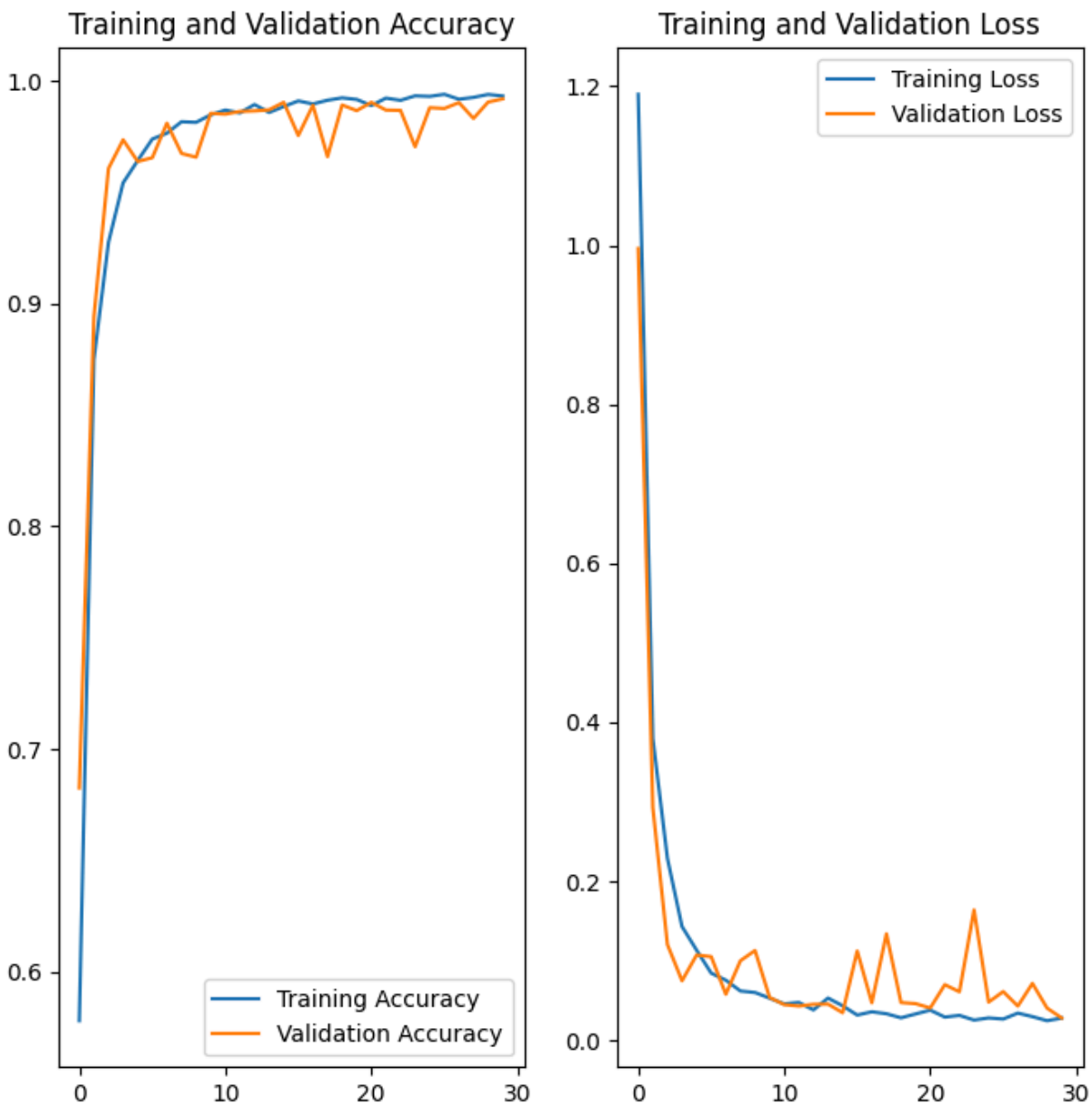
Avec Normalisation et Dropout: Bien que l'accuracy soit élevée (0.9924), la validation loss est extrêmement élevée (1.6741) et la validation accuracy relativement basse (0.8262), suggérant un surapprentissage.

Sans Dropout ni BatchNormalisation: Le modèle montre un bon équilibre avec une validation accuracy de 0.9833 et une validation loss de 0.0518.

Grâce à ces nouveaux tests, on peut écarter définitivement l'utilisation de bach normalisation qui nous donne les pires résultats de nos tests. On peut également voir que DropOut est réellement efficace sur la durée car elle fini par produire le meilleur resultat 0.9897 contre 0.9833 sur la val_accuracy et a l'air de grandement reduire l'overfitting que fait le modèle de base avec le temps.

Entraînement Final 30 epochs

Une fois que nous avons établi ce qui nous semble être la meilleure configuration, nous avons décidé de l'entraîner sur une durée significativement plus longue afin de pouvoir en extraire un maximum de potentiel.




```
loss: 0.0280 - accuracy: 0.9932 - val_loss: 0.0282
- val_accuracy: 0.9919
```

Influence de la Batch Normalisation

1 Sans Dropout et Avec Batch Normalisation:

- Pour les modèles à 3 couches, l'ajout de Batch Normalisation a conduit à une amélioration notable de la validation accuracy, passant de 0.9013 à 0.9422. Cela montre que Batch Normalisation aide à stabiliser et accélérer l'apprentissage, en réduisant les effets des covariances internes.

- Cependant, pour les modèles à 5 couches, bien que l'accuracy ait augmenté à 0.9855, la validation loss a considérablement augmenté à 0.6529, et la validation accuracy a chuté à 0.8421. Cela pourrait indiquer que Batch Normalisation, dans ce contexte, mène à un surapprentissage ou à un mauvais ajustement aux données de validation.

2 Avec Dropout et Batch Normalisation:

- Pour les modèles à 3 couches, cette combinaison n'a pas donné les meilleurs résultats, avec une validation accuracy de 0.9325 et une validation loss de 0.3476. Cela suggère que l'effet combiné peut dépendre du nombre de couches ou de la complexité du modèle.

- Pour les modèles à 5 couches, la validation accuracy est restée basse (0.8409) avec une validation loss élevée (0.6855), indiquant une possible incompatibilité ou un besoin d'ajustement plus fin des hyperparamètres.

Effet du Dropout

1 Avec Dropout, Sans Batch Normalisation:

- Pour les modèles à 3 couches, l'ajout de Dropout a permis d'atteindre une validation accuracy élevée (0.9698) et une validation loss basse (0.0912), montrant son efficacité dans la prévention du surapprentissage.

- Pour les modèles à 5 couches, Dropout a également donné de bons résultats, avec une validation accuracy de 0.9670 et une validation loss de 0.0986, confirmant son rôle bénéfique dans des architectures plus profondes.

Comparaison entre nos différentes architectures

1 3 Couches vs 5 Couches:

-Les modèles à 3 couches sans Dropout et sans Batch Normalisation ont montré une validation accuracy plus faible (0.9013) par rapport aux modèles à 5 couches (0.9746), suggérant que des architectures plus profondes peuvent mieux capturer la complexité des données.

-Cependant, l'ajout de techniques de régularisation (Dropout et Batch Normalisation) a permis d'améliorer les performances des modèles à 3 couches de manière significative.

2 Optimisation par la Durée de l'Entraînement:

- Les modèles entraînés pendant 15 epochs ont montré une performance variable. Par exemple, le modèle avec Dropout a atteint une validation accuracy de 0.9897, tandis que le modèle avec Normalisation et Dropout a vu sa validation loss s'envoler à 1.6741, malgré une accuracy élevée.

- L'entraînement final pendant 30 epochs avec le modèle le plus performant a permis d'obtenir des résultats excellents avec une validation accuracy de 0.9919 et une validation loss de 0.0282,

montrant l'importance de choisir la bonne architecture et la bonne durée d'entraînement.

Synthèse détaillée des tests

1 Batch Normalisation:

- Peut améliorer l'accuracy de l'entraînement mais nécessite une attention particulière pour éviter l'augmentation de la validation loss.
- Son efficacité peut être contextuelle, dépendant du nombre de couches et des configurations spécifiques.

2 Dropout:

- Montre une capacité constante à améliorer la généralisation en réduisant le surapprentissage.
- Particulièrement efficace dans les modèles à 3 couches.

3 Combinaison des Techniques:

- L'effet combiné de Batch Normalisation et Dropout n'est pas toujours bénéfique et peut nécessiter des ajustements fins.
- Les résultats montrent que ces techniques ne sont pas un remède universel et doivent être adaptées en fonction de la structure du modèle.

4 Nombre de Couches et Durée de l'Entraînement:

- Les architectures plus profondes (5 couches) ont montré une meilleure performance globale, mais avec un risque de surapprentissage.

- Un entraînement prolongé (30 epochs) avec le bon modèle a donné des résultats optimaux, soulignant l'importance de la durée d'entraînement.

Resultat de detection final

Le Résultat de notre modèle final étant très bon, nous avons décidé de le tester dans la dernière cellule de code du notebook qui génère et visualise aléatoirement des images avec leurs prédictions par le modèle pour pouvoir vérifier manuellement son efficacité

```
import random
import os
test_ds = os.listdir("tests")

for i in range(4):
  ax = plt.subplot(2, 2, i+1) #les 2 premiers parametres multipliés doivent etre egal a i
  reponse = random.choice(test_ds)
  rand_img = Path("tests/"+reponse)
  img = tf.keras.utils.load_img(
  rand_img, target_size=(IMG_H, IMG_W)
  )
  img_array = tf.keras.utils.img_to_array(img)
  plt.imshow(normalization(img_array))
  img_array = tf.expand_dims(img_array, 0) # Create a batch

  predictions = modelFinal.predict(img_array)
  score = tf.nn.softmax(predictions[0])



  plt.title("pred:"+class_names[np.argmax(score)]+"\n"+reponse)
  plt.axis("off")#desactiv les axes, osef on veut juste voir les images
plt.tight_layout()
```

```
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 24ms/step
```

pred:Plum
PlumyAB1174.png

pred:muskmelon
Muskmelon 001117.png

pred:Kiwi
Kiwi001119.png

pred:Mango
Mango001172.png

Au dessus, la prédiction, en dessous, le nom de l'image
On peut voir ici que le modèle prédit avec justesse les fruits que nous lui présentons à partir du dossier de tests.