Projet Machine learning/Deep learning

Feuille de route

Ce projet a été réalisé par N'ZAOU Renaud, SERRA Mathieu et LAURENT Gautier

Le but du projet est de construire des modèles basés sur des différentes architectures dans un but de classification d'images de fruits provenant d'un dataset de 961 images constitué par nous mêmes.

Nous avons trois classes à notre disposition, réparties en jeu d'entraînement et de validation :

- Une classe « Orange » (244 échantillons pour l'entraînement, 132 pour la validation)
- Une classe « Kiwi » (260 échantillons pour l'entraînement, 105 pour la validation)
- Une classe « Banane » (134 échantillons pour l'entraînement, 86 pour la validation)

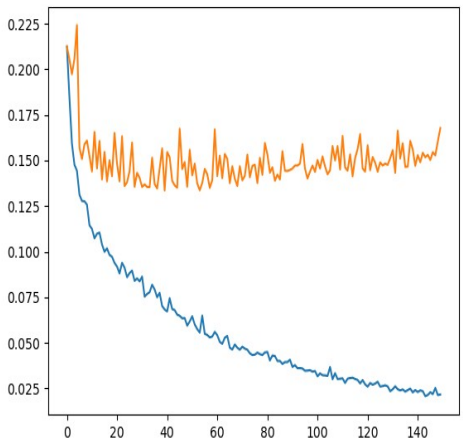
Les images sont transformées en matrices de 32x32 pixels RGB avant l'apprentissage.

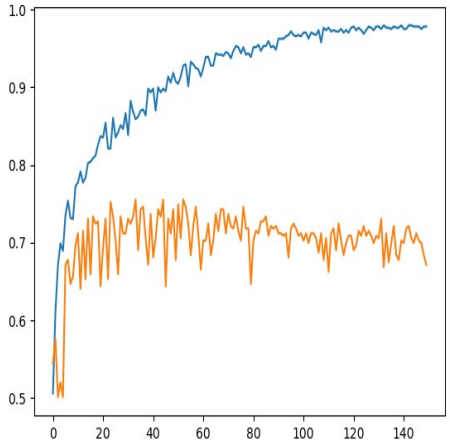
Implémentation d'un premier modèle linéaire

Fonction d'activation de sortie : sigmoïde
Nombre de cycles d'entraînements : 150

```
def create_linear_model():
    m = keras.models.Sequential()
    m.add(keras.layers.Flatten())
    m.add(keras.layers.Dense(3, activation=keras.activations.sigmoid))
    m.compile(optimizer=keras.optimizers.Adam(lr=0.0001),
              loss=keras.losses.mean_squared_error,
              metrics=['accuracy'])
    return m
```

Ce modèle ne contient qu'une seule couche qui est aussi la couche de sortie + une simple couche « Flatten » qui permet de transformer les données d'entrée (tableau de pixels eux mêmes composés de 3 valeurs par pixel) en un seul vecteur linéaire.





Nom du modèle : linear1.keras

accuracy : 0.9781
val_accuracy : 0.6718

Résultat observés : le modèle arrive très rapidement en situation d'overfit (à partir de seulement quelques cycles d'entraînement). Ce modèle paraît trop simpliste, il semble que même si on essayait de réduire fortement la longueur de l'apprentissage, on obtiendrait tout de même de mauvais résultats.

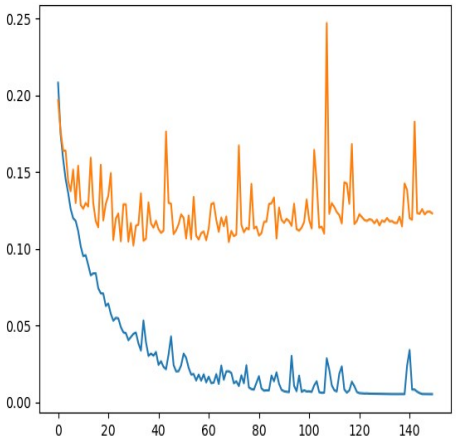
Implémentation d'un modèle de type perceptron multi-couche

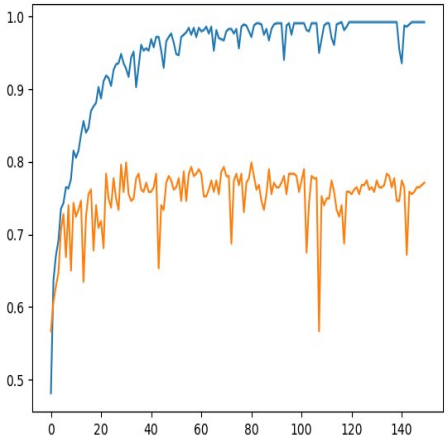
Modèle #1

Fonction d'activation de sortie : sigmoïde
Fonction d'activation pour les couches cachées : tanh
Nombre de couches cachées : 3
Nombre de neurones par couche : 32
Nombre de cycles d'entraînements : 150

```
def create_mlp_model():
    m = keras.models.Sequential()
    m.add(keras.layers.Flatten())
    m.add(keras.layers.Dense(32, activation=keras.activations.tanh))
    m.add(keras.layers.Dense(32, activation=keras.activations.tanh))
    m.add(keras.layers.Dense(32, activation=keras.activations.tanh))
    m.add(keras.layers.Dense(3, activation=keras.activations.sigmoid))
    m.compile(optimizer=keras.optimizers.Adam(lr=0.0001),
              loss=keras.losses.mean_squared_error,
              metrics=['accuracy'])
    return m
```

Sur ce modèle, on utilise des couches cachées qui propagent l'information de chaque neurone vers l'ensemble de ceux de la couche suivante. Cette architecture devrait être plus efficace pour résoudre notre problème non linéaire de classification d'image.





Nom du modèle : mlp1.keras

accuracy : 0.9922
val_accuracy : 0.7709

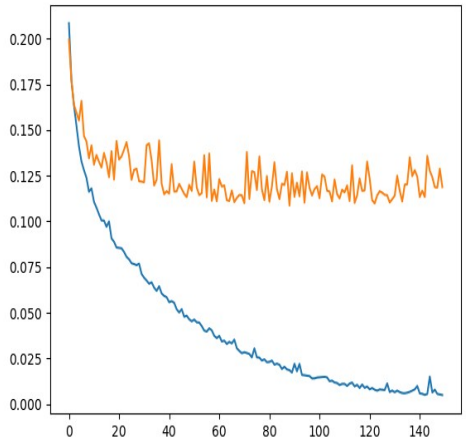
Résultat observés : Les résultats finaux sont meilleurs que lors de l'essai avec un modèle linéaire (+10 % dans val_accuracy). Le modèle est tout de même insuffisant pour correctement classifier. On a de fortes variations dans les courbes loss et accuracy côté jeu de test. En jouant avec le paramètre « lr » de l'optimizer Adam, on obtient des courbes plus lisses au prix d'un apprentissage plus long, mais les résultats finaux sont similaires.

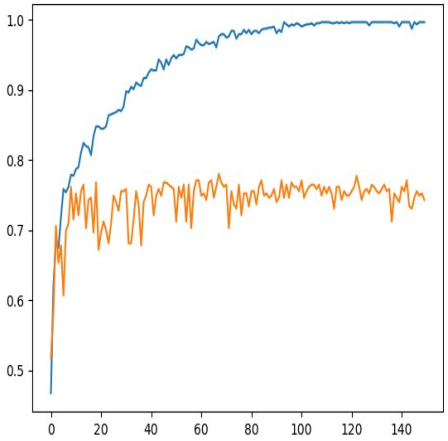
Modèle #2

Fonction d'activation de sortie : sigmoïde
Fonction d'activation pour les couches cachées : tanh
Nombre de couches cachées : 1
Nombre de neurones par couche : 32
Nombre de cycles d'entraînements : 150

```
def create_mlp_model():
    m = keras.models.Sequential()
    m.add(keras.layers.Flatten())
    m.add(keras.layers.Dense(32, activation=keras.activations.tanh))
    m.add(keras.layers.Dense(3, activation=keras.activations.sigmoid))
    m.compile(optimizer=keras.optimizers.Adam(lr=0.0001),
              loss=keras.losses.mean_squared_error,
              metrics=['accuracy'])
    return m
```

On cherche ici à simplifier le modèle en supprimant des couches.





Nom du modèle : mlp2.keras

accuracy : 0.9969
val_accuracy : 0.7430

Résultat observés :L'apprentissage trace des courbes aux formes moins erratiques mais les résultats sont sensiblement les mêmes.

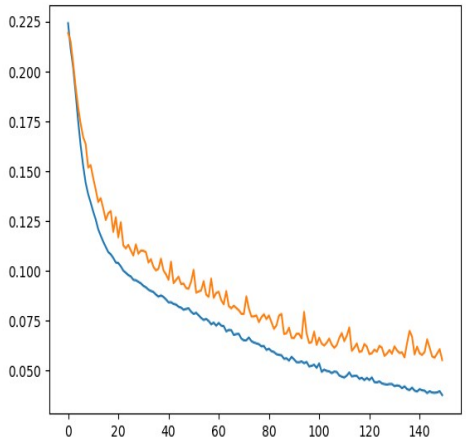
Implémentation d'un modèle de type réseau de neurones avec couches de convolutions

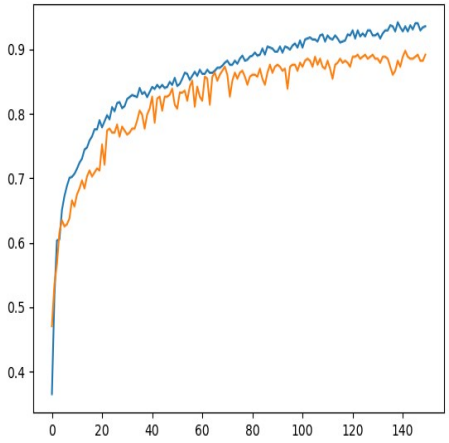
Fonction d'activation de sortie : sigmoïde
Nombre de couches cachées post-convolution : 1 (64 neurones)
Nombre de couches de convolution: 4 (nombres de filtres partant de 4, multiplié par 2 à chaque couche)
Taille des filtres : 2x2
Pooling : 2x2 en mode Max
Nombre de cycles d'entraînements : 100

```
def create_conv_nn_model():
    m = keras.models.Sequential()

    for i in range(1, 5):
        m.add(keras.layers.Conv2D(4 * i, kernel_size=(2, 2), activation=keras.activations.relu, padding='same'))
        m.add(keras.layers.MaxPool2D((2, 2)))

    m.add(keras.layers.Flatten())

    m.add(keras.layers.Dense(64, activation=keras.activations.tanh))
    m.add(keras.layers.Dense(3, activation=keras.activations.sigmoid))
    m.compile(optimizer=keras.optimizers.Adam(lr=0.0001),
              loss=keras.losses.mean_squared_error,
              metrics=['accuracy'])
    return m
```

Ce modèle devrait fonctionner beaucoup mieux sur des images que les modèles précédents car les filtres de convolution vont potentiellement mettre en évidence des formes spécifiques

pour chaque fruit, du moins entre banane (forme ovale) et kiwi/orange (forme ronde). Des formes plus détaillées pourraient être détectées sur les couches de filtres les plus complexes (par exemple, la différence de texture de la peau d'une orange et celle d'un kiwi).





Nom du modèle : cnn.keras

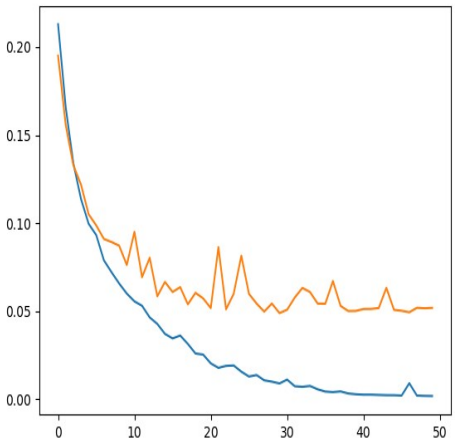
accuracy : 0.9357
val_accuracy : 0.8916

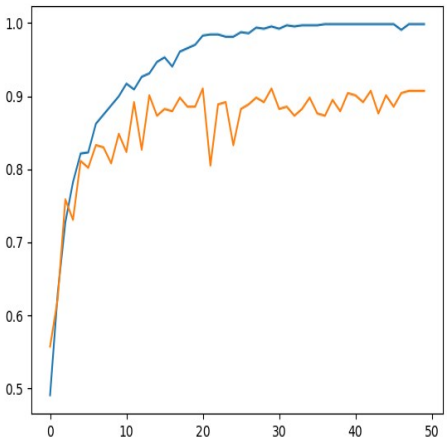
Résultat observés : Les résultats sont plutôt convaincants. L'accuracy sur le jeu d'entraînement est moindre par rapports aux modèles précédents, ce qui est normal étant donné les transformations effectuées lors des convolutions. On obtient près de 15 % de plus d'accuracy finale sur le jeu de test par rapport au modèles de type MLP.
Au delà de 150 cycles d'entraînement, le modèle avait tendance à être en situation d'overfitting (la courbe val_accuracy avait tendance à baisser).

Implémentation d’un modèle de type réseau de neurones avec couche de convolution non séquentiel (ResNet)

```
def create_residual_nn_model():
    input_tensor = keras.layers.Input((target_size[0], target_size[1], 3))

    previous_tensor = input_tensor

    next_tensor = keras.layers.Conv2D(16, kernel_size=(2, 2), activation=keras.activations.relu, padding='same')(input_tensor)
    previous_tensor = keras.layers.Concatenate()([previous_tensor, next_tensor])
    previous_tensor = keras.layers.MaxPool2D((2, 2))(previous_tensor)

    next_tensor = keras.layers.Conv2D(32, kernel_size=(2, 2), activation=keras.activations.relu, padding='same')(previous_tensor)
    previous_tensor = keras.layers.Concatenate()([previous_tensor, next_tensor])
    previous_tensor = keras.layers.MaxPool2D((2, 2))(previous_tensor)

    next_tensor = keras.layers.Conv2D(64, kernel_size=(2, 2), activation=keras.activations.relu, padding='same')(previous_tensor)
    previous_tensor = keras.layers.Concatenate()([previous_tensor, next_tensor])
    previous_tensor = keras.layers.MaxPool2D((2, 2))(previous_tensor)

    previous_tensor = keras.layers.Flatten()(previous_tensor)
    next_tensor = keras.layers.Dense(64, activation=keras.activations.tanh)(previous_tensor)
    next_tensor = keras.layers.Dense(3, activation=keras.activations.sigmoid)(next_tensor)

    m = keras.models.Model(input_tensor, next_tensor)
    m.compile(optimizer=keras.optimizers.Adam(lr=0.0001),
              loss=keras.losses.mean_squared_error,
              metrics=['accuracy'])
    return m
```

Fonction d’activation de sortie : sigmoïde
Nombre de couches cachées post-convolution : 1 (64 neurones)
Nombre de couches de convolution: 3 (nombres de filtres partant de 16, multiplié par 2 à chaque couche)
Taille des filtres : 2x2
Pooling : 2x2 en mode Max
Nombre de cycles d’entraînements : 50





Nom du modèle : resnet1.keras

accuracy : 0.9984
val_accuracy : 0.9071

Résultat observés : Les résultats sont similaires à ceux du modèle de type convolutional neural network, avec un apprentissage qui semble beaucoup plus rapide. Le nombre de cycles d’entraînement a du être revu à la baisse pour éviter l’overtfitting.