Extraction de Fichiers à Partir d'une Archive ZIP en Python

```
import zipfile

zip_path = '/content/archive (1).zip'  # Remplacez par le nom exact de votre fichier
extract_path = '/content/FER13'  # Répertoire cible pour extraire les fichiers

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print(f"Fichiers extraits dans : {extract_path}")
```

```
Fichiers extraits dans : /content/FER13
```

Classification des Émotions avec VGG16 : Modèle Personnalisé pour FER2013

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

# Charger le modèle VGG16 pré-entraîné, sans la dernière couche (include_top=False)
vgg_base = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))

# Geler les couches du modèle pré-entraîné pour ne pas les réentraîner
for layer in vgg_base.layers:
    layer.trainable = False

# Ajouter de nouvelles couches pour la classification personnalisée
x = vgg_base.output
x = Flatten()(x)
x = Dense(256, activation='relu')(x)  # Couche dense avec 256 neurones
x = Dense(7, activation='softmax')(x)  # Couche de sortie avec 7 neurones et activation softmax pour la classification multi-classes

# Créer un nouveau modèle
model = Model(inputs=vgg_base.input, outputs=x)

# Compiler le modèle
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

# Résumé du modèle
model.summary()

# Préparation des données avec un générateur
train_datagen = ImageDataGenerator(
    rescale=1./255,  # Normalisation des images
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    'FER13/train',  # Dossier contenant les images d'entraînement (correspondant au dossier "train" dans FER13)
    target_size=(150, 150),  # Redimensionnement des images à 150x150
    batch_size=32,
    class_mode='categorical'  # Classification multi-classes
)

validation_generator = test_datagen.flow_from_directory(
    'FER13/test',  # Dossier contenant les images de test (correspondant au dossier "test" dans FER13)
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'  # Classification multi-classes
)

# Entraînement du modèle sur les nouvelles données
history = model.fit(
    train_generator,
    steps_per_epoch=100,  # Nombre de lots par époque
    epochs=10,  # Nombre d'époques d'entraînement
    validation_data=validation_generator,
    validation_steps=50
)

# Évaluation du modèle sur le jeu de validation
```

```
score = model.evaluate(validation_generator)
print(f"Accuracy sur le jeu de validation : {score[1] * 100:.2f}%")
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1,792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36,928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73,856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147,584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295,168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590,080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590,080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
flatten_1 (Flatten)	(None, 8192)	0
dense_2 (Dense)	(None, 256)	2,097,408
dense_3 (Dense)	(None, 7)	1,799

```
 Total params: 16,813,895 (64.14 MB)
 Trainable params: 2,099,207 (8.01 MB)
 Non-trainable params: 14,714,688 (56.13 MB)
Found 28709 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` cl
  self._warn_if_super_not_called()
100/100 ━━━━━━━━━━━━━━━━━━━━ 1201s 12s/step - accuracy: 0.2771 - loss: 1.8158 - val_accuracy: 0.4081 - val_loss: 1.5502
Epoch 2/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1180s 12s/step - accuracy: 0.3959 - loss: 1.5886 - val_accuracy: 0.4150 - val_loss: 1.5299
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1151s 12s/step - accuracy: 0.4226 - loss: 1.5070 - val_accuracy: 0.4437 - val_loss: 1.4802
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1156s 12s/step - accuracy: 0.4612 - loss: 1.4623 - val_accuracy: 0.4594 - val_loss: 1.4377
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 8s/step - accuracy: 0.4491 - loss: 1.4625/usr/lib/python3.10/contextlib.py:153: UserWarning: Your i
  self.gen.throw(typ, value, traceback)
100/100 ━━━━━━━━━━━━━━━━━━━━ 1008s 10s/step - accuracy: 0.4491 - loss: 1.4625 - val_accuracy: 0.4177 - val_loss: 1.5405
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1159s 12s/step - accuracy: 0.4589 - loss: 1.4372 - val_accuracy: 0.4319 - val_loss: 1.4541
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1208s 12s/step - accuracy: 0.4709 - loss: 1.3970 - val_accuracy: 0.4700 - val_loss: 1.3945
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1166s 12s/step - accuracy: 0.4794 - loss: 1.3969 - val_accuracy: 0.4588 - val_loss: 1.4249
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 1139s 11s/step - accuracy: 0.4442 - loss: 1.4216 - val_accuracy: 0.4531 - val_loss: 1.3998
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 963s 10s/step - accuracy: 0.4861 - loss: 1.3811 - val_accuracy: 0.4614 - val_loss: 1.4320
225/225 ━━━━━━━━━━━━━━━━━━━━ 1715s 8s/step - accuracy: 0.4732 - loss: 1.3865
Accuracy sur le jeu de validation : 47.62%
```

Importations

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import MobileNetV2
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

Amélioration avec Fine-Tuning

```
# Dégeler quelques couches du modèle pré-entraîné (fine-tuning)
for layer in vgg_base.layers[-4:]:  # Dégeler les 4 dernières couches convolutives
    layer.trainable = True

# Afficher un résumé des couches pour s'assurer des couches dégélées
print("Couches dégélées :")
for i, layer in enumerate(vgg_base.layers):
    print(f"Layer {i} - {layer.name} : {'Trainable' if layer.trainable else 'Frozen'}")

# Recompiler le modèle après avoir dégeler des couches
model.compile(
    optimizer=Adam(learning_rate=1e-5),  # Taux d'apprentissage réduit pour éviter un surajustement
    loss='categorical_crossentropy',    # Garder categorical_crossentropy pour plusieurs classes
    metrics=['accuracy']
)

# Ajouter des callbacks pour un meilleur contrôle de l'entraînement
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

# Sauvegarder le meilleur modèle pendant l'entraînement
checkpoint = ModelCheckpoint(
    'best_model.keras',   # Changer l'extension en .keras
    monitor='val_accuracy',  # Surveiller l'accuracy de validation
    save_best_only=True,     # Sauvegarder uniquement le modèle ayant la meilleure accuracy
    mode='max',              # Maximiser l'accuracy
    verbose=1
)

# Arrêter l'entraînement tôt si l'amélioration stagne
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=3,            # Nombre d'époques sans amélioration avant l'arrêt
    restore_best_weights=True,  # Restaurer les poids du meilleur modèle
    verbose=1
)

# Réentraîner le modèle (fine-tuning)
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=5,  # Réduire le nombre d'époques pour accélérer l'entraînement
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // validation_generator.batch_size,
    callbacks=[checkpoint, early_stopping]  # Utilisation des callbacks
)

# Évaluation finale
score = model.evaluate(validation_generator)
print(f"Accuracy finale sur le jeu de validation : {score[1] * 100:.2f}%")
```

```
Couches dégélées :
Layer 0 - input_layer_1 : Frozen
Layer 1 - block1_conv1 : Frozen
Layer 2 - block1_conv2 : Frozen
Layer 3 - block1_pool : Frozen
Layer 4 - block2_conv1 : Frozen
Layer 5 - block2_conv2 : Frozen
Layer 6 - block2_pool : Frozen
Layer 7 - block3_conv1 : Frozen
Layer 8 - block3_conv2 : Frozen
Layer 9 - block3_conv3 : Frozen
Layer 10 - block3_pool : Frozen
Layer 11 - block4_conv1 : Frozen
Layer 12 - block4_conv2 : Frozen
Layer 13 - block4_conv3 : Frozen
Layer 14 - block4_pool : Frozen
Layer 15 - block5_conv1 : Trainable
Layer 16 - block5_conv2 : Trainable
Layer 17 - block5_conv3 : Trainable
Layer 18 - block5_pool : Trainable
Epoch 1/5
897/897 ━━━━━━━━━━━━━━━━━━━━ 0s 9s/step - accuracy: 0.5057 - loss: 1.2949
Epoch 1: val_accuracy improved from -inf to 0.52651, saving model to best_model.keras
897/897 ━━━━━━━━━━━━━━━━━━━━ 9830s 11s/step - accuracy: 0.5057 - loss: 1.2949 - val_accuracy: 0.5265 - val_loss: 1.2264
Epoch 2/5
  1/897 ━━━━━━━━━━━━━━━━━━━━ 2:23:51 10s/step - accuracy: 0.7500 - loss: 0.9343
Epoch 2: val_accuracy did not improve from 0.52651
897/897 ━━━━━━━━━━━━━━━━━━━━ 35s 28ms/step - accuracy: 0.7500 - loss: 0.9343 - val_accuracy: 0.5000 - val_loss: 1.2467
```

```
Epoch 3/5
159/897 ━━━━━━━━━━━━━━━━━━━━ 1:51:59 9s/step - accuracy: 0.5568 - loss: 1.1721
```

Tracage des courbes

```
def plot_training_curves(history, title):
    # Courbes d'accuracy
    plt.figure(figsize=(12, 6))

    # Précision
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.title(f'{title} - Accuracy')
    plt.legend()

    # Perte
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title(f'{title} - Loss')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Tracer les courbes pour l'entraînement initial
plot_training_curves(history, "Initial Training")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-f6eaa0e48f8c> in <cell line: 27>()
     25
     26 # Tracer les courbes pour l'entraînement initial
---> 27 plot_training_curves(history, "Initial Training")
     28
     29

NameError: name 'history' is not defined
```

Étapes suivantes : Expliquer l'erreur

Double-cliquez (ou appuyez sur Entrée) pour modifier