In [1]:
```
# Import des librairies necessaires pour le transfer learning avec MobileNetV2
# ainsi que le suivi de l'entrainement et des ressourcesimport tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.callbacks import EarlyStopping, Callback
import psutil
import time
import GPUtil
import matplotlib.pyplot as plt
```

In [2]:
```
# Chargement des images pour le transfer learning avec redimensionnement
# Les donnees sont chargees depuis les dossiers d'entrainement et de validation
img_size = (224, 224)
batch_size = 32

train_ds = tf.keras.utils.image_dataset_from_directory(
    "../dataset/seg_train",
    image_size=img_size,
    batch_size=batch_size,
    label_mode="int"   # labels en int car on utilise une loss sparse en multicl
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    "../dataset/seg_test",
    image_size=img_size,
    batch_size=batch_size,
    label_mode="int"   # meme format de labels que pour le train
)
```

```
Found 14034 files belonging to 6 classes.
Found 3000 files belonging to 6 classes.
```

In [3]:
```
# Optimisation du chargement des donnees avec prefetch
# Permet d'accelerer l'entrainement en preparant les batches a l'avance
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.prefetch(buffer_size=AUTOTUNE)
```

In [4]:
```
# Fonctions pour recuperer l'utilisation du CPU, de la RAM et du GPU
# pendant l'entrainement du modele en transfer learning
def get_cpu_ram():
    cpu_percent = psutil.cpu_percent(interval=1)
    ram = psutil.virtual_memory()
    ram_used_mb = ram.used / (1024 ** 2)
    return cpu_percent, ram_used_mb
def get_gpu_stats():

    try:
        gpus = GPUtil.getGPUs()
        if not gpus:
            return None, None, None
        gpu = gpus[0]
        return gpu.load * 100, gpu.memoryUsed, gpu.memoryTotal
    except:
        return None, None, None
```

In [5]:
```
# Affichage de l'utilisation des ressources avant le debut de l'entrainement
cpu_before, ram_before = get_cpu_ram()
gpu_before, vram_used_before, vram_total_before = get_gpu_stats()

msg = f"AVANT entraînement → CPU: {cpu_before:.1f}% | RAM: {ram_before:.0f} MB"
if gpu_before is not None:
    msg += f" | GPU: {gpu_before:.1f}% | VRAM: {vram_used_before}/{vram_total_be
print(msg)
```

```
AVANT entraînement → CPU: 3.4% | RAM: 14949 MB | GPU: 45.0% | VRAM: 627.0/16303.0
MB
```

In [6]:
```
# Chargement du modele MobileNetV2 pre-entraine sur ImageNet
# Les poids sont figes pour utiliser le reseau comme extracteur de caracteristiq
base_model = MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    weights="imagenet"
)

base_model.trainable = False
```

In [7]:
```
# Construction du modele final a partir de MobileNetV2
# Ajout de couches denses pour adapter le modele a la classification binaire
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation="relu"),
    layers.Dropout(0.3),
    layers.Dense(6, activation="softmax")
])
```

In [8]:
```
# Compilation du modele avec les parametres adaptes a la classification binaire
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",# Fonction de perte adaptee a la clas
    metrics=["accuracy"]
)
```

In [9]:
```
# Mise en place de l'early stopping et d'un callback personnalise
# pour suivre le temps et l'utilisation des ressources pendant l'entrainement
early_stop = EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True
)

class PerformanceCallback(Callback):
    def on_epoch_begin(self, epoch, logs=None):
        self.start_time = time.time()

    def on_epoch_end(self, epoch, logs=None):
        cpu, ram = get_cpu_ram()
        gpu, vram_used, vram_total = get_gpu_stats()
        duration = time.time() - self.start_time

        msg = f" | CPU: {cpu:.1f}% | RAM: {ram:.0f} MB | Time: {duration:.1f}s"
        if gpu is not None:
```

```
            msg += f" | GPU: {gpu:.1f}% | VRAM: {vram_used}/{vram_total} MB"
        print(msg)
```

In [10]:
```
print("TensorFlow version:", tf.__version__)
print("GPUs disponibles :", tf.config.list_physical_devices("GPU"))
```

```
TensorFlow version: 2.10.1
GPUs disponibles : []
```

In [11]:
```
# Entrainement du modele en transfert learning avec suivi des performances
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=10,
    callbacks=[early_stop, PerformanceCallback()]
)
```

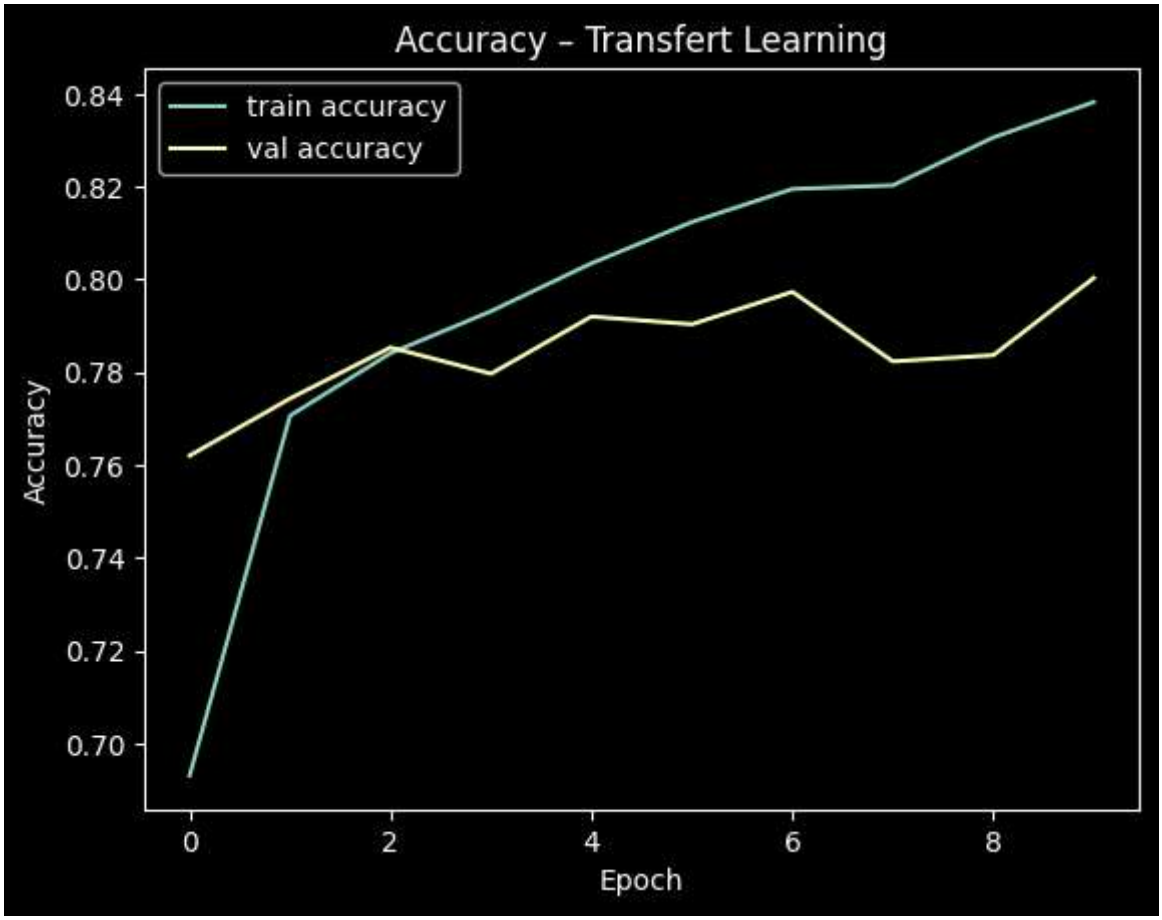
```
Epoch 1/10
439/439 [==============================] - ETA: 0s - loss: 0.7979 - accuracy: 0.6
930 | CPU: 2.0% | RAM: 15328 MB | Time: 96.9s | GPU: 0.0% | VRAM: 627.0/16303.0 M
B
439/439 [==============================] - 97s 219ms/step - loss: 0.7979 - accura
cy: 0.6930 - val_loss: 0.6240 - val_accuracy: 0.7620
Epoch 2/10
439/439 [==============================] - ETA: 0s - loss: 0.6109 - accuracy: 0.7
706 | CPU: 2.1% | RAM: 15388 MB | Time: 94.3s | GPU: 0.0% | VRAM: 627.0/16303.0 M
B
439/439 [==============================] - 94s 215ms/step - loss: 0.6109 - accura
cy: 0.7706 - val_loss: 0.5946 - val_accuracy: 0.7743
Epoch 3/10
439/439 [==============================] - ETA: 0s - loss: 0.5691 - accuracy: 0.7
841 | CPU: 1.5% | RAM: 15585 MB | Time: 95.6s | GPU: 1.0% | VRAM: 627.0/16303.0 M
B
439/439 [==============================] - 96s 218ms/step - loss: 0.5691 - accura
cy: 0.7841 - val_loss: 0.5698 - val_accuracy: 0.7853
Epoch 4/10
439/439 [==============================] - ETA: 0s - loss: 0.5396 - accuracy: 0.7
931 | CPU: 1.1% | RAM: 15621 MB | Time: 95.4s | GPU: 0.0% | VRAM: 627.0/16303.0 M
B
439/439 [==============================] - 95s 217ms/step - loss: 0.5396 - accura
cy: 0.7931 - val_loss: 0.5906 - val_accuracy: 0.7797
Epoch 5/10
439/439 [==============================] - ETA: 0s - loss: 0.5166 - accuracy: 0.8
035 | CPU: 4.7% | RAM: 15527 MB | Time: 94.0s | GPU: 0.0% | VRAM: 627.0/16303.0 M
B
439/439 [==============================] - 94s 214ms/step - loss: 0.5166 - accura
cy: 0.8035 - val_loss: 0.5606 - val_accuracy: 0.7920
Epoch 6/10
439/439 [==============================] - ETA: 0s - loss: 0.4909 - accuracy: 0.8
124 | CPU: 2.7% | RAM: 15497 MB | Time: 96.0s | GPU: 3.0% | VRAM: 627.0/16303.0 M
B
439/439 [==============================] - 96s 219ms/step - loss: 0.4909 - accura
cy: 0.8124 - val_loss: 0.5559 - val_accuracy: 0.7903
Epoch 7/10
439/439 [==============================] - ETA: 0s - loss: 0.4749 - accuracy: 0.8
195 | CPU: 2.0% | RAM: 16434 MB | Time: 99.2s | GPU: 1.0% | VRAM: 739.0/16303.0 M
B
439/439 [==============================] - 99s 226ms/step - loss: 0.4749 - accura
cy: 0.8195 - val_loss: 0.5587 - val_accuracy: 0.7973
Epoch 8/10
439/439 [==============================] - ETA: 0s - loss: 0.4641 - accuracy: 0.8
203 | CPU: 7.3% | RAM: 16665 MB | Time: 98.7s | GPU: 0.0% | VRAM: 774.0/16303.0 M
B
439/439 [==============================] - 99s 225ms/step - loss: 0.4641 - accura
cy: 0.8203 - val_loss: 0.5827 - val_accuracy: 0.7823
Epoch 9/10
439/439 [==============================] - ETA: 0s - loss: 0.4414 - accuracy: 0.8
306 | CPU: 2.0% | RAM: 16563 MB | Time: 99.0s | GPU: 0.0% | VRAM: 686.0/16303.0 M
B
439/439 [==============================] - 99s 225ms/step - loss: 0.4414 - accura
cy: 0.8306 - val_loss: 0.6026 - val_accuracy: 0.7837
Epoch 10/10
439/439 [==============================] - ETA: 0s - loss: 0.4270 - accuracy: 0.8
382 | CPU: 6.4% | RAM: 16393 MB | Time: 100.1s | GPU: 1.0% | VRAM: 887.0/16303.0
MB
439/439 [==============================] - 100s 228ms/step - loss: 0.4270 - accur
acy: 0.8382 - val_loss: 0.5538 - val_accuracy: 0.8003
```

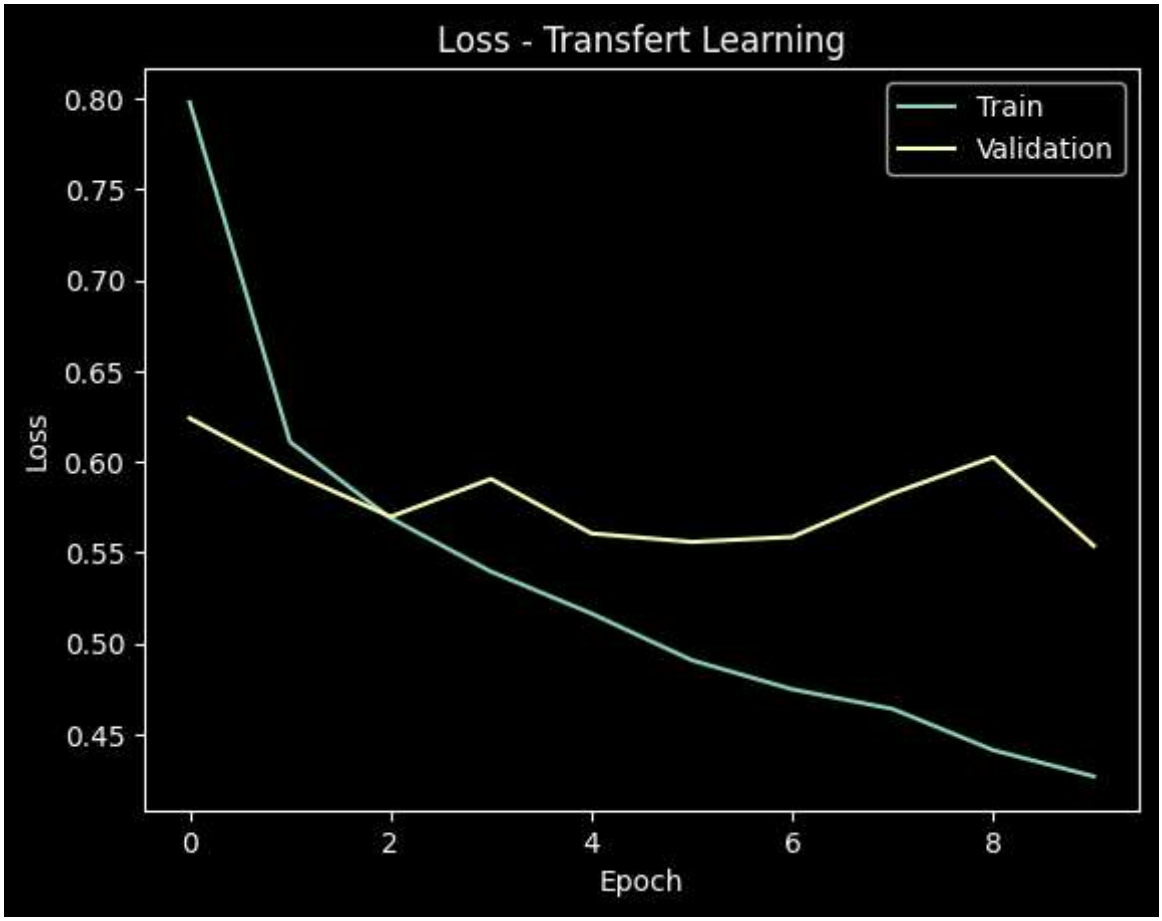
In [12]:
```
# Affichage de l'utilisation des ressources apres la fin de l'entrainement
cpu_after, ram_after = get_cpu_ram()
gpu_after, vram_used_after, vram_total_after = get_gpu_stats()

msg = f"APRÈS entraînement → CPU: {cpu_after:.1f}% | RAM: {ram_after:.0f} MB"
if gpu_after is not None:
    msg += f" | GPU: {gpu_after:.1f}% | VRAM: {vram_used_after}/{vram_total_afte
print(msg)
```

```
APRÈS entraînement → CPU: 4.4% | RAM: 16465 MB | GPU: 1.0% | VRAM: 953.0/16303.0
MB
```

In [13]:
```
plt.figure()
plt.plot(history.history["accuracy"], label="train accuracy")
plt.plot(history.history["val_accuracy"], label="val accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.title("Accuracy - Transfert Learning")
plt.show()
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(["Train", "Validation"])
plt.title("Loss - Transfert Learning")
plt.show()
```





Conclusion

L'entraînement en transfer learning sur ce dernier dataset montre une progression régulière des performances, avec une précision de validation maximale d'environ 80,0 % atteinte à l'epoch 10. La perte de validation diminue globalement pour se stabiliser autour de 0,55, ce qui traduit une généralisation correcte, malgré de légères fluctuations et un début de surapprentissage en fin d'entraînement.

L'utilisation des ressources matérielles reste globalement maîtrisée. Le CPU est peu sollicité (environ 1 à 7 %), la RAM reste stable entre 15 et 16,5 Go, et la VRAM utilisée demeure inférieure à 1 Go. Le GPU est faiblement exploité sur ce jeu de données, tandis que le temps d'entraînement par epoch est élevé (≈ 95 à 100 secondes), principalement dû à la taille du dataset et au modèle pré-entraîné.

In [13]: