In [1]:
```
# Import des librairies necessaires pour le transfer learning avec MobileNetV2
# ainsi que le suivi de l'entrainement et des ressources
import 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/training_set",
    image_size=img_size,
    batch_size=batch_size
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    "../dataset/test_set",
    image_size=img_size,
    batch_size=batch_size
)
```

```
Found 8000 files belonging to 2 classes.
Found 2000 files belonging to 2 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: 3.7% | RAM: 14331 MB | GPU: 21.0% | VRAM: 569.0/16303.
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
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9406464/9406464 [==============================] - 0s 0us/step
```

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(1, activation="sigmoid")
])
```

In [8]:
```
# Compilation du modele avec les parametres adaptes a la classification binaire
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    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]:
```
# 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()]
)
```

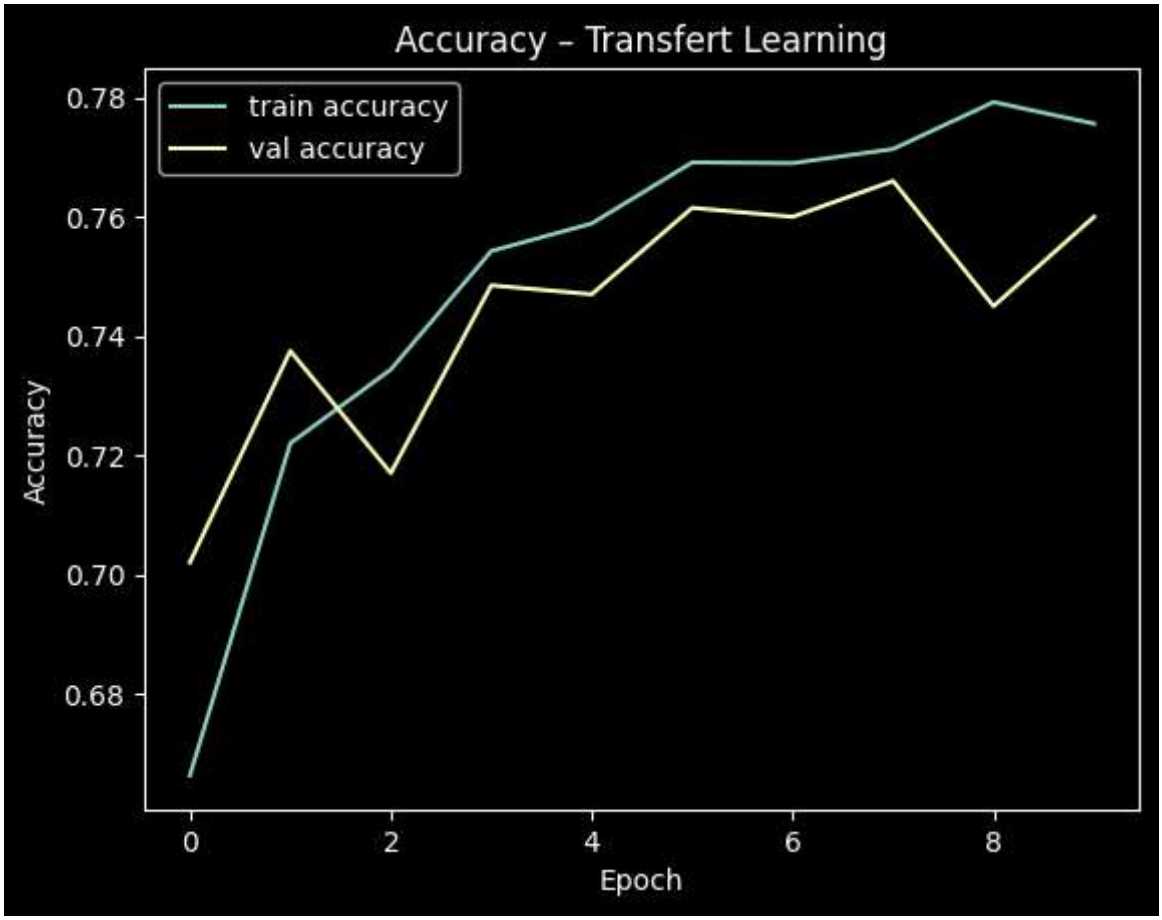
```
Epoch 1/10
250/250 [==============================] - ETA: 0s - loss: 0.6185 - accuracy: 0.6
662 | CPU: 2.0% | RAM: 14575 MB | Time: 57.9s | GPU: 1.0% | VRAM: 569.0/16303.0 M
B
250/250 [==============================] - 58s 228ms/step - loss: 0.6185 - accura
cy: 0.6662 - val_loss: 0.5607 - val_accuracy: 0.7020
Epoch 2/10
250/250 [==============================] - ETA: 0s - loss: 0.5484 - accuracy: 0.7
220 | CPU: 2.1% | RAM: 14644 MB | Time: 56.7s | GPU: 2.0% | VRAM: 569.0/16303.0 M
B
250/250 [==============================] - 57s 227ms/step - loss: 0.5484 - accura
cy: 0.7220 - val_loss: 0.5244 - val_accuracy: 0.7375
Epoch 3/10
250/250 [==============================] - ETA: 0s - loss: 0.5268 - accuracy: 0.7
344 | CPU: 2.5% | RAM: 14853 MB | Time: 62.0s | GPU: 0.0% | VRAM: 570.0/16303.0 M
B
250/250 [==============================] - 62s 248ms/step - loss: 0.5268 - accura
cy: 0.7344 - val_loss: 0.5495 - val_accuracy: 0.7170
Epoch 4/10
250/250 [==============================] - ETA: 0s - loss: 0.5106 - accuracy: 0.7
542 | CPU: 1.4% | RAM: 14702 MB | Time: 57.8s | GPU: 1.0% | VRAM: 599.0/16303.0 M
B
250/250 [==============================] - 58s 231ms/step - loss: 0.5106 - accura
cy: 0.7542 - val_loss: 0.5019 - val_accuracy: 0.7485
Epoch 5/10
250/250 [==============================] - ETA: 0s - loss: 0.4938 - accuracy: 0.7
589 | CPU: 1.0% | RAM: 14562 MB | Time: 57.9s | GPU: 0.0% | VRAM: 612.0/16303.0 M
B
250/250 [==============================] - 58s 232ms/step - loss: 0.4938 - accura
cy: 0.7589 - val_loss: 0.5069 - val_accuracy: 0.7470
Epoch 6/10
250/250 [==============================] - ETA: 0s - loss: 0.4791 - accuracy: 0.7
691 | CPU: 3.3% | RAM: 14492 MB | Time: 57.8s | GPU: 1.0% | VRAM: 612.0/16303.0 M
B
250/250 [==============================] - 58s 231ms/step - loss: 0.4791 - accura
cy: 0.7691 - val_loss: 0.4895 - val_accuracy: 0.7615
Epoch 7/10
250/250 [==============================] - ETA: 0s - loss: 0.4780 - accuracy: 0.7
690 | CPU: 3.8% | RAM: 14498 MB | Time: 57.9s | GPU: 39.0% | VRAM: 612.0/16303.0
MB
250/250 [==============================] - 58s 232ms/step - loss: 0.4780 - accura
cy: 0.7690 - val_loss: 0.4904 - val_accuracy: 0.7600
Epoch 8/10
250/250 [==============================] - ETA: 0s - loss: 0.4730 - accuracy: 0.7
714 | CPU: 4.1% | RAM: 14486 MB | Time: 57.6s | GPU: 5.0% | VRAM: 612.0/16303.0 M
B
250/250 [==============================] - 58s 230ms/step - loss: 0.4730 - accura
cy: 0.7714 - val_loss: 0.4874 - val_accuracy: 0.7660
Epoch 9/10
250/250 [==============================] - ETA: 0s - loss: 0.4590 - accuracy: 0.7
793 | CPU: 4.3% | RAM: 14617 MB | Time: 58.7s | GPU: 15.0% | VRAM: 627.0/16303.0
MB
250/250 [==============================] - 59s 235ms/step - loss: 0.4590 - accura
cy: 0.7793 - val_loss: 0.5007 - val_accuracy: 0.7450
Epoch 10/10
250/250 [==============================] - ETA: 0s - loss: 0.4558 - accuracy: 0.7
756 | CPU: 2.6% | RAM: 14625 MB | Time: 57.5s | GPU: 31.0% | VRAM: 627.0/16303.0
MB
250/250 [==============================] - 57s 230ms/step - loss: 0.4558 - accura
cy: 0.7756 - val_loss: 0.4937 - val_accuracy: 0.7600
```

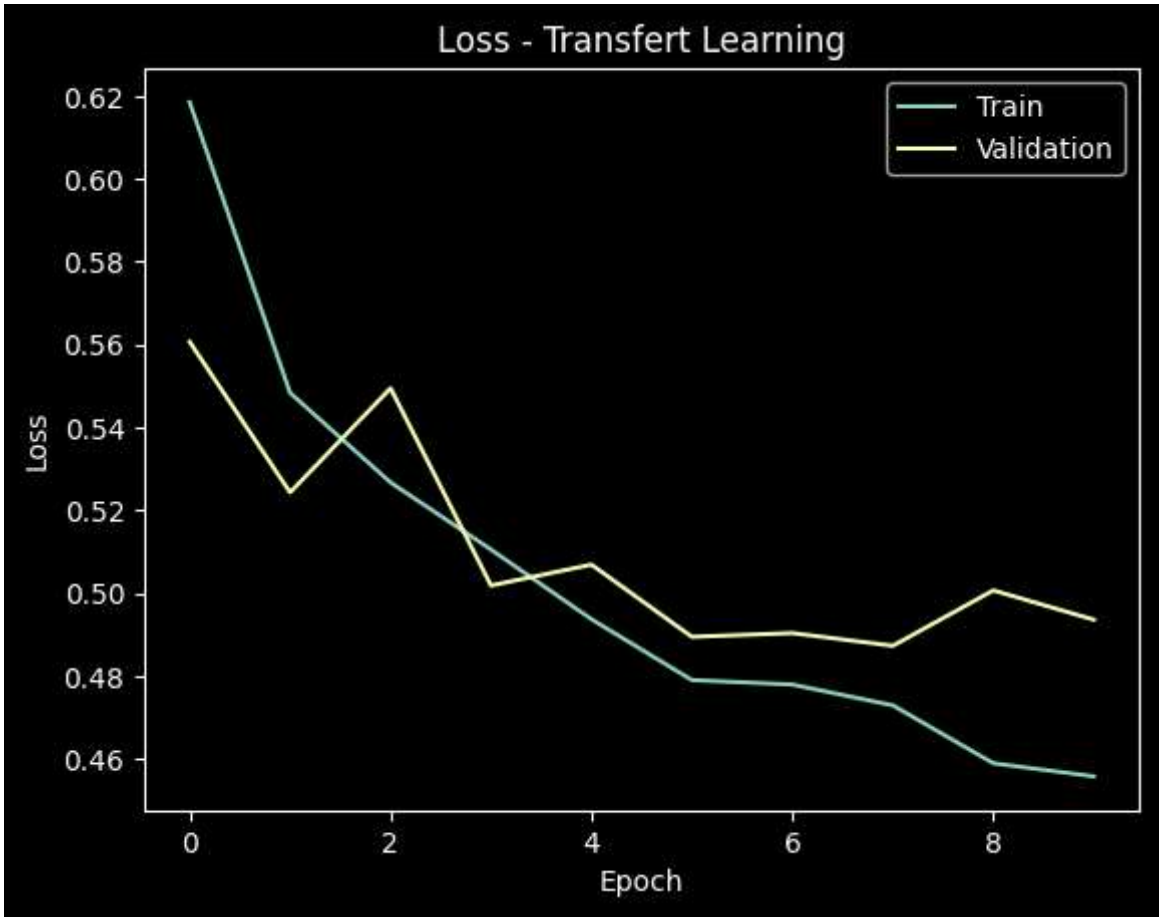
In [11]:
```
# Affichage de l'utilisation des ressources apres la fin de l'entrainement
cpu_after, ram_after = get_cpu_ram()
gpu_after, vram_used_after, vram_total_after = get_gpu_stats()

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: 4.5% | RAM: 14624 MB | GPU: 3.0% | VRAM: 627.0/16303.0
MB
```

In [12]:
```
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 du modèle en transfer learning montre une amélioration rapide des performances dès les premières époques, avec une précision maximale de validation d'environ 76,6 % atteinte à l'epoch 8. La perte de validation diminue globalement et reste relativement stable, indiquant une généralisation correcte sans surapprentissage marqué.

L'utilisation des ressources matérielles reste maîtrisée tout au long de l'entraînement. Le CPU est faiblement sollicité (environ 1 à 4 %), le GPU présente une utilisation modérée avec quelques pics ponctuels, et la mémoire RAM reste stable autour de 14,5 à 14,8 Go. La VRAM utilisée reste inférieure à 1 Go, malgré la complexité du modèle pré-entraîné. Le temps d'entraînement par epoch est d'environ 58 secondes.

Ainsi, le transfer learning permet d'obtenir des performances satisfaisantes avec un nombre limité d'epochs et une convergence rapide, tout en maintenant une utilisation des ressources raisonnable