Actividad: Transfer Learning

- Cynthia Cristal Quijas Flores - A01655996
- Alejandro Sanchez Flores - A01662783
- Carlos Adrian Palmieri Álvarez - A01635776
- Dabria Camila Carrillo Meneses - A01656716

In [6]:
```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import Adam
```

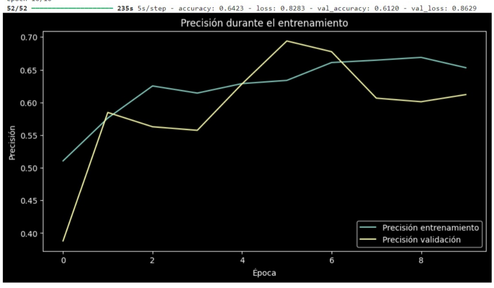
In [10]:
```
# DEFINIMOS LAS RUTAS DE NUESTRO DATASET PERSONALIZADO
train_dir = r'./Imagenes_Class/train/'
val_dir = r'./Imagenes_Class/val/'
```

In [11]:
```
# PROCESAMOS LAS IMAGENES Y GENERAMOS LOS LOTES
train_datagen = ImageDataGenerator(rescale=1./255, horizontal_flip=True, rotation_range=20, zoom_range=0.2)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir, target_size=(224, 224), batch_size=32, class_mode='categorical')
val_generator = val_datagen.flow_from_directory(val_dir, target_size=(224, 224), batch_size=32, class_mode='categorical')

# VERIFICAMOS LAS CLASES ENCONTRADAS
print("Clases encontradas en entrenamiento:", train_generator.class_indices)
print("Número de imágenes de entrenamiento:", train_generator.samples)
print("Número de imágenes de validación:", val_generator.samples)
```

```
Found 1649 images belonging to 3 classes.
Found 183 images belonging to 3 classes.
Clases encontradas en entrenamiento: {'keyboard': 0, 'monitor': 1, 'mouse': 2}
Número de imágenes de entrenamiento: 1649
Número de imágenes de validación: 183
```

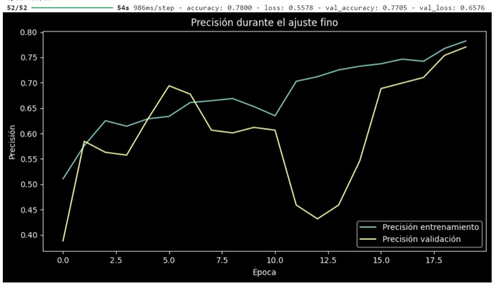
In [12]:
```
# IMPORTAMOS EL MODELO ResNet50 PREENTRENADO EN ImageNet Y EXCLUIMOS LAS CAPAS SUPERIORES
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# CONGELAMOS LAS CAPAS DEL MODELO BASE PARA CONSERVAR LOS PESOS PREENTRENADOS
base_model.trainable = False

# AGREGAMOS LAS NUEVAS CAPAS DE CLASIFICACION
model = models.Sequential()
model.add(base_model)
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dropout(0.5))
# DEFINIMOS LAS 3 CLASES (Mouse, Teclado, Monitor)
model.add(layers.Dense(3, activation='softmax'))

# COMPILAMOS EL MODELO
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

# ENTRENAMOS EL MODELO
history = model.fit(train_generator, epochs=10, validation_data=val_generator)

# GRAFICAMOS LOS RESULTADOS DE ENTRENAMIENTO
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Precisión entrenamiento')
plt.plot(history.history['val_accuracy'], label='Precisión validación')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.legend(loc='lower right')
plt.title('Precisión durante el entrenamiento')
plt.show()
```

```
Epoch 1/10
c:\Users\aless\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
52/52 ━━━━━━━━━━━━━━━━━━━━ 103s 2s/step - accuracy: 0.4608 - loss: 1.2578 - val_accuracy: 0.3880 - val_loss: 1.1752
Epoch 2/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 64s 1s/step - accuracy: 0.5665 - loss: 0.9579 - val_accuracy: 0.5847 - val_loss: 0.9063
Epoch 3/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 179s 3s/step - accuracy: 0.6438 - loss: 0.8483 - val_accuracy: 0.5628 - val_loss: 0.9091
Epoch 4/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 50s 920ms/step - accuracy: 0.6222 - loss: 0.8551 - val_accuracy: 0.5574 - val_loss: 0.9419
Epoch 5/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 58s 1s/step - accuracy: 0.6295 - loss: 0.8636 - val_accuracy: 0.6284 - val_loss: 0.8546
Epoch 6/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 230s 4s/step - accuracy: 0.6453 - loss: 0.8188 - val_accuracy: 0.6940 - val_loss: 0.8145
Epoch 7/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 47s 869ms/step - accuracy: 0.6558 - loss: 0.7991 - val_accuracy: 0.6776 - val_loss: 0.8134
Epoch 8/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 49s 912ms/step - accuracy: 0.6829 - loss: 0.7646 - val_accuracy: 0.6066 - val_loss: 0.8487
Epoch 9/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 50s 916ms/step - accuracy: 0.6811 - loss: 0.7468 - val_accuracy: 0.6011 - val_loss: 0.8645
Epoch 10/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 235s 5s/step - accuracy: 0.6423 - loss: 0.8283 - val_accuracy: 0.6120 - val_loss: 0.8629
```



In [13]:
```
# DESCONGELAMOS ALGUNAS CAPAS DEL MODELO BASE
base_model.trainable = True

# DESCONGELAMOS LAS ULTIMAS 10 CAPAS
for layer in base_model.layers[:-10]:
    layer.trainable = False

# COMPILAMOS NUEVAMENTE EL MODELO CON UN learning_rate BAJO
model.compile(optimizer=Adam(learning_rate=1e-5), loss='categorical_crossentropy', metrics=['accuracy'])

# ENTRENAMOS NUEVAMENTE EL MODELO PARA AJUSTE FINO
fine_tune_history = model.fit(train_generator, epochs=10, validation_data=val_generator)

# GRAFICAMOS LOS RESULTADOS DEL ENTRENAMIENTO (entrenamiento + ajuste fino)
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'] + fine_tune_history.history['accuracy'], label='Precisión entrenamiento')
plt.plot(history.history['val_accuracy'] + fine_tune_history.history['val_accuracy'], label='Precisión validación')
plt.xlabel('Época')
plt.ylabel('Precisión')
plt.legend(loc='lower right')
plt.title('Precisión durante el ajuste fino')
plt.show()
```

```
Epoch 1/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 79s 1s/step - accuracy: 0.5573 - loss: 1.2134 - val_accuracy: 0.6066 - val_loss: 0.8443
Epoch 2/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 0.6983 - loss: 0.7275 - val_accuracy: 0.4590 - val_loss: 1.1455
Epoch 3/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 847s 17s/step - accuracy: 0.7303 - loss: 0.6615 - val_accuracy: 0.4317 - val_loss: 1.3065
Epoch 4/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 54s 1s/step - accuracy: 0.7281 - loss: 0.6692 - val_accuracy: 0.4590 - val_loss: 1.1417
Epoch 5/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 53s 980ms/step - accuracy: 0.7180 - loss: 0.6478 - val_accuracy: 0.5464 - val_loss: 0.9785
Epoch 6/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 66s 1s/step - accuracy: 0.7423 - loss: 0.6126 - val_accuracy: 0.6885 - val_loss: 0.7259
Epoch 7/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 83s 2s/step - accuracy: 0.7489 - loss: 0.6064 - val_accuracy: 0.6995 - val_loss: 0.7429
Epoch 8/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 156s 3s/step - accuracy: 0.7537 - loss: 0.5812 - val_accuracy: 0.7104 - val_loss: 0.6918
Epoch 9/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 53s 986ms/step - accuracy: 0.7609 - loss: 0.5869 - val_accuracy: 0.7541 - val_loss: 0.6530
Epoch 10/10
52/52 ━━━━━━━━━━━━━━━━━━━━ 54s 986ms/step - accuracy: 0.7800 - loss: 0.5578 - val_accuracy: 0.7705 - val_loss: 0.6576
```



In [15]:
```
# OBTENEMOS UN LOTE DE IMAGENES ALEATORIAS DE VALIDACION
val_images, val_labels = next(val_generator)
predictions = model.predict(val_images)

# VISUALIZAMOS LAS IMAGENES CON SUS PREDICCIONES
class_names = ['keyboard', 'monitor', 'mouse']

plt.figure(figsize=(10,10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(val_images[i])
    plt.title(f"Verdadero: {class_names[np.argmax(val_labels[i])]} \nPredicción: {class_names[np.argmax(predictions[i])]}")
    plt.axis('off')
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 958ms/step
```

In [ ]:
```
```