In [1]:
```
print("Hola Mundo, Vamos a crear una nuevo proyecto para la clasificación de imáger
```

```
Hola Mundo, Vamos a crear una nuevo proyecto para la clasificación de imágenes...
```

Librerias para procesamiento de datos, visualización, imágenes

In [2]:
```
import os
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
from tqdm import tqdm
from pathlib import Path
import cv2
from math import ceil
import itertools
%matplotlib inline
```

Librerias para procesamiento de imágenes, modelos, capas, red neural

In [4]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_
from tensorflow.keras.models import Sequential, load_model,Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import ResNet50, EfficientNetB0
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheck
from sklearn.metrics import classification_report, confusion_matrix
```

In [68]:
```
data_dir =  'G:/Intel/'
data_train = 'G:/Intel/seg_train'
data_test = 'G:/Intel/seg_test'
data_path = 'G:/Intel/seg_train'

nuevo_data_path = 'G:/Intel/data_train'
nuevo_data_test = 'G:/Intel/data_test'
nuevo_data_pred = 'G:/Intel/data_pred'

train_dir = os.path.join(data_dir, 'seg_train')
test_dir = os.path.join(data_dir, 'seg_test')
pred_dir = os.path.join(data_dir, 'seg_pred')
```

Clases que contiene el dataset para train y test

In [8]:
```
clases_train = os.listdir(data_train)
print('Clases del Dataset: ', clases_train)
```

```
Clases del Dataset:  ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'stree
t']
```

In [44]:
```
clases_test = os.listdir(data_test)
print('Clases del Dataset: ', clases_test)
```

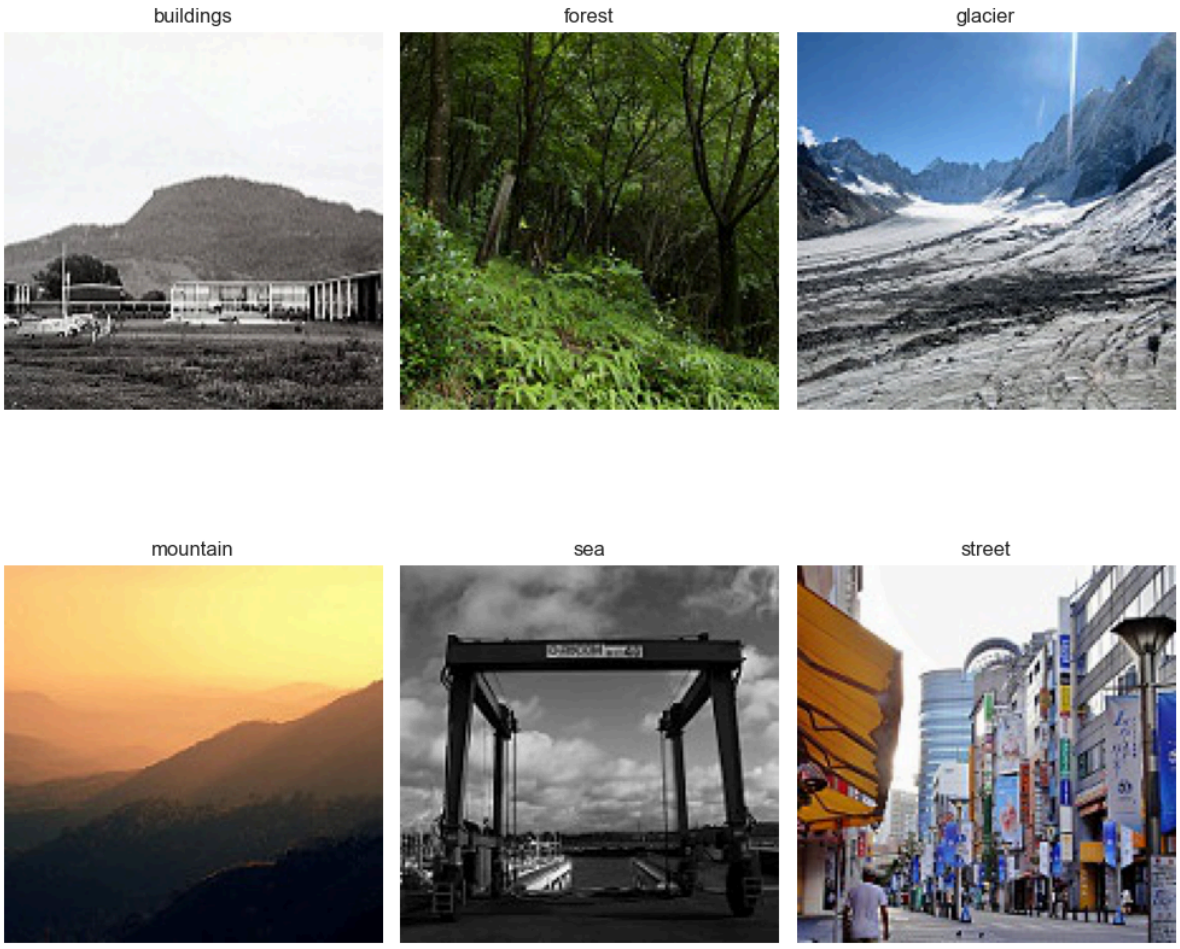
```
Clases del Dataset:  ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'stree
t']
```

In [22]:
```
clases = os.listdir(data_path)
print('Clases del Dataset: ', clases)
```

```
Clases del Dataset:  ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'stree
t']
```

Vamos a hacer una primera visualización de una imágen por cada clase.

In [21]:
```
plt.figure(figsize=(10, 10))

for i, clase in enumerate(clases_train):
    img_folder = os.path.join(data_train, clase)
    img_file = os.listdir(img_folder)[0]
    img_path = os.path.join(img_folder, img_file)

    imagen = Image.open(img_path)

    plt.subplot(2, 3, i + 1)
    plt.imshow(imagen)
    plt.title(clase)
    plt.axis('off')

plt.tight_layout()
plt.show()
```



Cuantas imágenes tenemos por cada clase

In [24]:
```
imagen_por_clase = {}

for clase in clases:
    ruta = os.path.join(data_path, clase)
    conteo = len(os.listdir(ruta))
    imagen_por_clase[clase] = conteo

print('Numero de imágenes por cada clase:')
for clase, cantidad in  imagen_por_clase.items():
    print(f'{clase}: {cantidad}')
```

```
Numero de imágenes por cada clase:
buildings: 2191
forest: 2271
glacier: 2404
mountain: 2512
sea: 2274
street: 2382
```

Distribución de Dimensiones

In [25]:
```
dimensiones = []

for clase in clases:
    ruta = os.path.join(data_path, clase)
    for img_file in tqdm(os.listdir(ruta), desc=f'Procesando {clase}'):
        img_path = os.path.join(ruta, img_file)
```

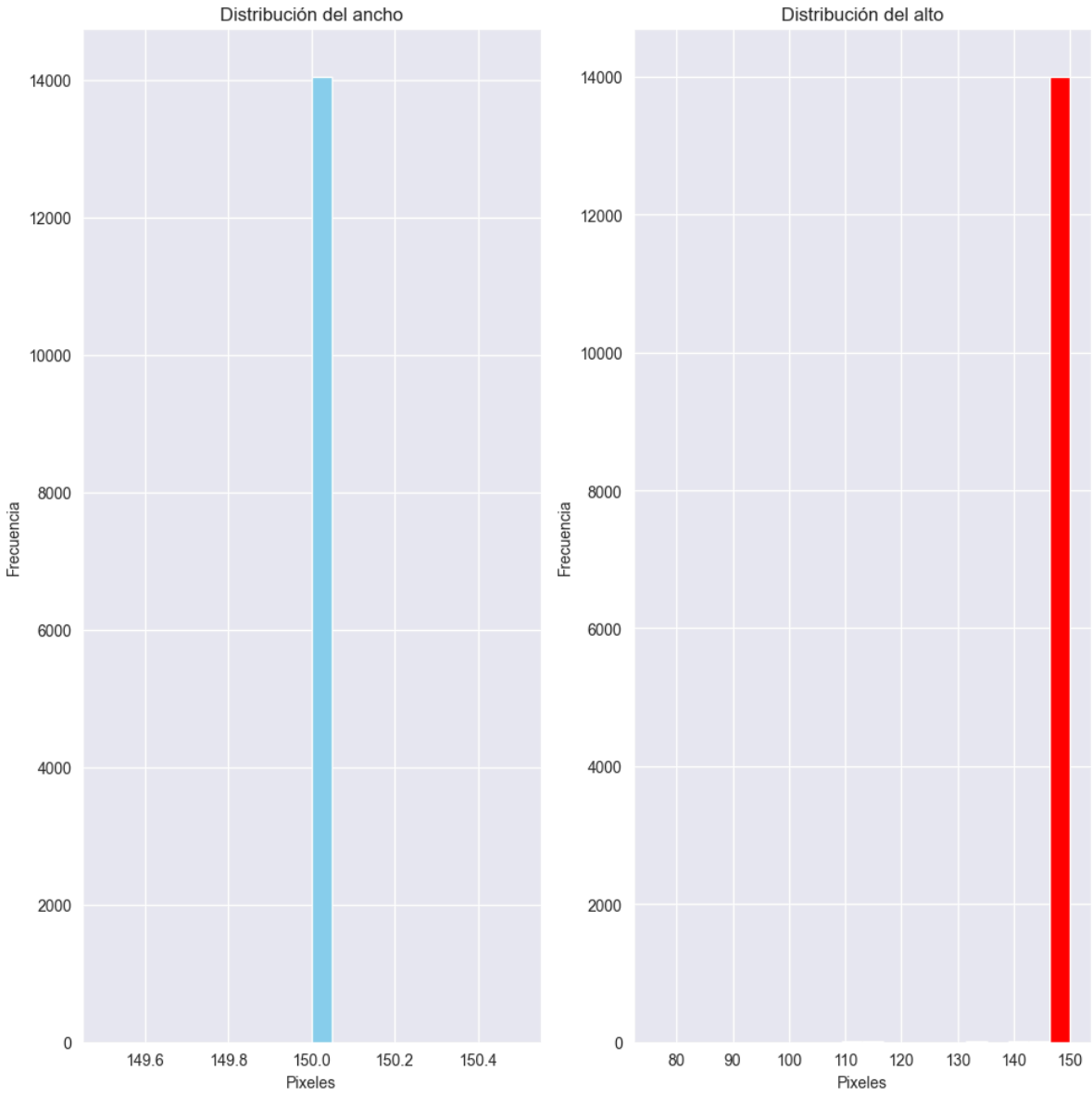
```
        try:
            with Image.open(img_path) as img:
                dimensiones.append(img.size)
        except:
            continue
```

```
Procesando buildings: 100%|██████████| 2191/2191 [00:49<00:00, 43.84it/s]
Procesando forest: 100%|██████████| 2271/2271 [00:47<00:00, 48.17it/s]
Procesando glacier: 100%|██████████| 2404/2404 [00:51<00:00, 46.90it/s]
Procesando mountain: 100%|██████████| 2512/2512 [00:47<00:00, 52.98it/s]
Procesando sea: 100%|██████████| 2274/2274 [00:43<00:00, 52.55it/s]
Procesando street: 100%|██████████| 2382/2382 [00:33<00:00, 71.25it/s]
```

In [26]:
```
dimensiones = np.array(dimensiones)
anchos = dimensiones[:, 0]
altos = dimensiones[:, 1]
```

Vamos a graficar en Histogramas loa valores de alto y ancho

In [29]:
```
plt.figure(figsize=(10,10))
plt.subplot(1, 2, 1)
plt.hist(anchos, bins=20, color='skyblue')
plt.title('Distribución del ancho')
plt.xlabel('Pixeles')
plt.ylabel('Frecuencia')

plt.subplot(1, 2, 2)
plt.hist(altos, bins=20, color='red')
plt.title('Distribución del alto')
plt.xlabel('Pixeles')
plt.ylabel('Frecuencia')

plt.tight_layout()
plt.show()
```



Vamos a crear una funcion que nos calcule el promedio RGB por cada clase

In [30]:
```
def rgb_clase(imagen):
    imagen = imagen.resize((64,64))
    imagen_np = np.array(imagen)
    if imagen_np.ndim == 3:
        return [np.mean(imagen_np[:,:,i]) for i in range(3)]
        return [0,0,0]
```

Guardamos los resultados de la función

In [31]:
```
promedios_rgb = {}
```

In [32]:
```
for clase in clases:
    carpeta = os.path.join(data_path, clase)
    rgb_totales = np.zeros(3)
    contador = 0

    for img_file in os.listdir(carpeta)[:100]:
```

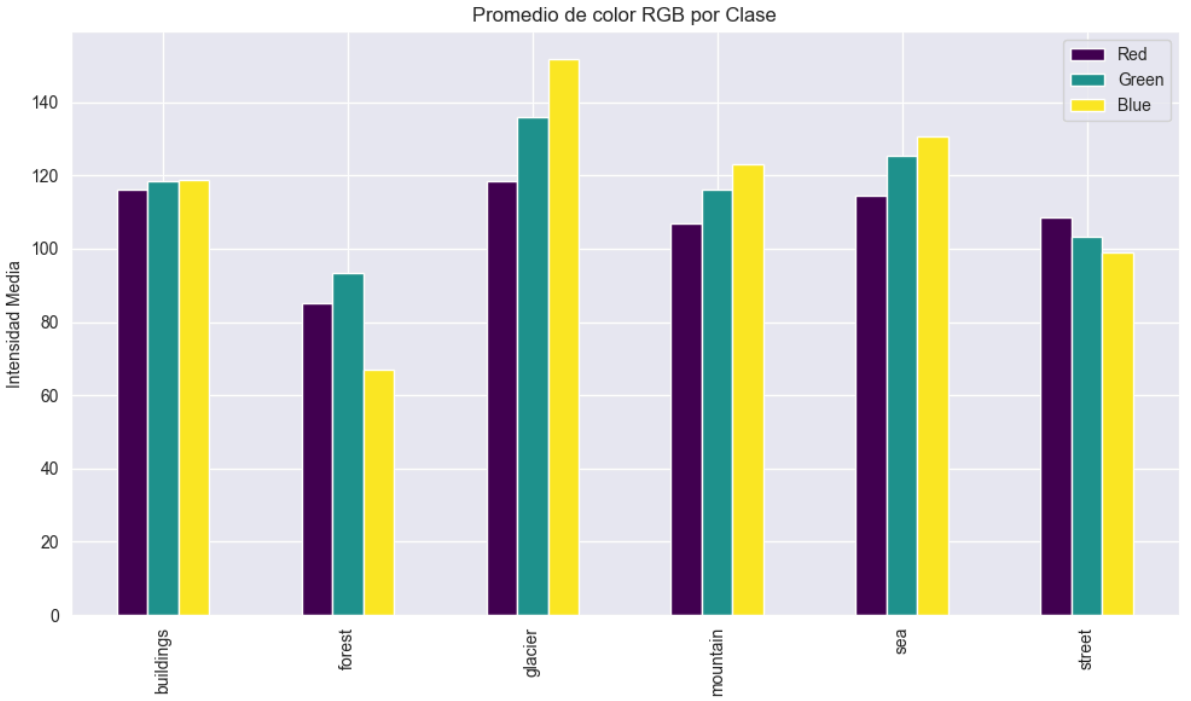
```
        ruta = os.path.join(carpeta, img_file)
        try:
            img = Image.open(ruta)
            rgb = rgb_clase(img)
            rgb_totales += rgb
            contador += 1
        except:
            continue

    promedios_rgb[clase] = rgb_totales/ max(contador, 1)
```

Creamos un nuevo DataFrame

In [33]:
```
df_rgb = pd.DataFrame(promedios_rgb, index=['Red', 'Green', 'Blue'])
```

Gráficamos los datos de nuestro nuevo DataFrame

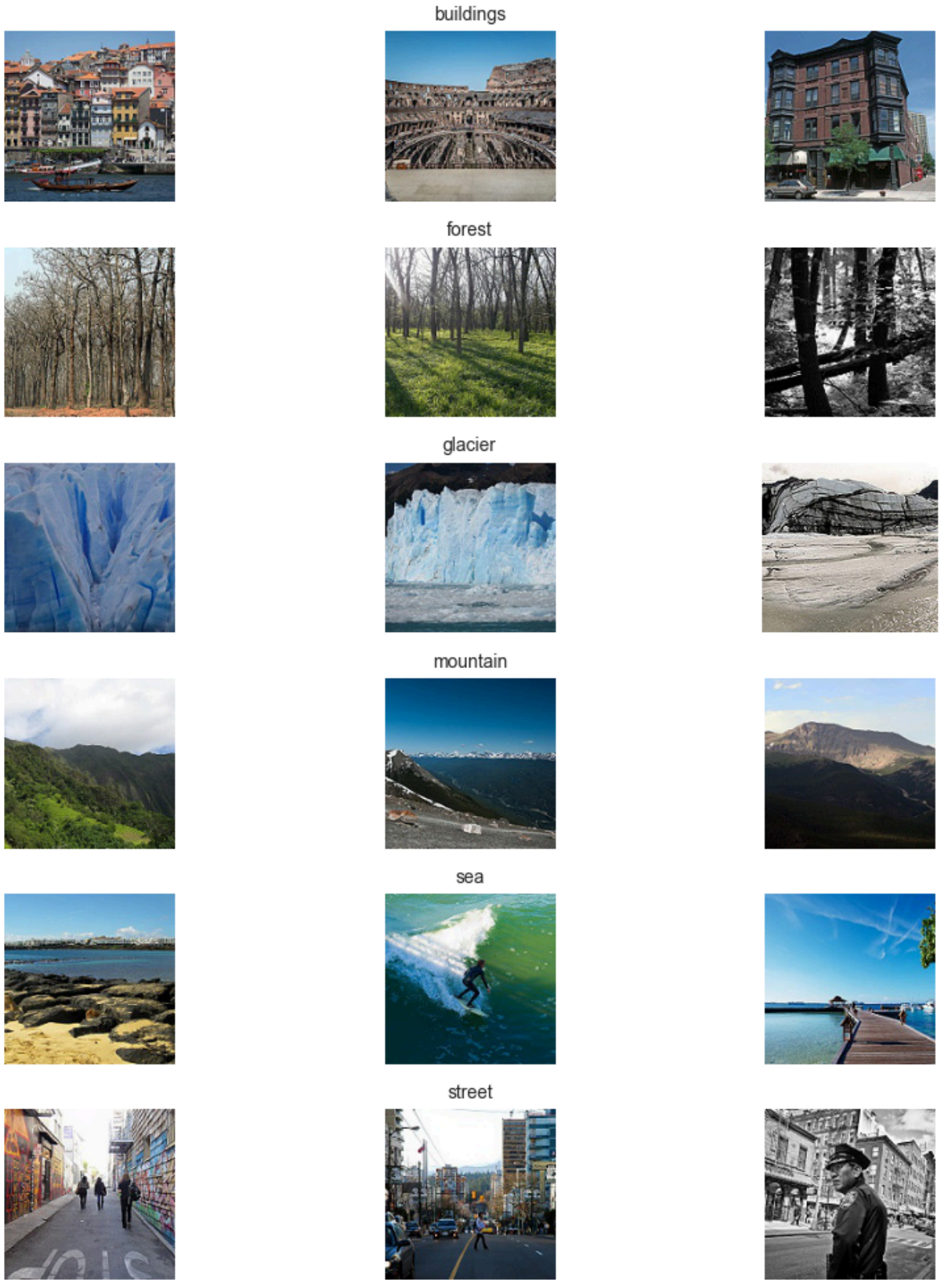
In [34]:
```
df_rgb.T.plot(kind='bar', figsize=(10,6), colormap='viridis')
plt.title('Promedio de color RGB por Clase')
plt.ylabel('Intensidad Media')
plt.xticks(rotation=90)
plt.grid=True
plt.tight_layout()
plt.show()
```



Vamos a crear una mosaico de 3 imágenes de forma aleatoria por cada clase

In [37]:
```
plt.figure(figsize=(10,10))
for i, clase in enumerate(clases):
    folder = os.path.join(data_path, clase)
```

```
    imagenes = os.listdir(folder)
    seleccionadas = random.sample(imagenes, 3)

    for j, img_name in enumerate(seleccionadas):
        img_path = os.path.join(folder, img_name)
        img = Image.open(img_path)
        plt.subplot(len(clases),3, i*3 + j + 1)
        plt.imshow(img)
        if j == 1:
            plt.title(clase, fontsize=10)
        plt.axis('off')

plt.tight_layout()
plt.show()
```



Vamos a verificar que no hay imagenes corruptas en el Dataset

In [38]:
```
errores = []

for clase in clases:
    ruta = os.path.join(data_path, clase)
    for archivo in os.listdir(ruta):
        path = os.path.join(ruta, archivo)
        try:
```

```
            img = Image.open(path)
            img.verify()
        except Exception as e:
            errores.append((path, str(e)))

print(f'Imagenes Corruptas Encontradas: {len(errores)}')
for i, (path, error) in enumerate(errores[:5]):
    print(f'{i+1}. {path}: {error}')
```

```
Imagenes Corruptas Encontradas: 0
```

A partir de este punto vamos a hacer uso de seg_test:

- Contiene menos imágenes que seg_train, por lo que podemos cargar mas rapido las imágenes, nos permite aplicar filtros, bordes, histogramas de una forma mas rápida.

In [73]:
```
ruta_imagenes = 'G:/Intel/seg_test'
imagenes = os.listdir(ruta_imagenes)
```

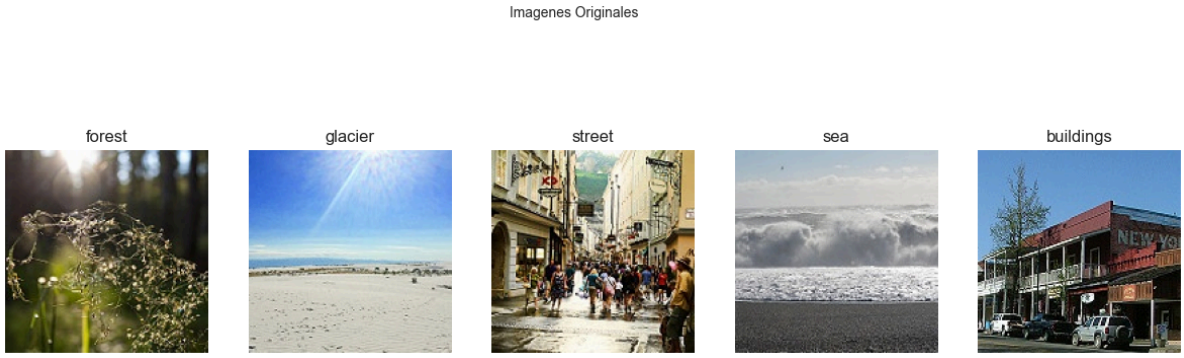
In [57]:
```
muestras = random.sample(imagenes, 5)
print('Imagenes seleccionadas')
print(muestras)
```

```
Imagenes seleccionadas
['forest', 'glacier', 'street', 'sea', 'buildings']
```

Visualizamos las imágenes.

In [58]:
```
def mostrar_imagenes(muestras, titulo='Imagenes Originales'):
    plt.figure(figsize=(15,5))

    for i, clase_nombre in enumerate(muestras):
        clase_folder = os.path.join(ruta_imagenes, clase_nombre)

        if os.path.exists(clase_folder):
            imagenes_en_clase = os.listdir(clase_folder)

            if imagenes_en_clase:
                imagen_nombre = random.choice(imagenes_en_clase)
                img_path = os.path.join(clase_folder, imagen_nombre)
                img = cv2.imread(img_path)

                if img is not None:
                    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                    plt.subplot(1, len(muestras), i+1)
                    plt.imshow(img)
                    plt.axis('off')
                    plt.title(f'{clase_nombre}')
                else:
                    print(f'Error al cargar imagen: {img_path}')
            else:
                print(f'No existen imagenes en la carpeta: {clase_folder}')
        else:
            print(f'No existe la carpeta: {clase_folder}')

    plt.suptitle(titulo, fontsize=10)
    plt.show()
```

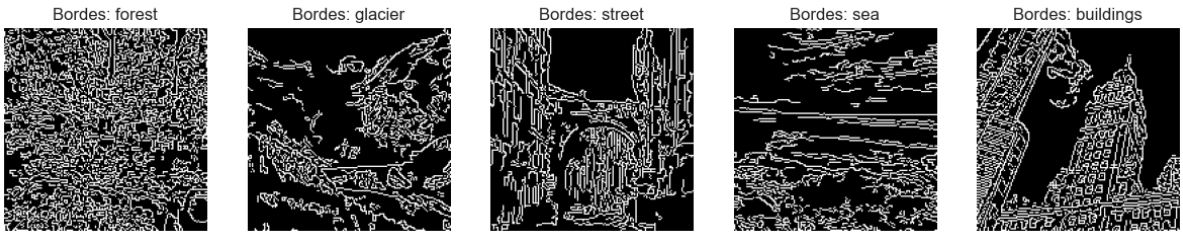
```
mostrar_imagenes(muestras)
```



Vamos a resaltar contornos y formas dominantes

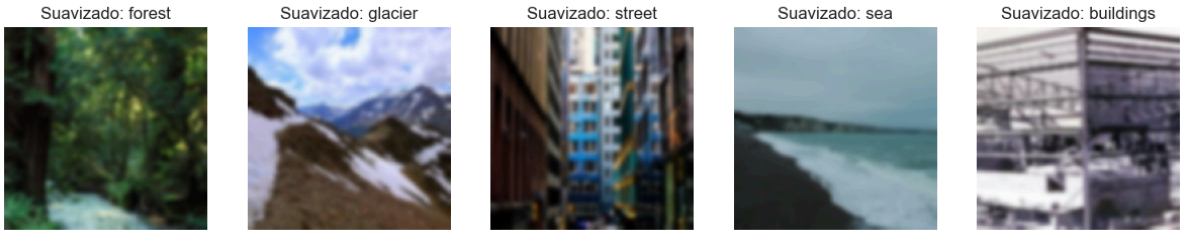
In [65]:
```
def mostrar_bordes(muestras):
    plt.figure(figsize=(15,5))

    for i, clase_nombre in enumerate(muestras):
        clase_folder = os.path.join(ruta_imagenes, clase_nombre)

        if os.path.exists(clase_folder):
            imagenes_clase = os.listdir(clase_folder)

            if imagenes_clase:
                imagen_nombre = random.choice(imagenes_clase)
                img_path = os.path.join(clase_folder, imagen_nombre)
                img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

                if img is not None:
                    bordes = cv2.Canny(img, 100, 200)
                    plt.subplot(1, len(muestras), i+1)
                    plt.imshow(bordes, cmap='gray')
                    plt.axis('off')
                    plt.title(f'Bordes: {clase_nombre}')
                else:
                    print(f'No existen imagenes en la carpeta: {img_path}')
            else:
                print(f'No existen imagenes en la carpeta: {clase_folder}')
        else:
            print(f'No existe la carpeta: {clase_folder}')

    plt.suptitle('Deteccion de bordes (Canny)', fontsize=10)
    plt.show()

mostrar_bordes(muestras)
```

Deteccion de bordes (Canny)



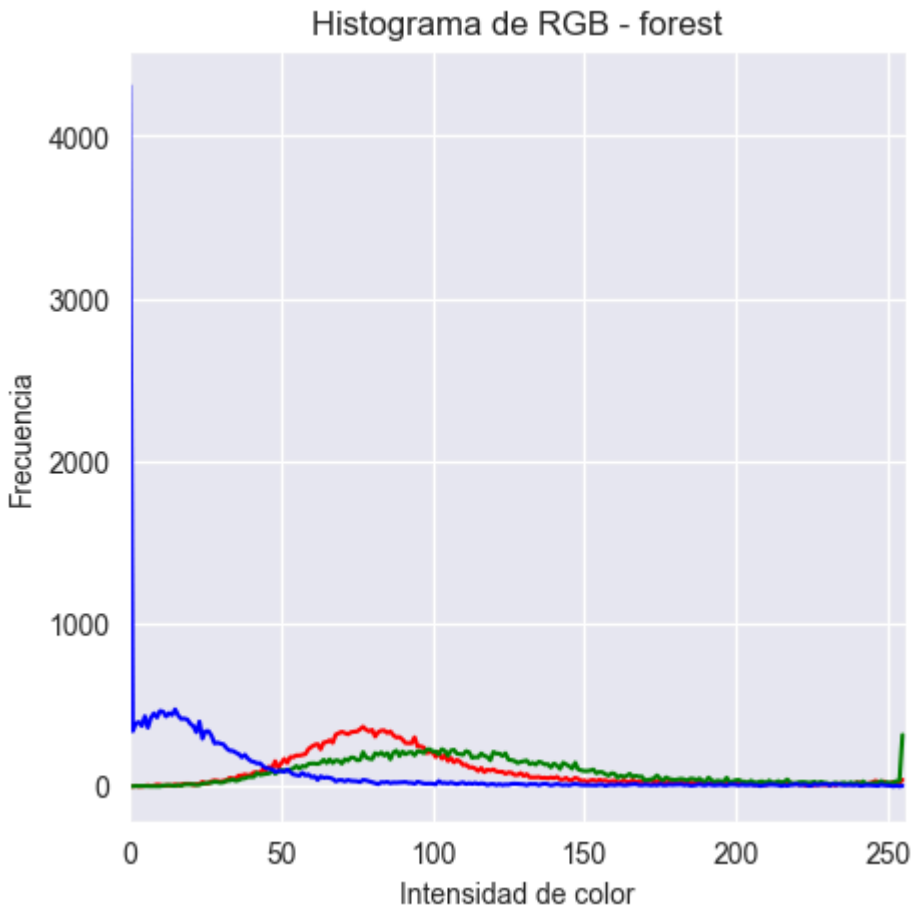
In [64]:
```python
def filtro_suavizado(muestras):
    plt.figure(figsize=(15,5))

    for i, clase_nombre in enumerate(muestras):
        clase_folder = os.path.join(ruta_imagenes, clase_nombre)

        if os.path.exists(clase_folder):
            imagenes_clase = os.listdir(clase_folder)

            if imagenes_clase:
                imagen_nombre = random.choice(imagenes_clase)
                img_path = os.path.join(clase_folder, imagen_nombre)
                img = cv2.imread(img_path)

                if img is not None:
                    rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                    suavizado = cv2.GaussianBlur(rgb, (9, 9), 0)
                    plt.subplot(1, len(muestras), i+1)
                    plt.imshow(suavizado)
                    plt.axis('off')
                    plt.title(f'Suavizado: {clase_nombre}')
                else:
                    print(f'No existen imagenes en la carpeta: {img_path}')
            else:
                print(f'No existen imagenes en la carpeta: {clase_folder}')
        else:
            print(f'No existe la carpeta: {clase_folder}')

    plt.suptitle('filtro de suavizado (Gaussian Blur)', fontsize=10)
    plt.show()

filtro_suavizado(muestras)
```

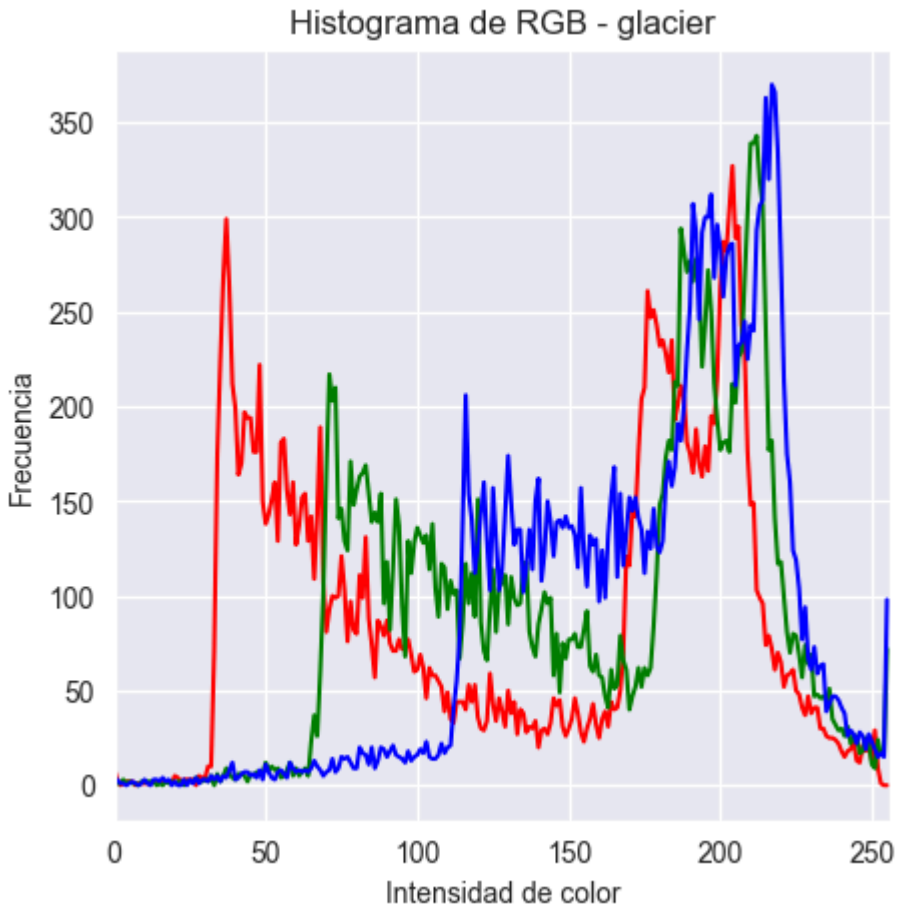
filtro de suavizado (Gaussian Blur)

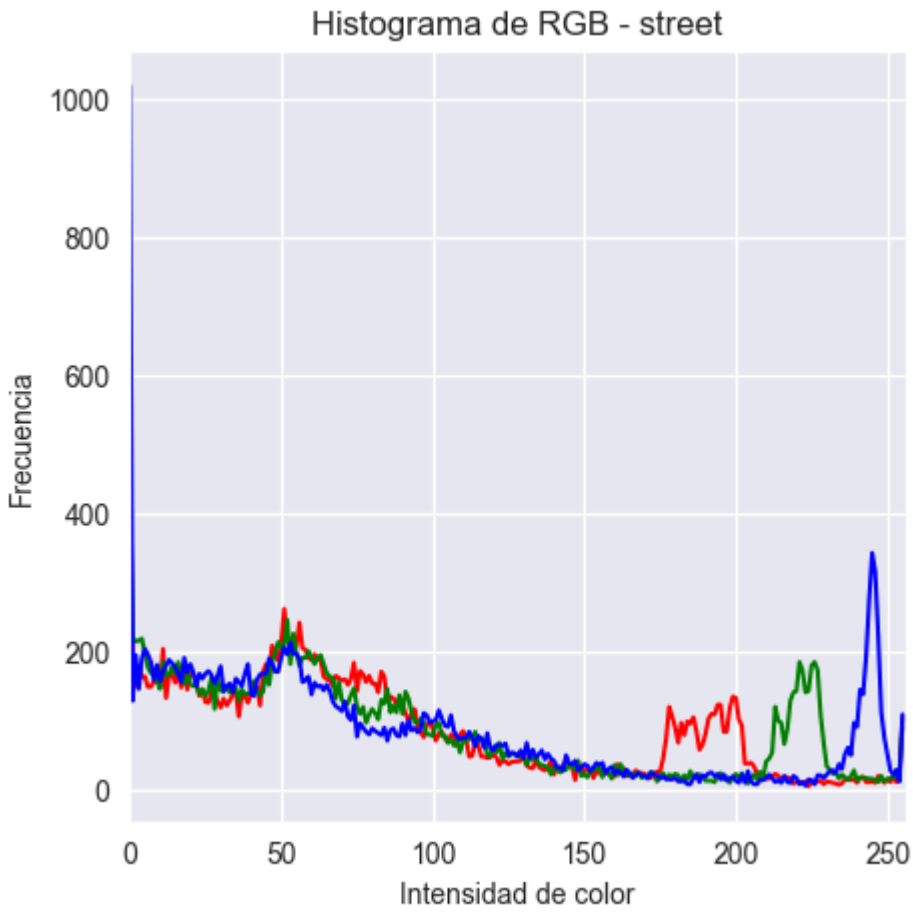


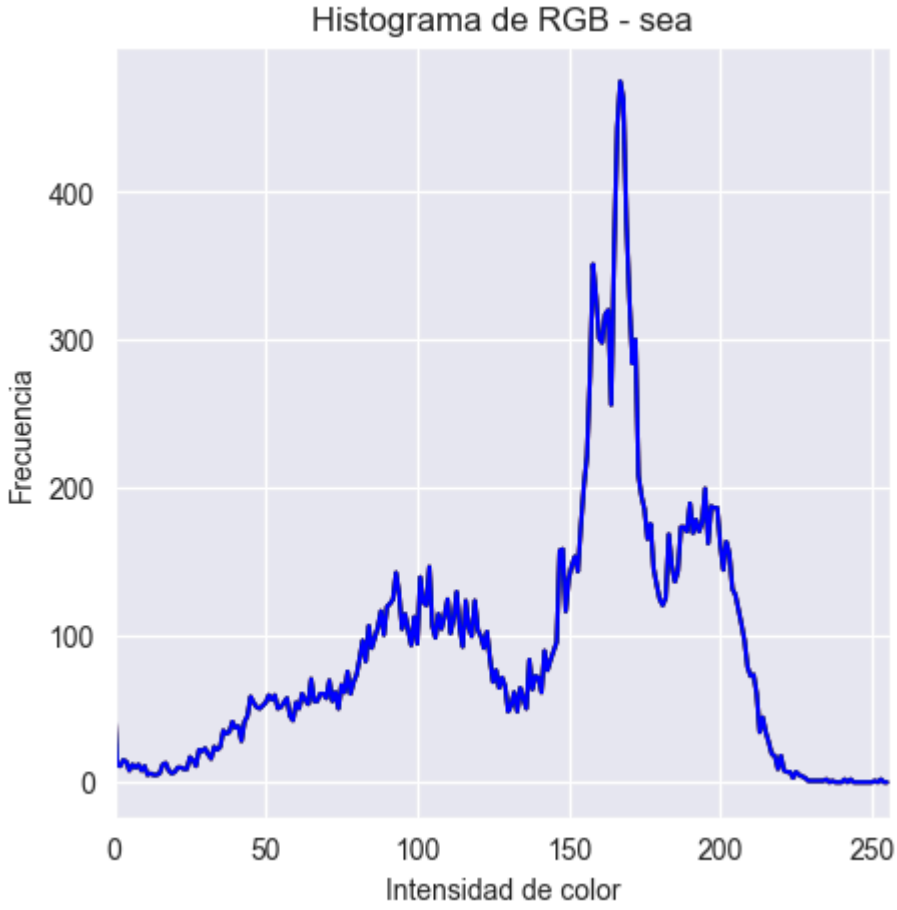
Histograms de RGB para mostrar la distribución de colore. útil para entender clases diminantes para ciertos tonos

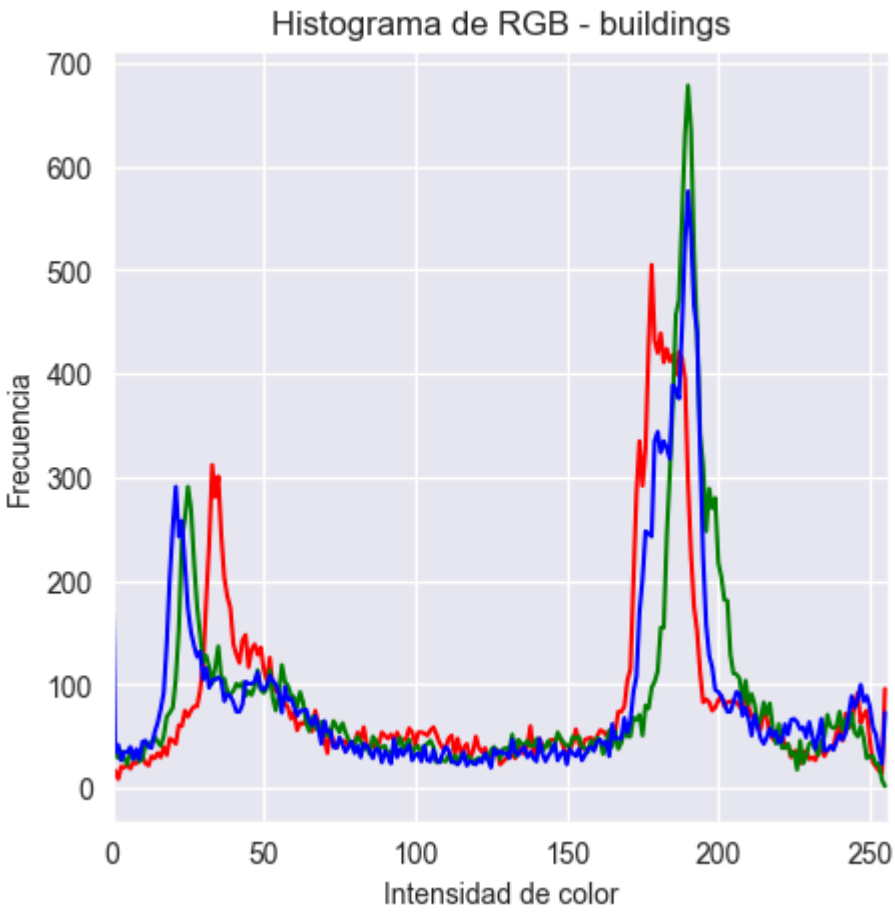
In [70]:
```
def histograma_rgb(muestras):
    for nombre in muestras:
        clase_folder = os.path.join(ruta_imagenes, nombre)

        if os.path.exists(clase_folder):
            img_clases = os.listdir(clase_folder)

            if img_clases:
                img_nombre = random.choice(img_clases)
                img_path = os.path.join(clase_folder, img_nombre)
                img = cv2.imread(img_path)

                if img is not None:
                    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                    plt.figure(figsize=(5,5))
                    colores = ('r', 'g', 'b')

                    for i, color in enumerate(colores):
                        hist = cv2.calcHist([img_rgb], [i], None, [256], [0, 256])
                        plt.plot(hist, color=color)
                    plt.title(f'Histograma de RGB - {nombre}')
                    plt.xlabel('Intensidad de color')
                    plt.ylabel('Frecuencia')
                    plt.xlim([0, 256])
                    #plt.grid(True)
                    plt.show()
                else:
                    print(f'Error al cargar imagen {img_path}')
            else:
                print(f'No existen imagenes en la carpeta {clase_folder}')
        else:
            print(f'No existe la carpeta {clase_folder}')


histograma_rgb(muestras)
```


Histograma de RGB - forest
Frecuencia
4000
3000
2000
1000
0
0
50
100
150
200
250
Intensidad de color


Histograma de RGB - glacier
Frecuencia
350
300
250
200
150
100
50
0
0
50
100
150
200
250
Intensidad de color


Histograma de RGB - street
Frecuencia
Intensidad de color
0
200
400
600
800
1000
0
50
100
150
200
250


Histograma de RGB - sea
Frecuencia
Intensidad de color
0
100
200
300
400
0
50
100
150
200
250

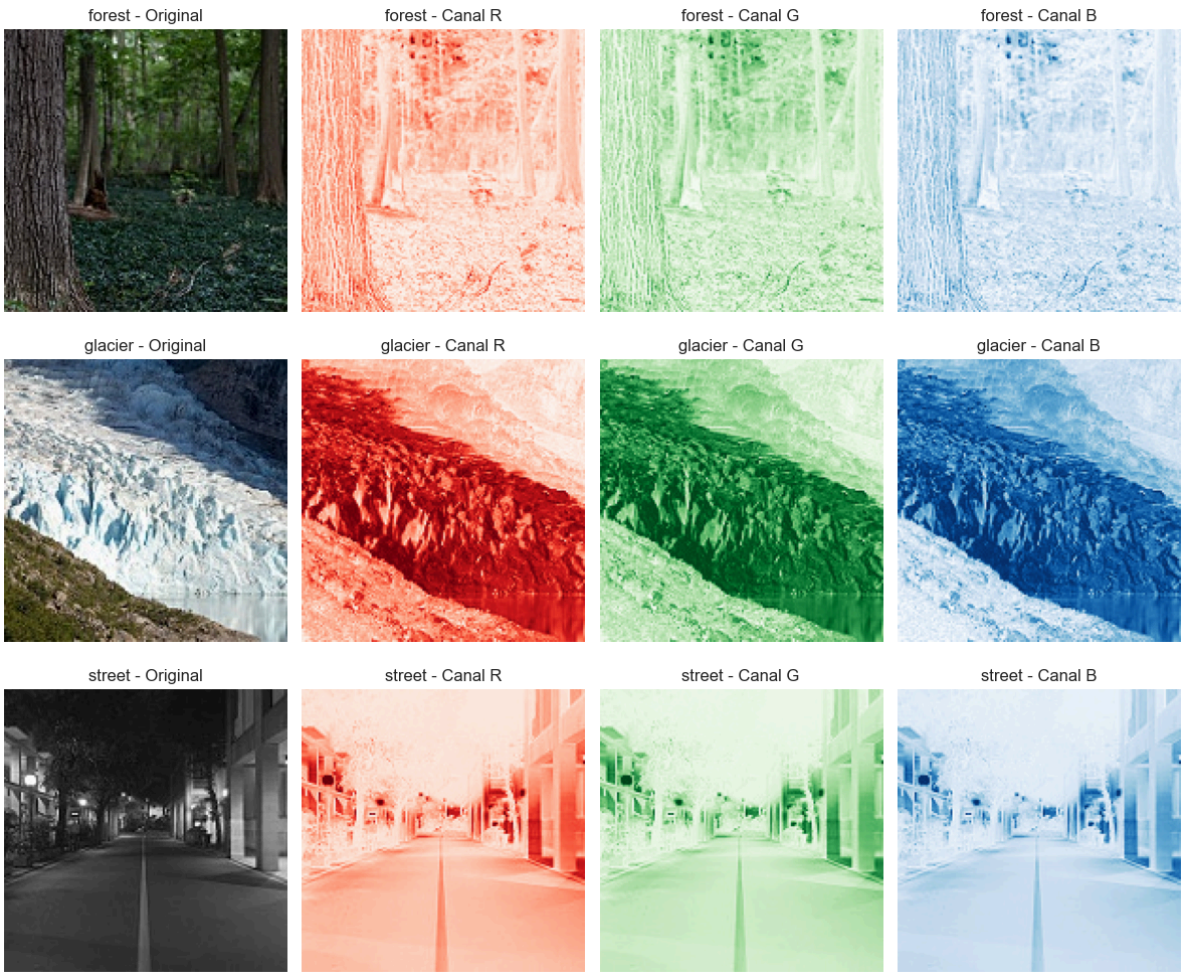


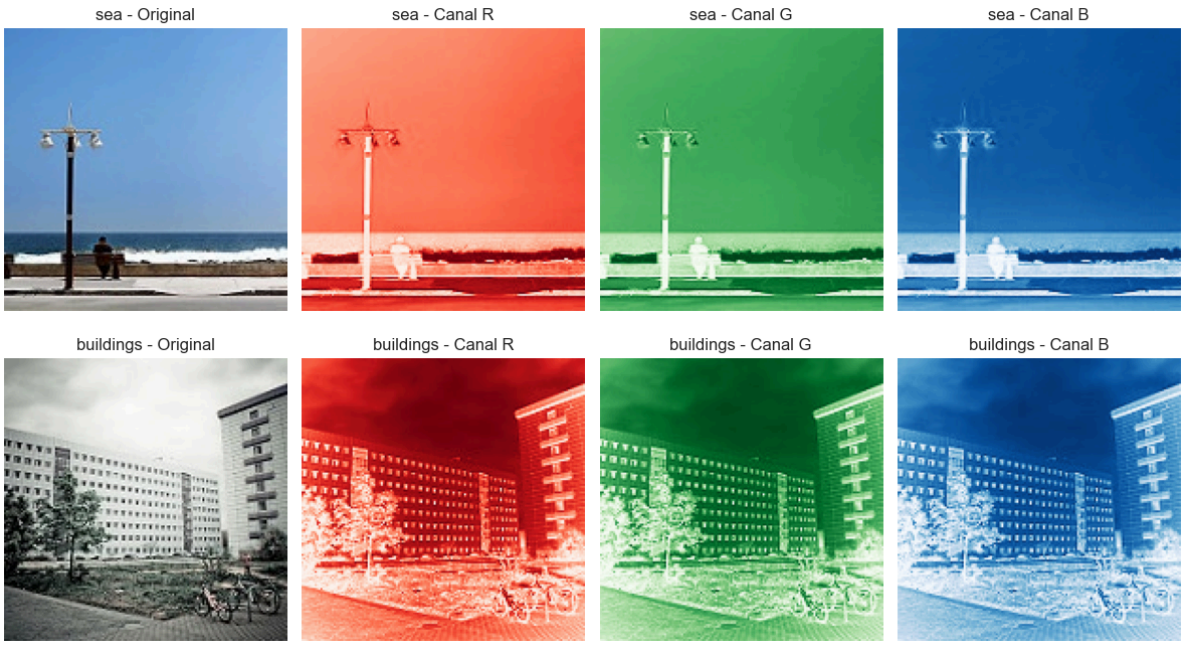
Vamos a mostrar una imágen por cada clase original y al mismo tiempo la distribución de canales RGB

In [76]:

```
def mostrar_canales_rgb(muestras):
    for nombre_clase in muestras:
        clase_folder = os.path.join(ruta_imagenes, nombre_clase)

        if os.path.exists(clase_folder):
            imagenes_en_clase = os.listdir(clase_folder)

            if imagenes_en_clase:
                imagen_nombre = random.choice(imagenes_en_clase)
                ruta_imagen = os.path.join(clase_folder, imagen_nombre)
                imagen = cv2.imread(ruta_imagen)

                if imagen is not None:
                    imagen_rgb = cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)


                    R, G, B = cv2.split(imagen_rgb)

                    plt.figure(figsize=(12, 6))

                    plt.subplot(1, 4, 1)
                    plt.imshow(imagen_rgb)
                    plt.title(f"{nombre_clase} - Original")
                    plt.axis('off')

                    plt.subplot(1, 4, 2)
```

```
                    plt.imshow(R, cmap='Reds')
                    plt.title(f"{nombre_clase} - Canal R")
                    plt.axis('off')

                    plt.subplot(1, 4, 3)
                    plt.imshow(G, cmap='Greens')
                    plt.title(f"{nombre_clase} - Canal G")
                    plt.axis('off')

                    plt.subplot(1, 4, 4)
                    plt.imshow(B, cmap='Blues')
                    plt.title(f"{nombre_clase} - Canal B")
                    plt.axis('off')

                    plt.tight_layout()
                    plt.show()
                else:
                    print(f"Error al cargar imagen: {ruta_imagen}")
            else:
                print(f"No existen imagenes en esta carpeta: {clase_folder}")
        else:
            print(f"No existe la carpeta: {clase_folder}")



mostrar_canales_rgb(muestras)
```





Procesamiento de datos

- Vamos a pasar al procesamiento de los datos, para entrenar nuestra red neuronal, nuestras capas, pero en este caso vamos a reducir un poco el tamaño del dataset para que no dure tanto el procesamiento.
- Vamos a usar ImageDataGenerator que es una clase de Keras que automatiza la carga y transformación de imágenes desde carpetas. Esto nos es util en este dataset que ya viene organizada por carpetas.

Vamos a crear los generadores de imágenes

- Uno para entrenamiento
- otro para validación

Parámetros importantes

- rescale: Normaliza los píxeles de 0-255 a 0-1
- rotation: Rota aleatoriamente las imágenes hasta 20 grados
- width_shift_range: Desplaza horizontalmente las imágenes
- height_shift_range: Desplaza verticalmente las imágenes
- shear_range: Aplica transformaciones de corte
- zoom_range: aplica zoom aleatorio
- horizontal_flip: invierte horizontalmente la imágen
- fill_mode: Rellena los pixeles faltantes tras la transformación
- validation_spli: Separa el 20% de los datos para validación

In [6]:
```
nuevo_data_path = 'G:/Intel/data_train'
nuevo_data_test = 'G:/Intel/data_test'
```

```
nuevo_data_pred = 'G:/Intel/data_pred'
```

In [7]:
```
train_datagen1 = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 45,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
    fill_mode = 'nearest',
    validation_split = 0.2,
)
```

In [8]:
```
val_datagen1 = ImageDataGenerator(
    rescale = 1./255,
    validation_split = 0.2,
)
```

Creación de los conjuntos (train y val)

- flow_from_factory: Busca sub-carpetas en data_train y cada una representa una clase
- class_mode: 'categorical':Es por qur hay mas de dos clases
- shuffle_True: Mezcla aleatoriamente las imágenes en el entrenamiento para que el modelo no aprenda en un orden espcifico

In [9]:
```
image_size = (150, 150)
batch_size = 32

train_generator1 = train_datagen1.flow_from_directory(
    nuevo_data_path,
    target_size = image_size,
    batch_size = batch_size,
    class_mode = 'categorical',
    subset = 'training',
    shuffle = True,
    seed=42
)
```

```
Found 4800 images belonging to 6 classes.
```

In [10]:
```
val_generator1 = val_datagen1.flow_from_directory(
    nuevo_data_test,
    target_size = image_size,
    batch_size = batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=False,
    seed=42
)
```

```
Found 598 images belonging to 6 classes.
```

In [11]:
```
print('Clases detectadas:', train_generator1.class_indices)
```

```
Clases detectadas: {'buildings': 0, 'forest': 1, 'glacier': 2, 'mountain': 3, 'se
a': 4, 'street': 5}
```

Construcción y entrenamiento del modelo CNN

Parámetros

- Conv2D: Extrae características visuales (bordes, texturas).
- MaxPooling2D: reduce el tamaño y la complejidad de las características.
- Flatten: Convierte el mapa 2D de activaciones en un vector 1D.
- Dense: Decide la clasificación a partir de las características.
- Dropout: Ayuda a que el modelo no memorice los datos.
- softmax: Para clasificación multiclase.

In [17]:
```
modelo1 = Sequential()
```

Capas convolucionales

Por que usamos 3 capas en este entrenamiento:

- Usamos 3 capas porque nuestra red es media o mediana.
- Tenemos una resolución baja/mediana en las imágenes que para nuestro caso son de 150*150
- El Dataset que estamos ocupando no estan grande
- Porque tambien no tenemos una GPU y hacemos uso solo de CPU

Características de las capas

- 1ra capa: borde, límites
- 2da capa: formas, texturas
- 3ra capa: patrones más abstractos

Filtros (32 -> 64 -> 128)

- La primera capa detecta los detalles pequeños
- La segunda capa que es la intermedia aprende combinaciones de los detalles
- La última capa necesita representar patrones mas complejos, esto aumenta la capacidad de representación del modelo.

Podemos agregar más capas

Podemos agregar más capas hasta llegar a un filtro del tamaño 256. Estos es incluir 4-5-6 o mas, dependiendo de factores como:

- El tamaño del Dataset, entre mas grande redes mas profundas
- Tamaño de las imágenes, mas grandes mas capas posibles
- Capacidad de procesamiento, es decir más capas, más parámetros más uso de GPU

In [18]:
```
modelo1.add(Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)))
modelo1.add(MaxPooling2D(pool_size=(2,2)))

modelo1.add(Conv2D(64, (3,3), activation='relu'))
modelo1.add(MaxPooling2D(pool_size=(2,2)))

modelo1.add(Conv2D(128, (3,3), activation='relu'))
modelo1.add(MaxPooling2D(pool_size=(2,2)))
```

Aplanamos la salida para conectarla a capas densas.

In [19]:
```
modelo1.add(Flatten())
```

Capa totalmente conectada

In [20]:
```
modelo1.add(Dense(128, activation='relu'))
modelo1.add(Dropout(0.5))
```

Capa de salida

In [21]:
```
modelo1.add(Dense(train_generator1.num_classes, activation='softmax'))
```

Compilación del modelo

- Adam: Optimizador eficiente que ajusta los pesos
- categorical_crossentropy: función de pérdida adecuada cuando usamos class_mode='categorical'

In [22]:
```
modelo1.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Resumen del modelo

In [23]:
```
modelo1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 148, 148, 32)      896

 max_pooling2d_5 (MaxPoolin  (None, 74, 74, 32)        0
 g2D)

 conv2d_8 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_6 (MaxPoolin  (None, 36, 36, 64)        0
 g2D)

 conv2d_9 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_7 (MaxPoolin  (None, 17, 17, 128)       0
 g2D)

 flatten (Flatten)           (None, 36992)             0

 dense (Dense)               (None, 128)               4735104

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 6)                 774

=================================================================
Total params: 4829126 (18.42 MB)
Trainable params: 4829126 (18.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Resumen

- Total de parámetros 4,829,126
- Entrenables 4,829,126
- No entrenables 0 Esto nos quiere decir que la red tiene mas de 4.8 millosnes de pesos que ira ajustando con el entrenamiento. ya que nuestro modelo es de tamaño medio.

In [25]:
```python
epochs1 = 10

modelo_entrenado1 = modelo1.fit(
    train_generator1,
    epochs=epochs1,
    validation_data=val_generator1
    #validation_step=val_generator.samples
)
```

```
Epoch 1/10
150/150 [==============================] - 273s 2s/step - loss: 1.4088 - accuracy:
0.4087 - val_loss: 1.1851 - val_accuracy: 0.5368
Epoch 2/10
150/150 [==============================] - 125s 830ms/step - loss: 1.1973 - accura
cy: 0.5331 - val_loss: 1.0145 - val_accuracy: 0.6154
Epoch 3/10
150/150 [==============================] - 119s 789ms/step - loss: 1.1028 - accura
cy: 0.5731 - val_loss: 1.0259 - val_accuracy: 0.6455
Epoch 4/10
150/150 [==============================] - 118s 787ms/step - loss: 1.0888 - accura
cy: 0.5863 - val_loss: 0.9696 - val_accuracy: 0.6371
Epoch 5/10
150/150 [==============================] - 119s 794ms/step - loss: 1.0673 - accura
cy: 0.5967 - val_loss: 0.9002 - val_accuracy: 0.6639
Epoch 6/10
150/150 [==============================] - 121s 802ms/step - loss: 0.9979 - accura
cy: 0.6229 - val_loss: 0.8908 - val_accuracy: 0.6722
Epoch 7/10
150/150 [==============================] - 119s 789ms/step - loss: 0.9765 - accura
cy: 0.6329 - val_loss: 0.8985 - val_accuracy: 0.6689
Epoch 8/10
150/150 [==============================] - 118s 782ms/step - loss: 0.9782 - accura
cy: 0.6363 - val_loss: 0.8213 - val_accuracy: 0.7107
Epoch 9/10
150/150 [==============================] - 118s 788ms/step - loss: 0.9401 - accura
cy: 0.6477 - val_loss: 0.8297 - val_accuracy: 0.7207
Epoch 10/10
150/150 [==============================] - 119s 794ms/step - loss: 0.9054 - accura
cy: 0.6615 - val_loss: 0.7598 - val_accuracy: 0.7559
```

In [26]:
```
history_dict1 = modelo_entrenado1.history
```

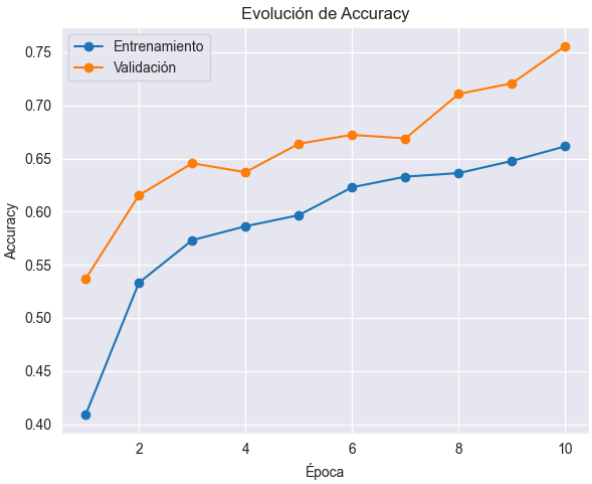
In [27]:
```
modelo_entrenado1.history
```

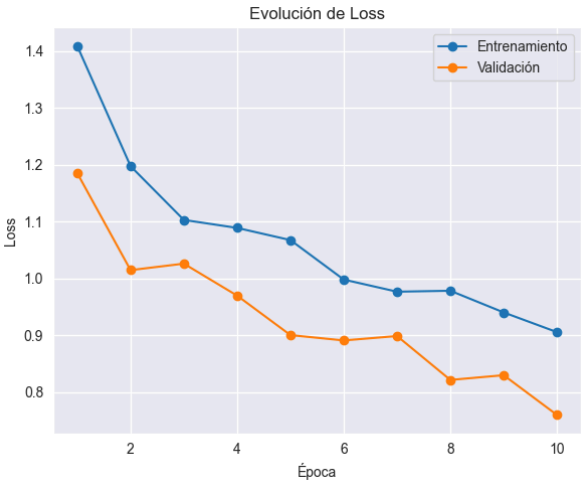
Out[27]:

```
{'loss': [1.4088014364242554,
  1.197329044342041,
  1.1027708053588867,
  1.088754653930664,
  1.0673149824142456,
  0.9978610277175903,
  0.9765427708625793,
  0.9781989455223083,
  0.9400767087936401,
  0.9054098725318909],
 'accuracy': [0.4087499976158142,
  0.5331249833106995,
  0.5731250047683716,
  0.5862500071525574,
  0.596666693687439,
  0.6229166388511658,
  0.6329166889190674,
  0.6362500190734863,
  0.6477083563804626,
  0.6614583134651184],
 'val_loss': [1.1850954294204712,
  1.0144895315170288,
  1.0258711576461792,
  0.9696007966995239,
  0.9001606702804565,
  0.89079350233078,
  0.898517906665802,
  0.8213281631469727,
  0.8296622633934021,
  0.7597572803497314],
 'val_accuracy': [0.5367892980575562,
  0.6153846383094788,
  0.6454849243164062,
  0.6371237635612488,
  0.6638795733451843,
  0.6722407937049866,
  0.6688963174819946,
  0.7107023596763611,
  0.7207357883453369,
  0.7558528184890747]}
```


In [28]:
```python
history_dict1 = modelo_entrenado1.history
epochs = range(1, len(history_dict1['accuracy']) + 1)

# Accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, history_dict1['accuracy'], 'o-', label='Entrenamiento')
plt.plot(epochs, history_dict1['val_accuracy'], 'o-', label='Validación')
plt.title('Evolución de Accuracy')
plt.xlabel('Época')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

# Loss
plt.subplot(1, 2, 2)
plt.plot(epochs, history_dict1['loss'], 'o-', label='Entrenamiento')
plt.plot(epochs, history_dict1['val_loss'], 'o-', label='Validación')
plt.title('Evolución de Loss')
plt.xlabel('Época')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```





Segundo entrenamiento

In [9]:
```
data_train = 'G:/Intel/seg_train'
data_test = 'G:/Intel/seg_test'
data_path = 'G:/Intel/seg_train'
```

In [10]:
```
train_datagen2 = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 45,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
    fill_mode = 'nearest',
    validation_split = 0.2,
)
```

In [11]:
```
val_datagen2 = ImageDataGenerator(
    rescale = 1./255,
    validation_split = 0.2,
)
```

In [32]:
```
image_size = (150, 150)
batch_size = 32

train_generator2 = train_datagen2.flow_from_directory(
    data_train,
    target_size = image_size,
    batch_size = batch_size,
    class_mode = 'categorical',
    subset = 'training',
    shuffle = True,
    seed=42
)
```

```
Found 11230 images belonging to 6 classes.
```

In [33]:
```
val_generator2 = val_datagen2.flow_from_directory(
    data_test,
    target_size = image_size,
```

```
    batch_size = batch_size,
    class_mode='categorical',
    subset='validation',
    shuffle=False,
    seed=42
)
```

```
Found 598 images belonging to 6 classes.
```

In [34]:
```
print('Clases detectadas:', train_generator2.class_indices)
```

```
Clases detectadas: {'buildings': 0, 'forest': 1, 'glacier': 2, 'mountain': 3, 'se
a': 4, 'street': 5}
```

In [35]:
```
modelo2 = Sequential()
```

In [36]:
```
modelo2.add(Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)))
modelo2.add(MaxPooling2D(pool_size=(2,2)))

modelo2.add(Conv2D(64, (3,3), activation='relu'))
modelo2.add(MaxPooling2D(pool_size=(2,2)))

modelo2.add(Conv2D(128, (3,3), activation='relu'))
modelo2.add(MaxPooling2D(pool_size=(2,2)))
```

In [37]:
```
modelo2.add(Flatten())
```

In [38]:
```
modelo2.add(Dense(128, activation='relu'))
modelo2.add(Dropout(0.5))
```

In [39]:
```
modelo2.add(Dense(train_generator2.num_classes, activation='softmax'))
```

In [40]:
```
modelo2.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [41]:
```
modelo2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 148, 148, 32)      896

 max_pooling2d_8 (MaxPoolin  (None, 74, 74, 32)        0
 g2D)

 conv2d_11 (Conv2D)          (None, 72, 72, 64)        18496

 max_pooling2d_9 (MaxPoolin  (None, 36, 36, 64)        0
 g2D)

 conv2d_12 (Conv2D)          (None, 34, 34, 128)       73856

 max_pooling2d_10 (MaxPooli  (None, 17, 17, 128)       0
 ng2D)

 flatten_1 (Flatten)         (None, 36992)             0

 dense_2 (Dense)             (None, 128)               4735104

 dropout_1 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 6)                 774

=================================================================
Total params: 4829126 (18.42 MB)
Trainable params: 4829126 (18.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [42]:

```
epochs2 = 20

modelo_entrenado2 = modelo2.fit(
    train_generator2,
    epochs=epochs2,
    validation_data=val_generator2
    #validation_step=val_generator.samples
)
```

```
Epoch 1/20
351/351 [==============================] - 660s 2s/step - loss: 1.2980 - accuracy:
0.4805 - val_loss: 1.0673 - val_accuracy: 0.5619
Epoch 2/20
351/351 [==============================] - 284s 807ms/step - loss: 1.1033 - accura
cy: 0.5738 - val_loss: 1.1376 - val_accuracy: 0.6087
Epoch 3/20
351/351 [==============================] - 281s 800ms/step - loss: 1.0035 - accura
cy: 0.6195 - val_loss: 0.8541 - val_accuracy: 0.7007
Epoch 4/20
351/351 [==============================] - 282s 802ms/step - loss: 0.9385 - accura
cy: 0.6508 - val_loss: 0.8968 - val_accuracy: 0.6957
Epoch 5/20
351/351 [==============================] - 281s 801ms/step - loss: 0.9043 - accura
cy: 0.6613 - val_loss: 0.7504 - val_accuracy: 0.7324
Epoch 6/20
351/351 [==============================] - 283s 806ms/step - loss: 0.8596 - accura
cy: 0.6889 - val_loss: 0.7220 - val_accuracy: 0.7425
Epoch 7/20
351/351 [==============================] - 282s 802ms/step - loss: 0.8267 - accura
cy: 0.7028 - val_loss: 0.6980 - val_accuracy: 0.7592
Epoch 8/20
351/351 [==============================] - 281s 798ms/step - loss: 0.7850 - accura
cy: 0.7179 - val_loss: 0.6613 - val_accuracy: 0.7692
Epoch 9/20
351/351 [==============================] - 282s 801ms/step - loss: 0.7720 - accura
cy: 0.7183 - val_loss: 0.6030 - val_accuracy: 0.7943
Epoch 10/20
351/351 [==============================] - 281s 801ms/step - loss: 0.7520 - accura
cy: 0.7302 - val_loss: 0.6221 - val_accuracy: 0.7742
Epoch 11/20
351/351 [==============================] - 282s 803ms/step - loss: 0.7352 - accura
cy: 0.7382 - val_loss: 0.6438 - val_accuracy: 0.7742
Epoch 12/20
351/351 [==============================] - 281s 799ms/step - loss: 0.7202 - accura
cy: 0.7412 - val_loss: 0.6023 - val_accuracy: 0.7843
Epoch 13/20
351/351 [==============================] - 280s 798ms/step - loss: 0.7068 - accura
cy: 0.7476 - val_loss: 0.5572 - val_accuracy: 0.8110
Epoch 14/20
351/351 [==============================] - 293s 834ms/step - loss: 0.6906 - accura
cy: 0.7531 - val_loss: 0.6187 - val_accuracy: 0.7893
Epoch 15/20
351/351 [==============================] - 280s 797ms/step - loss: 0.6780 - accura
cy: 0.7608 - val_loss: 0.5336 - val_accuracy: 0.8211
Epoch 16/20
351/351 [==============================] - 279s 794ms/step - loss: 0.6672 - accura
cy: 0.7607 - val_loss: 0.5796 - val_accuracy: 0.8010
Epoch 17/20
351/351 [==============================] - 280s 796ms/step - loss: 0.6728 - accura
cy: 0.7639 - val_loss: 0.6068 - val_accuracy: 0.7910
Epoch 18/20
351/351 [==============================] - 280s 796ms/step - loss: 0.6492 - accura
cy: 0.7686 - val_loss: 0.6354 - val_accuracy: 0.7592
Epoch 19/20
351/351 [==============================] - 283s 804ms/step - loss: 0.6422 - accura
cy: 0.7711 - val_loss: 0.5552 - val_accuracy: 0.8010
Epoch 20/20
351/351 [==============================] - 279s 794ms/step - loss: 0.6292 - accura
cy: 0.7739 - val_loss: 0.5713 - val_accuracy: 0.8110
```

In [43]:
```
history_dict2 = modelo_entrenado2.history
```

In [44]:
```
modelo_entrenado2.history.keys()
```

Out[44]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

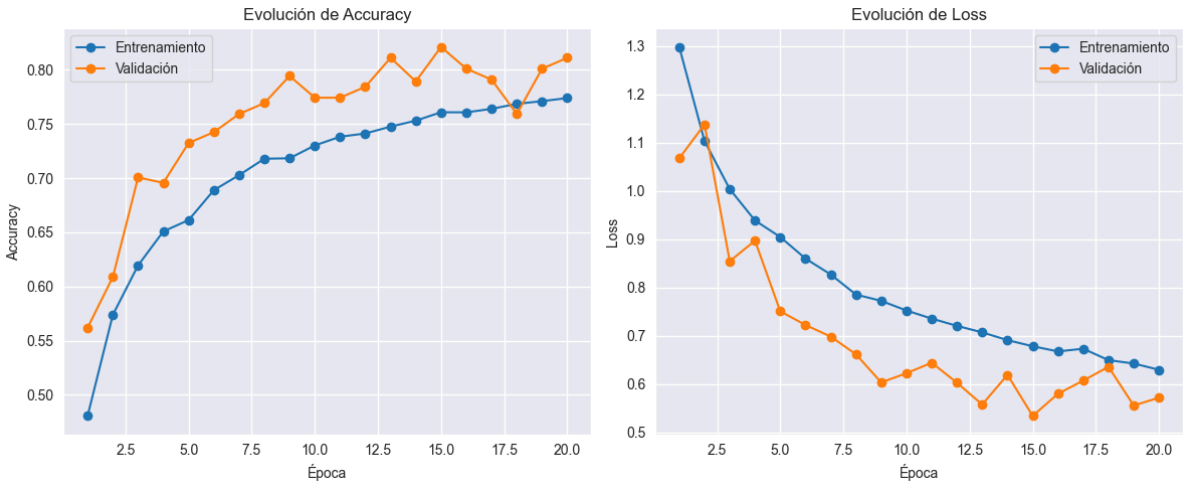
In [45]:
```
modelo_entrenado2.history
```

Out[45]:

```
{'loss': [1.2980388402938843,
  1.1033177375793457,
  1.0035395622253418,
  0.9384883642196655,
  0.9043436646461487,
  0.859642505645752,
  0.8266555666923523,
  0.78504478931427,
  0.7720282077789307,
  0.7519802451133728,
  0.7352477312088013,
  0.7201619148254395,
  0.7067561745643616,
  0.6906139254570007,
  0.6779631972312927,
  0.6672425270080566,
  0.6728276014328003,
  0.6491921544075012,
  0.6421696543693542,
  0.6292039752006531],
 'accuracy': [0.48049867153167725,
  0.5738201141357422,
  0.6195013523101807,
  0.6508459448814392,
  0.6612644791603088,
  0.6888691186904907,
  0.7027604579925537,
  0.7178984880447388,
  0.7183437347412109,
  0.7301869988441467,
  0.7382012605667114,
  0.7412288784980774,
  0.7475512027740479,
  0.7530721426010132,
  0.7608192563056946,
  0.7607302069664001,
  0.7639358639717102,
  0.7685663104057312,
  0.7710596323013306,
  0.7739091515541077],
 'val_loss': [1.0672651529312134,
  1.137558937072754,
  0.8541373610496521,
  0.8967649340629578,
  0.7503944039344788,
  0.722000777721405,
  0.6980310082435608,
  0.6612836718559265,
  0.6030112504959106,
  0.6220883131027222,
  0.643780529499054,
  0.6022869944572449,
  0.557243287563324,
  0.6187296509742737,
  0.5336061716079712,
  0.5795917510986328,
  0.6067537665367126,
  0.6353688836097717,
  0.5552242398262024,
  0.5712815523147583],
 'val_accuracy': [0.5618728995323181,
  0.6086956262588501,
  0.7006688714027405,
  0.695652186870575,
```

```
0.7324414849281311,
0.7424749135971069,
0.7591972947120667,
0.7692307829856873,
0.7943143844604492,
0.7742474675178528,
0.7742474675178528,
0.7842809557914734,
0.8110367655754089,
0.7892976403236389,
0.8210702538490295,
0.8010033369064331,
0.7909699082374573,
0.7591972947120667,
0.8010033369064331,
0.8110367655754089]}
```

In [65]:
```python
history_dict2 = modelo_entrenado2.history
epochs2 = range(1, len(history_dict2['accuracy']) + 1)

# Accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs2, history_dict2['accuracy'], 'o-', label='Entrenamiento')
plt.plot(epochs2, history_dict2['val_accuracy'], 'o-', label='Validación')
plt.title('Evolución de Accuracy')
plt.xlabel('Época')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

# Loss
plt.subplot(1, 2, 2)
plt.plot(epochs2, history_dict2['loss'], 'o-', label='Entrenamiento')
plt.plot(epochs2, history_dict2['val_loss'], 'o-', label='Validación')
plt.title('Evolución de Loss')
plt.xlabel('Época')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



Tercer entranamiento:

In [12]:
```
data_train = 'G:/Intel/seg_train'
data_test = 'G:/Intel/seg_test'
data_path = 'G:/Intel/seg_train'
```

In [26]:
```
train_datagen3 = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 25,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.15,
    zoom_range = 0.2,
    horizontal_flip = True,
    fill_mode = 'nearest',
    validation_split = 0.2
)
```

In [27]:
```
val_datagen3 = ImageDataGenerator(
    rescale = 1./255
)
```

In [28]:
```
image_size = (150, 150)
batch_size = 32

train_generator3 = train_datagen3.flow_from_directory(
    data_train,
    target_size = image_size,
    batch_size = batch_size,
    class_mode = 'categorical'
)
```

```
Found 14034 images belonging to 6 classes.
```

In [29]:
```
val_generator3 = val_datagen3.flow_from_directory(
    data_test,
    target_size = image_size,
    batch_size = batch_size,
    class_mode='categorical'
)
```

```
Found 3000 images belonging to 6 classes.
```

In [31]:
```
modelo3 = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(pool_size=(2,2)),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),

    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(train_generator3.num_classes, activation='softmax')
])
```

In [32]:
```
modelo3.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [33]:
```
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

In [34]:
```
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=3,
    min_lr=1e-6
)
```

In [36]:
```
epochs3 = 30

modelo_entrenado3 = modelo3.fit(
    train_generator3,
    epochs=epochs3,
    validation_data=val_generator3,
    callbacks=[early_stop, reduce_lr]
)
```

```
Epoch 1/30
439/439 [==============================] - 1062s 2s/step - loss: 1.1994 - accurac
y: 0.5202 - val_loss: 0.9189 - val_accuracy: 0.6377 - lr: 0.0010
Epoch 2/30
439/439 [==============================] - 518s 1s/step - loss: 1.0056 - accuracy:
0.6162 - val_loss: 0.8245 - val_accuracy: 0.6863 - lr: 0.0010
Epoch 3/30
439/439 [==============================] - 459s 1s/step - loss: 0.8840 - accuracy:
0.6719 - val_loss: 0.6798 - val_accuracy: 0.7593 - lr: 0.0010
Epoch 4/30
439/439 [==============================] - 466s 1s/step - loss: 0.8010 - accuracy:
0.7028 - val_loss: 0.7166 - val_accuracy: 0.7433 - lr: 0.0010
Epoch 5/30
439/439 [==============================] - 484s 1s/step - loss: 0.7538 - accuracy:
0.7288 - val_loss: 0.5603 - val_accuracy: 0.8070 - lr: 0.0010
Epoch 6/30
439/439 [==============================] - 459s 1s/step - loss: 0.7139 - accuracy:
0.7441 - val_loss: 0.5447 - val_accuracy: 0.7987 - lr: 0.0010
Epoch 7/30
439/439 [==============================] - 460s 1s/step - loss: 0.6898 - accuracy:
0.7543 - val_loss: 0.6379 - val_accuracy: 0.7753 - lr: 0.0010
Epoch 8/30
439/439 [==============================] - 438s 997ms/step - loss: 0.6565 - accura
cy: 0.7703 - val_loss: 0.6025 - val_accuracy: 0.7857 - lr: 0.0010
Epoch 9/30
439/439 [==============================] - 433s 984ms/step - loss: 0.6541 - accura
cy: 0.7686 - val_loss: 0.5621 - val_accuracy: 0.8137 - lr: 0.0010
Epoch 10/30
439/439 [==============================] - 464s 1s/step - loss: 0.5838 - accuracy:
0.7909 - val_loss: 0.4378 - val_accuracy: 0.8433 - lr: 5.0000e-04
Epoch 11/30
439/439 [==============================] - 493s 1s/step - loss: 0.5599 - accuracy:
0.7970 - val_loss: 0.5015 - val_accuracy: 0.8157 - lr: 5.0000e-04
Epoch 12/30
439/439 [==============================] - 462s 1s/step - loss: 0.5577 - accuracy:
0.8029 - val_loss: 0.4613 - val_accuracy: 0.8327 - lr: 5.0000e-04
Epoch 13/30
439/439 [==============================] - 462s 1s/step - loss: 0.5399 - accuracy:
0.8105 - val_loss: 0.4678 - val_accuracy: 0.8313 - lr: 5.0000e-04
Epoch 14/30
439/439 [==============================] - 462s 1s/step - loss: 0.5082 - accuracy:
0.8212 - val_loss: 0.4241 - val_accuracy: 0.8473 - lr: 2.5000e-04
Epoch 15/30
439/439 [==============================] - 451s 1s/step - loss: 0.4955 - accuracy:
0.8270 - val_loss: 0.4305 - val_accuracy: 0.8510 - lr: 2.5000e-04
Epoch 16/30
439/439 [==============================] - 453s 1s/step - loss: 0.4951 - accuracy:
0.8263 - val_loss: 0.4197 - val_accuracy: 0.8547 - lr: 2.5000e-04
Epoch 17/30
439/439 [==============================] - 472s 1s/step - loss: 0.4843 - accuracy:
0.8278 - val_loss: 0.4282 - val_accuracy: 0.8517 - lr: 2.5000e-04
Epoch 18/30
439/439 [==============================] - 452s 1s/step - loss: 0.4923 - accuracy:
0.8258 - val_loss: 0.4373 - val_accuracy: 0.8473 - lr: 2.5000e-04
Epoch 19/30
439/439 [==============================] - 455s 1s/step - loss: 0.4663 - accuracy:
0.8355 - val_loss: 0.4208 - val_accuracy: 0.8520 - lr: 2.5000e-04
Epoch 20/30
439/439 [==============================] - 458s 1s/step - loss: 0.4592 - accuracy:
0.8432 - val_loss: 0.4128 - val_accuracy: 0.8537 - lr: 1.2500e-04
Epoch 21/30
439/439 [==============================] - 467s 1s/step - loss: 0.4593 - accuracy:
0.8378 - val_loss: 0.4133 - val_accuracy: 0.8550 - lr: 1.2500e-04
Epoch 22/30
```

```
439/439 [==============================] - 464s 1s/step - loss: 0.4413 - accuracy:
0.8420 - val_loss: 0.4005 - val_accuracy: 0.8587 - lr: 1.2500e-04
Epoch 23/30
439/439 [==============================] - 450s 1s/step - loss: 0.4425 - accuracy:
0.8413 - val_loss: 0.4149 - val_accuracy: 0.8533 - lr: 1.2500e-04
Epoch 24/30
439/439 [==============================] - 456s 1s/step - loss: 0.4421 - accuracy:
0.8467 - val_loss: 0.3873 - val_accuracy: 0.8660 - lr: 1.2500e-04
Epoch 25/30
439/439 [==============================] - 454s 1s/step - loss: 0.4401 - accuracy:
0.8413 - val_loss: 0.4115 - val_accuracy: 0.8597 - lr: 1.2500e-04
Epoch 26/30
439/439 [==============================] - 596s 1s/step - loss: 0.4382 - accuracy:
0.8453 - val_loss: 0.4106 - val_accuracy: 0.8573 - lr: 1.2500e-04
Epoch 27/30
439/439 [==============================] - 536s 1s/step - loss: 0.4310 - accuracy:
0.8501 - val_loss: 0.3832 - val_accuracy: 0.8653 - lr: 1.2500e-04
Epoch 28/30
439/439 [==============================] - 478s 1s/step - loss: 0.4289 - accuracy:
0.8486 - val_loss: 0.3904 - val_accuracy: 0.8633 - lr: 1.2500e-04
Epoch 29/30
439/439 [==============================] - 471s 1s/step - loss: 0.4258 - accuracy:
0.8517 - val_loss: 0.3754 - val_accuracy: 0.8683 - lr: 1.2500e-04
Epoch 30/30
439/439 [==============================] - 461s 1s/step - loss: 0.4292 - accuracy:
0.8506 - val_loss: 0.4251 - val_accuracy: 0.8560 - lr: 1.2500e-04
```

In [38]:
```
history_dict3 = modelo_entrenado3.history
```

In [39]:
```
modelo_entrenado3.history.keys()
```

Out[39]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy', 'lr'])
```

In [40]:
```
modelo_entrenado3.history
```

Out[40]:

```
{'loss': [1.1994311809539795,
  1.0056487321853638,
  0.8840314149856567,
  0.8009738922119141,
  0.753781259059906,
  0.7139289379119873,
  0.689836323261261,
  0.6565174460411072,
  0.6540573835372925,
  0.5838224291801453,
  0.5599307417869568,
  0.5576876997947693,
  0.5398508906364441,
  0.5081998109817505,
  0.4955448508262634,
  0.4951328933238983,
  0.4842793643474579,
  0.4923228919506073,
  0.4662681221961975,
  0.4592363536357879 6,
  0.459299772977829,
  0.44129329919815063,
  0.44251206517219543,
  0.44209861755371094,
  0.4400606155395508,
  0.4381740689277649,
  0.4310492277145386,
  0.4289337992668152,
  0.42584457993507385,
  0.4291658103466034],
 'accuracy': [0.5202365517616272,
  0.6162177324295044,
  0.6718683242797852,
  0.7027932405471802,
  0.7288014888763428,
  0.7441214323043823,
  0.7543109655380249,
  0.7702721953392029,
  0.7685620784759521,
  0.7908650636672974,
  0.7969930171966553,
  0.8029072284698486,
  0.8104603290557861,
  0.8212199211120605,
  0.826991617679596,
  0.8262790441513062,
  0.8278466463088989,
  0.8257802724838257,
  0.8355422616004944,
  0.8432378768920898,
  0.8377511501312256,
  0.8420265316963196,
  0.8413139581680298,
  0.8466581106185913,
  0.8413139581680298,
  0.8453042507171631,
  0.8501496315002441,
  0.8485820293426514,
  0.8517172336578369,
  0.8506484031677246],
 'val_loss': [0.9189009070396423,
  0.8244810104370117,
  0.679813802242279,
  0.7165949940681458,
```

```
 0.5602884292602539,
 0.5447397232055664,
 0.6379279494285583,
 0.602499783039093,
 0.5620777010917664,
 0.4378431737422943,
 0.5014508962631226,
 0.46128031611442566,
 0.46779489517211914,
 0.4241483211517334,
 0.43049007654190063,
 0.41970834136009216,
 0.4282259941101074,
 0.4372844099998474,
 0.4207509756088257,
 0.41282787919044495,
 0.41326066851615906,
 0.40053218603134155,
 0.41491174697875977,
 0.38728466629981995,
 0.4115121364593506,
 0.41064995527267456,
 0.38317057490348816,
 0.3903634548187256,
 0.3754015564918518,
 0.4250618815422058],
'val_accuracy': [0.637666642665863,
 0.6863333582878113,
 0.7593333125114441,
 0.7433333396911621,
 0.8069999814033508,
 0.7986666560173035,
 0.7753333449363708,
 0.7856666445732117,
 0.8136666417121887,
 0.8433333039283752,
 0.815666675567627,
 0.8326666951179504,
 0.831333339214325,
 0.8473333120346069,
 0.8510000109672546,
 0.8546666502952576,
 0.8516666889190674,
 0.8473333120346069,
 0.8519999980926514,
 0.8536666631698608,
 0.8550000190734863,
 0.8586666584014893,
 0.8533333539962769,
 0.8659999966621399,
 0.859666645526886,
 0.8573333621025085,
 0.8653333187103271,
 0.8633333444595337,
 0.8683333396911621,
 0.8560000061988831],
'lr': [0.001,
 0.001,
 0.001,
 0.001,
 0.001,
 0.001,
 0.001,
 0.001,
```

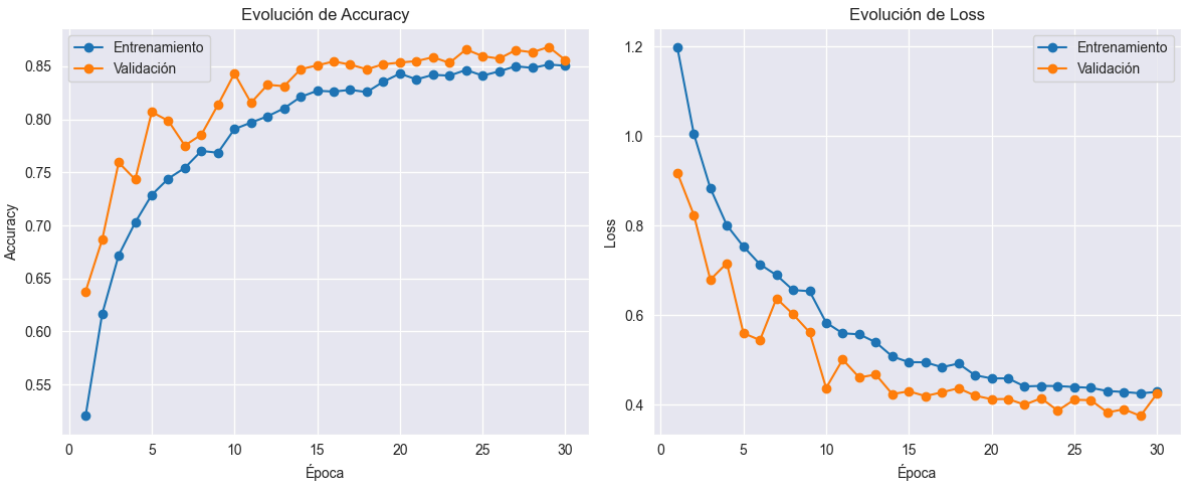
```
0.001,
0.0005,
0.0005,
0.0005,
0.0005,
0.00025,
0.00025,
0.00025,
0.00025,
0.00025,
0.00025,
0.000125,
0.000125,
0.000125,
0.000125,
0.000125,
0.000125,
0.000125,
0.000125,
0.000125,
0.000125]}
```

In [44]:
```
modelo_final = modelo_entrenado3
```

Guardamos y el modleo para poder despues poder reutilizarlo.

In [52]:
```
modelo3.save("modelo_entrenado3.keras")
```

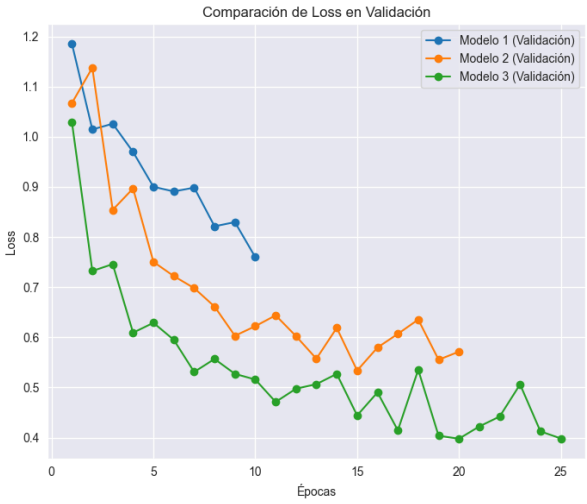
In [53]:
```
history_dict3 = modelo_entrenado3.history
epochs3 = range(1, len(history_dict3['accuracy']) + 1)

# Accuracy
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs3, history_dict3['accuracy'], 'o-', label='Entrenamiento')
plt.plot(epochs3, history_dict3['val_accuracy'], 'o-', label='Validación')
plt.title('Evolución de Accuracy')
plt.xlabel('Época')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

# Loss
plt.subplot(1, 2, 2)
plt.plot(epochs3, history_dict3['loss'], 'o-', label='Entrenamiento')
plt.plot(epochs3, history_dict3['val_loss'], 'o-', label='Validación')
plt.title('Evolución de Loss')
plt.xlabel('Época')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



Comparativa de los 3 entrenamietnos junto accuracy y loss

In [66]:

```
epochs = range(1, len(history_dict1['accuracy']) + 1)
epochs2 = range(1, len(history_dict2['accuracy']) + 1)
epochs3 = range(1, len(history_dict3['accuracy']) + 1)

plt.figure(figsize=(14, 6))


plt.subplot(1, 2, 1)
plt.plot(epochs, history_dict1['val_accuracy'], 'o-', label='Modelo 1 (Validación)'
plt.plot(epochs2, history_dict2['val_accuracy'], 'o-', label='Modelo 2 (Validación)
plt.plot(epochs3, history_dict3['val_accuracy'], 'o-', label='Modelo 3 (Validación)
plt.title('Comparación de Accuracy en Validación')
plt.xlabel('Épocas')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)


plt.subplot(1, 2, 2)
plt.plot(epochs, history_dict1['val_loss'], 'o-', label='Modelo 1 (Validación)')
plt.plot(epochs2, history_dict2['val_loss'], 'o-', label='Modelo 2 (Validación)')
plt.plot(epochs3, history_dict3['val_loss'], 'o-', label='Modelo 3 (Validación)')
plt.title('Comparación de Loss en Validación')
plt.xlabel('Épocas')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

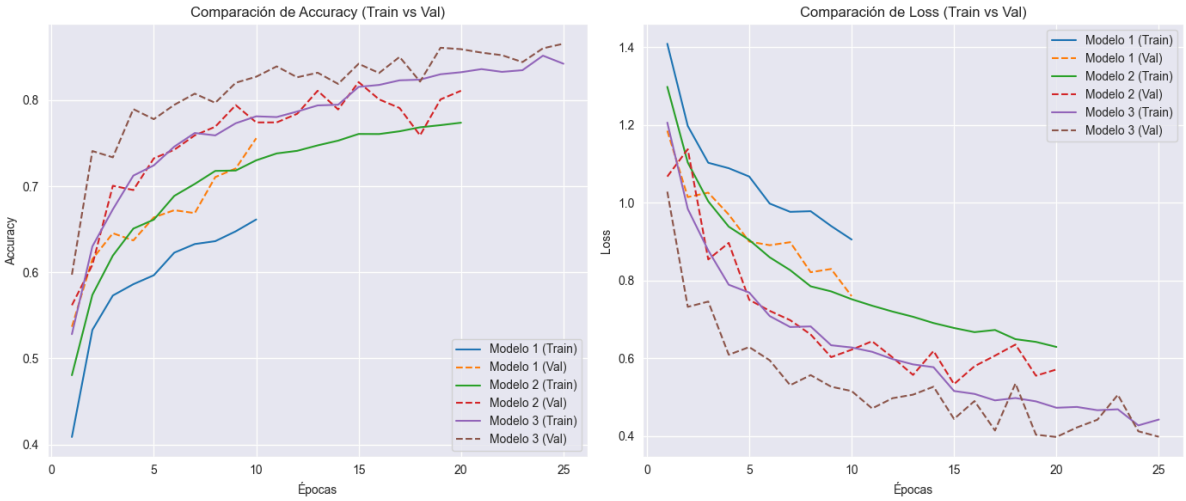


Curvas de entrenamiento y validación

In [67]:

```
epochs = range(1, len(history_dict1['accuracy']) + 1)
epochs2 = range(1, len(history_dict2['accuracy']) + 1)
epochs3 = range(1, len(history_dict3['accuracy']) + 1)

plt.figure(figsize=(14, 6))


plt.subplot(1, 2, 1)
# Modelo 1
plt.plot(epochs, history_dict1['accuracy'], '-', label='Modelo 1 (Train)')
plt.plot(epochs, history_dict1['val_accuracy'], '--', label='Modelo 1 (Val)')
# Modelo 2
plt.plot(epochs2, history_dict2['accuracy'], '-', label='Modelo 2 (Train)')
plt.plot(epochs2, history_dict2['val_accuracy'], '--', label='Modelo 2 (Val)')
# Modelo 3
plt.plot(epochs3, history_dict3['accuracy'], '-', label='Modelo 3 (Train)')
plt.plot(epochs3, history_dict3['val_accuracy'], '--', label='Modelo 3 (Val)')

plt.title('Comparación de Accuracy (Train vs Val)')
plt.xlabel('Épocas')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)


plt.subplot(1, 2, 2)
# Modelo 1
plt.plot(epochs, history_dict1['loss'], '-', label='Modelo 1 (Train)')
plt.plot(epochs, history_dict1['val_loss'], '--', label='Modelo 1 (Val)')
# Modelo 2
plt.plot(epochs2, history_dict2['loss'], '-', label='Modelo 2 (Train)')
plt.plot(epochs2, history_dict2['val_loss'], '--', label='Modelo 2 (Val)')
# Modelo 3
plt.plot(epochs3, history_dict3['loss'], '-', label='Modelo 3 (Train)')
plt.plot(epochs3, history_dict3['val_loss'], '--', label='Modelo 3 (Val)')

plt.title('Comparación de Loss (Train vs Val)')
plt.xlabel('Épocas')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```



Comparación general de los entrenamientos.

Modelo	Épocas	Accuracy final (train)	Val Accuracy final	Loss final (train)	Val loss (final)
1	10	~0.66	0.75	~0.90	0.76
2	20	~0.77	0.81	~0.63	0.57
3	25	~0.84	0.86	~0.44	0.39

Evaluación del rendimiento

- Entrenamiento 1:
  - Empezó con un accuracy bajo 0.40, pero subió rápido a 0.66 train y 0.75 val en solo 10 épocas.
  - Las curvas muestran que aun había margen para mejorar, pero el número de épocas fue limitado.
  - Se tiene buen resultado inicial, pero se quedó corto
- Entrenamiento 2:
  - Llegó a 0.77 train y 0.81 val en 20 épocas
  - La validación siguió mejorando sin señales claras de sobreajuste.
  - Muestra que más daos y más épocas mejoran notablemente el modelo
- Entrenamiento 3:
  - Se nota que fue el mejor por que alcanzo un 0.84 en train y 0.86 en val
  - Además, el val_loos bajó hasta 0.39, lo que indica que generaliza bien.
  - A partir de la época 20, el modelo ya tenia un rendimiento muy estable, con un ligero ajuste gracias a ReducelROnPlateau
  - La reducción del learning rate de 0.001 a 0.0005 a 0.00025 ayudó a pulir la convergencia.

-

- Podemos ver que cada entrenamiento supero al anterior en precision y estabilidad, ya que se fueron aumentado los datos, y las épocas.
- El modelo 3 no muestra un sobreajuste train y val accuracy estan muy cercanos 0.84 y 0.86
- En el modelo 3 despues de la época 20 no mejora mucho y ya no era necesario llegar hasta la apoca 30.
- Nos quedamos con el resultado del modelo 3, auqnue aun se le pueden hacer algunos ajustes para poder tener un accuaracy mas alto.

Análisis de aciertos y errores

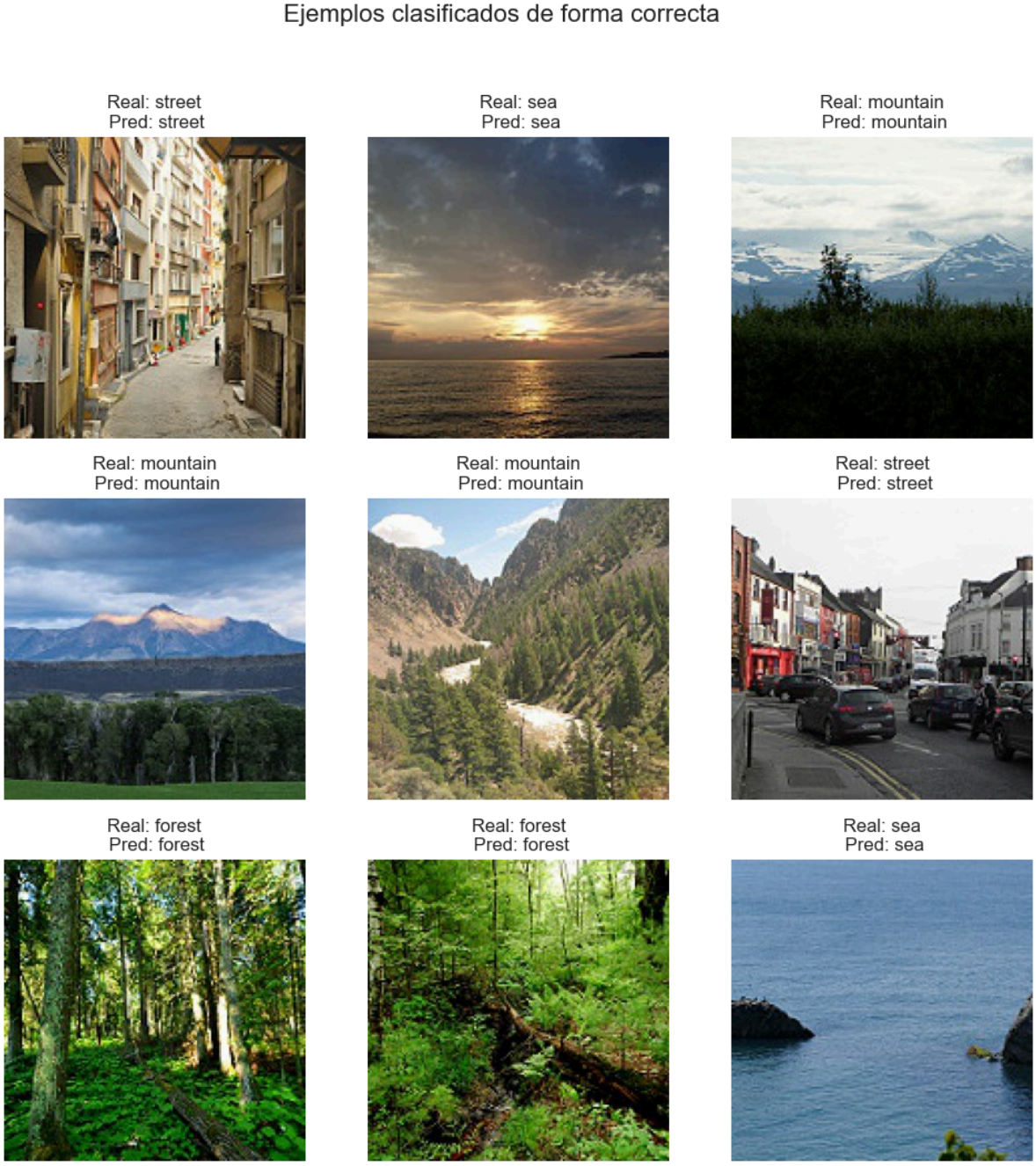
In [25]:
```
images_ = []
labels_ = []
for batch_images, batch_labels in val_generator3:
     images_.append(batch_images)
     labels_.append(batch_labels)
     if len(images_) * val_generator3.batch_size >= val_generator3.samples:
         break

images_ = np.vstack(images_)
labels_ = np.vstack(labels_)
```

In [26]:
```
#val_images, val_labels = next(iter(val_generator3))

predicciones = modelo3.predict(images_)
prediccion_clases = np.argmax(predicciones, axis=1)
clases_verdaderas =  np.argmax(labels_, axis=1)

correctos = np.where(prediccion_clases == clases_verdaderas)[0]
incorrectos = np.where(prediccion_clases != clases_verdaderas)[0]

clases = list(train_generator3.class_indices.keys())
```

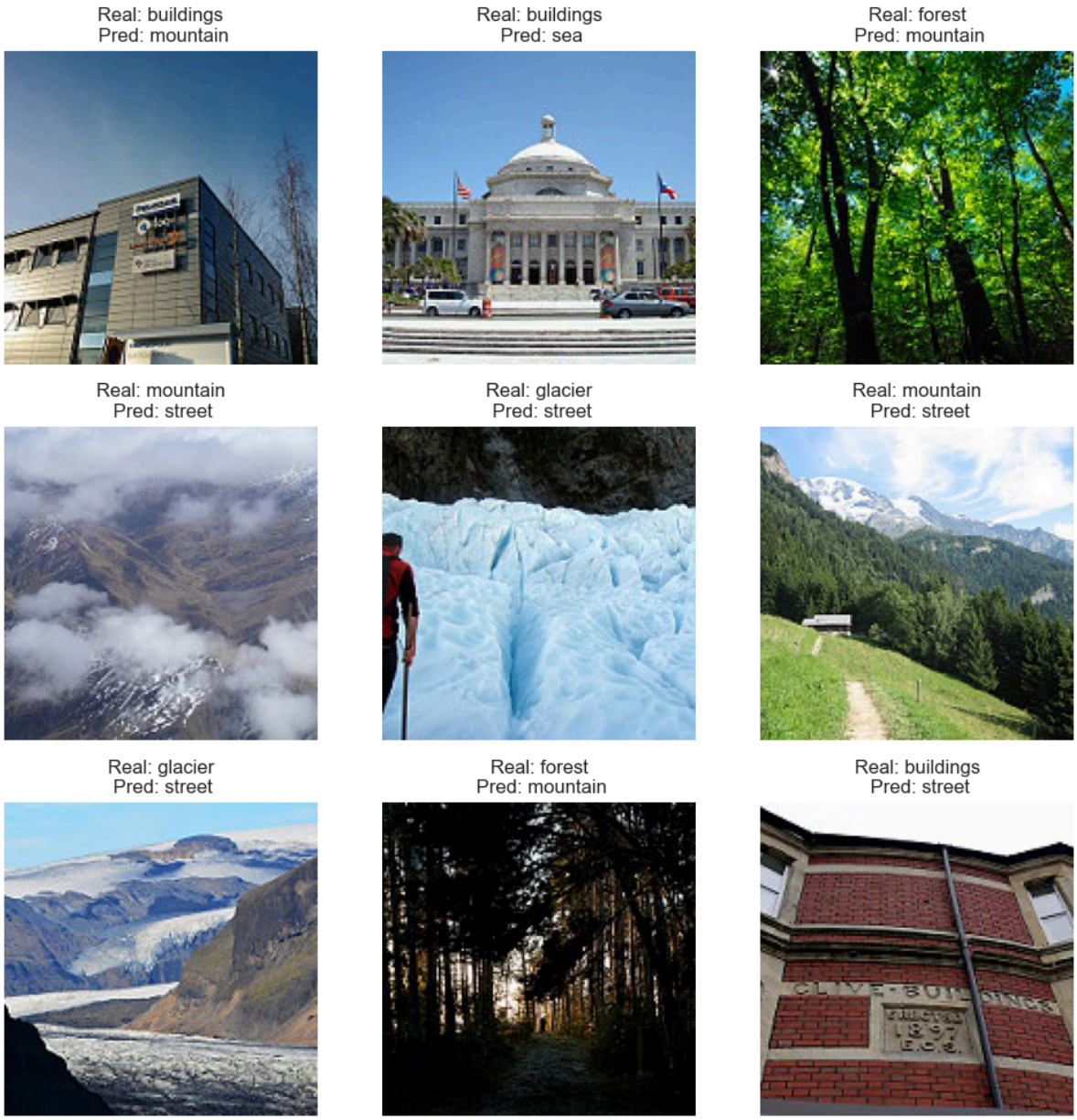
```
94/94 [==============================] - 18s 186ms/step
```

In [33]:
```
def mostrar_datos(indices, titulo, n=9):
    plt.figure(figsize=(12, 12))

    if len(indices) > n:
        selected = random.sample(list(indices), n)
    else:
        selected = indices

    for i, idx in enumerate(selected):
        plt.subplot(3,3, i+1)
        plt.imshow(images_[idx])
        plt.axis('off')
        plt.title(f'Real: {clases[clases_verdaderas[idx]]}\n Pred: {clases[predicci
    plt.suptitle(titulo, fontsize=16)
    plt.show()
```

In [35]:
```
mostrar_datos(correctos, 'Ejemplos clasificados de forma correcta', n=9)
```



In [36]:
```
mostrar_datos(incorrectos, 'Ejemplos clasificados de forma incorrecta', n=9)
```



In [59]:
```
vl3 = val_generator3
tr3 = train_generator3
data_test = 'G:/Intel/seg_test'
seg_test = 'G:/Intel/seg_test'
```

In [60]:
```
test_datagen = ImageDataGenerator(rescale=1./255)

test_dir = 'G:/Intel/seg_test'
batch_size = 32
img_size = (150, 150)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)

modelo_final = modelo_entrenado3
```

```
Found 3000 images belonging to 6 classes.
```

Evaluación global (loss y accuracy) + predicciones

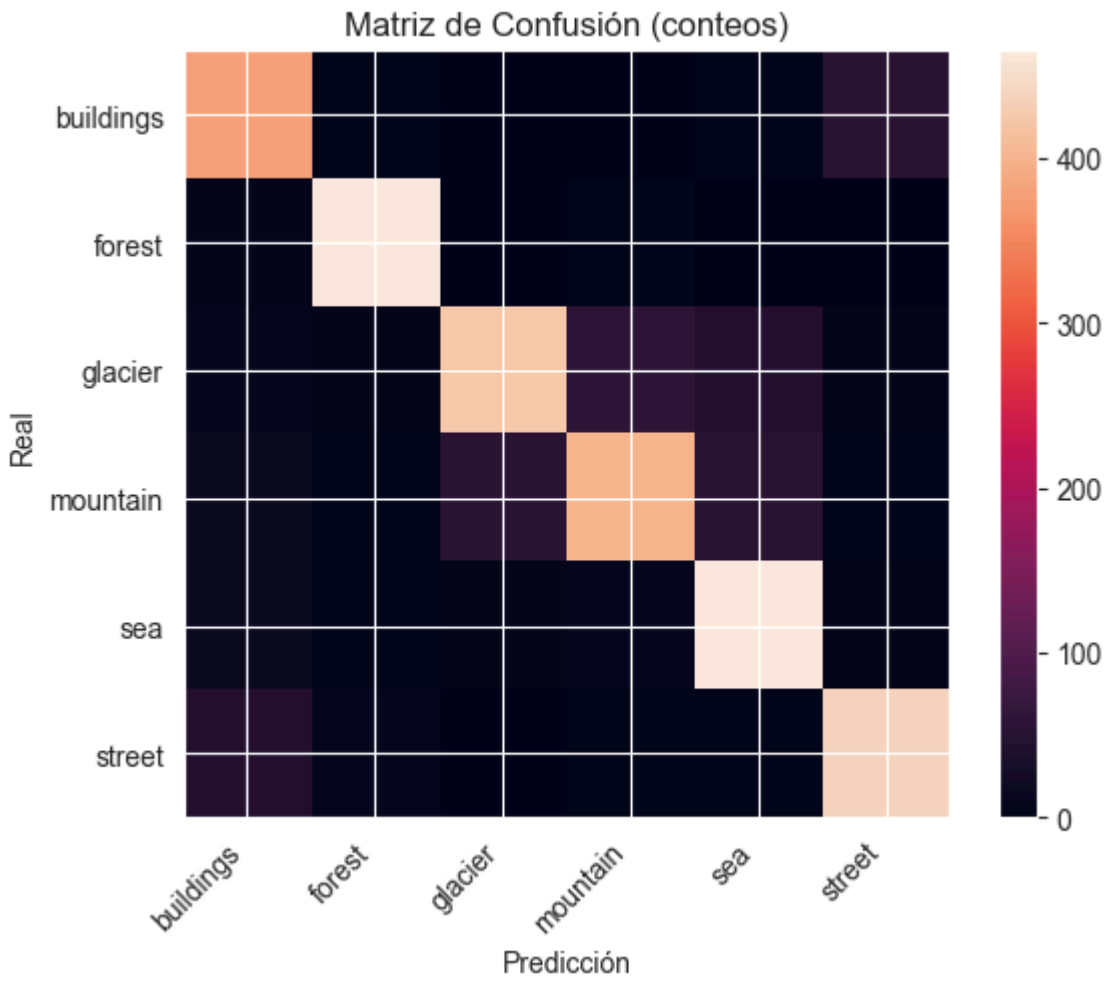
In [63]:
```
test_loss, test_acc = modelo3.evaluate(test_generator, verbose=1)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_acc:.4f}")


y_prob = modelo3.predict(test_generator, verbose=1)


y_pred = np.argmax(y_prob, axis=1)


y_true = test_generator.classes


idx_to_class = {v: k for k, v in test_generator.class_indices.items()}
class_names = [idx_to_class[i] for i in range(len(idx_to_class))]

print("Clases (orden):", class_names)
```

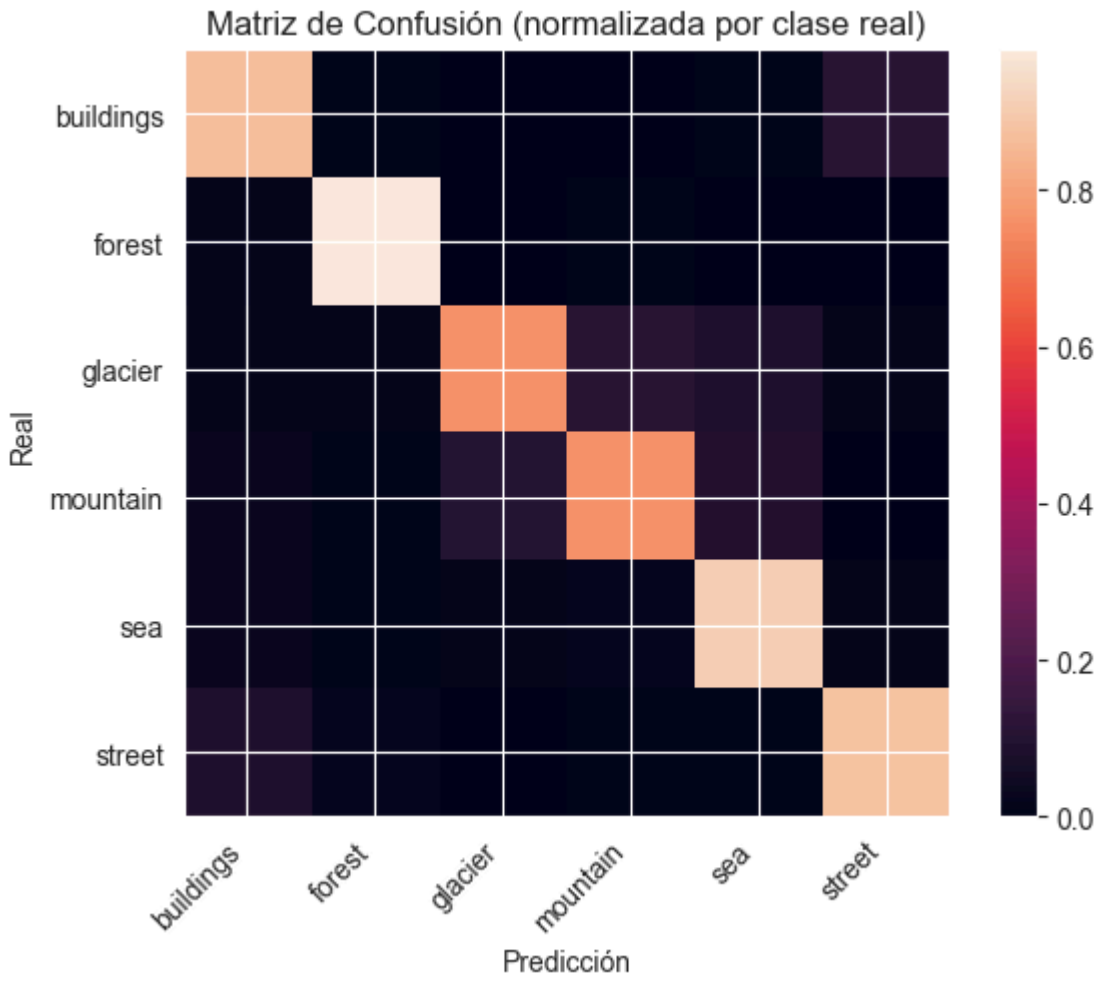
```
94/94 [==============================] - 41s 428ms/step - loss: 0.4251 - accuracy:
0.8560
Test Loss: 0.4251
Test Accuracy: 0.8560
94/94 [==============================] - 23s 237ms/step
Clases (orden): ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
```

Matriz e confusión (cruda y normalizada)

In [64]:
```
cm = confusion_matrix(y_true, y_pred)


cm_norm = cm.astype('float') / cm.sum(axis=1, keepdims=True)


plt.figure(figsize=(6, 5))
plt.imshow(cm, interpolation='nearest')
plt.title('Matriz de Confusión (conteos)')
plt.colorbar()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names, rotation=45, ha='right')
plt.yticks(tick_marks, class_names)
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.tight_layout()
plt.show()


plt.figure(figsize=(6, 5))
plt.imshow(cm_norm, interpolation='nearest')
plt.title('Matriz de Confusión (normalizada por clase real)')
plt.colorbar()
plt.xticks(tick_marks, class_names, rotation=45, ha='right')
plt.yticks(tick_marks, class_names)
plt.xlabel('Predicción')
```

```
plt.ylabel('Real')
plt.tight_layout()
plt.show()
```





Reporte de clasificación por clase y tabla de resumen

In [65]:
```
report_dict = classification_report(
    y_true, y_pred,
    target_names=class_names,
    output_dict=True
)
df_report = pd.DataFrame(report_dict).transpose()
df_report = df_report.round(4)
df_report
```

Out[65]:

	precision	recall	f1-score	support
buildings	0.8025	0.8650	0.8326	437.000
forest	0.9393	0.9789	0.9587	474.000
glacier	0.8737	0.7631	0.8147	553.000
mountain	0.8421	0.7619	0.8000	525.000
sea	0.8172	0.9118	0.8619	510.000
street	0.8642	0.8762	0.8702	501.000
accuracy	0.8560	0.8560	0.8560	0.856
macro avg	0.8565	0.8595	0.8563	3000.000
weighted avg	0.8570	0.8560	0.8548	3000.000

Entrenamiento con EfficientNetB0

In [5]:
```
data_train4 = 'G:/Intel/seg_train'
data_test4 = 'G:/Intel/seg_test'
data_path4 = 'G:/Intel/seg_train'
```

In [6]:
```
train_datagen4 = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 45,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

In [7]:
```
val_datagen4 = ImageDataGenerator(
    rescale = 1./255,
)
```

In [8]:
```
train_generator4 = train_datagen4.flow_from_directory(
    'G:/Intel/seg_train',
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 14034 images belonging to 6 classes.
```

In [10]:
```
val_generator4 = val_datagen4.flow_from_directory(
     'G:/Intel/seg_test',
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)
```

```
Found 3000 images belonging to 6 classes.
```

In [11]:
```
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=Tru
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e
```

```
checkpoint = ModelCheckpoint('mejor_modelo4.h5', monitor='val_accuracy', save_best_
```

In [12]:
```
base_model = EfficientNetB0(weights='imagenet', include_top=False, input_shape=(150
base_model.trainable = False
```

```
WARNING:tensorflow:From C:\Users\Lalo\AppData\Roaming\Python\Python311\site-packag
es\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_functions is d
eprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\Lalo\AppData\Roaming\Python\Python311\site-packag
es\keras\src\layers\normalization\batch_normalization.py:979: The name tf.nn.fused
_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.

Downloading data from https://storage.googleapis.com/keras-applications/efficientn
etb0_notop.h5
16705208/16705208 [==============================] - 2s 0us/step
```

In [13]:
```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.3)(x)
```

In [14]:
```
predictions = Dense(train_generator4.num_classes, activation='softmax')(x)
```

In [15]:
```
modelo4 = Model(inputs=base_model.input, outputs=predictions)
```

In [17]:
```
modelo4.compile(
    optimizer=Adam(learning_rate=1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [18]:
```
history4_fase1 = modelo4.fit(
    train_generator4,
    epochs=10,
    validation_data=val_generator4,
    callbacks=[early_stop, reduce_lr, checkpoint]
)
```

```
Epoch 1/10
WARNING:tensorflow:From C:\Users\Lalo\AppData\Roaming\Python\Python311\site-packag
es\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprec
ated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Lalo\AppData\Roaming\Python\Python311\site-packag
es\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprec
ated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Lalo\AppData\Roaming\Python\Python311\site-packag
es\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside
_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functi
ons instead.

WARNING:tensorflow:From C:\Users\Lalo\AppData\Roaming\Python\Python311\site-packag
es\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside
_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functi
ons instead.

439/439 [==============================] - ETA: 0s - loss: 1.8205 - accuracy: 0.17
08
Epoch 1: val_accuracy improved from -inf to 0.17500, saving model to mejor_modelo
4.h5
```

```
C:\Users\Lalo\AppData\Roaming\Python\Python311\site-packages\keras\src\engine\training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
439/439 [==============================] - 936s 2s/step - loss: 1.8205 - accuracy: 0.1708 - val_loss: 1.7932 - val_accuracy: 0.1750 - lr: 1.0000e-05
Epoch 2/10
439/439 [==============================] - ETA: 0s - loss: 1.8092 - accuracy: 0.1699
Epoch 2: val_accuracy did not improve from 0.17500
439/439 [==============================] - 371s 845ms/step - loss: 1.8092 - accuracy: 0.1699 - val_loss: 1.7903 - val_accuracy: 0.1750 - lr: 1.0000e-05
Epoch 3/10
439/439 [==============================] - ETA: 0s - loss: 1.8090 - accuracy: 0.1637
Epoch 3: val_accuracy did not improve from 0.17500
439/439 [==============================] - 363s 826ms/step - loss: 1.8090 - accuracy: 0.1637 - val_loss: 1.7902 - val_accuracy: 0.1750 - lr: 1.0000e-05
Epoch 4/10
439/439 [==============================] - ETA: 0s - loss: 1.8086 - accuracy: 0.1707
Epoch 4: val_accuracy did not improve from 0.17500
439/439 [==============================] - 363s 826ms/step - loss: 1.8086 - accuracy: 0.1707 - val_loss: 1.7897 - val_accuracy: 0.1750 - lr: 1.0000e-05
Epoch 5/10
439/439 [==============================] - ETA: 0s - loss: 1.8069 - accuracy: 0.1677
Epoch 5: val_accuracy did not improve from 0.17500
439/439 [==============================] - 359s 817ms/step - loss: 1.8069 - accuracy: 0.1677 - val_loss: 1.7898 - val_accuracy: 0.1750 - lr: 1.0000e-05
Epoch 6/10
439/439 [==============================] - ETA: 0s - loss: 1.8052 - accuracy: 0.1763
Epoch 6: val_accuracy did not improve from 0.17500
439/439 [==============================] - 359s 818ms/step - loss: 1.8052 - accuracy: 0.1763 - val_loss: 1.7896 - val_accuracy: 0.1750 - lr: 1.0000e-05
Epoch 7/10
439/439 [==============================] - ETA: 0s - loss: 1.8064 - accuracy: 0.1692
Epoch 7: val_accuracy did not improve from 0.17500
439/439 [==============================] - 358s 816ms/step - loss: 1.8064 - accuracy: 0.1692 - val_loss: 1.7900 - val_accuracy: 0.1750 - lr: 1.0000e-05
Epoch 8/10
439/439 [==============================] - ETA: 0s - loss: 1.8075 - accuracy: 0.1700
Epoch 8: val_accuracy did not improve from 0.17500
439/439 [==============================] - 358s 816ms/step - loss: 1.8075 - accuracy: 0.1700 - val_loss: 1.7895 - val_accuracy: 0.1750 - lr: 1.0000e-05
Epoch 9/10
439/439 [==============================] - ETA: 0s - loss: 1.8052 - accuracy: 0.1715
Epoch 9: val_accuracy did not improve from 0.17500
439/439 [==============================] - 357s 813ms/step - loss: 1.8052 - accuracy: 0.1715 - val_loss: 1.7894 - val_accuracy: 0.1750 - lr: 1.0000e-05
Epoch 10/10
439/439 [==============================] - ETA: 0s - loss: 1.8068 - accuracy: 0.1672
Epoch 10: val_accuracy did not improve from 0.17500
439/439 [==============================] - 376s 858ms/step - loss: 1.8068 - accuracy: 0.1672 - val_loss: 1.7893 - val_accuracy: 0.1750 - lr: 1.0000e-05
```

In [19]:

```
base_model.trainable = True
for layer in base_model.layers[:-50]:
```

```
    layer.trainable = False
```

In [20]:
```
modelo4.compile(
    optimizer=Adam(learning_rate=1e-5),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [21]:
```
history4_fase2 = modelo4.fit(
    train_generator4,
    epochs=20,
    validation_data=val_generator4,
    callbacks=[early_stop, reduce_lr, checkpoint]
)
```

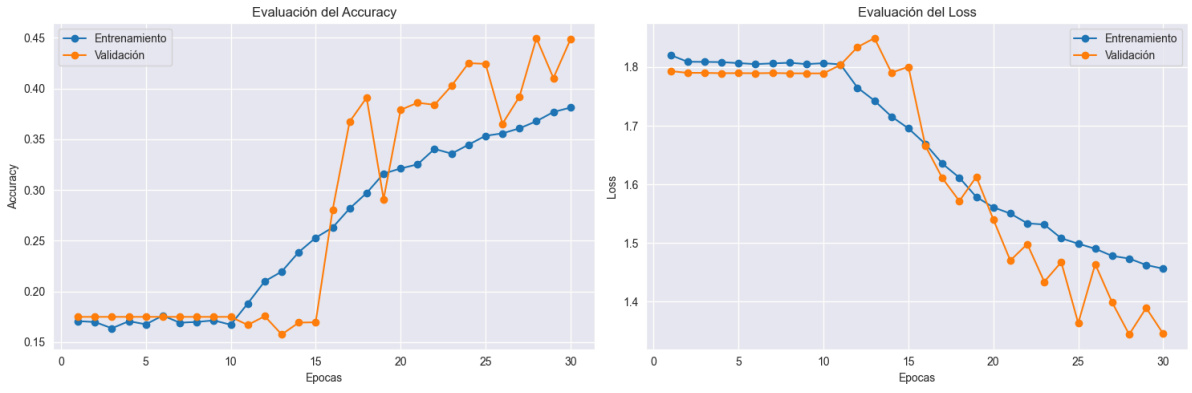
```
Epoch 1/20
439/439 [==============================] - ETA: 0s - loss: 1.8047 - accuracy: 0.18
82
Epoch 1: val_accuracy did not improve from 0.17500
439/439 [==============================] - 626s 1s/step - loss: 1.8047 - accuracy:
0.1882 - val_loss: 1.8038 - val_accuracy: 0.1670 - lr: 1.0000e-05
Epoch 2/20
439/439 [==============================] - ETA: 0s - loss: 1.7650 - accuracy: 0.21
01
Epoch 2: val_accuracy improved from 0.17500 to 0.17567, saving model to mejor_mode
lo4.h5
439/439 [==============================] - 445s 1s/step - loss: 1.7650 - accuracy:
0.2101 - val_loss: 1.8342 - val_accuracy: 0.1757 - lr: 1.0000e-05
Epoch 3/20
439/439 [==============================] - ETA: 0s - loss: 1.7429 - accuracy: 0.21
96
Epoch 3: val_accuracy did not improve from 0.17567
439/439 [==============================] - 440s 1s/step - loss: 1.7429 - accuracy:
0.2196 - val_loss: 1.8495 - val_accuracy: 0.1577 - lr: 1.0000e-05
Epoch 4/20
439/439 [==============================] - ETA: 0s - loss: 1.7158 - accuracy: 0.23
88
Epoch 4: val_accuracy did not improve from 0.17567
439/439 [==============================] - 442s 1s/step - loss: 1.7158 - accuracy:
0.2388 - val_loss: 1.7906 - val_accuracy: 0.1693 - lr: 1.0000e-05
Epoch 5/20
439/439 [==============================] - ETA: 0s - loss: 1.6955 - accuracy: 0.25
30
Epoch 5: val_accuracy did not improve from 0.17567
439/439 [==============================] - 443s 1s/step - loss: 1.6955 - accuracy:
0.2530 - val_loss: 1.8006 - val_accuracy: 0.1697 - lr: 1.0000e-05
Epoch 6/20
439/439 [==============================] - ETA: 0s - loss: 1.6689 - accuracy: 0.26
31
Epoch 6: val_accuracy improved from 0.17567 to 0.28067, saving model to mejor_mode
lo4.h5
439/439 [==============================] - 441s 1s/step - loss: 1.6689 - accuracy:
0.2631 - val_loss: 1.6656 - val_accuracy: 0.2807 - lr: 1.0000e-05
Epoch 7/20
439/439 [==============================] - ETA: 0s - loss: 1.6354 - accuracy: 0.28
18
Epoch 7: val_accuracy improved from 0.28067 to 0.36733, saving model to mejor_mode
lo4.h5
439/439 [==============================] - 453s 1s/step - loss: 1.6354 - accuracy:
0.2818 - val_loss: 1.6106 - val_accuracy: 0.3673 - lr: 1.0000e-05
Epoch 8/20
439/439 [==============================] - ETA: 0s - loss: 1.6115 - accuracy: 0.29
71
Epoch 8: val_accuracy improved from 0.36733 to 0.39100, saving model to mejor_mode
lo4.h5
439/439 [==============================] - 447s 1s/step - loss: 1.6115 - accuracy:
0.2971 - val_loss: 1.5716 - val_accuracy: 0.3910 - lr: 1.0000e-05
Epoch 9/20
439/439 [==============================] - ETA: 0s - loss: 1.5785 - accuracy: 0.31
62
Epoch 9: val_accuracy did not improve from 0.39100
439/439 [==============================] - 448s 1s/step - loss: 1.5785 - accuracy:
0.3162 - val_loss: 1.6130 - val_accuracy: 0.2910 - lr: 1.0000e-05
Epoch 10/20
439/439 [==============================] - ETA: 0s - loss: 1.5609 - accuracy: 0.32
12
Epoch 10: val_accuracy did not improve from 0.39100
439/439 [==============================] - 438s 999ms/step - loss: 1.5609 - accura
cy: 0.3212 - val_loss: 1.5402 - val_accuracy: 0.3790 - lr: 1.0000e-05
```

```
Epoch 11/20
439/439 [==============================] - ETA: 0s - loss: 1.5503 - accuracy: 0.32
53
Epoch 11: val_accuracy did not improve from 0.39100
439/439 [==============================] - 438s 999ms/step - loss: 1.5503 - accura
cy: 0.3253 - val_loss: 1.4702 - val_accuracy: 0.3860 - lr: 1.0000e-05
Epoch 12/20
439/439 [==============================] - ETA: 0s - loss: 1.5336 - accuracy: 0.34
05
Epoch 12: val_accuracy did not improve from 0.39100
439/439 [==============================] - 442s 1s/step - loss: 1.5336 - accuracy:
0.3405 - val_loss: 1.4983 - val_accuracy: 0.3840 - lr: 1.0000e-05
Epoch 13/20
439/439 [==============================] - ETA: 0s - loss: 1.5315 - accuracy: 0.33
58
Epoch 13: val_accuracy improved from 0.39100 to 0.40267, saving model to mejor_mod
elo4.h5
439/439 [==============================] - 437s 996ms/step - loss: 1.5315 - accura
cy: 0.3358 - val_loss: 1.4340 - val_accuracy: 0.4027 - lr: 1.0000e-05
Epoch 14/20
439/439 [==============================] - ETA: 0s - loss: 1.5081 - accuracy: 0.34
47
Epoch 14: val_accuracy improved from 0.40267 to 0.42500, saving model to mejor_mod
elo4.h5
439/439 [==============================] - 441s 1s/step - loss: 1.5081 - accuracy:
0.3447 - val_loss: 1.4674 - val_accuracy: 0.4250 - lr: 1.0000e-05
Epoch 15/20
439/439 [==============================] - ETA: 0s - loss: 1.4988 - accuracy: 0.35
34
Epoch 15: val_accuracy did not improve from 0.42500
439/439 [==============================] - 439s 999ms/step - loss: 1.4988 - accura
cy: 0.3534 - val_loss: 1.3635 - val_accuracy: 0.4243 - lr: 1.0000e-05
Epoch 16/20
439/439 [==============================] - ETA: 0s - loss: 1.4903 - accuracy: 0.35
59
Epoch 16: val_accuracy did not improve from 0.42500
439/439 [==============================] - 440s 1s/step - loss: 1.4903 - accuracy:
0.3559 - val_loss: 1.4640 - val_accuracy: 0.3653 - lr: 1.0000e-05
Epoch 17/20
439/439 [==============================] - ETA: 0s - loss: 1.4780 - accuracy: 0.36
08
Epoch 17: val_accuracy did not improve from 0.42500
439/439 [==============================] - 448s 1s/step - loss: 1.4780 - accuracy:
0.3608 - val_loss: 1.3985 - val_accuracy: 0.3920 - lr: 1.0000e-05
Epoch 18/20
439/439 [==============================] - ETA: 0s - loss: 1.4735 - accuracy: 0.36
78
Epoch 18: val_accuracy improved from 0.42500 to 0.44967, saving model to mejor_mod
elo4.h5
439/439 [==============================] - 473s 1s/step - loss: 1.4735 - accuracy:
0.3678 - val_loss: 1.3440 - val_accuracy: 0.4497 - lr: 1.0000e-05
Epoch 19/20
439/439 [==============================] - ETA: 0s - loss: 1.4625 - accuracy: 0.37
69
Epoch 19: val_accuracy did not improve from 0.44967
439/439 [==============================] - 481s 1s/step - loss: 1.4625 - accuracy:
0.3769 - val_loss: 1.3893 - val_accuracy: 0.4103 - lr: 1.0000e-05
Epoch 20/20
439/439 [==============================] - ETA: 0s - loss: 1.4562 - accuracy: 0.38
12
Epoch 20: val_accuracy did not improve from 0.44967
439/439 [==============================] - 476s 1s/step - loss: 1.4562 - accuracy:
0.3812 - val_loss: 1.3454 - val_accuracy: 0.4483 - lr: 1.0000e-05
```

In [22]:
```
loss4, acc4 = modelo4.evaluate(val_generator4, verbose=1)
print(f'Accuracy final: {acc4:.4f} | Loss final: {loss4:.4f}')
```

```
94/94 [==============================] - 54s 562ms/step - loss: 1.3454 - accuracy:
0.4483
Accuracy final: 0.4483 | Loss final: 1.3454
```

In [23]:
```
def curvas_entrenamiento(history1, history2):
    acc = history1.history['accuracy'] + history2.history['accuracy']
    val_acc = history1.history['val_accuracy'] + history2.history['val_accuracy']
    loss = history1.history['loss'] + history2.history['loss']
    val_loss = history1.history['val_loss'] + history2.history['val_loss']

    epochs = range(1, len(acc) + 1)

    plt.figure(figsize=(15, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, 'o-', label='Entrenamiento')
    plt.plot(epochs, val_acc, 'o-', label='Validación')
    plt.title('Evaluación del Accuracy')
    plt.xlabel('Epocas')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, 'o-', label='Entrenamiento')
    plt.plot(epochs, val_loss, 'o-', label='Validación')
    plt.title('Evaluación del Loss')
    plt.xlabel('Epocas')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()


curvas_entrenamiento(history4_fase1, history4_fase2)
```



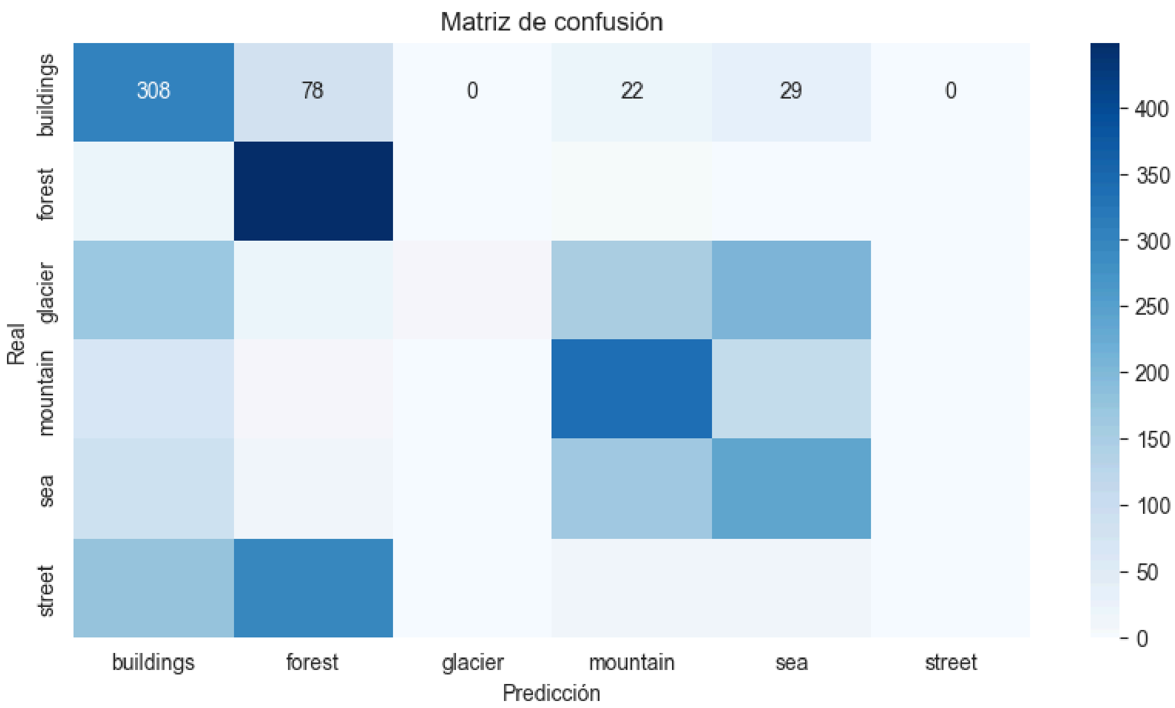
In [24]:
```
y_true = val_generator4.classes
y_pred = np.argmax(modelo4.predict(val_generator4), axis=1)
```

```
94/94 [==============================] - 55s 546ms/step
```

In [25]:
```
cm = confusion_matrix(y_true, y_pred)
labels = list(val_generator4.class_indices.keys())
```

In [26]:
```
plt.figure(figsize=(10, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=
plt.xlabel('Predicción')
```

```python
plt.ylabel('Real')
plt.title('Matriz de confusión')
plt.show()
```



In [28]:

```python
print('Reporte de clasificación')
print(classification_report(y_true, y_pred, target_names=labels))
```

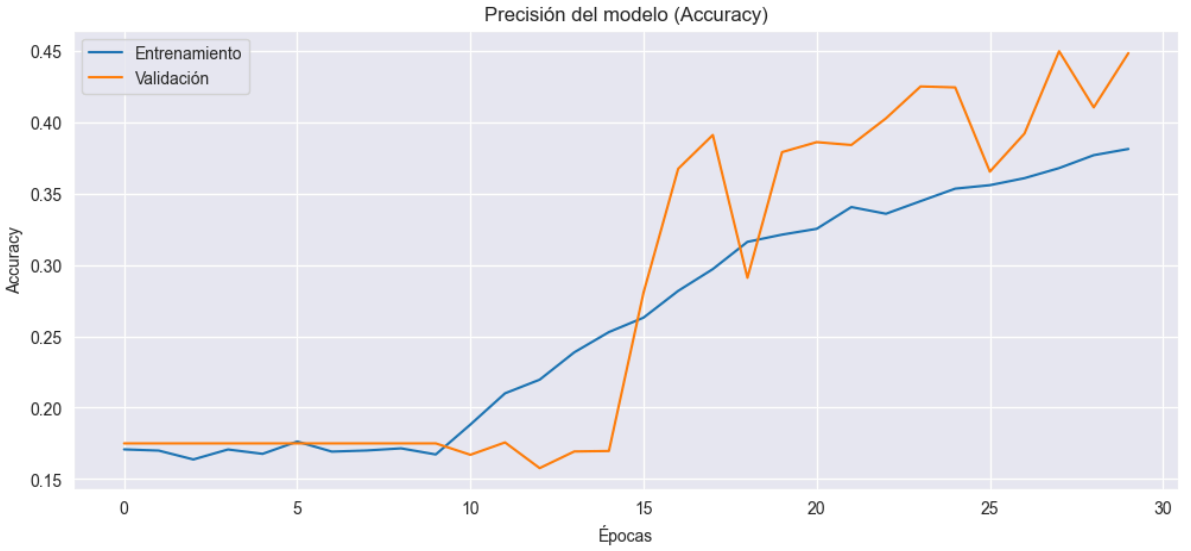
```
Reporte de clasificación
              precision    recall  f1-score   support

   buildings       0.37      0.70      0.49       437
      forest       0.52      0.95      0.67       474
     glacier       0.78      0.01      0.02       553
    mountain       0.49      0.65      0.56       525
         sea       0.40      0.47      0.44       510
      street       0.00      0.00      0.00       501

    accuracy                           0.45      3000
   macro avg       0.43      0.46      0.36      3000
weighted avg       0.43      0.45      0.35      3000
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:134
4: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:134
4: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:134
4: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

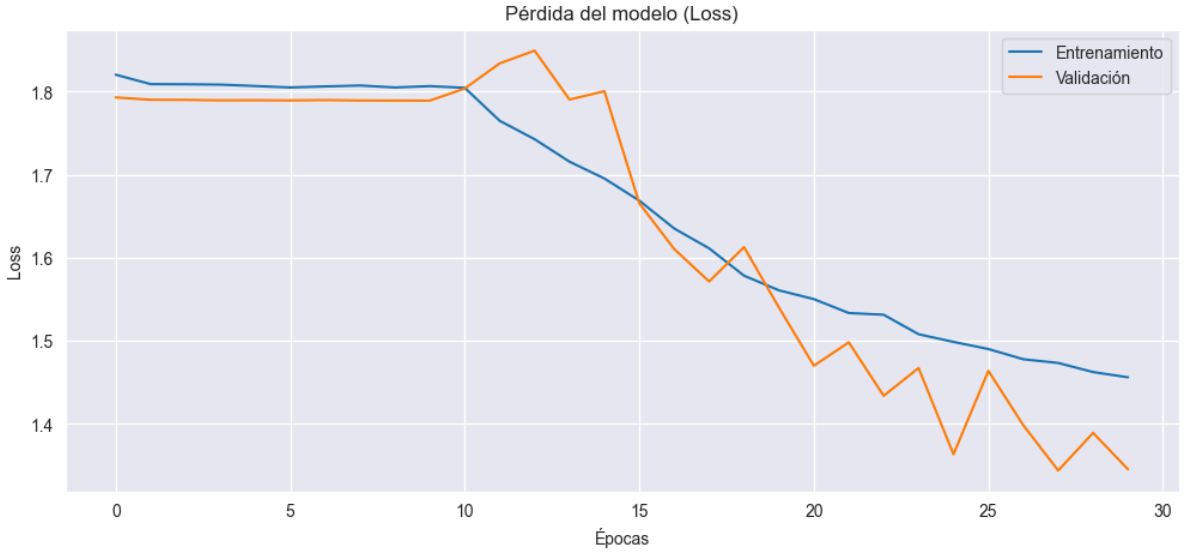
In [36]:

```python
plt.figure(figsize=(12, 5))
plt.plot(history4_fase1.history['accuracy'] + history4_fase2.history['accuracy'], l
plt.plot(history4_fase1.history['val_accuracy'] + history4_fase2.history['val_accur
plt.title("Precisión del modelo (Accuracy)")
```

```
plt.xlabel("Épocas")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



In [34]:
```
plt.figure(figsize=(12, 5))
plt.plot(history4_fase1.history['loss'] + history4_fase2.history['loss'], label='Er
plt.plot(history4_fase1.history['val_loss'] + history4_fase2.history['val_loss'], l
plt.title("Pérdida del modelo (Loss)")
plt.xlabel("Épocas")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



In [44]:
```
images_ = []
labels_ = []
for batch_img, batch_lbls in val_generator4:
    images_.append(batch_img)
    labels_.append(batch_lbls)
    if len(images_) * val_generator4.batch_size >= val_generator4.samples:
        break

images_ = np.vstack(images_)
labels_ = np.vstack(labels_)
```

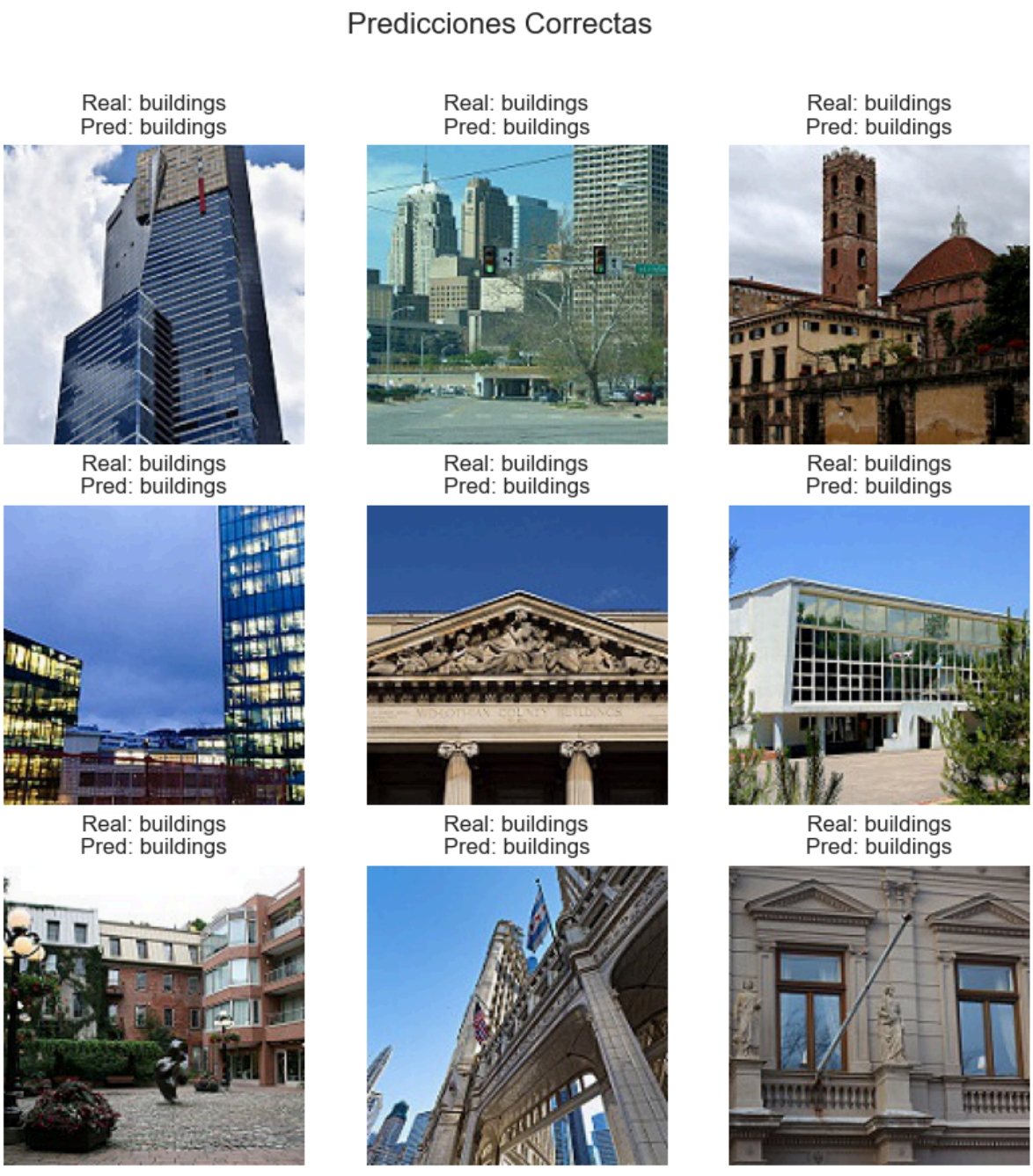
In [45]:
```
pred = modelo4.predict(images_)
true_classes = val_generator4.classes
```

```
94/94 [==============================] - 50s 531ms/step
```

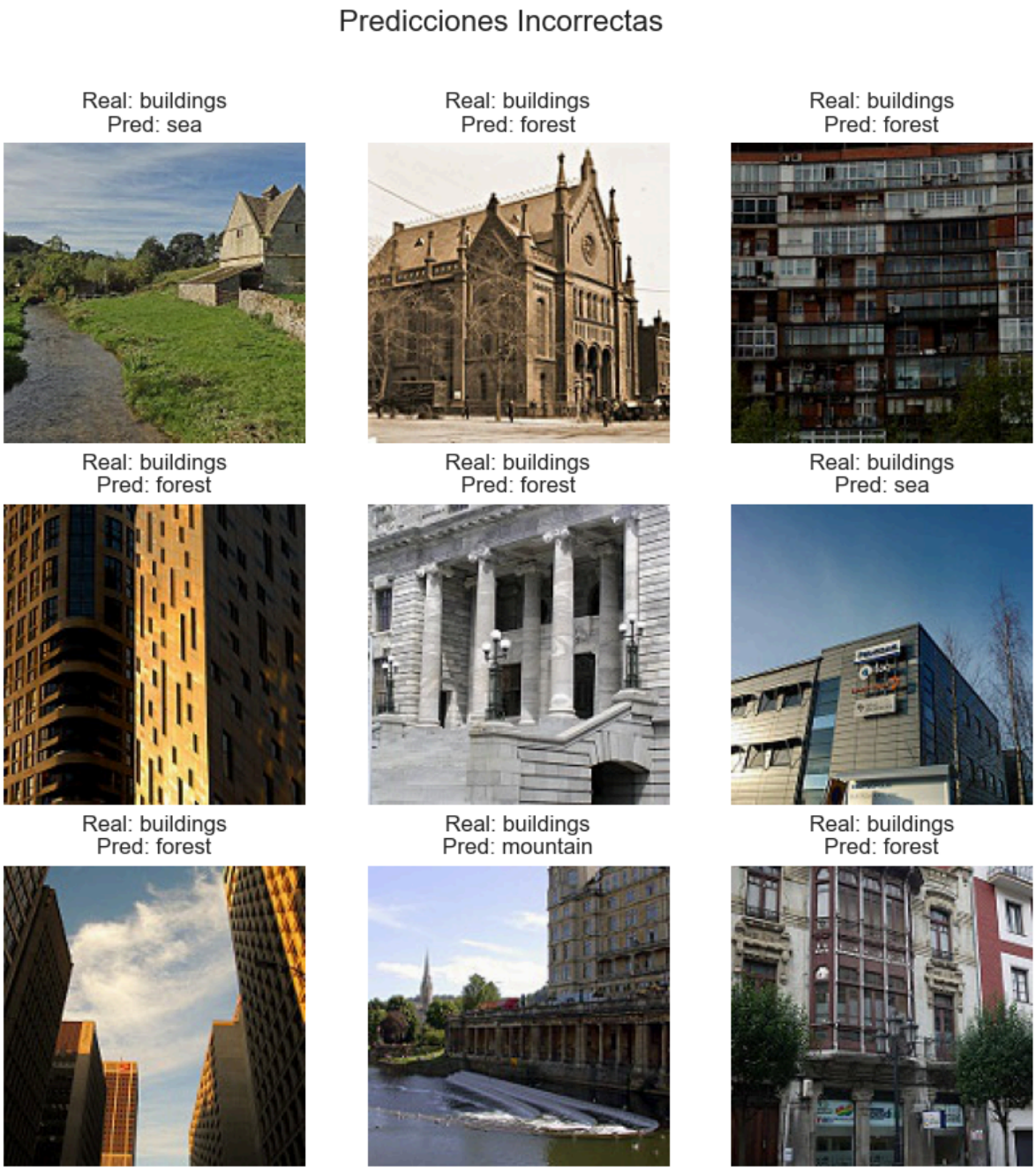
In [38]:
```
correctos = np.where(y_pred == true_classes)[0]
incorrectos = np.where(y_pred != true_classes)[0]
clases = list(train_generator4.class_indices.keys())
```

In [48]:
```
def resultados(indices, title, n=9):
    plt.figure(figsize=(10, 10))
    plt.suptitle(title, fontsize=16)
    for i, idx in enumerate(indices[:n]):
        plt.subplot(3, 3, i + 1)
        img_path = val_generator4.filepaths[idx]
        img = plt.imread(img_path)
        plt.imshow(img)
        plt.title(f"Real: {clases[true_classes[idx]]}\nPred: {clases[y_pred[idx]]}'
        plt.axis("off")
    plt.show()
```

In [51]:
```
resultados(correctos, "Predicciones Correctas", n=9)
```



In [50]:
```
resultados(incorrectos, "Predicciones Incorrectas", n=9)
```



Entrenamiento con ResNEt50

In [6]:
```
data_trainRS = 'G:/Intel/seg_train'
data_testRS = 'G:/Intel/seg_test'
data_pathRS = 'G:/Intel/seg_train'
```

In [8]:
```
img_size = (224, 224)
batch_size = 32
epochs = 15
#learnig_rate = 1e-4
lr = 1e-4
```

In [9]:
```
train_datagenRS = ImageDataGenerator(
    rescale=1./255,
    rotation_range=90,
    width_shift_range=0.2,
    height_shift_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
```

```
    fill_mode='nearest'
)
```

In [10]:
```
val_datagenRS = ImageDataGenerator(
    rescale=1./255,
)
```

In [11]:
```
train_generatorRS = train_datagenRS.flow_from_directory(
    data_trainRS,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',

)
```

```
Found 14034 images belonging to 6 classes.
```

In [13]:
```
val_generatorRS = val_datagenRS.flow_from_directory(
    data_testRS,
    target_size = img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)
```

```
Found 3000 images belonging to 6 classes.
```

In [14]:
```
base_model = ResNet50(
    weights='imagenet',
    include_top=False,
    input_shape=(img_size[0], img_size[1], 3)
)
```

```
WARNING:tensorflow:From C:\Users\Lalo\AppData\Roaming\Python\Python311\site-packag
es\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_functions is d
eprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\Lalo\AppData\Roaming\Python\Python311\site-packag
es\keras\src\layers\normalization\batch_normalization.py:979: The name tf.nn.fused
_batch_norm is deprecated. Please use tf.compat.v1.nn.fused_batch_norm instead.

Downloading data from https://storage.googleapis.com/tensorflow/keras-application
s/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 8s 0us/step
```

In [15]:
```
for layer in base_model.layers:
    layer.trainable = False
```

In [16]:
```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dropout(0.2)(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.2)(x)
```

In [17]:
```
predictios = Dense(train_generatorRS.num_classes, activation='softmax')(x)
```

In [18]:
```
modelo_resnet = Model(
    inputs=base_model.input,
    outputs=predictios
)
```

In [19]:
```
modelo_resnet.compile(
    optimizer=Adam(learning_rate=lr),
```

```
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

In [21]:
```
modelo_resnet.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         []

 conv1_pad (ZeroPadding2D)   (None, 230, 230, 3)       0         ['input_1[0][0]']

 conv1_conv (Conv2D)         (None, 112, 112, 64)      9472      ['conv1_pad[0][0]']

 conv1_bn (BatchNormalizati  (None, 112, 112, 64)      256       ['conv1_conv[0][0]']
 on)

 conv1_relu (Activation)     (None, 112, 112, 64)      0         ['conv1_bn[0][0]']

 pool1_pad (ZeroPadding2D)   (None, 114, 114, 64)      0         ['conv1_relu[0][0]']

 pool1_pool (MaxPooling2D)   (None, 56, 56, 64)        0         ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2  (None, 56, 56, 64)        4160      ['pool1_pool[0][0]']
 D)

 conv2_block1_1_bn (BatchNo  (None, 56, 56, 64)        256       ['conv2_block1_1_conv[0][0]']
 rmalization)

 conv2_block1_1_relu (Activ  (None, 56, 56, 64)        0         ['conv2_block1_1_bn[0][0]']
 ation)

 conv2_block1_2_conv (Conv2  (None, 56, 56, 64)        36928     ['conv2_block1_1_relu[0][0]']
 D)

 conv2_block1_2_bn (BatchNo  (None, 56, 56, 64)        256       ['conv2_block1_2_conv[0][0]']
 rmalization)

 conv2_block1_2_relu (Activ  (None, 56, 56, 64)        0         ['conv2_block1_2_bn[0][0]']
 ation)

 conv2_block1_0_conv (Conv2  (None, 56, 56, 256)       16640     ['pool1_pool[0][0]']
 D)

 conv2_block1_3_conv (Conv2  (None, 56, 56, 256)       16640     ['conv2_block1_2_relu[0][0]']
 D)

 conv2_block1_0_bn (BatchNo  (None, 56, 56, 256)       1024      ['conv2_block1_0_conv[0][0]']
 rmalization)

 conv2_block1_3_bn (BatchNo  (None, 56, 56, 256)       1024      ['conv2_block1
```

```
_3_conv[0][0]']
 rmalization)

 conv2_block1_add (Add)      (None, 56, 56, 256)          0         ['conv2_block1
_0_bn[0][0]',
                                                                     'conv2_block1
_3_bn[0][0]']

 conv2_block1_out (Activati  (None, 56, 56, 256)          0         ['conv2_block1
_add[0][0]']
 on)

 conv2_block2_1_conv (Conv2  (None, 56, 56, 64)           16448     ['conv2_block1
_out[0][0]']
 D)

 conv2_block2_1_bn (BatchNo  (None, 56, 56, 64)           256       ['conv2_block2
_1_conv[0][0]']
 rmalization)

 conv2_block2_1_relu (Activ  (None, 56, 56, 64)           0         ['conv2_block2
_1_bn[0][0]']
 ation)

 conv2_block2_2_conv (Conv2  (None, 56, 56, 64)           36928     ['conv2_block2
_1_relu[0][0]']
 D)

 conv2_block2_2_bn (BatchNo  (None, 56, 56, 64)           256       ['conv2_block2
_2_conv[0][0]']
 rmalization)

 conv2_block2_2_relu (Activ  (None, 56, 56, 64)           0         ['conv2_block2
_2_bn[0][0]']
 ation)

 conv2_block2_3_conv (Conv2  (None, 56, 56, 256)          16640     ['conv2_block2
_2_relu[0][0]']
 D)

 conv2_block2_3_bn (BatchNo  (None, 56, 56, 256)          1024      ['conv2_block2
_3_conv[0][0]']
 rmalization)

 conv2_block2_add (Add)      (None, 56, 56, 256)          0         ['conv2_block1
_out[0][0]',
                                                                     'conv2_block2
_3_bn[0][0]']

 conv2_block2_out (Activati  (None, 56, 56, 256)          0         ['conv2_block2
_add[0][0]']
 on)

 conv2_block3_1_conv (Conv2  (None, 56, 56, 64)           16448     ['conv2_block2
_out[0][0]']
 D)

 conv2_block3_1_bn (BatchNo  (None, 56, 56, 64)           256       ['conv2_block3
_1_conv[0][0]']
 rmalization)

 conv2_block3_1_relu (Activ  (None, 56, 56, 64)           0         ['conv2_block3
_1_bn[0][0]']
 ation)
```

```
 conv2_block3_2_conv (Conv2  (None, 56, 56, 64)          36928     ['conv2_block3
_1_relu[0][0]']
 D)

 conv2_block3_2_bn (BatchNo  (None, 56, 56, 64)          256       ['conv2_block3
_2_conv[0][0]']
 rmalization)

 conv2_block3_2_relu (Activ  (None, 56, 56, 64)          0         ['conv2_block3
_2_bn[0][0]']
 ation)

 conv2_block3_3_conv (Conv2  (None, 56, 56, 256)         16640     ['conv2_block3
_2_relu[0][0]']
 D)

 conv2_block3_3_bn (BatchNo  (None, 56, 56, 256)         1024      ['conv2_block3
_3_conv[0][0]']
 rmalization)

 conv2_block3_add (Add)      (None, 56, 56, 256)         0         ['conv2_block2
_out[0][0]',
                                                                     'conv2_block3
_3_bn[0][0]']

 conv2_block3_out (Activati  (None, 56, 56, 256)         0         ['conv2_block3
_add[0][0]']
 on)

 conv3_block1_1_conv (Conv2  (None, 28, 28, 128)         32896     ['conv2_block3
_out[0][0]']
 D)

 conv3_block1_1_bn (BatchNo  (None, 28, 28, 128)         512       ['conv3_block1
_1_conv[0][0]']
 rmalization)

 conv3_block1_1_relu (Activ  (None, 28, 28, 128)         0         ['conv3_block1
_1_bn[0][0]']
 ation)

 conv3_block1_2_conv (Conv2  (None, 28, 28, 128)         147584    ['conv3_block1
_1_relu[0][0]']
 D)

 conv3_block1_2_bn (BatchNo  (None, 28, 28, 128)         512       ['conv3_block1
_2_conv[0][0]']
 rmalization)

 conv3_block1_2_relu (Activ  (None, 28, 28, 128)         0         ['conv3_block1
_2_bn[0][0]']
 ation)

 conv3_block1_0_conv (Conv2  (None, 28, 28, 512)         131584    ['conv2_block3
_out[0][0]']
 D)

 conv3_block1_3_conv (Conv2  (None, 28, 28, 512)         66048     ['conv3_block1
_2_relu[0][0]']
 D)

 conv3_block1_0_bn (BatchNo  (None, 28, 28, 512)         2048      ['conv3_block1
_0_conv[0][0]']
```

```
 rmalization)

 conv3_block1_3_bn (BatchNo  (None, 28, 28, 512)          2048      ['conv3_block1
_3_conv[0][0]']
 rmalization)

 conv3_block1_add (Add)      (None, 28, 28, 512)          0         ['conv3_block1
_0_bn[0][0]',
                                                                     'conv3_block1
_3_bn[0][0]']

 conv3_block1_out (Activati  (None, 28, 28, 512)          0         ['conv3_block1
_add[0][0]']
 on)

 conv3_block2_1_conv (Conv2  (None, 28, 28, 128)          65664     ['conv3_block1
_out[0][0]']
 D)

 conv3_block2_1_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block2
_1_conv[0][0]']
 rmalization)

 conv3_block2_1_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block2
_1_bn[0][0]']
 ation)

 conv3_block2_2_conv (Conv2  (None, 28, 28, 128)          147584    ['conv3_block2
_1_relu[0][0]']
 D)

 conv3_block2_2_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block2
_2_conv[0][0]']
 rmalization)

 conv3_block2_2_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block2
_2_bn[0][0]']
 ation)

 conv3_block2_3_conv (Conv2  (None, 28, 28, 512)          66048     ['conv3_block2
_2_relu[0][0]']
 D)

 conv3_block2_3_bn (BatchNo  (None, 28, 28, 512)          2048      ['conv3_block2
_3_conv[0][0]']
 rmalization)

 conv3_block2_add (Add)      (None, 28, 28, 512)          0         ['conv3_block1
_out[0][0]',
                                                                     'conv3_block2
_3_bn[0][0]']

 conv3_block2_out (Activati  (None, 28, 28, 512)          0         ['conv3_block2
_add[0][0]']
 on)

 conv3_block3_1_conv (Conv2  (None, 28, 28, 128)          65664     ['conv3_block2
_out[0][0]']
 D)

 conv3_block3_1_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block3
_1_conv[0][0]']
 rmalization)
```

```
 conv3_block3_1_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block3
_1_bn[0][0]']
 ation)

 conv3_block3_2_conv (Conv2  (None, 28, 28, 128)          147584    ['conv3_block3
_1_relu[0][0]']
 D)

 conv3_block3_2_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block3
_2_conv[0][0]']
 rmalization)

 conv3_block3_2_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block3
_2_bn[0][0]']
 ation)

 conv3_block3_3_conv (Conv2  (None, 28, 28, 512)          66048     ['conv3_block3
_2_relu[0][0]']
 D)

 conv3_block3_3_bn (BatchNo  (None, 28, 28, 512)          2048      ['conv3_block3
_3_conv[0][0]']
 rmalization)

 conv3_block3_add (Add)      (None, 28, 28, 512)          0         ['conv3_block2
_out[0][0]',
                                                                     'conv3_block3
_3_bn[0][0]']

 conv3_block3_out (Activati  (None, 28, 28, 512)          0         ['conv3_block3
_add[0][0]']
 on)

 conv3_block4_1_conv (Conv2  (None, 28, 28, 128)          65664     ['conv3_block3
_out[0][0]']
 D)

 conv3_block4_1_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block4
_1_conv[0][0]']
 rmalization)

 conv3_block4_1_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block4
_1_bn[0][0]']
 ation)

 conv3_block4_2_conv (Conv2  (None, 28, 28, 128)          147584    ['conv3_block4
_1_relu[0][0]']
 D)

 conv3_block4_2_bn (BatchNo  (None, 28, 28, 128)          512       ['conv3_block4
_2_conv[0][0]']
 rmalization)

 conv3_block4_2_relu (Activ  (None, 28, 28, 128)          0         ['conv3_block4
_2_bn[0][0]']
 ation)

 conv3_block4_3_conv (Conv2  (None, 28, 28, 512)          66048     ['conv3_block4
_2_relu[0][0]']
 D)

 conv3_block4_3_bn (BatchNo  (None, 28, 28, 512)          2048      ['conv3_block4
_3_conv[0][0]']
 rmalization)
```

```
 conv3_block4_add (Add)      (None, 28, 28, 512)          0         ['conv3_block3
_out[0][0]',
                                                                     'conv3_block4
_3_bn[0][0]']

 conv3_block4_out (Activati  (None, 28, 28, 512)          0         ['conv3_block4
_add[0][0]']
 on)

 conv4_block1_1_conv (Conv2  (None, 14, 14, 256)          131328    ['conv3_block4
_out[0][0]']
 D)

 conv4_block1_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block1
_1_conv[0][0]']
 rmalization)

 conv4_block1_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block1
_1_bn[0][0]']
 ation)

 conv4_block1_2_conv (Conv2  (None, 14, 14, 256)          590080    ['conv4_block1
_1_relu[0][0]']
 D)

 conv4_block1_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block1
_2_conv[0][0]']
 rmalization)

 conv4_block1_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block1
_2_bn[0][0]']
 ation)

 conv4_block1_0_conv (Conv2  (None, 14, 14, 1024)         525312    ['conv3_block4
_out[0][0]']
 D)

 conv4_block1_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block1
_2_relu[0][0]']
 D)

 conv4_block1_0_bn (BatchNo  (None, 14, 14, 1024)         4096      ['conv4_block1
_0_conv[0][0]']
 rmalization)

 conv4_block1_3_bn (BatchNo  (None, 14, 14, 1024)         4096      ['conv4_block1
_3_conv[0][0]']
 rmalization)

 conv4_block1_add (Add)      (None, 14, 14, 1024)         0         ['conv4_block1
_0_bn[0][0]',
                                                                     'conv4_block1
_3_bn[0][0]']

 conv4_block1_out (Activati  (None, 14, 14, 1024)         0         ['conv4_block1
_add[0][0]']
 on)

 conv4_block2_1_conv (Conv2  (None, 14, 14, 256)          262400    ['conv4_block1
_out[0][0]']
 D)

 conv4_block2_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block2
```

```
_1_conv[0][0]']
 rmalization)

 conv4_block2_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block2
_1_bn[0][0]']
 ation)

 conv4_block2_2_conv (Conv2  (None, 14, 14, 256)          590080    ['conv4_block2
_1_relu[0][0]']
 D)

 conv4_block2_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block2
_2_conv[0][0]']
 rmalization)

 conv4_block2_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block2
_2_bn[0][0]']
 ation)

 conv4_block2_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block2
_2_relu[0][0]']
 D)

 conv4_block2_3_bn (BatchNo  (None, 14, 14, 1024)         4096      ['conv4_block2
_3_conv[0][0]']
 rmalization)

 conv4_block2_add (Add)      (None, 14, 14, 1024)         0         ['conv4_block1
_out[0][0]',
                                                                     'conv4_block2
_3_bn[0][0]']

 conv4_block2_out (Activati  (None, 14, 14, 1024)         0         ['conv4_block2
_add[0][0]']
 on)

 conv4_block3_1_conv (Conv2  (None, 14, 14, 256)          262400    ['conv4_block2
_out[0][0]']
 D)

 conv4_block3_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block3
_1_conv[0][0]']
 rmalization)

 conv4_block3_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block3
_1_bn[0][0]']
 ation)

 conv4_block3_2_conv (Conv2  (None, 14, 14, 256)          590080    ['conv4_block3
_1_relu[0][0]']
 D)

 conv4_block3_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block3
_2_conv[0][0]']
 rmalization)

 conv4_block3_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block3
_2_bn[0][0]']
 ation)

 conv4_block3_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block3
_2_relu[0][0]']
 D)
```

```
 conv4_block3_3_bn (BatchNo  (None, 14, 14, 1024)         4096      ['conv4_block3
_3_conv[0][0]']
 rmalization)

 conv4_block3_add (Add)      (None, 14, 14, 1024)         0         ['conv4_block2
_out[0][0]',
                                                                     'conv4_block3
_3_bn[0][0]']

 conv4_block3_out (Activati  (None, 14, 14, 1024)         0         ['conv4_block3
_add[0][0]']
 on)

 conv4_block4_1_conv (Conv2  (None, 14, 14, 256)          262400    ['conv4_block3
_out[0][0]']
 D)

 conv4_block4_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block4
_1_conv[0][0]']
 rmalization)

 conv4_block4_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block4
_1_bn[0][0]']
 ation)

 conv4_block4_2_conv (Conv2  (None, 14, 14, 256)          590080    ['conv4_block4
_1_relu[0][0]']
 D)

 conv4_block4_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block4
_2_conv[0][0]']
 rmalization)

 conv4_block4_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block4
_2_bn[0][0]']
 ation)

 conv4_block4_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block4
_2_relu[0][0]']
 D)

 conv4_block4_3_bn (BatchNo  (None, 14, 14, 1024)         4096      ['conv4_block4
_3_conv[0][0]']
 rmalization)

 conv4_block4_add (Add)      (None, 14, 14, 1024)         0         ['conv4_block3
_out[0][0]',
                                                                     'conv4_block4
_3_bn[0][0]']

 conv4_block4_out (Activati  (None, 14, 14, 1024)         0         ['conv4_block4
_add[0][0]']
 on)

 conv4_block5_1_conv (Conv2  (None, 14, 14, 256)          262400    ['conv4_block4
_out[0][0]']
 D)

 conv4_block5_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block5
_1_conv[0][0]']
 rmalization)

 conv4_block5_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block5
_1_bn[0][0]']
```

```
 ation)

 conv4_block5_2_conv (Conv2  (None, 14, 14, 256)          590080    ['conv4_block5
_1_relu[0][0]']
 D)

 conv4_block5_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block5
_2_conv[0][0]']
 rmalization)

 conv4_block5_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block5
_2_bn[0][0]']
 ation)

 conv4_block5_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block5
_2_relu[0][0]']
 D)

 conv4_block5_3_bn (BatchNo  (None, 14, 14, 1024)         4096      ['conv4_block5
_3_conv[0][0]']
 rmalization)

 conv4_block5_add (Add)      (None, 14, 14, 1024)         0         ['conv4_block4
_out[0][0]',
                                                                     'conv4_block5
_3_bn[0][0]']

 conv4_block5_out (Activati  (None, 14, 14, 1024)         0         ['conv4_block5
_add[0][0]']
 on)

 conv4_block6_1_conv (Conv2  (None, 14, 14, 256)          262400    ['conv4_block5
_out[0][0]']
 D)

 conv4_block6_1_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block6
_1_conv[0][0]']
 rmalization)

 conv4_block6_1_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block6
_1_bn[0][0]']
 ation)

 conv4_block6_2_conv (Conv2  (None, 14, 14, 256)          590080    ['conv4_block6
_1_relu[0][0]']
 D)

 conv4_block6_2_bn (BatchNo  (None, 14, 14, 256)          1024      ['conv4_block6
_2_conv[0][0]']
 rmalization)

 conv4_block6_2_relu (Activ  (None, 14, 14, 256)          0         ['conv4_block6
_2_bn[0][0]']
 ation)

 conv4_block6_3_conv (Conv2  (None, 14, 14, 1024)         263168    ['conv4_block6
_2_relu[0][0]']
 D)

 conv4_block6_3_bn (BatchNo  (None, 14, 14, 1024)         4096      ['conv4_block6
_3_conv[0][0]']
 rmalization)

 conv4_block6_add (Add)      (None, 14, 14, 1024)         0         ['conv4_block5
```

```
_out[0][0]',
                                                                 'conv4_block6
_3_bn[0][0]']

 conv4_block6_out (Activati  (None, 14, 14, 1024)         0         ['conv4_block6
_add[0][0]']
 on)

 conv5_block1_1_conv (Conv2  (None, 7, 7, 512)            524800    ['conv4_block6
_out[0][0]']
 D)

 conv5_block1_1_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block1
_1_conv[0][0]']
 rmalization)

 conv5_block1_1_relu (Activ  (None, 7, 7, 512)            0         ['conv5_block1
_1_bn[0][0]']
 ation)

 conv5_block1_2_conv (Conv2  (None, 7, 7, 512)            2359808   ['conv5_block1
_1_relu[0][0]']
 D)

 conv5_block1_2_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block1
_2_conv[0][0]']
 rmalization)

 conv5_block1_2_relu (Activ  (None, 7, 7, 512)            0         ['conv5_block1
_2_bn[0][0]']
 ation)

 conv5_block1_0_conv (Conv2  (None, 7, 7, 2048)           2099200   ['conv4_block6
_out[0][0]']
 D)

 conv5_block1_3_conv (Conv2  (None, 7, 7, 2048)           1050624   ['conv5_block1
_2_relu[0][0]']
 D)

 conv5_block1_0_bn (BatchNo  (None, 7, 7, 2048)           8192      ['conv5_block1
_0_conv[0][0]']
 rmalization)

 conv5_block1_3_bn (BatchNo  (None, 7, 7, 2048)           8192      ['conv5_block1
_3_conv[0][0]']
 rmalization)

 conv5_block1_add (Add)      (None, 7, 7, 2048)           0         ['conv5_block1
_0_bn[0][0]',
                                                                  'conv5_block1
_3_bn[0][0]']

 conv5_block1_out (Activati  (None, 7, 7, 2048)           0         ['conv5_block1
_add[0][0]']
 on)

 conv5_block2_1_conv (Conv2  (None, 7, 7, 512)            1049088   ['conv5_block1
_out[0][0]']
 D)

 conv5_block2_1_bn (BatchNo  (None, 7, 7, 512)            2048      ['conv5_block2
_1_conv[0][0]']
 rmalization)
```

```
 conv5_block2_1_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block2
_1_bn[0][0]']
 ation)

 conv5_block2_2_conv (Conv2  (None, 7, 7, 512)           2359808   ['conv5_block2
_1_relu[0][0]']
 D)

 conv5_block2_2_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block2
_2_conv[0][0]']
 rmalization)

 conv5_block2_2_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block2
_2_bn[0][0]']
 ation)

 conv5_block2_3_conv (Conv2  (None, 7, 7, 2048)          1050624   ['conv5_block2
_2_relu[0][0]']
 D)

 conv5_block2_3_bn (BatchNo  (None, 7, 7, 2048)          8192      ['conv5_block2
_3_conv[0][0]']
 rmalization)

 conv5_block2_add (Add)      (None, 7, 7, 2048)          0         ['conv5_block1
_out[0][0]',
                                                                     'conv5_block2
_3_bn[0][0]']

 conv5_block2_out (Activati  (None, 7, 7, 2048)          0         ['conv5_block2
_add[0][0]']
 on)

 conv5_block3_1_conv (Conv2  (None, 7, 7, 512)           1049088   ['conv5_block2
_out[0][0]']
 D)

 conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block3
_1_conv[0][0]']
 rmalization)

 conv5_block3_1_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block3
_1_bn[0][0]']
 ation)

 conv5_block3_2_conv (Conv2  (None, 7, 7, 512)           2359808   ['conv5_block3
_1_relu[0][0]']
 D)

 conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)           2048      ['conv5_block3
_2_conv[0][0]']
 rmalization)

 conv5_block3_2_relu (Activ  (None, 7, 7, 512)           0         ['conv5_block3
_2_bn[0][0]']
 ation)

 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)          1050624   ['conv5_block3
_2_relu[0][0]']
 D)

 conv5_block3_3_bn (BatchNo  (None, 7, 7, 2048)          8192      ['conv5_block3
_3_conv[0][0]']
```

```
 rmalization)

 conv5_block3_add (Add)      (None, 7, 7, 2048)           0         ['conv5_block2
_out[0][0]',
                                                                     'conv5_block3
_3_bn[0][0]']

 conv5_block3_out (Activati  (None, 7, 7, 2048)           0         ['conv5_block3
_add[0][0]']
 on)

 global_average_pooling2d (  (None, 2048)                 0         ['conv5_block3
_out[0][0]']
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 2048)                 0         ['global_avera
ge_pooling2d[0][
                                                                    0]']

 dense (Dense)               (None, 512)                  1049088   ['dropout[0]
[0]']

 dropout_1 (Dropout)         (None, 512)                  0         ['dense[0]
[0]']

 dense_1 (Dense)             (None, 6)                    3078      ['dropout_1[0]
[0]']

==================================================================================
================
Total params: 24639878 (93.99 MB)
Trainable params: 1052166 (4.01 MB)
Non-trainable params: 23587712 (89.98 MB)
__________________________________________________________________________________
________________
```

In [22]:
```
early_stop = EarlyStopping(monitor="val_loss", patience=4, restore_best_weights=Tru
reduce_lr = ReduceLROnPlateau(monitor="val_loss", factor=0.3, patience=2, verbose=1
checkpoint = ModelCheckpoint("mejor_modelo_resnet.h5", monitor="val_accuracy", save
```

In [24]:
```
history_resnet = modelo_resnet.fit(
    train_generatorRS,
    epochs=epochs,
    validation_data = val_generatorRS,
    callbacks=[early_stop, reduce_lr, checkpoint]
)
```

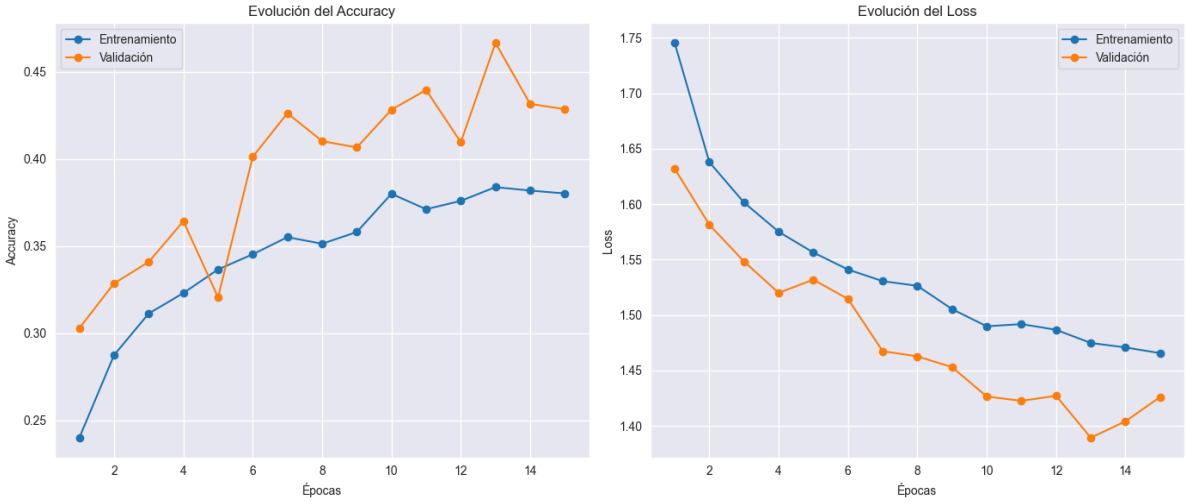
```
Epoch 1/15
439/439 [==============================] - ETA: 0s - loss: 1.7454 - accuracy: 0.24
01
C:\Users\Lalo\AppData\Roaming\Python\Python311\site-packages\keras\src\engine\trai
ning.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.sa
ve()`. This file format is considered legacy. We recommend using instead the nativ
e Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
439/439 [==============================] - 1930s 4s/step - loss: 1.7454 - accurac
y: 0.2401 - val_loss: 1.6320 - val_accuracy: 0.3030 - lr: 1.0000e-04
Epoch 2/15
439/439 [==============================] - 1869s 4s/step - loss: 1.6378 - accurac
y: 0.2875 - val_loss: 1.5814 - val_accuracy: 0.3287 - lr: 1.0000e-04
Epoch 3/15
439/439 [==============================] - 1848s 4s/step - loss: 1.6014 - accurac
y: 0.3114 - val_loss: 1.5482 - val_accuracy: 0.3410 - lr: 1.0000e-04
Epoch 4/15
439/439 [==============================] - 1882s 4s/step - loss: 1.5750 - accurac
y: 0.3231 - val_loss: 1.5199 - val_accuracy: 0.3643 - lr: 1.0000e-04
Epoch 5/15
439/439 [==============================] - 1952s 4s/step - loss: 1.5563 - accurac
y: 0.3367 - val_loss: 1.5318 - val_accuracy: 0.3207 - lr: 1.0000e-04
Epoch 6/15
439/439 [==============================] - 2455s 6s/step - loss: 1.5410 - accurac
y: 0.3454 - val_loss: 1.5144 - val_accuracy: 0.4013 - lr: 1.0000e-04
Epoch 7/15
439/439 [==============================] - 2406s 5s/step - loss: 1.5305 - accurac
y: 0.3552 - val_loss: 1.4673 - val_accuracy: 0.4263 - lr: 1.0000e-04
Epoch 8/15
439/439 [==============================] - 2369s 5s/step - loss: 1.5263 - accurac
y: 0.3514 - val_loss: 1.4627 - val_accuracy: 0.4103 - lr: 1.0000e-04
Epoch 9/15
439/439 [==============================] - 2380s 5s/step - loss: 1.5053 - accurac
y: 0.3581 - val_loss: 1.4529 - val_accuracy: 0.4067 - lr: 1.0000e-04
Epoch 10/15
439/439 [==============================] - 2373s 5s/step - loss: 1.4898 - accurac
y: 0.3801 - val_loss: 1.4265 - val_accuracy: 0.4283 - lr: 1.0000e-04
Epoch 11/15
439/439 [==============================] - 2391s 5s/step - loss: 1.4918 - accurac
y: 0.3712 - val_loss: 1.4226 - val_accuracy: 0.4397 - lr: 1.0000e-04
Epoch 12/15
439/439 [==============================] - 2356s 5s/step - loss: 1.4866 - accurac
y: 0.3760 - val_loss: 1.4271 - val_accuracy: 0.4097 - lr: 1.0000e-04
Epoch 13/15
439/439 [==============================] - 2351s 5s/step - loss: 1.4748 - accurac
y: 0.3839 - val_loss: 1.3893 - val_accuracy: 0.4667 - lr: 1.0000e-04
Epoch 14/15
439/439 [==============================] - 2357s 5s/step - loss: 1.4707 - accurac
y: 0.3819 - val_loss: 1.4040 - val_accuracy: 0.4317 - lr: 1.0000e-04
Epoch 15/15
439/439 [==============================] - ETA: 0s - loss: 1.4656 - accuracy: 0.38
03
Epoch 15: ReduceLROnPlateau reducing learning rate to 2.9999999242136255e-05.
439/439 [==============================] - 2337s 5s/step - loss: 1.4656 - accurac
y: 0.3803 - val_loss: 1.4260 - val_accuracy: 0.4287 - lr: 1.0000e-04
```

In [26]:
```
history_dict = history_resnet.history
epochs_range = range(1, len(history_dict['accuracy']) + 1)
```

In [28]:
```
plt.figure(figsize=(14,6))

plt.subplot(1,2,1)
plt.plot(epochs_range, history_dict['accuracy'], 'o-', label='Entrenamiento')
plt.plot(epochs_range, history_dict['val_accuracy'], 'o-', label='Validación')
plt.title('Evolución del Accuracy')
plt.xlabel('Épocas')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.subplot(1,2,2)
plt.plot(epochs_range, history_dict['loss'], 'o-', label='Entrenamiento')
```

```
plt.plot(epochs_range, history_dict['val_loss'], 'o-', label='Validación')
plt.title('Evolución del Loss')
plt.xlabel('Épocas')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



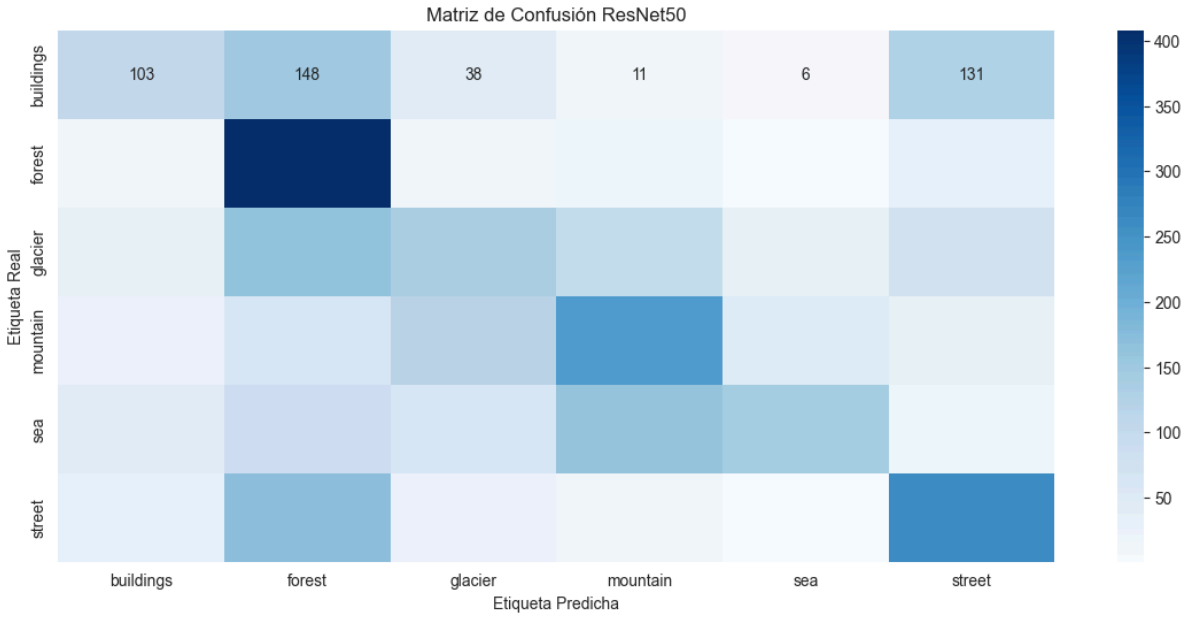
In [29]:
```
val_loss, val_acc = modelo_resnet.evaluate(val_generatorRS)
print(f'Perdida en validación {val_loss:.4f}')
print(f'Accuracy en validación {val_acc:.4f}')
```

```
94/94 [==============================] - 379s 4s/step - loss: 1.4260 - accuracy:
0.4287
Perdida en validación 1.4260
Accuracy en validación 0.4287
```

In [30]:
```
y_predRS =  modelo_resnet.predict(val_generatorRS)
y_pred_clasess = np.argmax(y_predRS, axis=1)
y_trueRS = val_generatorRS.classes
class_labels = list(val_generatorRS.class_indices.keys())
```

```
94/94 [==============================] - 386s 4s/step
```

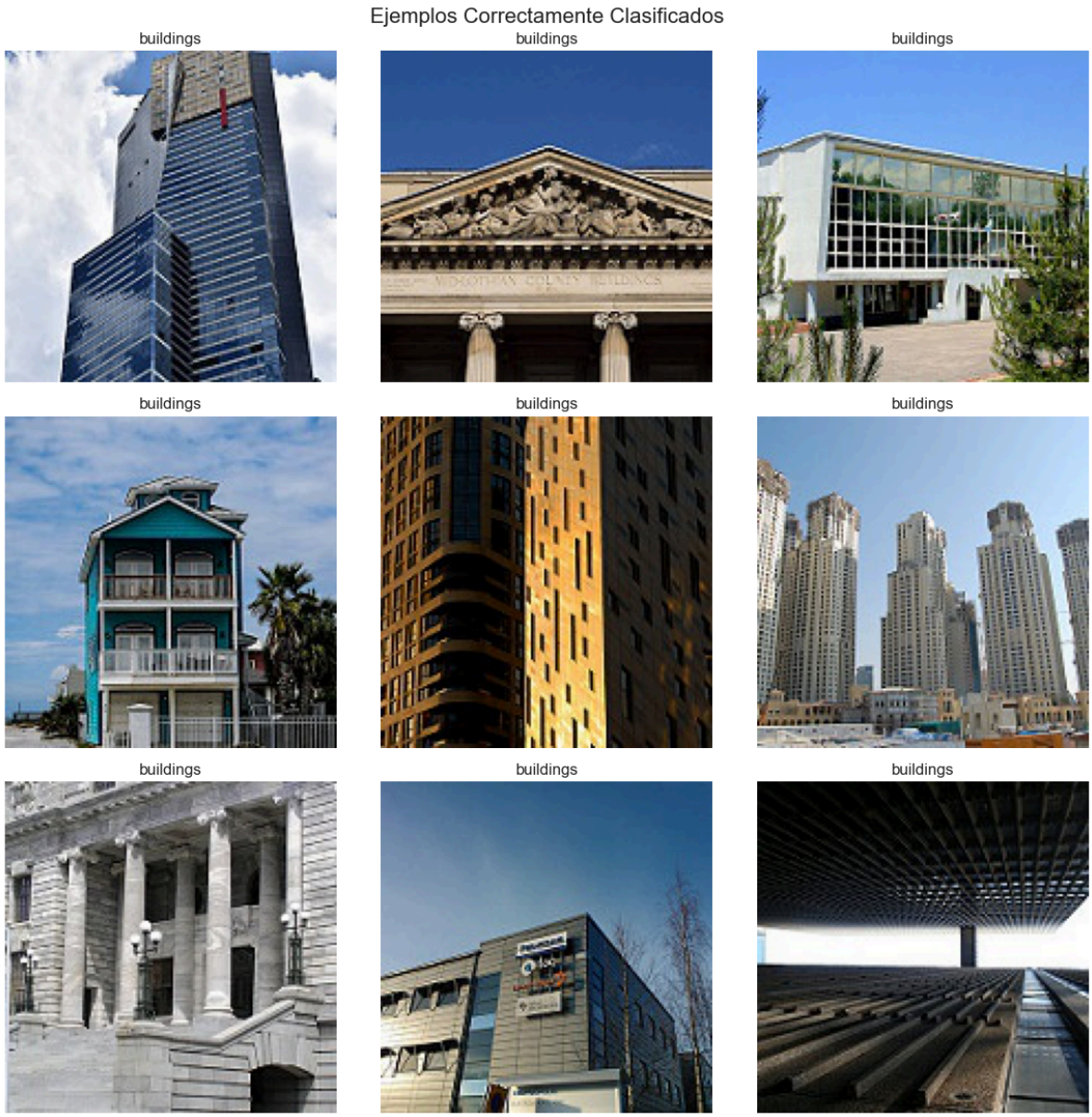
In [31]:
```
cm = confusion_matrix(y_trueRS, y_pred_clasess)
plt.figure(figsize=(14,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, ytickl
plt.title('Matriz de Confusión ResNet50')
plt.ylabel('Etiqueta Real')
plt.xlabel('Etiqueta Predicha')
plt.show()
```



In [32]:
```
print('Repoerte de Clasificación')
print(classification_report(y_trueRS, y_pred_clasess, target_names=class_labels))
```

```
Repoerte de Clasificación
              precision    recall  f1-score   support

   buildings       0.42      0.24      0.30       437
      forest       0.39      0.86      0.54       474
     glacier       0.36      0.25      0.30       553
    mountain       0.44      0.44      0.44       525
         sea       0.59      0.27      0.38       510
      street       0.47      0.53      0.50       501

    accuracy                           0.43      3000
   macro avg       0.45      0.43      0.41      3000
weighted avg       0.45      0.43      0.41      3000
```

In [36]:
```
correct_indices = np.where(y_pred_clasess == y_trueRS)[0]
plt.figure(figsize=(12, 12))
for i, idx in enumerate(correct_indices[:9]):
    plt.subplot(3, 3, i + 1)
    img_path = val_generatorRS.filepaths[idx]
    img = plt.imread(img_path)
    plt.imshow(img)
    plt.title(f"{class_labels[y_trueRS[idx]]}")
    plt.axis("off")
plt.suptitle("Ejemplos Correctamente Clasificados", fontsize=16)
plt.tight_layout()
plt.show()
```



In [35]:
```
incorrect_indices = np.where(y_pred_clasess != y_trueRS)[0]
plt.figure(figsize=(12, 12))
for i, idx in enumerate(incorrect_indices[:9]):
    plt.subplot(3, 3, i + 1)
    img_path = val_generatorRS.filepaths[idx]
    img = plt.imread(img_path)
    plt.imshow(img)
    plt.title(f"Pred: {class_labels[y_pred_clasess[idx]]}\nReal: {class_labels[y_tr
    plt.axis("off")
plt.suptitle("Ejemplos Incorrectamente Clasificados", fontsize=16)
plt.tight_layout()
plt.show()
```

In [38]:
```
modelo_resnet.save('mi_modelo_ResNet50.keras')
```

Los modelos EfficientNet y ResNet50 ya no fueron documentados ya que practicamente el lo mismo que los primeros 3, solo cambian un par de cosas. como la integracion de las aruitecturas, el dataset es es el mismo, las epocas 10 y 15. Si hay alguna duda, pueden ver lo documentado de los primero modelos.

Comparativa de todos los modelos entrenados

Modelo 1, 2 y 3 con CNN (10, 20 y 30 epochs)

EfficientNet (10 epochs)

Resnet (15 epochs)

#	Modelo	Épocas	Acc Train	Acc Val	Loss Train	Loss Val	Precisión Entrenamiento	Precisión Validacion
1	CNN capas propias	10	0.6615	0.7559	0.9054	0.7598	66.1%	75.6%
2	CNN capas propias	20	0.7739	0.8110	0.6292	0.5713	77.3$	81.1%
3	CNN capas propias	30	0.8506	0.8683	0.4292	0.3754	85.0%	86.8%
4	EfficentNet	10	0.1672	0.1750	1.8068	1.7893	17.0%	17.5%
4	ResNet50	15	0.3803	0.4287	1.4656	1.4260	38.0%	42.8%

Explicación

- Accuracy (precisión): Porcentaje de aciertos del modelo, Mientras mas alto mejor.
- Loss (pérdida): Mide qué tan equivocadas están las predicciones del modelo, mientras más bajo mejor.
- Val Accuacry: Precisión sobre dartos nuevos, los que no se ven durante el entrenamiento, permite saber si el modelo generaliza bien.
- Val Loss: Perdida en datos de validación, si es muy alta se tiene sobre ajustes, si es similar al loss de entrenamiento, el modelo generaliza bien.

Mejor modelo

- El tercer entrenamiento es el mejor modelo.
- Precisión en validación es del 86.6%
- Perdida en validación es del 0.3754
- Épocas 30
- Se tiene un buen balance entre el entrenamiento y la validación.

Peor modelo

- EfficientNet
- Precisión de solo 17.5%
- No logro aprender
- Se tiene una tasa de aprendizaje baja o la arquitectura es poco adecuada para los datos.

Resumen final

- Para este proyecto realizamos 5 entrenamientos, 3 con capas CNN, con arquitecturas ResNet5o y EfficentNet, despues de ver los resultados y comparar las métricas, la precisión, la generalización, el modelo final seleccionado fue el modelo 3 con 30 épocas, nos da un buen rendimiento estable y robusto.
- Esto nos dice que este modelo puede clasificar imágenes del dataset con alto nivel de confianza y generaliza bien sobre los datos.