modelS_2regularization

June 12, 2025

[9]:
```
from tensorflow import keras
from keras import layers
from keras.preprocessing import image_dataset_from_directory
import matplotlib.pyplot as plt
from keras.utils import to_categorical
import tensorflow as tf
import numpy as np
from keras.preprocessing import image
from sklearn.metrics import classification_report
import seaborn as sns
import pandas as pd
from sklearn.metrics import confusion_matrix
import os, shutil
```

1 Funções

[10]:
```
def get_true_pred(model, dataset):
    y_true = []
    y_pred = []
    for images, labels in dataset.unbatch().batch(1):
        y_true.append(np.argmax(labels.numpy()))
        pred = model.predict(images, verbose=0)
        y_pred.append(np.argmax(pred))
    return np.array(y_true), np.array(y_pred)
```

1.1 Carregamento do dataset

Carrega o dataset distribuido pelos diferentes conjuntos de dados.

[11]:
```
train_dir = 'Dataset/archive/seg_train'
validation_dir = 'Dataset/archive/seg_val'
test_dir = 'Dataset/archive/seg_test'

train_buildings_dir = 'Dataset/archive/seg_train/buildings/'
train_forest_dir = 'Dataset/archive/seg_train/forest'
train_glacier_dir = 'Dataset/archive/seg_train/glacier'
train_mountain_dir = 'Dataset/archive/seg_train/mountain'
```

```
train_sea_dir = 'Dataset/archive/seg_train/sea'
train_street_dir = 'Dataset/archive/seg_train/street'

val_buildings_dir = 'Dataset/archive/seg_val/buildings'
val_forest_dir = 'Dataset/archive/seg_val/forest'
val_glacier_dir = 'Dataset/archive/seg_val/glacier'
val_mountain_dir = 'Dataset/archive/seg_val/mountain'
val_sea_dir = 'Dataset/archive/seg_val/sea'
val_street_dir = 'Dataset/archive/seg_val/street'

test_buildings_dir = 'Dataset/archive/seg_test/buildings'
test_forest_dir = 'Dataset/archive/seg_test/forest'
test_glacier_dir = 'Dataset/archive/seg_test/glacier'
test_mountain_dir = 'Dataset/archive/seg_test/mountain'
test_sea_dir = 'Dataset/archive/seg_test/sea'
test_street_dir = 'Dataset/archive/seg_test/street'

print('total training buildings images:', len(os.listdir(train_buildings_dir)))
print('total training forest images:', len(os.listdir(train_forest_dir)))
print('total training glacier images:', len(os.listdir(train_glacier_dir)))
print('total training mountain images:', len(os.listdir(train_mountain_dir)))
print('total training sea images:', len(os.listdir(train_sea_dir)))
print('total training street images:', len(os.listdir(train_street_dir)))

print('total validation buildings images:', len(os.listdir(val_buildings_dir)))
print('total validation forest images:', len(os.listdir(val_forest_dir)))
print('total validation glacier images:', len(os.listdir(val_glacier_dir)))
print('total validation mountain images:', len(os.listdir(val_mountain_dir)))
print('total validation sea images:', len(os.listdir(val_sea_dir)))
print('total validation street images:', len(os.listdir(val_street_dir)))

print('total test buildings images:', len(os.listdir(test_buildings_dir)))
print('total test forest images:', len(os.listdir(test_forest_dir)))
print('total test glacier images:', len(os.listdir(test_glacier_dir)))
print('total test mountain images:', len(os.listdir(test_mountain_dir)))
print('total test sea images:', len(os.listdir(test_sea_dir)))
print('total test street images:', len(os.listdir(test_street_dir)))
```

```
total training buildings images: 1691
total training forest images: 1771
total training glacier images: 1904
total training mountain images: 2012
total training sea images: 1774
total training street images: 1882
total validation buildings images: 500
total validation forest images: 500
total validation glacier images: 500
total validation mountain images: 500
```

```
total validation sea images: 500
total validation street images: 500
total test buildings images: 437
total test forest images: 474
total test glacier images: 553
total test mountain images: 525
total test sea images: 510
total test street images: 501
```

1.2 Distribuição de imagens por classe e por conjunto de dados

As imagens estão distribuidas por 3 conjuntos de dados: train, validation e test. Cada um desses conjuntos está distribuido por 6 classes: buildings, forest, glacier, mountain, sea e street.

1.2.1 Número total de imagens por classe:

Classe	Treino	Validação	Teste	Total
Buildings	1691	500	437	2628
Forest	1771	500	474	2745
Glacier	1904	500	553	2957
Mountain	2012	500	525	3037
Sea	1774	500	510	2784
Street	1882	500	501	2883
Total	**11034**	**3000**	**3000**	**17034**

1.2.2 Número total de imagens por conjunto de dados:

Conjunto de dados	Total
Treino	11034
Validação	3000
Teste	3000
Total geral	**17034**

2 Processamento dos dados

Carrega, redimensiona e organiza imagens em batches com rótulos one-hot, preparando os dados de treino, validação e teste.

[12]:
```
IMG_SIZE = 150
BATCH_SIZE = 32

train_dataset = image_dataset_from_directory(
    train_dir,
    label_mode='categorical',
    image_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE)
```

```
validation_dataset = image_dataset_from_directory(
    validation_dir,
    label_mode='categorical',
    image_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE)

test_dataset = image_dataset_from_directory(
    test_dir,
    label_mode='categorical',
    image_size=(IMG_SIZE, IMG_SIZE),
    batch_size=BATCH_SIZE)

print(test_dataset)
class_names = train_dataset.class_names
print("Classes:", class_names)
```

```
Found 11034 files belonging to 6 classes.
Found 3000 files belonging to 6 classes.
Found 3000 files belonging to 6 classes.
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 150, 150, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None, 6), dtype=tf.float32,
name=None))>
Classes: ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
```

3 Modelo (regularization: Dropout)

3.1 Criação da CNN

Criação da CNN que irá receber imagens de 150x150 píxeis, aplica normalização e passa por quatro camadas convolucionais com max pooling para extrair características, integrando camadas Dropout para reduzir overfitting (desligando aleatoriamente 50% dos neurónios durante o treino, como forma de regularização). A rede termina com uma camada densa com 512 unidades e uma camada de saída softmax para classificação multiclasse.

[13]:
```
inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(32, 3, activation="relu")(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(64, 3, activation="relu")(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation="relu")(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation="relu")(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(512, activation="relu")(x)
```

```
outputs = layers.Dense(len(class_names), activation="softmax")(x)
model_dropout = keras.Model(inputs, outputs)

print(model_dropout.summary())
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 150, 150, 3)	0
rescaling_1 (Rescaling)	(None, 150, 150, 3)	0
conv2d_4 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_4 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_5 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_5 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_6 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_6 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_7 (Conv2D)	(None, 15, 15, 128)	147,584
max_pooling2d_7 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten_1 (Flatten)	(None, 6272)	0
dropout_1 (Dropout)	(None, 6272)	0
dense_2 (Dense)	(None, 512)	3,211,776
dense_3 (Dense)	(None, 6)	3,078

Total params: 3,455,686 (13.18 MB)

Trainable params: 3,455,686 (13.18 MB)

Non-trainable params: 0 (0.00 B)

```
None
```

3.2 Compilação da CNN

Compilação da CNN utilizando a loss **categorical crossentropy** e o optimizer **RMSprop**.

[14]:
```
model_dropout.compile(optimizer=tf.keras.optimizers.
↪RMSprop(learning_rate=1e-4), loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

3.3 Definição do callback

Definição de um callback que guarda automaticamente o modelo com a menor perda (loss) de validação durante o treino.

[15]:
```
checkpoint_filepath = 'modelS_CatCross_RMS_dropout.keras'
model_checkpoint_callback = keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    monitor='val_loss',
    save_best_only=True)
```

3.4 Treino da CNN

Treino da CNN durante 50 épocas utilizando o dataset de validação e o callback para guardar o melhor modelo.

[16]:
```
history_dropout = model_dropout.fit(
train_dataset,
epochs=50,
validation_data=validation_dataset,
callbacks=[model_checkpoint_callback])
```

```
Epoch 1/50
345/345              21s 45ms/step -
accuracy: 0.4426 - loss: 1.3753 - val_accuracy: 0.5703 - val_loss: 1.1022
Epoch 2/50
345/345              12s 36ms/step -
accuracy: 0.6253 - loss: 0.9863 - val_accuracy: 0.6890 - val_loss: 0.8312
Epoch 3/50
345/345              15s 44ms/step -
accuracy: 0.6762 - loss: 0.8629 - val_accuracy: 0.7187 - val_loss: 0.7614
Epoch 4/50
345/345              12s 36ms/step -
accuracy: 0.7157 - loss: 0.7780 - val_accuracy: 0.7403 - val_loss: 0.7058
Epoch 5/50
345/345              15s 44ms/step -
accuracy: 0.7359 - loss: 0.7181 - val_accuracy: 0.7410 - val_loss: 0.7101
Epoch 6/50
345/345              12s 35ms/step -
```

```
accuracy: 0.7618 - loss: 0.6633 - val_accuracy: 0.7117 - val_loss: 0.7699
Epoch 7/50
345/345              13s 37ms/step -
accuracy: 0.7794 - loss: 0.6241 - val_accuracy: 0.7193 - val_loss: 0.7779
Epoch 8/50
345/345              20s 58ms/step -
accuracy: 0.7910 - loss: 0.5869 - val_accuracy: 0.7310 - val_loss: 0.7266
Epoch 9/50
345/345              13s 37ms/step -
accuracy: 0.8069 - loss: 0.5492 - val_accuracy: 0.7847 - val_loss: 0.5986
Epoch 10/50
345/345              18s 52ms/step -
accuracy: 0.8184 - loss: 0.5187 - val_accuracy: 0.7503 - val_loss: 0.6917
Epoch 11/50
345/345              14s 40ms/step -
accuracy: 0.8293 - loss: 0.4868 - val_accuracy: 0.8077 - val_loss: 0.5454
Epoch 12/50
345/345              17s 50ms/step -
accuracy: 0.8367 - loss: 0.4730 - val_accuracy: 0.7993 - val_loss: 0.5538
Epoch 13/50
345/345              15s 43ms/step -
accuracy: 0.8411 - loss: 0.4459 - val_accuracy: 0.8173 - val_loss: 0.5046
Epoch 14/50
345/345              13s 38ms/step -
accuracy: 0.8471 - loss: 0.4289 - val_accuracy: 0.8203 - val_loss: 0.5058
Epoch 15/50
345/345              17s 49ms/step -
accuracy: 0.8541 - loss: 0.4198 - val_accuracy: 0.8033 - val_loss: 0.5352
Epoch 16/50
345/345              12s 34ms/step -
accuracy: 0.8612 - loss: 0.3930 - val_accuracy: 0.8200 - val_loss: 0.5121
Epoch 17/50
345/345              15s 43ms/step -
accuracy: 0.8732 - loss: 0.3742 - val_accuracy: 0.8250 - val_loss: 0.5069
Epoch 18/50
345/345              12s 35ms/step -
accuracy: 0.8760 - loss: 0.3588 - val_accuracy: 0.8357 - val_loss: 0.4656
Epoch 19/50
345/345              14s 41ms/step -
accuracy: 0.8774 - loss: 0.3497 - val_accuracy: 0.8357 - val_loss: 0.4636
Epoch 20/50
345/345              15s 44ms/step -
accuracy: 0.8821 - loss: 0.3356 - val_accuracy: 0.7933 - val_loss: 0.5976
Epoch 21/50
345/345              12s 35ms/step -
accuracy: 0.8876 - loss: 0.3286 - val_accuracy: 0.8353 - val_loss: 0.4769
Epoch 22/50
345/345              15s 44ms/step -
```

```
accuracy: 0.8889 - loss: 0.3143 - val_accuracy: 0.8260 - val_loss: 0.5241
Epoch 23/50
345/345             15s 43ms/step -
accuracy: 0.8968 - loss: 0.2975 - val_accuracy: 0.8267 - val_loss: 0.5161
Epoch 24/50
345/345             14s 39ms/step -
accuracy: 0.8995 - loss: 0.2850 - val_accuracy: 0.8427 - val_loss: 0.4812
Epoch 25/50
345/345             18s 51ms/step -
accuracy: 0.9032 - loss: 0.2735 - val_accuracy: 0.8503 - val_loss: 0.4568
Epoch 26/50
345/345             13s 39ms/step -
accuracy: 0.9129 - loss: 0.2665 - val_accuracy: 0.8410 - val_loss: 0.4671
Epoch 27/50
345/345             15s 44ms/step -
accuracy: 0.9153 - loss: 0.2445 - val_accuracy: 0.8380 - val_loss: 0.4799
Epoch 28/50
345/345             13s 38ms/step -
accuracy: 0.9145 - loss: 0.2428 - val_accuracy: 0.8267 - val_loss: 0.5130
Epoch 29/50
345/345             16s 46ms/step -
accuracy: 0.9143 - loss: 0.2387 - val_accuracy: 0.8253 - val_loss: 0.5518
Epoch 30/50
345/345             13s 38ms/step -
accuracy: 0.9203 - loss: 0.2263 - val_accuracy: 0.8347 - val_loss: 0.5003
Epoch 31/50
345/345             12s 35ms/step -
accuracy: 0.9274 - loss: 0.2115 - val_accuracy: 0.8477 - val_loss: 0.4689
Epoch 32/50
345/345             16s 48ms/step -
accuracy: 0.9297 - loss: 0.2033 - val_accuracy: 0.8477 - val_loss: 0.4951
Epoch 33/50
345/345             12s 35ms/step -
accuracy: 0.9365 - loss: 0.1894 - val_accuracy: 0.8413 - val_loss: 0.5072
Epoch 34/50
345/345             16s 47ms/step -
accuracy: 0.9357 - loss: 0.1891 - val_accuracy: 0.8330 - val_loss: 0.5603
Epoch 35/50
345/345             12s 35ms/step -
accuracy: 0.9397 - loss: 0.1789 - val_accuracy: 0.8460 - val_loss: 0.4890
Epoch 36/50
345/345             13s 39ms/step -
accuracy: 0.9372 - loss: 0.1730 - val_accuracy: 0.8323 - val_loss: 0.5264
Epoch 37/50
345/345             18s 51ms/step -
accuracy: 0.9423 - loss: 0.1659 - val_accuracy: 0.8413 - val_loss: 0.4951
Epoch 38/50
345/345             13s 39ms/step -
```

```
accuracy: 0.9448 - loss: 0.1592 - val_accuracy: 0.8497 - val_loss: 0.5041
Epoch 39/50
345/345              15s 44ms/step -
accuracy: 0.9468 - loss: 0.1484 - val_accuracy: 0.8517 - val_loss: 0.4858
Epoch 40/50
345/345              13s 39ms/step -
accuracy: 0.9467 - loss: 0.1506 - val_accuracy: 0.8487 - val_loss: 0.4911
Epoch 41/50
345/345              13s 39ms/step -
accuracy: 0.9526 - loss: 0.1405 - val_accuracy: 0.8467 - val_loss: 0.5276
Epoch 42/50
345/345              16s 45ms/step -
accuracy: 0.9533 - loss: 0.1319 - val_accuracy: 0.8447 - val_loss: 0.5212
Epoch 43/50
345/345              12s 35ms/step -
accuracy: 0.9539 - loss: 0.1280 - val_accuracy: 0.8210 - val_loss: 0.7326
Epoch 44/50
345/345              15s 44ms/step -
accuracy: 0.9512 - loss: 0.1310 - val_accuracy: 0.8533 - val_loss: 0.4999
Epoch 45/50
345/345              12s 35ms/step -
accuracy: 0.9628 - loss: 0.1122 - val_accuracy: 0.8567 - val_loss: 0.5042
Epoch 46/50
345/345              12s 35ms/step -
accuracy: 0.9634 - loss: 0.1089 - val_accuracy: 0.8497 - val_loss: 0.5295
Epoch 47/50
345/345              15s 44ms/step -
accuracy: 0.9617 - loss: 0.1052 - val_accuracy: 0.8390 - val_loss: 0.6200
Epoch 48/50
345/345              13s 36ms/step -
accuracy: 0.9632 - loss: 0.1049 - val_accuracy: 0.8503 - val_loss: 0.5236
Epoch 49/50
345/345              17s 49ms/step -
accuracy: 0.9652 - loss: 0.1029 - val_accuracy: 0.8433 - val_loss: 0.5867
Epoch 50/50
345/345              12s 36ms/step -
accuracy: 0.9676 - loss: 0.0972 - val_accuracy: 0.8447 - val_loss: 0.5763
```

[17]:
```
best_epoch = np.argmin(history_dropout.history['val_loss']) + 1
print(f"Melhor época (menor val_loss): {best_epoch}")
```

```
Melhor época (menor val_loss): 25
```

3.5 Carregamento do modelo e validação

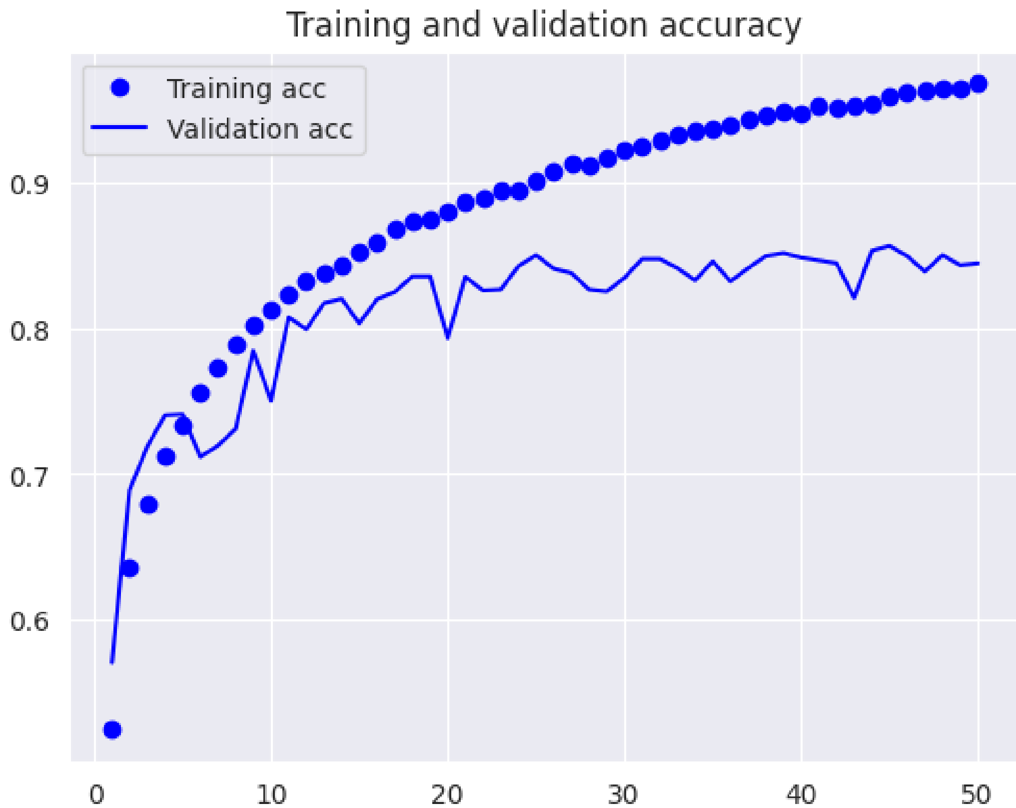
Carregamento e avaliação do modelo através do valor da accuracy.

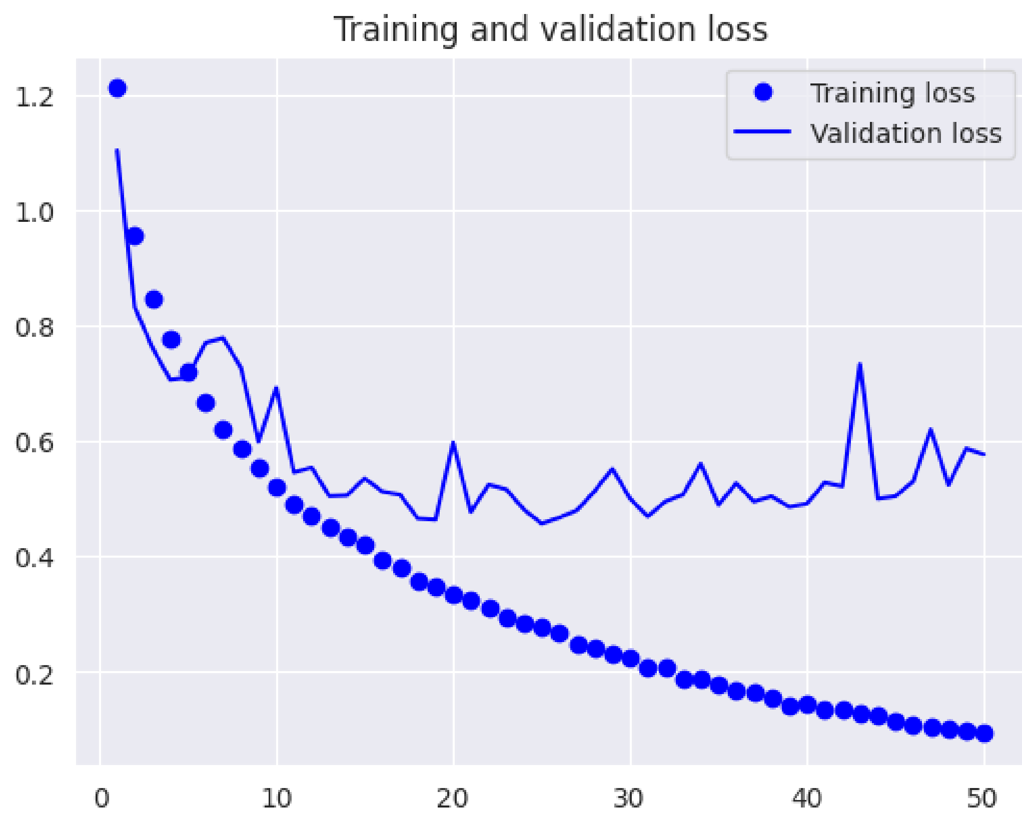
[18]:
```
model_dropout = keras.models.load_model('modelS_CatCross_RMS_dropout.keras')
val_loss, val_acc = model_dropout.evaluate(validation_dataset)
print('val_acc:', val_acc)
```

```
94/94            3s 21ms/step -
accuracy: 0.8406 - loss: 0.4978
val_acc: 0.8503333330154419
```

Representação gráfica dos valores da accuracy e da loss ao longo das épocas.

[19]:
```
acc = history_dropout.history['accuracy']
val_acc = history_dropout.history['val_accuracy']
loss = history_dropout.history['loss']
val_loss = history_dropout.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.9
0.8
0.7
0.6
0
10
20
30
40
50



Avaliação da performance do modelo no conjunto de teste, utilizando o relatório de classificação. O relatório apresenta, para cada classe, as métricas precision, recall e F1-score, permitindo analisar detalhadamente os acertos e erros por classe.

[20]:
```
y_true, y_pred = get_true_pred(model_dropout, test_dataset)
report = classification_report(y_true, y_pred, target_names=class_names,␣
    ↪output_dict=True)
class_only_report = {k: v for k, v in report.items() if k in class_names}
df = pd.DataFrame(class_only_report).T
print(df[['precision', 'recall', 'f1-score']].round(3))
```

```
2025-06-12 22:02:18.308742: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:549]
Omitted potentially buggy algorithm eng14{} for conv %cudnn-conv-bias-
activation.12 = (f32[1,32,148,148]{3,2,1,0}, u8[0]{0}) custom-
call(f32[1,3,150,150]{3,2,1,0} %bitcast.262, f32[32,3,3,3]{3,2,1,0}
%bitcast.269, f32[32]{0} %bitcast.271), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward",
metadata={op_type="Conv2D" op_name="functional_1_1/conv2d_4_1/convolution"
source_file="/home/diogo/.pyenv/versions/3.10.18/lib/python3.10/site-
packages/tensorflow/python/framework/ops.py" source_line=1200}, backend_config={
```

```
"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_confi
g":{"conv_result_scale":1,"activation_mode":"kRelu","side_input_scale":0,"leakyr
elu_alpha":0},"force_earliest_schedule":false}
2025-06-12 22:02:18.371909: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:549]
Omitted potentially buggy algorithm eng14{} for conv %cudnn-conv-bias-
activation.13 = (f32[1,64,72,72]{3,2,1,0}, u8[0]{0}) custom-
call(f32[1,32,74,74]{3,2,1,0} %bitcast.278, f32[64,32,3,3]{3,2,1,0}
%bitcast.285, f32[64]{0} %bitcast.287), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward",
metadata={op_type="Conv2D" op_name="functional_1_1/conv2d_5_1/convolution"
source_file="/home/diogo/.pyenv/versions/3.10.18/lib/python3.10/site-
packages/tensorflow/python/framework/ops.py" source_line=1200}, backend_config={
"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_confi
g":{"conv_result_scale":1,"activation_mode":"kRelu","side_input_scale":0,"leakyr
elu_alpha":0},"force_earliest_schedule":false}
2025-06-12 22:02:18.425038: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:549]
Omitted potentially buggy algorithm eng14{} for conv %cudnn-conv-bias-
activation.14 = (f32[1,128,34,34]{3,2,1,0}, u8[0]{0}) custom-
call(f32[1,64,36,36]{3,2,1,0} %bitcast.293, f32[128,64,3,3]{3,2,1,0}
%bitcast.300, f32[128]{0} %bitcast.302), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward",
metadata={op_type="Conv2D" op_name="functional_1_1/conv2d_6_1/convolution"
source_file="/home/diogo/.pyenv/versions/3.10.18/lib/python3.10/site-
packages/tensorflow/python/framework/ops.py" source_line=1200}, backend_config={
"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_confi
g":{"conv_result_scale":1,"activation_mode":"kRelu","side_input_scale":0,"leakyr
elu_alpha":0},"force_earliest_schedule":false}
2025-06-12 22:02:18.482526: I
external/local_xla/xla/service/gpu/autotuning/conv_algorithm_picker.cc:549]
Omitted potentially buggy algorithm eng14{} for conv %cudnn-conv-bias-
activation.15 = (f32[1,128,15,15]{3,2,1,0}, u8[0]{0}) custom-
call(f32[1,128,17,17]{3,2,1,0} %bitcast.308, f32[128,128,3,3]{3,2,1,0}
%bitcast.315, f32[128]{0} %bitcast.317), window={size=3x3},
dim_labels=bf01_oi01->bf01,
custom_call_target="__cudnn$convBiasActivationForward",
metadata={op_type="Conv2D" op_name="functional_1_1/conv2d_7_1/convolution"
source_file="/home/diogo/.pyenv/versions/3.10.18/lib/python3.10/site-
packages/tensorflow/python/framework/ops.py" source_line=1200}, backend_config={
"operation_queue_id":"0","wait_on_operation_queues":[],"cudnn_conv_backend_confi
g":{"conv_result_scale":1,"activation_mode":"kRelu","side_input_scale":0,"leakyr
elu_alpha":0},"force_earliest_schedule":false}
```

```
           precision  recall  f1-score
buildings      0.864   0.796     0.829
```

```
forest          0.942    0.951      0.946
glacier         0.832    0.821      0.826
mountain        0.886    0.758      0.817
sea             0.776    0.931      0.847
street          0.863    0.880      0.872
```

```
2025-06-12 22:05:59.173953: I tensorflow/core/framework/local_rendezvous.cc:407]
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

4 Modelo (regularization: L2)

4.1 Criação da CNN

Criação da CNN que irá receber imagens de 150x150 píxeis, aplica normalização e passa por quatro camadas convolucionais com max pooling para extrair características, integrando regularização L2 nas camadas convolucionais para penalizar pesos excessivamente elevados e reduzir o overfitting. A rede termina com uma camada densa com 512 unidades e uma camada de saída softmax para classificação multiclasse.

[21]:
```
inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(32, 3, activation="relu", kernel_regularizer='l2')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(64, 3, activation="relu", kernel_regularizer='l2')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation="relu", kernel_regularizer='l2')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation="relu", kernel_regularizer='l2')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
x = layers.Dense(512, activation="relu", kernel_regularizer='l2')(x)
outputs = layers.Dense(len(class_names), activation="softmax")(x)
model_l2 = keras.Model(inputs, outputs)

print(model_l2.summary())
```

Model: "functional_2"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 150, 150, 3)	0
rescaling_2 (Rescaling)	(None, 150, 150, 3)	0
conv2d_8 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_8 (MaxPooling2D)	(None, 74, 74, 32)	0

```
conv2d_9 (Conv2D)                 (None, 72, 72, 64)          18,496

max_pooling2d_9 (MaxPooling2D)    (None, 36, 36, 64)               0

conv2d_10 (Conv2D)                (None, 34, 34, 128)         73,856

max_pooling2d_10 (MaxPooling2D)   (None, 17, 17, 128)              0

conv2d_11 (Conv2D)                (None, 15, 15, 128)        147,584

max_pooling2d_11 (MaxPooling2D)   (None, 7, 7, 128)                0

flatten_2 (Flatten)               (None, 6272)                     0

dense_4 (Dense)                   (None, 512)              3,211,776

dense_5 (Dense)                   (None, 6)                    3,078


Total params: 3,455,686 (13.18 MB)

Trainable params: 3,455,686 (13.18 MB)

Non-trainable params: 0 (0.00 B)

None
```

4.2 Compilação da CNN

Compilação da CNN utilizando a loss **categorical crossentropy** e o optimizer **RMSprop**.

[22]:
```
model_l2.compile(
loss='categorical_crossentropy',
optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),
metrics=['acc'])
```

4.3 Definição do callback

Definição de um callback que guarda automaticamente o modelo com a menor perda (loss) de validação durante o treino.

[23]:
```
checkpoint_filepath = 'modelS_CatCross_RMS_L2.keras'
model_checkpoint_callback = keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    monitor='val_loss',
```

```
    save_best_only=True)
```

4.4 Treino da CNN

Treino da CNN durante 50 épocas utilizando o dataset de validação e o callback para guardar o melhor modelo.

[24]:
```
history_l2 = model_l2.fit(
train_dataset,
epochs=50,
validation_data=validation_dataset,
callbacks=[model_checkpoint_callback])
```

```
Epoch 1/50
345/345              24s 57ms/step -
acc: 0.4528 - loss: 9.2746 - val_acc: 0.5573 - val_loss: 3.6021
Epoch 2/50
345/345              14s 40ms/step -
acc: 0.6091 - loss: 3.0461 - val_acc: 0.6103 - val_loss: 2.2840
Epoch 3/50
345/345              15s 44ms/step -
acc: 0.6483 - loss: 2.1270 - val_acc: 0.6237 - val_loss: 1.8955
Epoch 4/50
345/345              13s 39ms/step -
acc: 0.6690 - loss: 1.7784 - val_acc: 0.6527 - val_loss: 1.6544
Epoch 5/50
345/345              15s 45ms/step -
acc: 0.6777 - loss: 1.5969 - val_acc: 0.6843 - val_loss: 1.5193
Epoch 6/50
345/345              13s 38ms/step -
acc: 0.6881 - loss: 1.4691 - val_acc: 0.6867 - val_loss: 1.4160
Epoch 7/50
345/345              13s 36ms/step -
acc: 0.6998 - loss: 1.3742 - val_acc: 0.6973 - val_loss: 1.3263
Epoch 8/50
345/345              16s 46ms/step -
acc: 0.7055 - loss: 1.3055 - val_acc: 0.6417 - val_loss: 1.3917
Epoch 9/50
345/345              15s 43ms/step -
acc: 0.7095 - loss: 1.2612 - val_acc: 0.6767 - val_loss: 1.3005
Epoch 10/50
345/345              16s 48ms/step -
acc: 0.7205 - loss: 1.2088 - val_acc: 0.6947 - val_loss: 1.2515
Epoch 11/50
345/345              13s 38ms/step -
acc: 0.7268 - loss: 1.1717 - val_acc: 0.6557 - val_loss: 1.3328
Epoch 12/50
345/345              12s 36ms/step -
```

```
acc: 0.7234 - loss: 1.1442 - val_acc: 0.7117 - val_loss: 1.1288
Epoch 13/50
345/345              16s 45ms/step -
acc: 0.7330 - loss: 1.1078 - val_acc: 0.7387 - val_loss: 1.1108
Epoch 14/50
345/345              12s 36ms/step -
acc: 0.7431 - loss: 1.0844 - val_acc: 0.7290 - val_loss: 1.0947
Epoch 15/50
345/345              15s 45ms/step -
acc: 0.7467 - loss: 1.0525 - val_acc: 0.7427 - val_loss: 1.0504
Epoch 16/50
345/345              12s 36ms/step -
acc: 0.7504 - loss: 1.0380 - val_acc: 0.7607 - val_loss: 1.0250
Epoch 17/50
345/345              12s 35ms/step -
acc: 0.7539 - loss: 1.0255 - val_acc: 0.7500 - val_loss: 1.0428
Epoch 18/50
345/345              15s 44ms/step -
acc: 0.7574 - loss: 1.0046 - val_acc: 0.7377 - val_loss: 1.0705
Epoch 19/50
345/345              13s 37ms/step -
acc: 0.7621 - loss: 0.9876 - val_acc: 0.7437 - val_loss: 1.0381
Epoch 20/50
345/345              17s 49ms/step -
acc: 0.7615 - loss: 0.9803 - val_acc: 0.7690 - val_loss: 0.9599
Epoch 21/50
345/345              15s 44ms/step -
acc: 0.7686 - loss: 0.9601 - val_acc: 0.6663 - val_loss: 1.1906
Epoch 22/50
345/345              13s 37ms/step -
acc: 0.7692 - loss: 0.9513 - val_acc: 0.7447 - val_loss: 0.9894
Epoch 23/50
345/345              17s 48ms/step -
acc: 0.7700 - loss: 0.9360 - val_acc: 0.7713 - val_loss: 0.9422
Epoch 24/50
345/345              14s 39ms/step -
acc: 0.7770 - loss: 0.9198 - val_acc: 0.7597 - val_loss: 0.9706
Epoch 25/50
345/345              17s 48ms/step -
acc: 0.7735 - loss: 0.9177 - val_acc: 0.7570 - val_loss: 0.9797
Epoch 26/50
345/345              13s 38ms/step -
acc: 0.7800 - loss: 0.9060 - val_acc: 0.7603 - val_loss: 0.9483
Epoch 27/50
345/345              16s 46ms/step -
acc: 0.7804 - loss: 0.8963 - val_acc: 0.7573 - val_loss: 0.9666
Epoch 28/50
345/345              13s 38ms/step -
```

```
acc: 0.7846 - loss: 0.8868 - val_acc: 0.7767 - val_loss: 0.9004
Epoch 29/50
345/345              13s 38ms/step -
acc: 0.7788 - loss: 0.8784 - val_acc: 0.7787 - val_loss: 0.9022
Epoch 30/50
345/345              16s 47ms/step -
acc: 0.7855 - loss: 0.8691 - val_acc: 0.7713 - val_loss: 0.9269
Epoch 31/50
345/345              13s 38ms/step -
acc: 0.7904 - loss: 0.8646 - val_acc: 0.7317 - val_loss: 1.0009
Epoch 32/50
345/345              16s 47ms/step -
acc: 0.7848 - loss: 0.8606 - val_acc: 0.7390 - val_loss: 0.9471
Epoch 33/50
345/345              13s 39ms/step -
acc: 0.7880 - loss: 0.8504 - val_acc: 0.7880 - val_loss: 0.8730
Epoch 34/50
345/345              15s 43ms/step -
acc: 0.7976 - loss: 0.8468 - val_acc: 0.7977 - val_loss: 0.8554
Epoch 35/50
345/345              16s 46ms/step -
acc: 0.8033 - loss: 0.8367 - val_acc: 0.7833 - val_loss: 0.8614
Epoch 36/50
345/345              13s 38ms/step -
acc: 0.7959 - loss: 0.8393 - val_acc: 0.7850 - val_loss: 0.8654
Epoch 37/50
345/345              15s 43ms/step -
acc: 0.8020 - loss: 0.8177 - val_acc: 0.7797 - val_loss: 0.8835
Epoch 38/50
345/345              12s 36ms/step -
acc: 0.8020 - loss: 0.8264 - val_acc: 0.7907 - val_loss: 0.8437
Epoch 39/50
345/345              13s 38ms/step -
acc: 0.8036 - loss: 0.8110 - val_acc: 0.7697 - val_loss: 0.8976
Epoch 40/50
345/345              17s 49ms/step -
acc: 0.8031 - loss: 0.8145 - val_acc: 0.7913 - val_loss: 0.8512
Epoch 41/50
345/345              13s 39ms/step -
acc: 0.8071 - loss: 0.7967 - val_acc: 0.7913 - val_loss: 0.8296
Epoch 42/50
345/345              15s 44ms/step -
acc: 0.8074 - loss: 0.8000 - val_acc: 0.7937 - val_loss: 0.8305
Epoch 43/50
345/345              12s 36ms/step -
acc: 0.8044 - loss: 0.7901 - val_acc: 0.7920 - val_loss: 0.8220
Epoch 44/50
345/345              12s 35ms/step -
```

```
acc: 0.8097 - loss: 0.7814 - val_acc: 0.7517 - val_loss: 0.9344
Epoch 45/50
345/345                18s 51ms/step -
acc: 0.8110 - loss: 0.7858 - val_acc: 0.7743 - val_loss: 0.8539
Epoch 46/50
345/345                14s 40ms/step -
acc: 0.8159 - loss: 0.7818 - val_acc: 0.8013 - val_loss: 0.8134
Epoch 47/50
345/345                15s 44ms/step -
acc: 0.8148 - loss: 0.7744 - val_acc: 0.7523 - val_loss: 0.9336
Epoch 48/50
345/345                12s 36ms/step -
acc: 0.8151 - loss: 0.7692 - val_acc: 0.7953 - val_loss: 0.8094
Epoch 49/50
345/345                12s 36ms/step -
acc: 0.8191 - loss: 0.7626 - val_acc: 0.7987 - val_loss: 0.8033
Epoch 50/50
345/345                15s 44ms/step -
acc: 0.8134 - loss: 0.7701 - val_acc: 0.7700 - val_loss: 0.8718
```

[25]:
```
best_epoch = np.argmin(history_l2.history['val_loss']) + 1
print(f"Melhor época (menor val_loss): {best_epoch}")
```

```
Melhor época (menor val_loss): 49
```

4.5 Carregamento do modelo e validação

Carregamento e avaliação do modelo através do valor da accuracy.

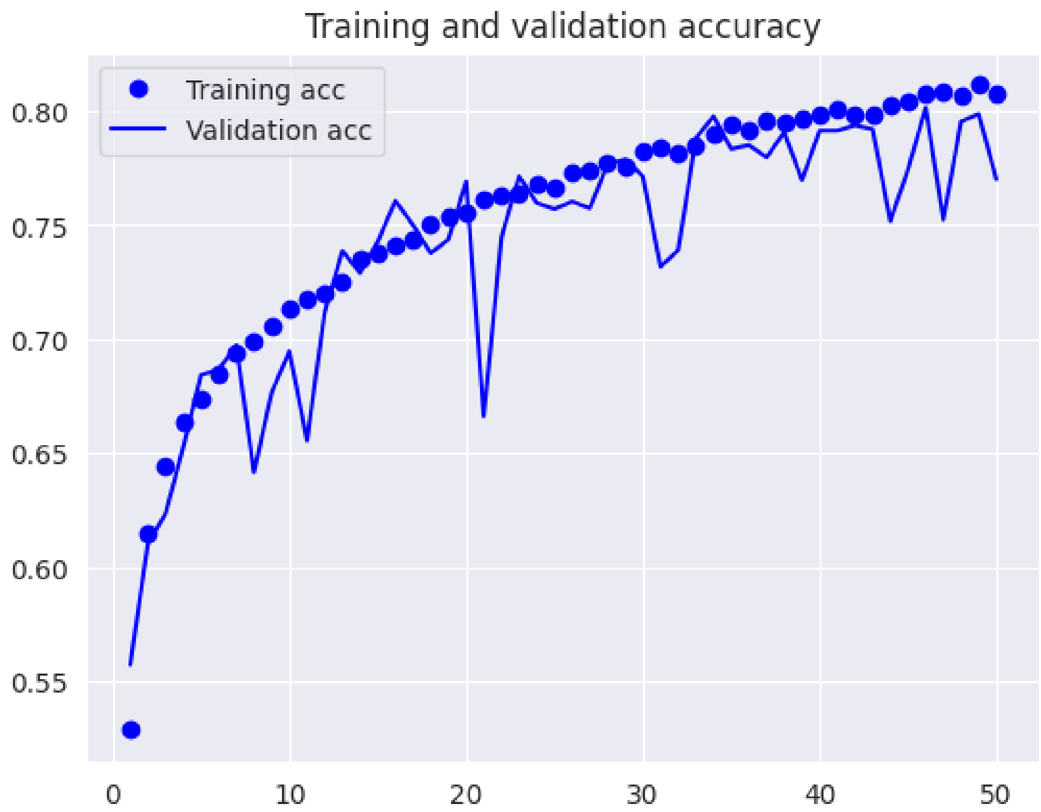
[26]:
```
model_l2 = keras.models.load_model('modelS_CatCross_RMS_L2.keras')
val_loss, val_acc = model_l2.evaluate(validation_dataset)
print('val_acc:', val_acc)
```

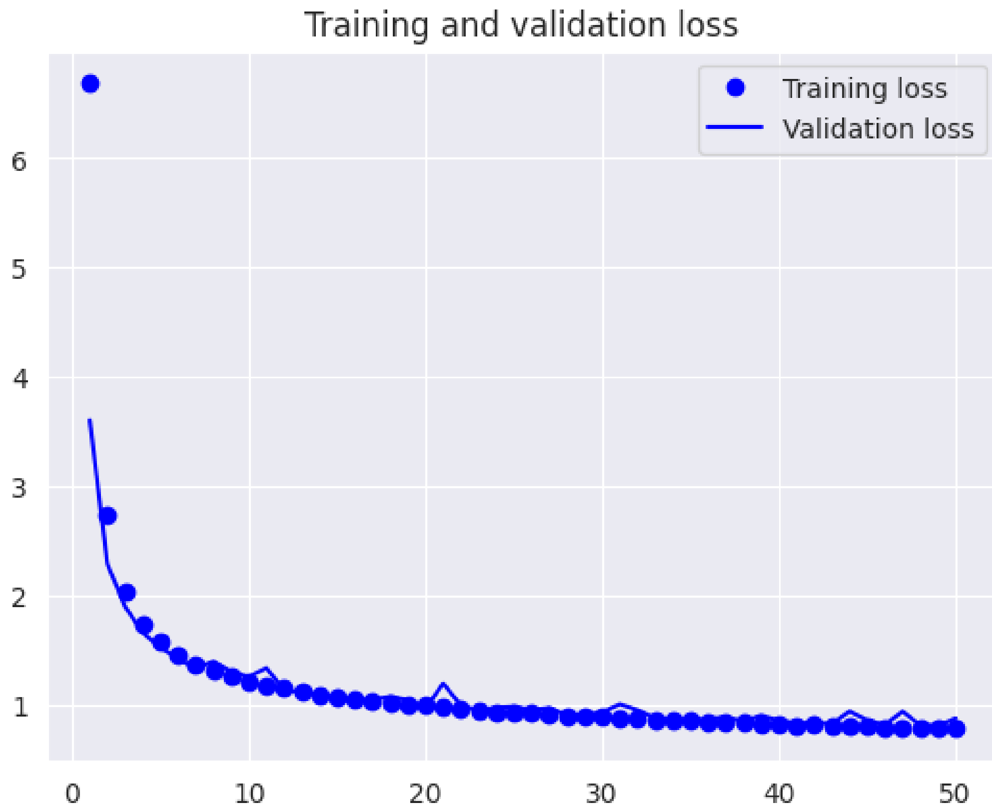
```
94/94               3s 21ms/step - acc:
0.7978 - loss: 0.8087
val_acc: 0.7986666560173035
```

Representação gráfica dos valores da accuracy e da loss ao longo das épocas.

[27]:
```
acc = history_l2.history['acc']
val_acc = history_l2.history['val_acc']
loss = history_l2.history['loss']
val_loss = history_l2.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
```

```
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





Avaliação da performance do modelo no conjunto de teste, utilizando o relatório de classificação. O relatório apresenta, para cada classe, as métricas precision, recall e F1-score, permitindo analisar detalhadamente os acertos e erros por classe.

[28]:

```
y_true, y_pred = get_true_pred(model_l2, test_dataset)
report = classification_report(y_true, y_pred, target_names=class_names,␣
  ↪output_dict=True)
class_only_report = {k: v for k, v in report.items() if k in class_names}
df = pd.DataFrame(class_only_report).T
print(df[['precision', 'recall', 'f1-score']].round(3))
```

```
           precision  recall  f1-score
buildings      0.789   0.787     0.788
forest         0.946   0.888     0.916
glacier        0.803   0.754     0.778
mountain       0.803   0.712     0.755
sea            0.748   0.843     0.793
street         0.778   0.868     0.821
```

```
2025-06-12 22:22:18.626637: I tensorflow/core/framework/local_rendezvous.cc:407]
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

5 Modelo (regularization: Dropout e L2)

5.1 Criação da CNN

Criação da CNN que irá receber imagens de 150x150 píxeis, aplica normalização e passa por quatro camadas convolucionais com max pooling para extrair características. Integra regularização L2 nas camadas convolucionais para penalizar pesos elevados e reduzir o overfitting, e camadas Dropout que desligam aleatoriamente 50% dos neurónios durante o treino. Termina com uma camada densa com 512 unidades e uma camada de saída softmax para classificação multiclasse.

[29]:
```
inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(32, 3, activation="relu", kernel_regularizer='l2')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(64, 3, activation="relu", kernel_regularizer='l2')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation="relu", kernel_regularizer='l2')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Conv2D(128, 3, activation="relu", kernel_regularizer='l2')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(512, activation="relu", kernel_regularizer='l2')(x)
outputs = layers.Dense(len(class_names), activation="softmax")(x)
model_both = keras.Model(inputs, outputs)

print(model_both.summary())
```

Model: "functional_3"

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 150, 150, 3)	0
rescaling_3 (Rescaling)	(None, 150, 150, 3)	0
conv2d_12 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_12 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_13 (Conv2D)	(None, 72, 72, 64)	18,496
max_pooling2d_13 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_14 (Conv2D)	(None, 34, 34, 128)	73,856
max_pooling2d_14 (MaxPooling2D)	(None, 17, 17, 128)	0

```
 conv2d_15 (Conv2D)              (None, 15, 15, 128)          147,584

 max_pooling2d_15 (MaxPooling2D)  (None, 7, 7, 128)                  0

 flatten_3 (Flatten)             (None, 6272)                        0

 dropout_2 (Dropout)             (None, 6272)                        0

 dense_6 (Dense)                 (None, 512)                 3,211,776

 dense_7 (Dense)                 (None, 6)                       3,078


 Total params: 3,455,686 (13.18 MB)

 Trainable params: 3,455,686 (13.18 MB)

 Non-trainable params: 0 (0.00 B)

None
```

5.2 Compilação da CNN

Compilação da CNN utilizando a loss **categorical crossentropy** e o optimizer **RMSprop**.

[30]:
```
model_both.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=1e-4),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
```

5.3 Definição do callback

Definição de um callback que guarda automaticamente o modelo com a menor perda (loss) de validação durante o treino.

[31]:
```
checkpoint_filepath = 'modelS_CatCross_RMS_dropout_L2.keras'
model_checkpoint_callback = keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    monitor='val_loss',
    save_best_only=True)
```

5.4 Treino da CNN

Treino da CNN durante 50 épocas utilizando o dataset de validação e o callback para guardar o melhor modelo.

[32]:
```
history_both = model_both.fit(
train_dataset,
epochs=50,
validation_data=validation_dataset,
callbacks=[model_checkpoint_callback])
```

```
Epoch 1/50
345/345             23s 55ms/step -
accuracy: 0.4214 - loss: 9.5866 - val_accuracy: 0.5413 - val_loss: 3.9742
Epoch 2/50
345/345             12s 36ms/step -
accuracy: 0.5933 - loss: 3.3751 - val_accuracy: 0.6030 - val_loss: 2.4343
Epoch 3/50
345/345             17s 50ms/step -
accuracy: 0.6382 - loss: 2.2580 - val_accuracy: 0.6430 - val_loss: 1.9492
Epoch 4/50
345/345             14s 40ms/step -
accuracy: 0.6604 - loss: 1.8401 - val_accuracy: 0.6463 - val_loss: 1.7115
Epoch 5/50
345/345             12s 36ms/step -
accuracy: 0.6727 - loss: 1.6196 - val_accuracy: 0.6677 - val_loss: 1.5477
Epoch 6/50
345/345             15s 43ms/step -
accuracy: 0.6845 - loss: 1.4849 - val_accuracy: 0.6790 - val_loss: 1.4433
Epoch 7/50
345/345             12s 34ms/step -
accuracy: 0.6917 - loss: 1.4102 - val_accuracy: 0.6963 - val_loss: 1.3490
Epoch 8/50
345/345             15s 43ms/step -
accuracy: 0.7043 - loss: 1.3289 - val_accuracy: 0.7130 - val_loss: 1.2866
Epoch 9/50
345/345             12s 34ms/step -
accuracy: 0.7116 - loss: 1.2796 - val_accuracy: 0.7273 - val_loss: 1.2266
Epoch 10/50
345/345             12s 36ms/step -
accuracy: 0.7154 - loss: 1.2340 - val_accuracy: 0.7120 - val_loss: 1.2114
Epoch 11/50
345/345             16s 47ms/step -
accuracy: 0.7235 - loss: 1.2049 - val_accuracy: 0.7253 - val_loss: 1.1907
Epoch 12/50
345/345             13s 37ms/step -
accuracy: 0.7254 - loss: 1.1677 - val_accuracy: 0.7343 - val_loss: 1.1421
Epoch 13/50
345/345             13s 37ms/step -
accuracy: 0.7339 - loss: 1.1366 - val_accuracy: 0.7480 - val_loss: 1.1096
Epoch 14/50
345/345             15s 44ms/step -
accuracy: 0.7413 - loss: 1.1195 - val_accuracy: 0.7410 - val_loss: 1.0936
```

```
Epoch 15/50
345/345              12s 36ms/step -
accuracy: 0.7451 - loss: 1.0907 - val_accuracy: 0.7560 - val_loss: 1.0573
Epoch 16/50
345/345              15s 44ms/step -
accuracy: 0.7547 - loss: 1.0684 - val_accuracy: 0.7287 - val_loss: 1.1081
Epoch 17/50
345/345              13s 36ms/step -
accuracy: 0.7589 - loss: 1.0490 - val_accuracy: 0.7550 - val_loss: 1.0485
Epoch 18/50
345/345              12s 35ms/step -
accuracy: 0.7567 - loss: 1.0374 - val_accuracy: 0.7027 - val_loss: 1.1904
Epoch 19/50
345/345              15s 45ms/step -
accuracy: 0.7523 - loss: 1.0343 - val_accuracy: 0.7653 - val_loss: 1.0116
Epoch 20/50
345/345              12s 35ms/step -
accuracy: 0.7671 - loss: 1.0063 - val_accuracy: 0.7590 - val_loss: 1.0176
Epoch 21/50
345/345              15s 44ms/step -
accuracy: 0.7688 - loss: 1.0036 - val_accuracy: 0.7453 - val_loss: 1.0283
Epoch 22/50
345/345              12s 36ms/step -
accuracy: 0.7721 - loss: 0.9823 - val_accuracy: 0.7770 - val_loss: 0.9633
Epoch 23/50
345/345              12s 35ms/step -
accuracy: 0.7773 - loss: 0.9660 - val_accuracy: 0.7680 - val_loss: 0.9873
Epoch 24/50
345/345              15s 44ms/step -
accuracy: 0.7855 - loss: 0.9553 - val_accuracy: 0.7550 - val_loss: 1.0057
Epoch 25/50
345/345              14s 41ms/step -
accuracy: 0.7747 - loss: 0.9548 - val_accuracy: 0.7657 - val_loss: 0.9789
Epoch 26/50
345/345              16s 47ms/step -
accuracy: 0.7827 - loss: 0.9414 - val_accuracy: 0.7717 - val_loss: 0.9654
Epoch 27/50
345/345              15s 43ms/step -
accuracy: 0.7881 - loss: 0.9302 - val_accuracy: 0.7860 - val_loss: 0.9220
Epoch 28/50
345/345              15s 42ms/step -
accuracy: 0.7917 - loss: 0.9171 - val_accuracy: 0.7773 - val_loss: 0.9257
Epoch 29/50
345/345              18s 52ms/step -
accuracy: 0.7876 - loss: 0.9184 - val_accuracy: 0.7900 - val_loss: 0.9193
Epoch 30/50
345/345              16s 46ms/step -
accuracy: 0.7851 - loss: 0.9170 - val_accuracy: 0.7693 - val_loss: 0.9448
```

```
Epoch 31/50
345/345             19s 56ms/step -
accuracy: 0.7961 - loss: 0.8937 - val_accuracy: 0.7847 - val_loss: 0.9126
Epoch 32/50
345/345             16s 46ms/step -
accuracy: 0.8031 - loss: 0.8880 - val_accuracy: 0.7923 - val_loss: 0.8883
Epoch 33/50
345/345             19s 55ms/step -
accuracy: 0.7948 - loss: 0.8906 - val_accuracy: 0.7843 - val_loss: 0.9122
Epoch 34/50
345/345             16s 47ms/step -
accuracy: 0.7957 - loss: 0.8809 - val_accuracy: 0.8060 - val_loss: 0.8578
Epoch 35/50
345/345             19s 55ms/step -
accuracy: 0.8061 - loss: 0.8708 - val_accuracy: 0.7693 - val_loss: 0.9256
Epoch 36/50
345/345             16s 48ms/step -
accuracy: 0.7998 - loss: 0.8646 - val_accuracy: 0.7983 - val_loss: 0.8543
Epoch 37/50
345/345             19s 55ms/step -
accuracy: 0.8014 - loss: 0.8579 - val_accuracy: 0.7773 - val_loss: 0.8929
Epoch 38/50
345/345             17s 50ms/step -
accuracy: 0.7971 - loss: 0.8637 - val_accuracy: 0.7817 - val_loss: 0.8797
Epoch 39/50
345/345             19s 54ms/step -
accuracy: 0.8071 - loss: 0.8506 - val_accuracy: 0.7843 - val_loss: 0.8952
Epoch 40/50
345/345             15s 45ms/step -
accuracy: 0.8056 - loss: 0.8435 - val_accuracy: 0.7850 - val_loss: 0.8862
Epoch 41/50
345/345             19s 53ms/step -
accuracy: 0.8092 - loss: 0.8330 - val_accuracy: 0.7770 - val_loss: 0.8985
Epoch 42/50
345/345             17s 49ms/step -
accuracy: 0.8050 - loss: 0.8409 - val_accuracy: 0.7893 - val_loss: 0.8632
Epoch 43/50
345/345             20s 57ms/step -
accuracy: 0.8116 - loss: 0.8290 - val_accuracy: 0.7970 - val_loss: 0.8492
Epoch 44/50
345/345             17s 50ms/step -
accuracy: 0.8162 - loss: 0.8223 - val_accuracy: 0.7787 - val_loss: 0.9015
Epoch 45/50
345/345             19s 54ms/step -
accuracy: 0.8147 - loss: 0.8233 - val_accuracy: 0.7230 - val_loss: 1.0055
Epoch 46/50
345/345             21s 60ms/step -
accuracy: 0.8109 - loss: 0.8214 - val_accuracy: 0.7993 - val_loss: 0.8313
```

```
Epoch 47/50
345/345              16s 45ms/step -
accuracy: 0.8149 - loss: 0.8134 - val_accuracy: 0.8003 - val_loss: 0.8389
Epoch 48/50
345/345              20s 57ms/step -
accuracy: 0.8117 - loss: 0.8128 - val_accuracy: 0.7687 - val_loss: 0.9084
Epoch 49/50
345/345              17s 48ms/step -
accuracy: 0.8175 - loss: 0.8061 - val_accuracy: 0.8057 - val_loss: 0.8178
Epoch 50/50
345/345              22s 63ms/step -
accuracy: 0.8126 - loss: 0.7987 - val_accuracy: 0.7853 - val_loss: 0.8561
```

5.5 Carregamento do modelo e validação

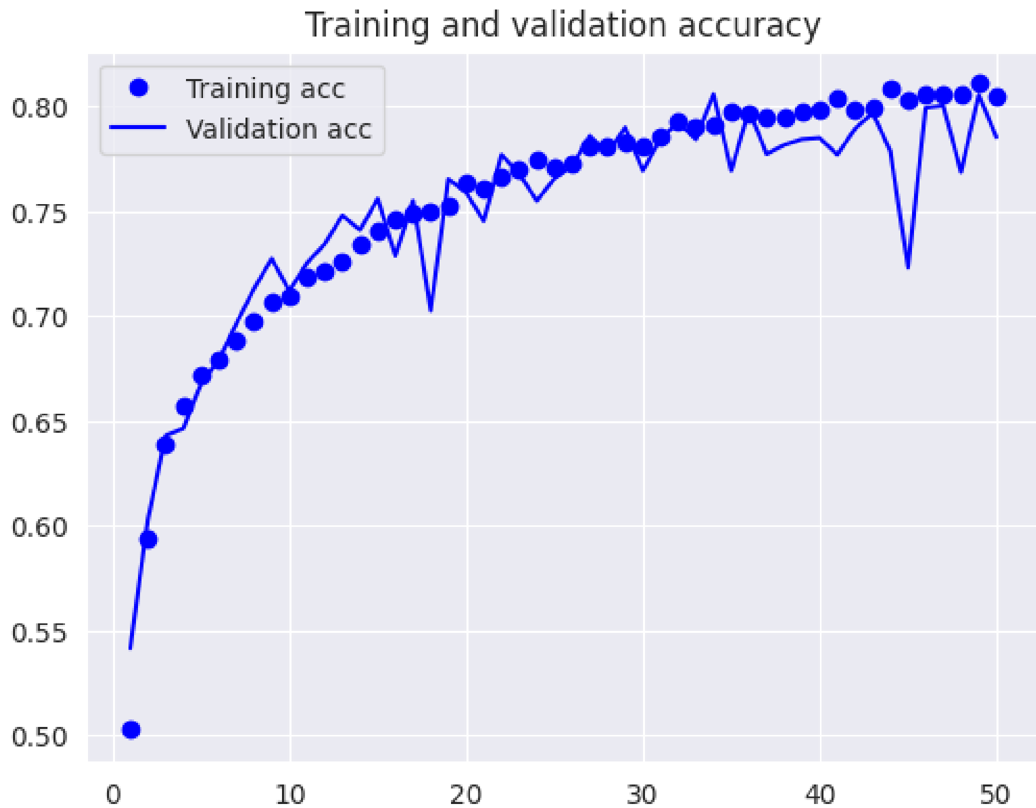
Carregamento e avaliação do modelo através do valor da accuracy.

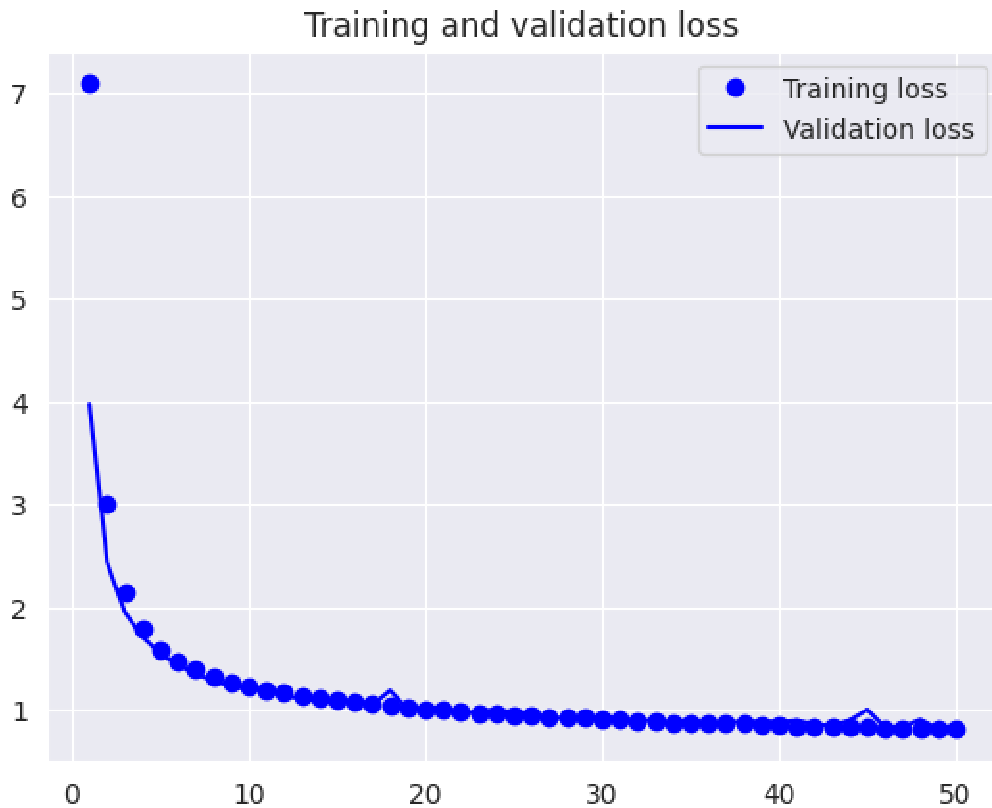
[34]:
```
model_both = keras.models.load_model('modelS_CatCross_RMS_dropout_L2.keras')
val_loss, val_acc = model_both.evaluate(validation_dataset)
print('val_acc:', val_acc)
```

```
94/94              4s 24ms/step -
accuracy: 0.8075 - loss: 0.8293
val_acc: 0.8056666851043701
```

Representação gráfica dos valores da accuracy e da loss ao longo das épocas.

[36]:
```
acc = history_both.history['accuracy']
val_acc = history_both.history['val_accuracy']
loss_CatCros = history_both.history['loss']
val_loss = history_both.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss_CatCros, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```


Training and validation accuracy
Training acc
Validation acc
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0
10
20
30
40
50



Avaliação da performance do modelo no conjunto de teste, utilizando o relatório de classificação. O relatório apresenta, para cada classe, as métricas precision, recall e F1-score, permitindo analisar detalhadamente os acertos e erros por classe.

[37]:
```
y_true, y_pred = get_true_pred(model_both, test_dataset)
report = classification_report(y_true, y_pred, target_names=class_names,␣
↪output_dict=True)
class_only_report = {k: v for k, v in report.items() if k in class_names}
df = pd.DataFrame(class_only_report).T
print(df[['precision', 'recall', 'f1-score']].round(3))
```

```
           precision  recall  f1-score
buildings      0.820   0.721     0.767
forest         0.853   0.958     0.903
glacier        0.766   0.817     0.791
mountain       0.815   0.737     0.774
sea            0.786   0.829     0.807
street         0.840   0.806     0.823
```

6 Avaliação do melhor modelo

6.1 Comparação dos modelos utilizando a accuracy

[38]:
```
val_loss_CatCross_RMS_dropout, val_acc_CatCross_RMS_dropout = model_dropout.
 ↪evaluate(validation_dataset)
val_loss_CatCross_RMS_l2, val_acc_CatCross_RMS_l2 = model_l2.
 ↪evaluate(validation_dataset)
val_loss_CatCross_RMS_dropout_l2, val_acc_CatCross_RMS_dropout_l2 = model_both.
 ↪evaluate(validation_dataset)

print("Validation Accuracy dos modelos:")
print(f"CatCross + RMSprop + Dropout: {val_loss_CatCross_RMS_dropout:.4f}")
print(f"CatCross + RMSprop + L2: {val_loss_CatCross_RMS_l2:.4f}")
print(f"CatCross + RMSprop + Dropout + L2: {val_loss_CatCross_RMS_dropout_l2:.
 ↪4f}")

results = {
    'CatCross_RMS_Dropout': val_acc_CatCross_RMS_dropout,
    'CatCross_RMS_L2': val_acc_CatCross_RMS_l2,
    'CatCross_RMS_Dropout_L2': val_acc_CatCross_RMS_dropout_l2
}

# Identificar o melhor modelo com base na maior val_accuracy
best_model = max(results, key=results.get)
best_accuracy = results[best_model]

print(f"\nMelhor modelo: {best_model} com val_accuracy = {best_accuracy:.4f}")
```

```
94/94               4s 35ms/step -
accuracy: 0.8360 - loss: 0.4948
94/94               3s 30ms/step - acc:
0.7997 - loss: 0.8191
94/94               3s 29ms/step -
accuracy: 0.8049 - loss: 0.8338
Validation Accuracy dos modelos:
CatCross + RMSprop + Dropout: 0.4568
CatCross + RMSprop + L2: 0.8033
CatCross + RMSprop + Dropout + L2: 0.8178

Melhor modelo: CatCross_RMS_Dropout com val_accuracy = 0.8503
```

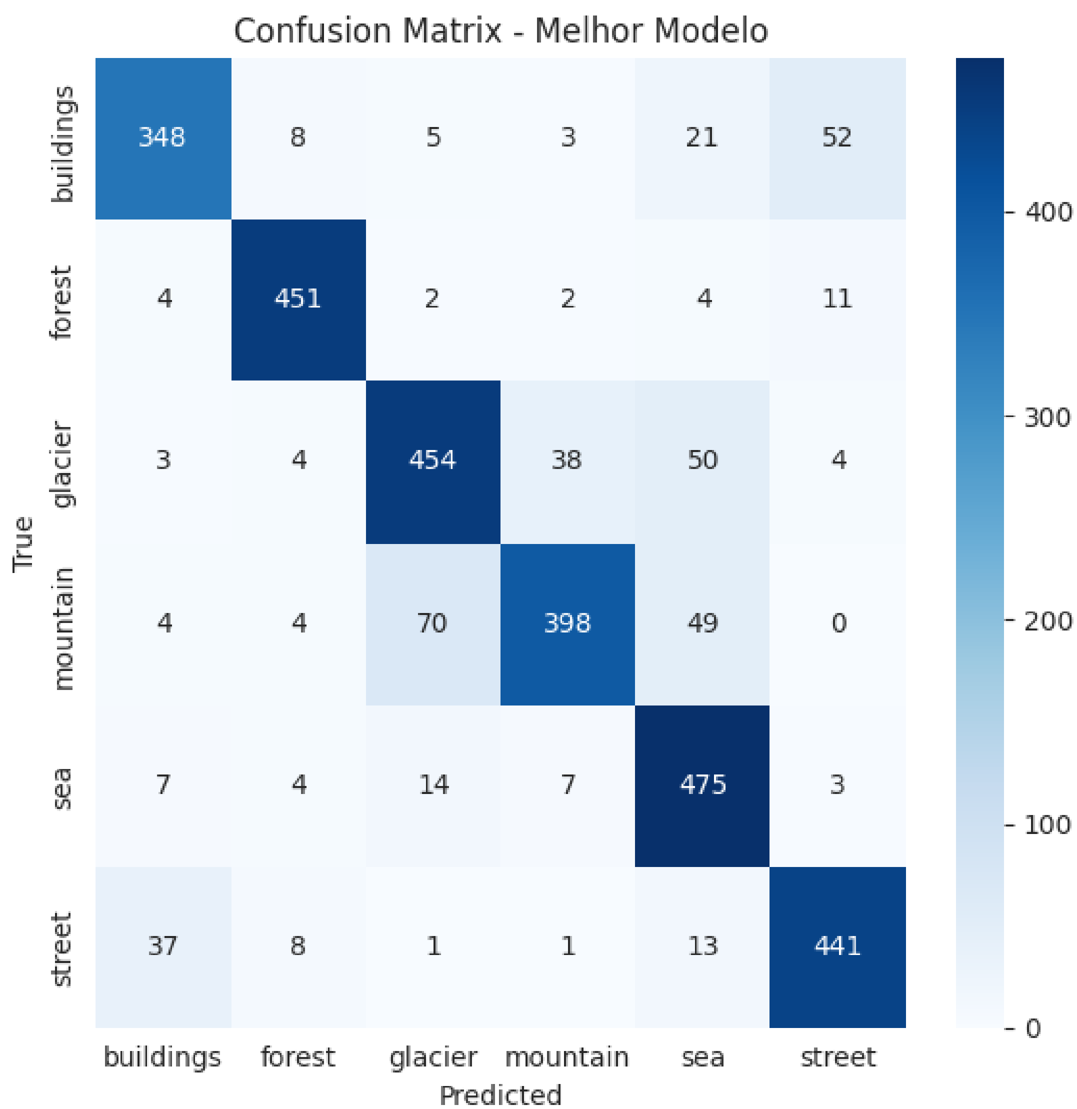
6.2 Matriz de confusão do melhor modelo

[39]:
```
y_true, y_pred = get_true_pred(model_dropout, test_dataset)
cm = confusion_matrix(y_true, y_pred)

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

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix - Melhor Modelo')
plt.tight_layout()
plt.show()
```

```
2025-06-12 22:52:36.305303: I tensorflow/core/framework/local_rendezvous.cc:407]
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```



6.3 Calcular saída do modelo para uma imagem

[40]:
```
img_path = 'Dataset/archive/seg_test/sea/20072.jpg'

img = tf.keras.preprocessing.image.load_img(
    img_path,
    target_size=(150, 150),
    interpolation='bilinear'
)

plt.imshow(img)
plt.axis('off')
plt.title("Imagem de Teste")
plt.show()

img_array = tf.keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)

# Previsão
result = model_dropout.predict(img_array)

class_names = ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
print("Probabilidades por classe:")
for i, prob in enumerate(result[0]):
    print(f"{class_names[i]:>10s}: {prob:.4f}")

# Classe prevista
predicted_class = np.argmax(result)
print(f"\nClasse prevista: {class_names[predicted_class]}␣
↪({result[0][predicted_class]:.4f})")
```


```
1/1              0s 41ms/step
Probabilidades por classe:
 buildings: 0.0069
    forest: 0.0004
   glacier: 0.0563
  mountain: 0.0155
       sea: 0.9200
    street: 0.0009

Classe prevista: sea (0.9200)
```