lab5-rishita-143

October 25, 2024

[1]:
```
import tensorflow as tf

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import os
import random
```

1 Loading the dataset

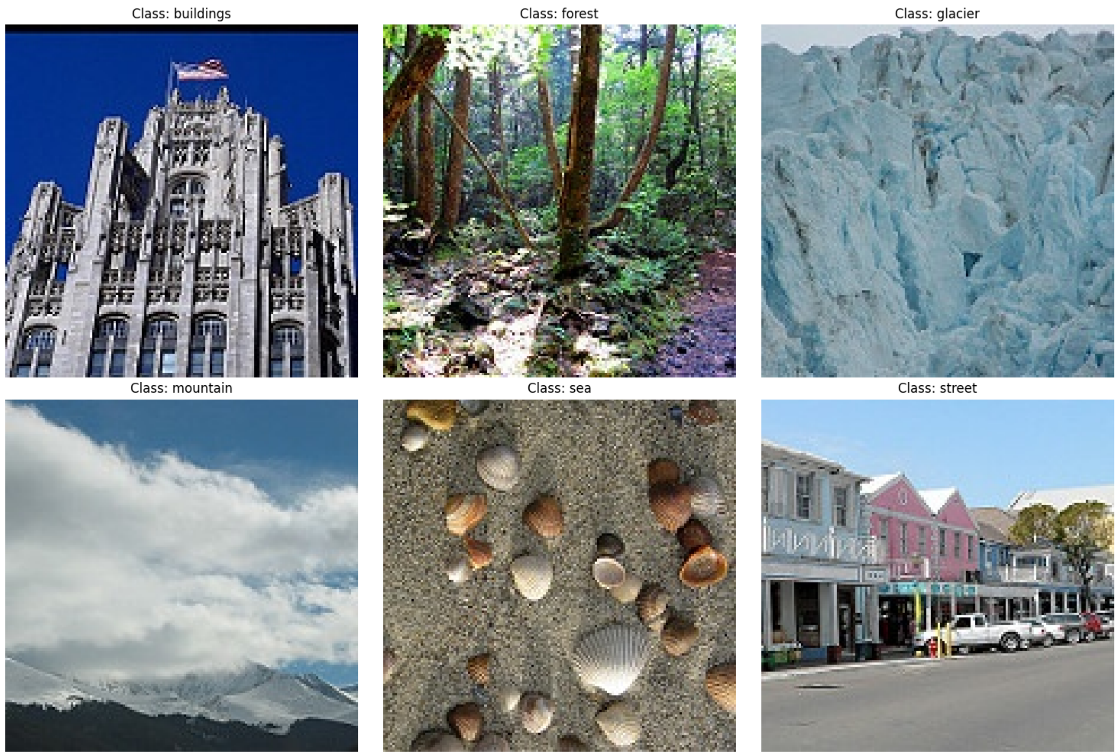
[2]:
```
# Define paths for your data folders
TRAIN_DIR = 'C:/Users/shikh/Desktop/5th trimester/NNDL/seg_train/seg_train'
TEST_DIR = 'C:/Users/shikh/Desktop/5th trimester/NNDL/seg_test/seg_test'
PRED_DIR = 'C:/Users/shikh/Desktop/5th trimester/NNDL/seg_pred/seg_pred'
```

[3]:
```
# Constants
IMG_HEIGHT = 150
IMG_WIDTH = 150
BATCH_SIZE = 32
NUM_CLASSES = 6
EPOCHS = 20

# Class mapping
class_names = {
    'buildings': 0,
    'forest': 1,
    'glacier': 2,
    'mountain': 3,
    'sea': 4,
    'street': 5
}
```

2 Displaying few images from the dataset(training set)

[4]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img,␣
 ↪img_to_array
# Function to display random samples from each class
def display_sample_images():
    plt.figure(figsize=(15, 10))
    for idx, class_name in enumerate(class_names.keys()):
        # Get path to class folder in training set
        class_path = os.path.join(TRAIN_DIR, class_name)
        # Get random image from class
        images = os.listdir(class_path)
        random_image = random.choice(images)
        img_path = os.path.join(class_path, random_image)

        # Load and display image
        img = load_img(img_path)
        plt.subplot(2, 3, idx + 1)
        plt.imshow(img)
        plt.title(f'Class: {class_name}')
        plt.axis('off')

    plt.tight_layout()
    plt.show()

# Display sample images
print("Displaying sample images from each class:")
display_sample_images()
```

```
Displaying sample images from each class:
```



3 Model Architecture:

[5]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator

model = Sequential()

# First Convolutional Layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

# Second Convolutional Layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

# Third Convolutional Layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(BatchNormalization())

# Flatten and Fully Connected Layers
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))
```

```
c:\Users\shikh\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:99: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(
```

- Three convolutional layers are used with ReLU activation and MaxPooling.
- Batch normalization is applied to speed up training and make it more stable.
- Dropout helps prevent overfitting by randomly disabling some neurons.

[6]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

- Adam optimizer is used for efficient gradient-based optimization.
- Categorical crossentropy is the loss function as this is a multi-class classification problem.

[7]:
```
# Data augmentation for training set
train_datagen = ImageDataGenerator(
    rescale=1./255

)

# Data generator for validation set (without augmentation)
test_datagen = ImageDataGenerator(rescale=1./255)

# Load training and validation data
train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)

validation_generator = test_datagen.flow_from_directory(
    TEST_DIR,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='categorical'
)
```

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

- Data augmentation in `train_datagen` introduces variability and prevents overfitting.
- Both generators rescale pixel values to range [0, 1] for faster learning.

4 Model Training:

[8]:
```python
# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=None,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=None
)
```

```
Epoch 1/20
```

```
c:\Users\shikh\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:122:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

```
439/439              152s 341ms/step -
accuracy: 0.4429 - loss: 5.0966 - val_accuracy: 0.4920 - val_loss: 2.2417
Epoch 2/20
439/439              285s 650ms/step -
accuracy: 0.5745 - loss: 1.2402 - val_accuracy: 0.7207 - val_loss: 0.8987
Epoch 3/20
439/439              325s 738ms/step -
accuracy: 0.6455 - loss: 0.9499 - val_accuracy: 0.7397 - val_loss: 0.8335
Epoch 4/20
439/439              862s 2s/step -
accuracy: 0.6707 - loss: 0.8868 - val_accuracy: 0.7373 - val_loss: 0.8644
Epoch 5/20
439/439              210s 477ms/step -
accuracy: 0.7055 - loss: 0.8192 - val_accuracy: 0.4853 - val_loss: 2.5290
Epoch 6/20
439/439              130s 297ms/step -
accuracy: 0.7471 - loss: 0.7086 - val_accuracy: 0.7550 - val_loss: 0.7245
Epoch 7/20
439/439              185s 421ms/step -
accuracy: 0.7646 - loss: 0.6454 - val_accuracy: 0.7717 - val_loss: 0.6527
Epoch 8/20
```

```
439/439              145s 330ms/step -
accuracy: 0.7846 - loss: 0.6062 - val_accuracy: 0.7897 - val_loss: 0.6331
Epoch 9/20
439/439              283s 645ms/step -
accuracy: 0.7883 - loss: 0.5758 - val_accuracy: 0.7840 - val_loss: 0.6630
Epoch 10/20
439/439              103s 234ms/step -
accuracy: 0.8061 - loss: 0.5239 - val_accuracy: 0.7467 - val_loss: 0.8786
Epoch 11/20
439/439              112s 255ms/step -
accuracy: 0.8282 - loss: 0.4807 - val_accuracy: 0.8160 - val_loss: 0.5720
Epoch 12/20
439/439              111s 252ms/step -
accuracy: 0.8456 - loss: 0.4252 - val_accuracy: 0.8203 - val_loss: 0.5436
Epoch 13/20
439/439              117s 266ms/step -
accuracy: 0.8488 - loss: 0.4085 - val_accuracy: 0.7897 - val_loss: 0.7170
Epoch 14/20
439/439              335s 763ms/step -
accuracy: 0.8657 - loss: 0.3707 - val_accuracy: 0.7027 - val_loss: 0.8941
Epoch 15/20
439/439              107s 243ms/step -
accuracy: 0.8773 - loss: 0.3379 - val_accuracy: 0.7823 - val_loss: 0.6937
Epoch 16/20
439/439              683s 2s/step -
accuracy: 0.8813 - loss: 0.3104 - val_accuracy: 0.8187 - val_loss: 0.7942
Epoch 17/20
439/439              395s 901ms/step -
accuracy: 0.8971 - loss: 0.2854 - val_accuracy: 0.8017 - val_loss: 0.7633
Epoch 18/20
439/439              104s 236ms/step -
accuracy: 0.9020 - loss: 0.2725 - val_accuracy: 0.7743 - val_loss: 0.7907
Epoch 19/20
439/439              109s 248ms/step -
accuracy: 0.9168 - loss: 0.2411 - val_accuracy: 0.8343 - val_loss: 0.6153
Epoch 20/20
439/439              112s 255ms/step -
accuracy: 0.9145 - loss: 0.2362 - val_accuracy: 0.8013 - val_loss: 0.6799
```

- The model trains over a specified number of epochs (20) to improve accuracy.
- `steps_per_epoch` and `validation_steps` are used to iterate over the entire dataset.

5 Evaluate the model

[9]:
```python
test_loss, test_accuracy = model.evaluate(validation_generator)
print(f'Test accuracy: {test_accuracy:.2f}')
```

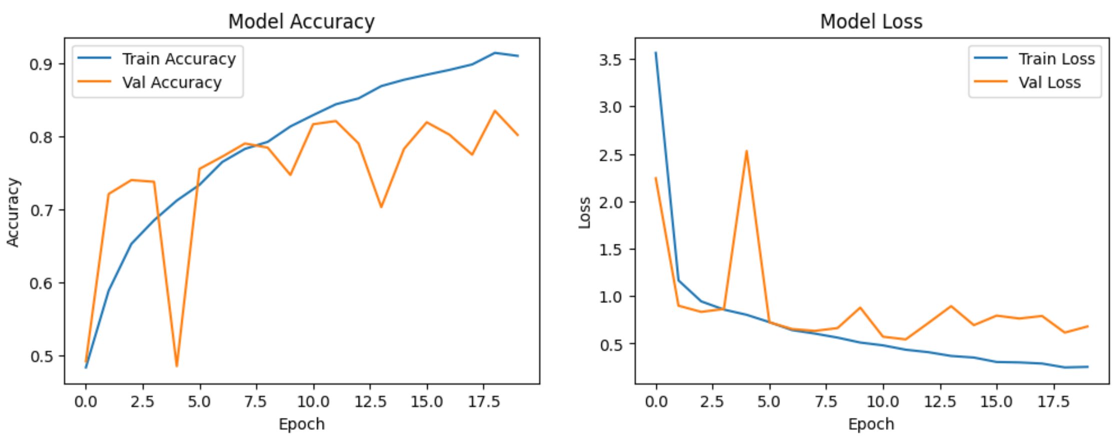
```
94/94              6s 63ms/step -
```

```
accuracy: 0.7981 - loss: 0.7125
Test accuracy: 0.80
```

- Test accuracy gives an indication of how well the model generalizes to unseen data.

6 Plot Accuracy and Loss Curves

[10]:
```python
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')

plt.show()
```

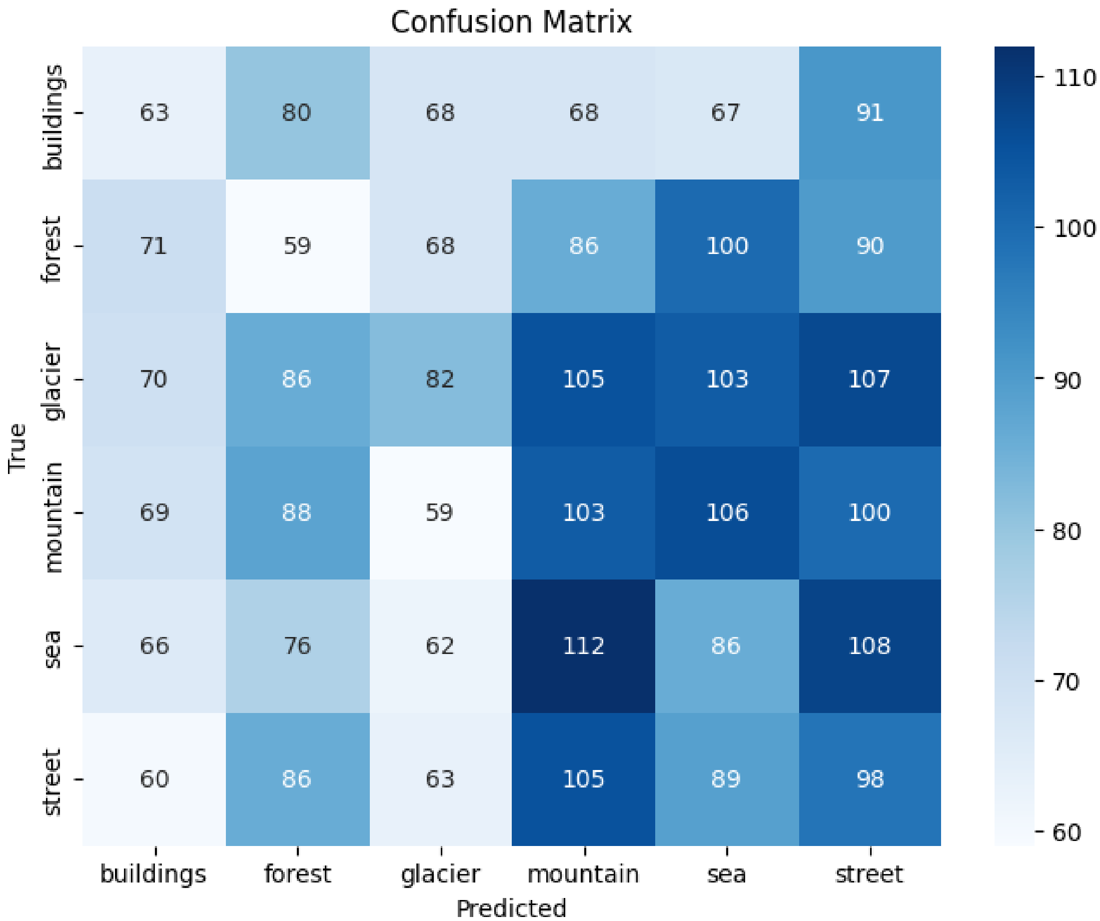


- These plots show how the model accuracy and loss evolve over time.
- Helps to spot overfitting (if validation loss starts increasing while training loss decreases).

[11]:

```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import seaborn as sns

# Generate predictions and true labels
Y_pred = model.predict(validation_generator)
y_pred = np.argmax(Y_pred, axis=1)
y_true = validation_generator.classes

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names.
 ↪keys(), yticklabels=class_names.keys())
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Classification report
print(classification_report(y_true, y_pred, target_names=class_names.keys()))
```

```
94/94                  6s 62ms/step
```



	precision	recall	f1-score	support
buildings	0.16	0.14	0.15	437
forest	0.12	0.12	0.12	474
glacier	0.20	0.15	0.17	553
mountain	0.18	0.20	0.19	525
sea	0.16	0.17	0.16	510
street	0.16	0.20	0.18	501
accuracy			0.16	3000
macro avg	0.16	0.16	0.16	3000
weighted avg	0.17	0.16	0.16	3000

- The confusion matrix shows the distribution of true vs. predicted labels.
- The classification report includes precision, recall, and F1-score for each class.

7 5. Optimization :

[22]:
```
# Enhanced data augmentation for training set
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,          # Rotate images by up to 30 degrees
    width_shift_range=0.2,      # Horizontal shift by up to 20%
    height_shift_range=0.2,     # Vertical shift by up to 20%
    shear_range=0.2,            # Shear intensity (slanting)
    zoom_range=0.2,             # Zoom in by up to 20%

)
```

[23]:
```
# No augmentation for validation set
test_datagen = ImageDataGenerator(rescale=1./255)
```

[24]:
```
# Load training and validation data with new augmentation settings
train_generator = train_datagen.flow_from_directory(
    TRAIN_DIR,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=32,
    class_mode='categorical'
)
validation_generator = test_datagen.flow_from_directory(
    TEST_DIR,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=32, # Changed the batch size to 16
    class_mode='categorical'
)
```

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

[25]:
```
from tensorflow.keras.optimizers import Adam

# Rebuild model with more filters per layer
model = Sequential()

# First Convolutional Layer with 32 filters
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(IMG_HEIGHT,␣
↪IMG_WIDTH, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

# Second Convolutional Layer with 64 filters
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(BatchNormalization())

# Third Convolutional Layer with 128 filters
model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())

# Flatten and Fully Connected Layers
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))  # To avoid overfitting
model.add(Dense(NUM_CLASSES, activation='softmax'))

# Compile the model with a lower learning rate
model.compile(optimizer=Adam(learning_rate=0.0001),␣
↪loss='categorical_crossentropy', metrics=['accuracy'])
```

[26]:
```
# Train the model with enhanced data augmentation and new hyperparameters
history = model.fit(
    train_generator,
    steps_per_epoch=None,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=None
)
```

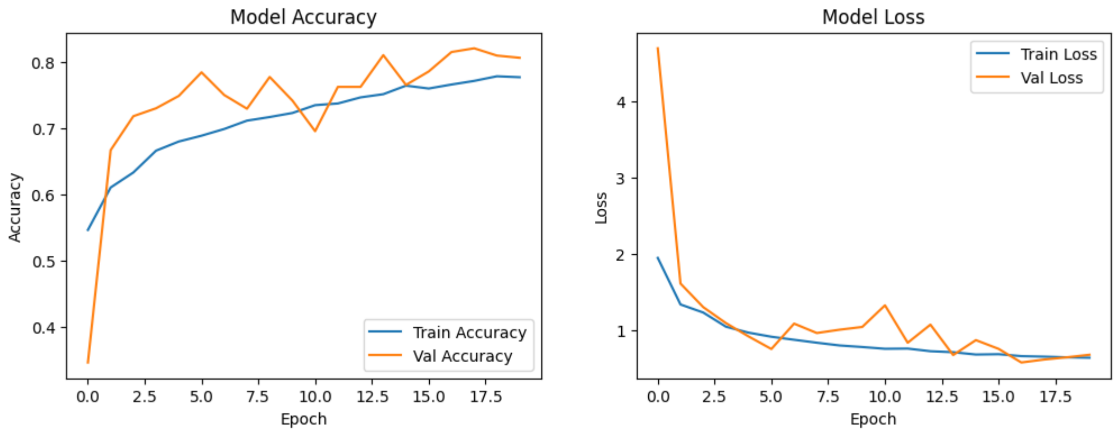
```
Epoch 1/20
439/439              419s 949ms/step -
accuracy: 0.5080 - loss: 2.3390 - val_accuracy: 0.3463 - val_loss: 4.6967
Epoch 2/20
439/439              690s 2s/step -
accuracy: 0.6032 - loss: 1.3870 - val_accuracy: 0.6673 - val_loss: 1.6094
Epoch 3/20
439/439              1099s 3s/step -
accuracy: 0.6324 - loss: 1.2374 - val_accuracy: 0.7187 - val_loss: 1.2998
Epoch 4/20
439/439              2578s 6s/step -
accuracy: 0.6618 - loss: 1.0590 - val_accuracy: 0.7307 - val_loss: 1.0901
Epoch 5/20
439/439              368s 836ms/step -
accuracy: 0.6845 - loss: 0.9403 - val_accuracy: 0.7493 - val_loss: 0.9147
Epoch 6/20
439/439              399s 906ms/step -
accuracy: 0.6770 - loss: 0.9391 - val_accuracy: 0.7850 - val_loss: 0.7501
Epoch 7/20
439/439              424s 965ms/step -
accuracy: 0.7047 - loss: 0.8536 - val_accuracy: 0.7507 - val_loss: 1.0833
```

```
Epoch 8/20
439/439              409s 928ms/step -
accuracy: 0.7130 - loss: 0.8138 - val_accuracy: 0.7300 - val_loss: 0.9598
Epoch 9/20
439/439              426s 968ms/step -
accuracy: 0.7120 - loss: 0.8094 - val_accuracy: 0.7780 - val_loss: 1.0037
Epoch 10/20
439/439              522s 1s/step -
accuracy: 0.7274 - loss: 0.7611 - val_accuracy: 0.7423 - val_loss: 1.0398
Epoch 11/20
439/439              431s 980ms/step -
accuracy: 0.7367 - loss: 0.7440 - val_accuracy: 0.6960 - val_loss: 1.3237
Epoch 12/20
439/439              441s 1s/step -
accuracy: 0.7279 - loss: 0.7804 - val_accuracy: 0.7630 - val_loss: 0.8323
Epoch 13/20
439/439              433s 983ms/step -
accuracy: 0.7472 - loss: 0.7152 - val_accuracy: 0.7630 - val_loss: 1.0701
Epoch 14/20
439/439              562s 1s/step -
accuracy: 0.7516 - loss: 0.7199 - val_accuracy: 0.8110 - val_loss: 0.6708
Epoch 15/20
439/439              645s 1s/step -
accuracy: 0.7602 - loss: 0.6774 - val_accuracy: 0.7660 - val_loss: 0.8668
Epoch 16/20
439/439              1023s 2s/step -
accuracy: 0.7616 - loss: 0.6758 - val_accuracy: 0.7863 - val_loss: 0.7522
Epoch 17/20
439/439              1084s 2s/step -
accuracy: 0.7665 - loss: 0.6589 - val_accuracy: 0.8157 - val_loss: 0.5731
Epoch 18/20
439/439              447s 1s/step -
accuracy: 0.7723 - loss: 0.6442 - val_accuracy: 0.8213 - val_loss: 0.6117
Epoch 19/20
439/439              654s 1s/step -
accuracy: 0.7738 - loss: 0.6486 - val_accuracy: 0.8103 - val_loss: 0.6412
Epoch 20/20
439/439              410s 931ms/step -
accuracy: 0.7752 - loss: 0.6403 - val_accuracy: 0.8070 - val_loss: 0.6748
```

[27]:
```python
# Evaluate the model
test_loss, test_accuracy = model.evaluate(validation_generator)
print(f'Optimized Test accuracy: {test_accuracy:.2f}')

# Plot training & validation accuracy/loss values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
```

```
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')

plt.show()
```

```
94/94                22s 238ms/step -
accuracy: 0.8070 - loss: 0.7005
Optimized Test accuracy: 0.81
```

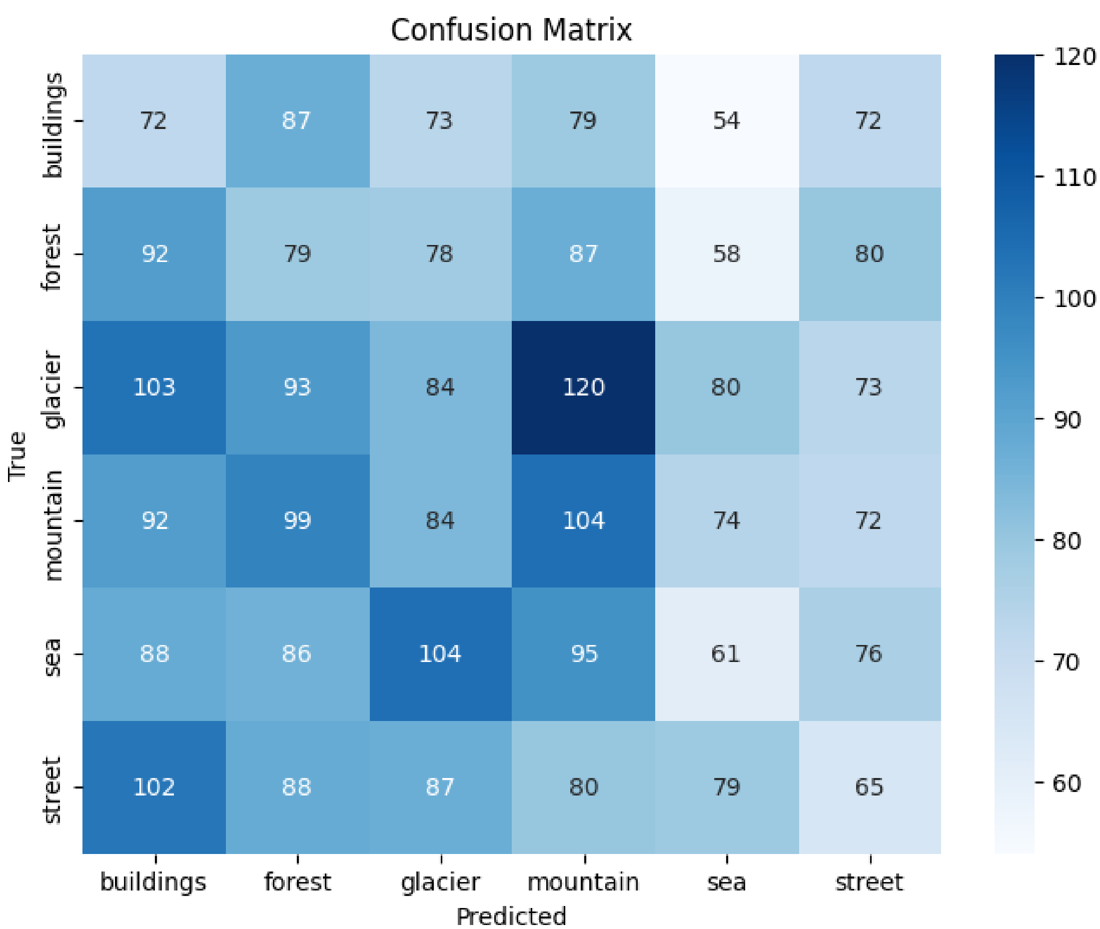


- After Optimizing the accuracy we get on the test data is **0.81**

[28]:
```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np
import seaborn as sns

# Generate predictions and true labels
Y_pred = model.predict(validation_generator)
y_pred = np.argmax(Y_pred, axis=1)
y_true = validation_generator.classes
```

```
# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names.
↪keys(), yticklabels=class_names.keys())
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Classification report
print(classification_report(y_true, y_pred, target_names=class_names.keys()))
```

```
94/94                 18s 187ms/step
```




```
              precision    recall  f1-score   support

   buildings       0.13      0.16      0.15       437
```

forest	0.15	0.17	0.16	474
glacier	0.16	0.15	0.16	553
mountain	0.18	0.20	0.19	525
sea	0.15	0.12	0.13	510
street	0.15	0.13	0.14	501
accuracy			0.15	3000
macro avg	0.15	0.16	0.15	3000
weighted avg	0.16	0.15	0.15	3000