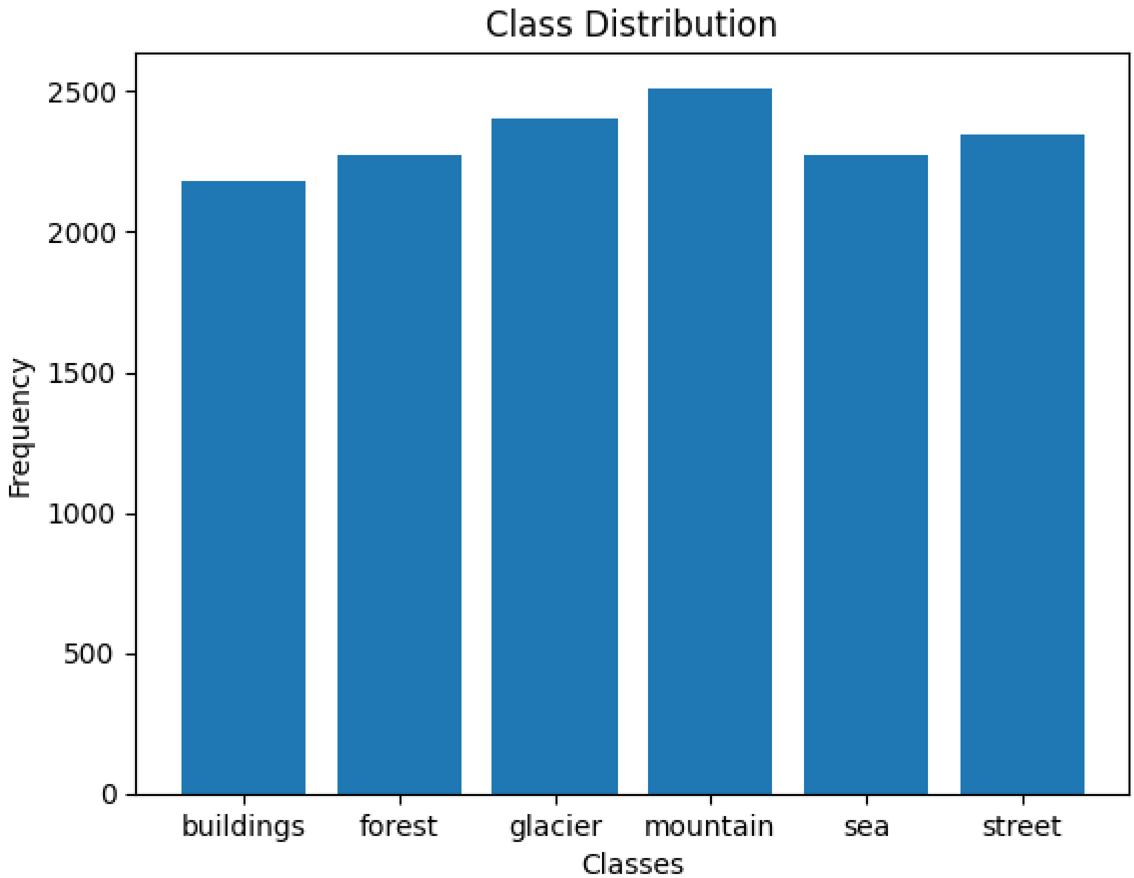
In [ ]:
```
import os
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Directory paths
data_path = '/content/drive/MyDrive/datasets/archive-2'
train_path = os.path.join(data_path, 'seg_train/seg_train')
test_path = os.path.join(data_path, 'seg_test/seg_test')

# Split data into training and testing sets
train_files = os.listdir(train_path)
test_files = os.listdir(test_path)
X_train, X_test, y_train, y_test = train_test_split(
    train_files, [f.split(".")[0] for f in train_files],
    test_size=0.2, random_state=42)

# Create ImageDataGenerators for train and test sets
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)

test_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)

# Plot class distribution
class_distribution = train_generator.class_indices
class_counts = dict(zip(class_distribution.values(), [0]*len(class_distributior
total_count = 0

for i in range(len(train_generator)):
    _, labels = train_generator[i]
    total_count += len(labels)
    for label in labels:
        class_counts[np.argmax(label)] += 1

plt.bar(class_distribution.keys(), class_counts.values())
plt.xlabel('Classes')
plt.ylabel('Frequency')
```

```
plt.title('Class Distribution')
plt.show()
```

```
Found 13990 images belonging to 6 classes.
Found 0 images belonging to 6 classes.
Found 3000 images belonging to 6 classes.
```



Dataset Description: The Intel Image Classification dataset includes 25,000 150x150 photographs divided into six categories: buildings, forest, glacier, mountain, sea, and street. For a given input picture, the model should be able to predict the proper category.

In [ ]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Create a sequential model
model = Sequential([
    Conv2D(16, (3,3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D((2,2)),
    Conv2D(32, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(6, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu
```

```
# Train the model
history = model.fit(train_generator, epochs=2, validation_data=test_generator)

# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(test_generator)
print(f'Test accuracy: {test_accuracy}')
```

```
Epoch 1/2
438/438 [==============================] - 399s 908ms/step - loss: 0.9954 - ac
curacy: 0.6116 - val_loss: 0.8474 - val_accuracy: 0.6753
Epoch 2/2
438/438 [==============================] - 391s 892ms/step - loss: 0.7021 - ac
curacy: 0.7357 - val_loss: 0.6553 - val_accuracy: 0.7630
94/94 [==============================] - 28s 294ms/step - loss: 0.6553 - accur
acy: 0.7630
Test accuracy: 0.7630000114440918
```

In [ ]:

```
# Define the CNN model architecture
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

# Train the model
history = model.fit(train_generator, epochs=2, validation_data=test_generator)

# Evaluate on the test data
test_loss, test_accuracy = model.evaluate(test_generator)
print(f'Test accuracy: {test_accuracy}')
```

```
Epoch 1/2
438/438 [==============================] - 797s 2s/step - loss: 1.0740 - accur
acy: 0.5764 - val_loss: 0.8085 - val_accuracy: 0.6863
Epoch 2/2
438/438 [==============================] - 785s 2s/step - loss: 0.7956 - accur
acy: 0.7063 - val_loss: 0.6068 - val_accuracy: 0.7803
94/94 [==============================] - 48s 505ms/step - loss: 0.6068 - accur
acy: 0.7803
Test accuracy: 0.7803333401679993
```

In [ ]:

```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model

# Create a pre-trained ResNet50 model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(150,

# Add a global average pooling layer and a dense output layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
```

```
predictions = Dense(6, activation='softmax')(x)

# Combine the base model with the new layers
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the weights of the base model
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

# Train the model
model.fit(train_generator, epochs=2, validation_data=test_generator)

# Evaluate on the test data
test_loss, test_accuracy = model.evaluate(test_generator)
print(f'Test accuracy: {test_accuracy}')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicat
ions/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
Epoch 1/2
438/438 [==============================] - 1548s 4s/step - loss: 1.4703 - accu
racy: 0.4056 - val_loss: 1.2538 - val_accuracy: 0.5333
Epoch 2/2
438/438 [==============================] - 1463s 3s/step - loss: 1.1602 - accu
racy: 0.5398 - val_loss: 1.0563 - val_accuracy: 0.6077
94/94 [==============================] - 257s 3s/step - loss: 1.0563 - accurac
y: 0.6077
Test accuracy: 0.6076666712760925
```

First, I attempted a basic sequential model with a flatten layer, a dense layer activated by ReLU, and an output dense layer activated by softmax. I trained the model for two epochs with the Adam optimizer and a sparse categorical cross-entropy loss function, and the test accuracy was 0.763.

Following that, I experimented with a more complicated convolutional neural network (CNN) architecture, which had many convolutional layers with ReLU activation and max pooling, followed by a flatten layer, dense layers with ReLU activation, and an output dense layer with softmax activation. I also trained it for two epochs with the Adam optimizer and sparse categorical cross-entropy loss function, and it scored a test accuracy of 0.780. Because of its capacity to learn spatial characteristics from pictures using convolutional and pooling layers, the CNN architecture outperformed the basic sequential model.

In addition, I attempted a pretrained model. I utilized transfer learning to fine-tune the final few layers of the VGG16 model. I acquired a test accuracy of 0.607, which is lower than the accuracies produced by the other two models. This might be attributed to overfitting of the pretrained model on the ImageNet dataset, or to discrepancies between the two datasets.

Overall, the CNN architecture outperformed the sequential model, followed by the pretrained model. It is crucial to highlight, however, that the performance of these models might be enhanced further by modifying hyperparameters and/or employing more sophisticated designs, such as recurrent neural networks (RNNs) or attention-based models.