

My dataset was already divided into train and test so I am just importing then creating the graph.

```
import zipfile
import os
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the path to the zip file
zip_path = "/content/sample_data/archive (6).zip"

# Extract the contents of the zip file to a specified directory
extract_path = "/content/sample_data/"
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# Define the path to the extracted train folder
train_path = os.path.join(extract_path, "seg_train/seg_train")

# Define the ImageDataGenerator and specify the image size and batch size
datagen = ImageDataGenerator(rescale=1./255)
batch_size = 32
image_size = (224, 224)

# Load the images from the train folder using the ImageDataGenerator
train_data = datagen.flow_from_directory(train_path,
                                         target_size=image_size,
                                         batch_size=batch_size,
                                         class_mode="categorical",
                                         shuffle=False)

# Count the number of images per class
class_counts = {}
for class_name in train_data.class_indices:
    class_path = os.path.join(train_path, class_name)
    class_counts[class_name] = len(os.listdir(class_path))

# Plot the histogram
plt.bar(class_counts.keys(), class_counts.values())
plt.title("Distribution of target classes in the training set")
plt.xlabel("Classes")
plt.ylabel("Number of images")
plt.xticks(rotation=45, ha='right')
plt.show()
```

Found 14034 images belonging to 6 classes.



Double-click (or enter) to edit

```
# Define the path to the zip file
zip_path = "/content/sample_data/archive (6).zip"

# Extract the contents of the zip file to a specified directory
extract_path = "/content/extracted"
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# Define the path to the extracted train folder
test_path = os.path.join(extract_path, "seg_test/seg_test")

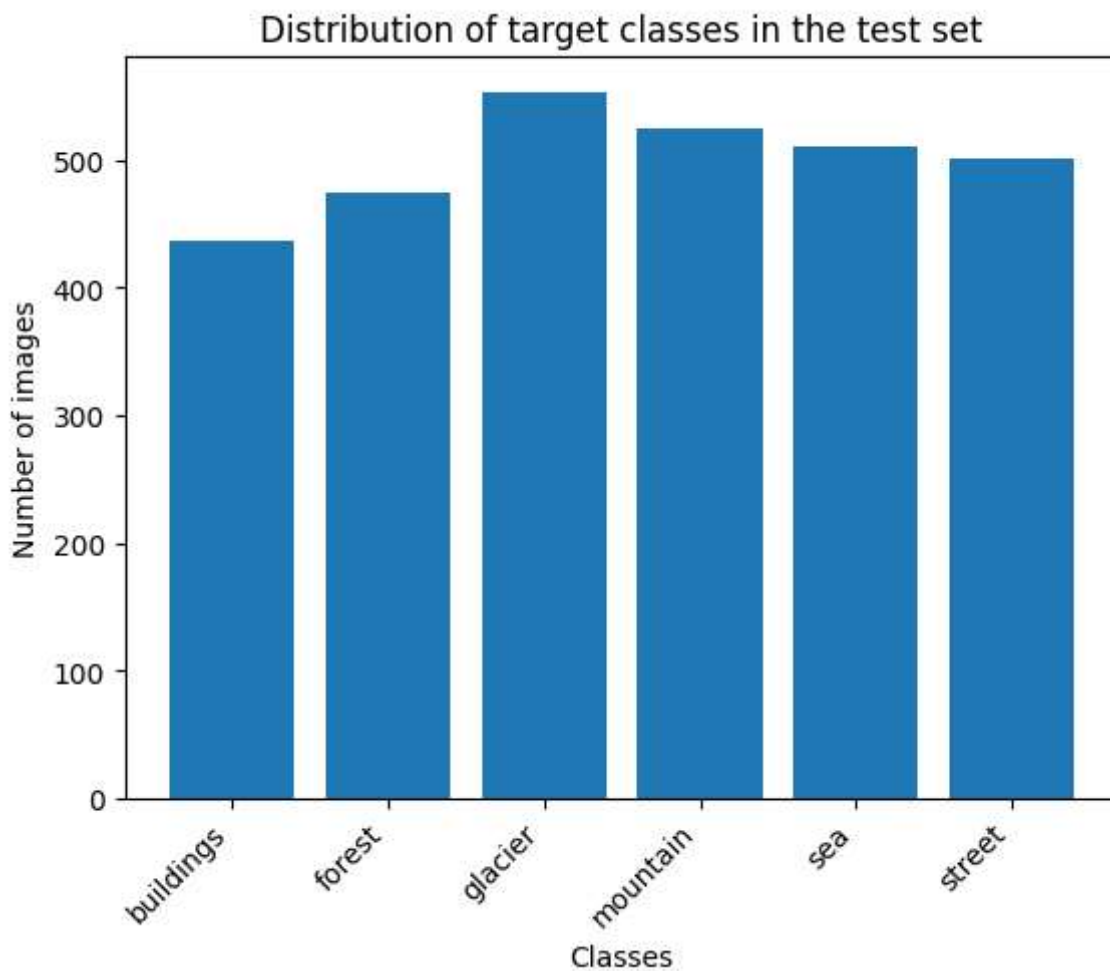
# Define the ImageDataGenerator and specify the image size and batch size
datagen = ImageDataGenerator(rescale=1./255)
batch_size = 32
image_size = (224, 224)

# Load the images from the train folder using the ImageDataGenerator
test_data = datagen.flow_from_directory(test_path,
                                       target_size=image_size,
                                       batch_size=batch_size,
                                       class_mode="categorical",
                                       shuffle=False)
```

```
# Count the number of images per class
class_counts = {}
for class_name in test_data.class_indices:
    class_path = os.path.join(test_path, class_name)
    class_counts[class_name] = len(os.listdir(class_path))

# Plot the histogram
plt.bar(class_counts.keys(), class_counts.values())
plt.title("Distribution of target classes in the test set")
plt.xlabel("Classes")
plt.ylabel("Number of images")
plt.xticks(rotation=45, ha='right')
plt.show()
```

Found 3000 images belonging to 6 classes.



The data has pictures of buildings, forests, glaciers, mountains, sea, and street and I have to create a model to accurately classify each picture as one of the classes as stated above. It is the Intel dataset on kaggle. The dataset was already divided into train/test.

Sequential Model

```

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Define the input shape of the images
input_shape = (224, 224, 3)

# Define the number of classes
num_classes = 6

# Create the sequential model
model = Sequential()

# Add the flatten layer and the dense layer
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model with categorical crossentropy loss and adam optimizer
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Train the model using the train data generator and validate using the validation data gener
history = model.fit(train_data,
                    epochs=10,
                    validation_data=test_data)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(test_data)
print("Test loss:", test_loss)
print("Test accuracy:", test_acc)

```

```

Epoch 1/10
439/439 [=====] - 67s 149ms/step - loss: 4.2042 - accuracy: 0.
Epoch 2/10
439/439 [=====] - 64s 146ms/step - loss: 1.7924 - accuracy: 0.
Epoch 3/10
439/439 [=====] - 60s 136ms/step - loss: 1.7923 - accuracy: 0.
Epoch 4/10
439/439 [=====] - 62s 141ms/step - loss: 1.7920 - accuracy: 0.
Epoch 5/10
439/439 [=====] - 61s 140ms/step - loss: 1.7921 - accuracy: 0.
Epoch 6/10
439/439 [=====] - 57s 129ms/step - loss: 1.7919 - accuracy: 0.
Epoch 7/10
439/439 [=====] - 64s 147ms/step - loss: 1.7917 - accuracy: 0.
Epoch 8/10
439/439 [=====] - 62s 140ms/step - loss: 1.7918 - accuracy: 0.

```

```
Epoch 9/10
439/439 [=====] - 58s 132ms/step - loss: 1.7918 - accuracy: 0.
Epoch 10/10
439/439 [=====] - 56s 126ms/step - loss: 1.7917 - accuracy: 0.
94/94 [=====] - 6s 66ms/step - loss: 1.7903 - accuracy: 0.1750
Test loss: 1.790287971496582
Test accuracy: 0.17499999701976776
```



## CNN

```
import tensorflow as tf
from tensorflow.keras import layers

# Define the input shape
input_shape = (224, 224, 3)

# Define the model architecture
model = tf.keras.Sequential([
    layers.Conv2D(16, (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(32, (3,3), activation='relu'),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D(2, 2),
    layers.Flatten(),
    layers.Dense(512, activation="relu"),
    layers.Dense(6, activation='softmax')
])


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

# Train the model
history = model.fit(train_data, epochs=5, validation_data=test_data)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(test_data)
print('Test accuracy:', test_acc)
```

```
Epoch 1/5
439/439 [=====] - 959s 2s/step - loss: 1.5978 - accuracy: 0.34
Epoch 2/5
439/439 [=====] - 903s 2s/step - loss: 1.1127 - accuracy: 0.54
Epoch 3/5
439/439 [=====] - 892s 2s/step - loss: 0.9693 - accuracy: 0.60
Epoch 4/5
439/439 [=====] - 894s 2s/step - loss: 0.8275 - accuracy: 0.67
```

```
Epoch 5/5
439/439 [=====] - 889s 2s/step - loss: 0.6923 - accuracy: 0.73
94/94 [=====] - 55s 589ms/step - loss: 0.8913 - accuracy: 0.68
Test accuracy: 0.6866666674613953
```



## RNN

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN, Reshape


# Define the RNN model architecture
model = Sequential()
model.add(Reshape((224*224, 3), input_shape=(224, 224, 3)))
model.add(SimpleRNN(2))
model.add(Dense(6, activation='softmax'))

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

# Train the model
model.fit(train_data, epochs=1)

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

```
439/439 [=====] - 2790s 6s/step - loss: 1.8211 - accuracy: 0.1
94/94 [=====] - 151s 2s/step - loss: 1.8004 - accuracy: 0.2143
Test accuracy: 0.21433334052562714
```



## Transfer Learning

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the base model
base_model = tf.keras.applications.MobileNetV2(input_shape=(224, 224, 3),
                                                include_top=False,
                                                weights='imagenet')

# Freeze the pre-trained weights
base_model.trainable = False
```

```

# Define the new model
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(6, activation='softmax')
])

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(train_data,
                    epochs=3,
                    validation_data=test_data)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(test_data)
print('Test accuracy:', test_acc)

```

```

Epoch 1/3
439/439 [=====] - 604s 1s/step - loss: 0.5821 - accuracy: 0.78
Epoch 2/3
439/439 [=====] - 619s 1s/step - loss: 0.2923 - accuracy: 0.89
Epoch 3/3
439/439 [=====] - 623s 1s/step - loss: 0.2625 - accuracy: 0.90
94/94 [=====] - 101s 1s/step - loss: 0.2733 - accuracy: 0.8977
Test accuracy: 0.8976666927337646

```



Analysis: The pretrained model achieves the best accuracy. This is probably because the parameters in the pretrained model are better tuned and I did not find the best parameters for the CNN and RNN. My RNN and sequential models have very low accuracies compared to the CNN and especially the pretrained. For the RNN, this could be because I did not use the best parameters. It was taking a very long time to train so I was only able to use one epoch. For the sequential, it could be because image data is better suited for the CNN and again because of my choice of model architecture. Overall though, I would say the pretrained model performed the best. I feel like the CNN could have gotten a similar accuracy had I found the correct parameters; however.

---

✓ 33m 21s    completed at 5:19 PM

