Assignment 6: Image Classification

Authors: Aarya Patil and Austin Girouard

[Image_Classification_Data.zip]

We are using an intel image classifiation dataset of natural scenes around the world. Make sure to download the zip file and upload the zip file to the local directory as 'Image_Classification_Data.zip'.

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import os
import tensorflow as tf
#import pathlib
import zipfile
import matplotlib.pyplot as plt
```

In [ ]:
```
# Extract zip file into content/data directory
zip_ref = zipfile.ZipFile("/content/Image_Classification_Data.zip", 'r')
zip_ref.extractall("data")
zip_ref.close()
```

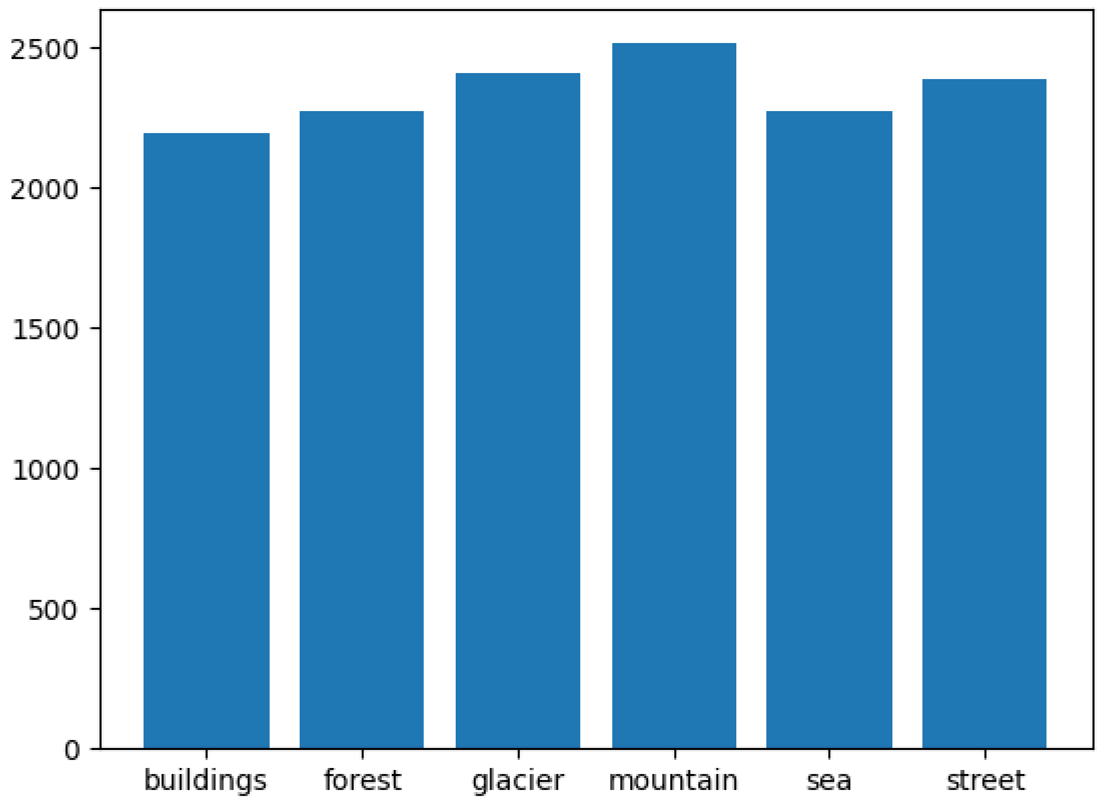
In [ ]:
```
PATH = '/content/data'
train_dir = os.path.join(PATH, 'seg_train/seg_train')
test_dir = os.path.join(PATH, 'seg_test/seg_test')

BATCH_SIZE = 64
IMG_SIZE = (150, 150)
IMG_SHAPE = (150, 150, 3)
NUM_CLASSES = 6
NUM_EPOCHS = 20

# Split data into training and testing datasets
train_dataset = tf.keras.utils.image_dataset_from_directory(train_dir, shuffle=True
test_dataset = tf.keras.utils.image_dataset_from_directory(test_dir, shuffle=True,
```

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

In [ ]:
```
# Get list of class names from training dataset
class_names = train_dataset.class_names
print(class_names)

# Unbatch data to more easily iterate through images and their corresponding labels
unbatch_data = train_dataset.unbatch()

# Initialize dict to store image classification count
dict = {key: 0 for key in class_names}
```

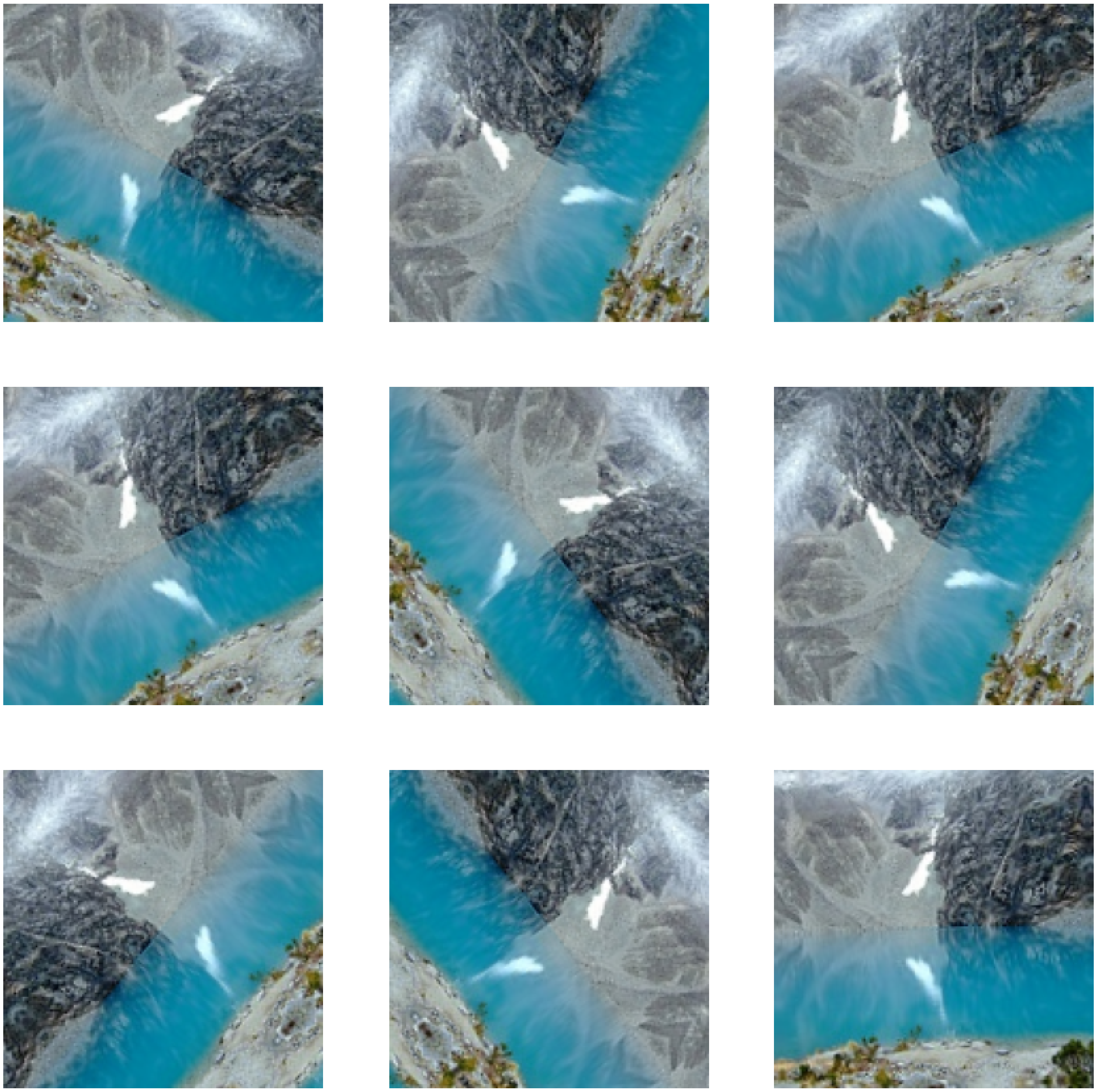
```
# Count number of instances of each categorical class, store in dict for plotting
for images, labels in unbatch_data:
  dict[class_names[tf.argmax(labels).numpy()]] += 1

dict_values = list(dict.values())

plt.bar(range(len(dict)), dict_values, label = 'Distribution of Target Classes', ti
print(dict)
#tf.keras.utils.plot_model()
```

```
['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
{'buildings': 2191, 'forest': 2271, 'glacier': 2404, 'mountain': 2512, 'sea': 227
4, 'street': 2382}
```



In [ ]:

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),
])

for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tf.expand_dims(first_image, 0))
    plt.imshow(augmented_image[0] / 255)
    plt.axis('off')
```



In [ ]:

```
# Tune data for performance
AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.cache().prefetch(buffer_size=AUTOTUNE)
```

Our dataset contains images categorized by their scenic description (Ex. buildings, forest, glacier, etc.). After training, our model should be able to take an image and correctly classify it as its scenic description.

Using a Sequential Model

In [ ]:

```
# Creating a Sequential Model

model1 = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(150, 150, 3)),
tf.keras.layers.Dense(512, activation='relu'),
```

```python
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(6, activation='softmax'),
])

model1.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten_12 (Flatten)        (None, 67500)             0         
                                                                 
 dense_33 (Dense)            (None, 512)               34560512  
                                                                 
 dropout_21 (Dropout)        (None, 512)               0         
                                                                 
 dense_34 (Dense)            (None, 6)                 3078      
                                                                 
=================================================================
Total params: 34,563,590
Trainable params: 34,563,590
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```python
# Compile our Sequential model
model1.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

history1 = model1.fit(train_dataset,
                    batch_size=BATCH_SIZE,
                    epochs=NUM_EPOCHS,
                    steps_per_epoch = 10,
                    verbose=1,
                    validation_data=test_dataset)
```

```
Epoch 1/20
10/10 [==============================] - 17s 2s/step - loss: 58060.5820 - accurac
y: 0.1562 - val_loss: 12857.4150 - val_accuracy: 0.2393
Epoch 2/20
10/10 [==============================] - 16s 2s/step - loss: 13411.9785 - accurac
y: 0.2219 - val_loss: 7820.0000 - val_accuracy: 0.2783
Epoch 3/20
10/10 [==============================] - 17s 2s/step - loss: 11041.9131 - accurac
y: 0.2313 - val_loss: 8904.9893 - val_accuracy: 0.2213
Epoch 4/20
10/10 [==============================] - 17s 2s/step - loss: 8211.9473 - accuracy:
0.2313 - val_loss: 4953.8765 - val_accuracy: 0.2813
Epoch 5/20
10/10 [==============================] - 23s 2s/step - loss: 4259.6616 - accuracy:
0.2500 - val_loss: 4341.7661 - val_accuracy: 0.1933
Epoch 6/20
10/10 [==============================] - 16s 2s/step - loss: 2858.6196 - accuracy:
0.2609 - val_loss: 1113.6483 - val_accuracy: 0.2417
Epoch 7/20
10/10 [==============================] - 16s 2s/step - loss: 402.3287 - accuracy:
0.2531 - val_loss: 10.5736 - val_accuracy: 0.1817
Epoch 8/20
10/10 [==============================] - 17s 2s/step - loss: 38.9456 - accuracy:
0.1703 - val_loss: 6.3982 - val_accuracy: 0.1713
Epoch 9/20
10/10 [==============================] - 12s 1s/step - loss: 3.7404 - accuracy: 0.
1734 - val_loss: 3.9253 - val_accuracy: 0.1667
Epoch 10/20
10/10 [==============================] - 17s 2s/step - loss: 3.1951 - accuracy: 0.
1688 - val_loss: 2.7214 - val_accuracy: 0.1780
Epoch 11/20
10/10 [==============================] - 16s 2s/step - loss: 9.2014 - accuracy: 0.
2109 - val_loss: 2.9578 - val_accuracy: 0.1787
Epoch 12/20
10/10 [==============================] - 16s 2s/step - loss: 3.8733 - accuracy: 0.
2062 - val_loss: 2.1997 - val_accuracy: 0.1730
Epoch 13/20
10/10 [==============================] - 17s 2s/step - loss: 1.8393 - accuracy: 0.
1734 - val_loss: 2.2274 - val_accuracy: 0.1737
Epoch 14/20
10/10 [==============================] - 16s 2s/step - loss: 2.7502 - accuracy: 0.
1625 - val_loss: 2.1549 - val_accuracy: 0.1670
Epoch 15/20
10/10 [==============================] - 16s 2s/step - loss: 1.9430 - accuracy: 0.
1922 - val_loss: 830.1097 - val_accuracy: 0.1673
Epoch 16/20
10/10 [==============================] - 18s 2s/step - loss: 89.3975 - accuracy:
0.1797 - val_loss: 2.0706 - val_accuracy: 0.1670
Epoch 17/20
10/10 [==============================] - 12s 1s/step - loss: 1.7915 - accuracy: 0.
1562 - val_loss: 2.0705 - val_accuracy: 0.1670
Epoch 18/20
10/10 [==============================] - 12s 1s/step - loss: 1.9540 - accuracy: 0.
1688 - val_loss: 2.0637 - val_accuracy: 0.1763
Epoch 19/20
10/10 [==============================] - 17s 2s/step - loss: 2.6448 - accuracy: 0.
```

```
1625 - val_loss: 1.8460 - val_accuracy: 0.1760
Epoch 20/20
10/10 [==============================] - 16s 2s/step - loss: 1.7868 - accuracy: 0.
1562 - val_loss: 1.8460 - val_accuracy: 0.1760
```

Using the CNN Model

In [ ]:

```python
# Creating a CNN model

model2 = tf.keras.models.Sequential(
    [
        tf.keras.Input(shape=(150, 150, 3)),
        tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(NUM_CLASSES, activation="softmax"),
    ]
)

model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 148, 148, 32)      896

 max_pooling2d_2 (MaxPooling  (None, 74, 74, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 72, 72, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 flatten_2 (Flatten)         (None, 82944)             0

 dropout_3 (Dropout)         (None, 82944)             0

 dense_4 (Dense)             (None, 6)                 497670

=================================================================
Total params: 517,062
Trainable params: 517,062
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```python
# Compile our CNN model
model2.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
history2 = model2.fit(train_dataset,
                     batch_size=BATCH_SIZE,
                     epochs=NUM_EPOCHS,
                     steps_per_epoch = 10,
                     verbose=1,
                     validation_data=test_dataset)
```

```
Epoch 1/20
10/10 [==============================] - 69s 7s/step - loss: 6.3426 - accuracy: 0.
3688 - val_loss: 2.0029 - val_accuracy: 0.3437
Epoch 2/20
10/10 [==============================] - 66s 7s/step - loss: 1.6472 - accuracy: 0.
3781 - val_loss: 1.5976 - val_accuracy: 0.4057
Epoch 3/20
10/10 [==============================] - 66s 7s/step - loss: 1.6388 - accuracy: 0.
3812 - val_loss: 1.5164 - val_accuracy: 0.4127
Epoch 4/20
10/10 [==============================] - 66s 7s/step - loss: 1.5903 - accuracy: 0.
4266 - val_loss: 1.4711 - val_accuracy: 0.4300
Epoch 5/20
10/10 [==============================] - 66s 7s/step - loss: 1.5709 - accuracy: 0.
4375 - val_loss: 1.5237 - val_accuracy: 0.4203
Epoch 6/20
10/10 [==============================] - 67s 7s/step - loss: 1.5442 - accuracy: 0.
4000 - val_loss: 1.4536 - val_accuracy: 0.4437
Epoch 7/20
10/10 [==============================] - 60s 6s/step - loss: 1.4732 - accuracy: 0.
4203 - val_loss: 1.4270 - val_accuracy: 0.4520
Epoch 8/20
10/10 [==============================] - 65s 7s/step - loss: 1.4325 - accuracy: 0.
4578 - val_loss: 1.3092 - val_accuracy: 0.5137
Epoch 9/20
10/10 [==============================] - 66s 7s/step - loss: 1.5179 - accuracy: 0.
4531 - val_loss: 1.4365 - val_accuracy: 0.4440
Epoch 10/20
10/10 [==============================] - 67s 7s/step - loss: 1.4972 - accuracy: 0.
4313 - val_loss: 1.3393 - val_accuracy: 0.5030
Epoch 11/20
10/10 [==============================] - 57s 6s/step - loss: 1.4099 - accuracy: 0.
4984 - val_loss: 1.3213 - val_accuracy: 0.5027
Epoch 12/20
10/10 [==============================] - 56s 6s/step - loss: 1.4467 - accuracy: 0.
4391 - val_loss: 1.5528 - val_accuracy: 0.4377
Epoch 13/20
10/10 [==============================] - 56s 6s/step - loss: 1.3451 - accuracy: 0.
5078 - val_loss: 1.3098 - val_accuracy: 0.5013
Epoch 14/20
10/10 [==============================] - 64s 7s/step - loss: 1.2359 - accuracy: 0.
5141 - val_loss: 1.2471 - val_accuracy: 0.5520
Epoch 15/20
10/10 [==============================] - 54s 6s/step - loss: 1.4673 - accuracy: 0.
4672 - val_loss: 1.5482 - val_accuracy: 0.3983
Epoch 16/20
10/10 [==============================] - 65s 7s/step - loss: 1.5056 - accuracy: 0.
3906 - val_loss: 1.4394 - val_accuracy: 0.4383
Epoch 17/20
10/10 [==============================] - 65s 7s/step - loss: 1.5091 - accuracy: 0.
4313 - val_loss: 1.4229 - val_accuracy: 0.4470
Epoch 18/20
10/10 [==============================] - 65s 7s/step - loss: 1.3765 - accuracy: 0.
4594 - val_loss: 1.4220 - val_accuracy: 0.4637
Epoch 19/20
10/10 [==============================] - 59s 6s/step - loss: 1.4540 - accuracy: 0.
```

```
4469 - val_loss: 1.4278 - val_accuracy: 0.4560
Epoch 20/20
10/10 [==============================] - 70s 7s/step - loss: 1.4198 - accuracy: 0.
4453 - val_loss: 1.3754 - val_accuracy: 0.4743
```

In [ ]:

```
from tensorflow.keras.applications import InceptionV3

base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=IMG_SHA

# Set all layer.trainable to false to prevent the weights in a given layer from bei
for layer in base_model.layers:
    layer.trainable = False

# Build our InceptionV3 model
model3 = tf.keras.models.Sequential([
    tf.keras.layers.experimental.preprocessing.Rescaling(1./255, input_shape=IMG_SH
    base_model,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(6, activation='softmax')
])

# Compile model
model3.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

Using a pretrained model on the data (InceptionV3)

After much trial and error getting other pretrained models to output desirable accuracies, InceptionV3 functioned as intended so we settled on it as our external trained model.

In [ ]:

```
history3 = model3.fit(train_dataset, validation_data = test_dataset, steps_per_epoc
```

```
Epoch 1/10
10/10 [==============================] - 158s 17s/step - loss: 1.0519 - accuracy:
0.7203 - val_loss: 2.2384 - val_accuracy: 0.7973
Epoch 2/10
10/10 [==============================] - 169s 19s/step - loss: 0.8107 - accuracy:
0.7828 - val_loss: 0.9051 - val_accuracy: 0.8593
Epoch 3/10
10/10 [==============================] - 169s 19s/step - loss: 0.5026 - accuracy:
0.8344 - val_loss: 0.7443 - val_accuracy: 0.8597
Epoch 4/10
10/10 [==============================] - 169s 19s/step - loss: 0.4734 - accuracy:
0.8391 - val_loss: 0.7843 - val_accuracy: 0.8433
Epoch 5/10
10/10 [==============================] - 169s 18s/step - loss: 0.4532 - accuracy:
0.8453 - val_loss: 0.5840 - val_accuracy: 0.8630
Epoch 6/10
10/10 [==============================] - 151s 16s/step - loss: 0.4798 - accuracy:
0.8250 - val_loss: 0.5141 - val_accuracy: 0.8717
Epoch 7/10
10/10 [==============================] - 148s 16s/step - loss: 0.4195 - accuracy:
0.8516 - val_loss: 0.4258 - val_accuracy: 0.8843
Epoch 8/10
10/10 [==============================] - 170s 19s/step - loss: 0.4249 - accuracy:
0.8438 - val_loss: 0.4181 - val_accuracy: 0.8803
Epoch 9/10
10/10 [==============================] - 151s 16s/step - loss: 0.3887 - accuracy:
0.8703 - val_loss: 0.3918 - val_accuracy: 0.8893
Epoch 10/10
10/10 [==============================] - 169s 18s/step - loss: 0.4943 - accuracy:
0.8375 - val_loss: 0.3936 - val_accuracy: 0.8810
```

Performance Analysis

- The regular Sequential Model produced more sporadic accuracies ranging from 15% to 26%. This model ran very quickly, taking less than 10 seconds on average per epoch, but the major drawback is the poor accuracy.
- The CNN model performed much better than the Sequential Model, giving a max accuracy of 50%, which is 3 times more accurate than guessing. The CNN model was very inconsistent across runs, however, sometimes producing average accuracies as low as 25%. CNN also took 3 times longer to process on average.
- Finally, the pre-trained InceptionV3 model performed the best by far, resulting in a max accuracy of 87%. This is because the model has been trained on a far more complex architecture with a much larger variety of data. The drawback to this model is the training time, which took over 2 times as long as CNN and 8 times as long as the Sequential Model.