Image Classification

For this assignment I chose this dataset: https://www.kaggle.com/datasets/puneet6060/intel-image-classification

This dataset is already divided into train/test, and each set contains images from 6 different categories/folders: 'buildings', 'forest', 'glacier', 'mountain', 'sea', 'street'. The goal of this assignment is to read those images as arrays of float values and make models that can accurately classify each image to one of the previously mentioned categories.

Note: I used code from the notebook listed below to read images form each folder, convert then into an array of float values, and graph the distribution of target classes.

Referenced Notebook: https://www.kaggle.com/code/vincee/intel-image-classification-cnn-keras

Import Required Packages

```
import os
import cv2
import numpy as np
import pandas as pd
from PIL import Image
from tqdm import tqdm
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
import seaborn as sn; sn.set(font_scale=1.4)
from sklearn.metrics import confusion_matrix
```

Assign Class Labels and Define Image Size

```
class_names = ['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
class_labels = {class_name:i for i, class_name in enumerate(class_names)}
num_classes = len(class_names)

IMAGE_SIZE = (150, 150)
```

Function to Read Data

```
def load_data():
    """
        Load the data:
            - 14,034 images to train the network.
            - 3,000 images to evaluate how accurately the network learned to classify images.
    """

    datasets = ['../input/intel-image-classification/seg_train/seg_train', '../input/intel-image-classification/seg_test/seg_test']
    output = []

    for dataset in datasets:
        images = []
        labels = []

        print("Loading {}".format(dataset))

        for folder in os.listdir(dataset):
            label = class_labels[folder]

            for file in tqdm(os.listdir(os.path.join(dataset, folder))):
                img_path = os.path.join(os.path.join(dataset, folder), file)
                image = Image.open(img_path).convert('RGB')
                image = image.resize(size = IMAGE_SIZE)
                image = np.array(image, dtype = np.float32)

                images.append(image)
                labels.append(label)

        images = np.array(images, dtype = 'float32') / 255.0
        labels = np.array(labels, dtype = 'int32')
        output.append((images, labels))

    return output
```

Load the Data

```
(train_images, train_labels), (test_images, test_labels) = load_data()
```

```
Loading ../input/intel-image-classification/seg_train/seg_train
100%|██████████| 2512/2512 [00:05<00:00, 471.84it/s]
100%|██████████| 2382/2382 [00:05<00:00, 441.30it/s]
100%|██████████| 2191/2191 [00:04<00:00, 513.50it/s]
100%|██████████| 2274/2274 [00:04<00:00, 554.61it/s]
100%|██████████| 2271/2271 [00:04<00:00, 521.46it/s]
100%|██████████| 2404/2404 [00:04<00:00, 552.71it/s]
Loading ../input/intel-image-classification/seg_test/seg_test
100%|██████████| 525/525 [00:00<00:00, 579.20it/s]
100%|██████████| 501/501 [00:00<00:00, 620.49it/s]
100%|██████████| 437/437 [00:00<00:00, 628.20it/s]
100%|██████████| 510/510 [00:00<00:00, 678.69it/s]
100%|██████████| 474/474 [00:00<00:00, 573.83it/s]
100%|██████████| 553/553 [00:00<00:00, 665.30it/s]
```

Print the shape of Train/Test

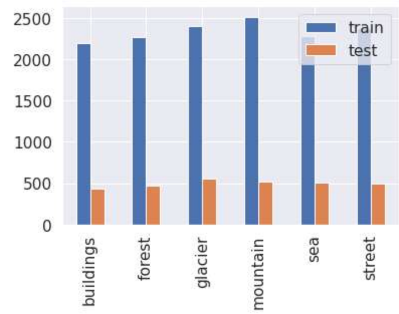
```
num_train = train_labels.shape[0]
num_test = test_labels.shape[0]

print ("Number of training examples: {}".format(num_train))
print ("Number of testing examples: {}".format(num_test))
```

```
Number of training examples: 14034
Number of testing examples: 3000
```

Graph the distribution of target classes

Looking at the graph below, we can see that all the target classes have almost equal number of instances, with 'mountain' having a little more train examples and 'glacier' having a little more test examples

```
_, train_counts = np.unique(train_labels, return_counts=True)
_, test_counts = np.unique(test_labels, return_counts=True)
pd.DataFrame({'train': train_counts, 'test': test_counts}, index=class_names).plot.bar()
plt.show()
```



Build a Sequential Model

```
seq = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(150, 150, 3)),
                                  tf.keras.layers.Dense(512, activation = 'relu'),
                                  tf.keras.layers.Dense(1028, activation = 'relu'),
                                  tf.keras.layers.Dense(num_classes, activation = 'softmax')])
```

Compile and Train the model

```
seq.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])
history = seq.fit(train_images, train_labels, batch_size = 128, epochs = 20, validation_split = 0.2)
```

```
Epoch 1/20
88/88 [==============================] - 21s 237ms/step - loss: 8.7675 - accuracy: 0.3786 - val_loss: 31.1768 - val_accuracy: 0.0762
Epoch 2/20
88/88 [==============================] - 20s 233ms/step - loss: 1.1399 - accuracy: 0.5305 - val_loss: 28.7248 - val_accuracy: 0.0930
Epoch 3/20
88/88 [==============================] - 20s 228ms/step - loss: 1.0413 - accuracy: 0.5766 - val_loss: 33.1513 - val_accuracy: 0.0844
Epoch 4/20
88/88 [==============================] - 21s 235ms/step - loss: 0.9889 - accuracy: 0.6001 - val_loss: 29.3127 - val_accuracy: 0.1193
Epoch 5/20
88/88 [==============================] - 20s 228ms/step - loss: 0.9790 - accuracy: 0.6101 - val_loss: 37.1936 - val_accuracy: 0.0741
Epoch 6/20
88/88 [==============================] - 21s 232ms/step - loss: 0.9478 - accuracy: 0.6205 - val_loss: 36.0047 - val_accuracy: 0.0830
Epoch 7/20
88/88 [==============================] - 21s 233ms/step - loss: 0.9113 - accuracy: 0.6371 - val_loss: 41.5027 - val_accuracy: 0.1161
Epoch 8/20
88/88 [==============================] - 20s 228ms/step - loss: 0.8930 - accuracy: 0.6456 - val_loss: 35.8625 - val_accuracy: 0.0759
Epoch 9/20
88/88 [==============================] - 21s 234ms/step - loss: 0.8492 - accuracy: 0.6658 - val_loss: 40.0645 - val_accuracy: 0.0802
Epoch 10/20
88/88 [==============================] - 20s 231ms/step - loss: 0.8603 - accuracy: 0.6562 - val_loss: 40.3066 - val_accuracy: 0.0894
Epoch 11/20
88/88 [==============================] - 20s 227ms/step - loss: 0.8449 - accuracy: 0.6653 - val_loss: 35.3879 - val_accuracy: 0.1115
Epoch 12/20
88/88 [==============================] - 20s 233ms/step - loss: 0.7819 - accuracy: 0.6908 - val_loss: 38.7746 - val_accuracy: 0.0905
Epoch 13/20
88/88 [==============================] - 21s 234ms/step - loss: 0.7721 - accuracy: 0.6989 - val_loss: 33.0109 - val_accuracy: 0.1015
Epoch 14/20
88/88 [==============================] - 20s 226ms/step - loss: 0.7449 - accuracy: 0.7058 - val_loss: 34.4429 - val_accuracy: 0.1108
Epoch 15/20
88/88 [==============================] - 21s 233ms/step - loss: 0.7370 - accuracy: 0.7103 - val_loss: 30.5901 - val_accuracy: 0.1158
Epoch 16/20
88/88 [==============================] - 21s 240ms/step - loss: 0.7244 - accuracy: 0.7164 - val_loss: 30.5407 - val_accuracy: 0.0944
Epoch 17/20
88/88 [==============================] - 22s 250ms/step - loss: 0.7789 - accuracy: 0.6892 - val_loss: 34.5409 - val_accuracy: 0.1008
Epoch 18/20
88/88 [==============================] - 20s 232ms/step - loss: 0.7064 - accuracy: 0.7306 - val_loss: 28.5624 - val_accuracy: 0.1051
Epoch 19/20
88/88 [==============================] - 20s 228ms/step - loss: 0.6827 - accuracy: 0.7355 - val_loss: 35.0675 - val_accuracy: 0.1051
Epoch 20/20
88/88 [==============================] - 20s 232ms/step - loss: 0.6182 - accuracy: 0.7641 - val_loss: 35.4567 - val_accuracy: 0.1112
```

Evaluate on the test data

```
seq_score = seq.evaluate(test_images, test_labels)
print('Test loss:', seq_score[0])
print('Test accuracy:', seq_score[1])
```

```
94/94 [==============================] - 3s 28ms/step - loss: 8.4860 - accuracy: 0.4973
Test loss: 8.48598575592041
Test accuracy: 0.4973333477973938
```

Build a CNN Model

```
cnn = tf.keras.Sequential([tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
                           tf.keras.layers.MaxPooling2D(2,2),
                           tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
                           tf.keras.layers.MaxPooling2D(2,2),
                           tf.keras.layers.Flatten(),
                           tf.keras.layers.Dense(64, activation = 'relu'),
                           tf.keras.layers.Dense(num_classes, activation = 'softmax')])
```

Compile and Train the model

```
cnn.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])
history = cnn.fit(train_images, train_labels, batch_size = 128, epochs = 20, validation_split = 0.2)
```

```
Epoch 1/20
88/88 [==============================] - 71s 798ms/step - loss: 1.1115 - accuracy: 0.5619 - val_loss: 12.1333 - val_accuracy: 0.1254
Epoch 2/20
88/88 [==============================] - 68s 776ms/step - loss: 0.6643 - accuracy: 0.7428 - val_loss: 13.6063 - val_accuracy: 0.1275
Epoch 3/20
88/88 [==============================] - 68s 771ms/step - loss: 0.5324 - accuracy: 0.7974 - val_loss: 13.3572 - val_accuracy: 0.1243
Epoch 4/20
88/88 [==============================] - 68s 774ms/step - loss: 0.4547 - accuracy: 0.8314 - val_loss: 13.0877 - val_accuracy: 0.1072
Epoch 5/20
88/88 [==============================] - 68s 773ms/step - loss: 0.3729 - accuracy: 0.8676 - val_loss: 14.1853 - val_accuracy: 0.1364
Epoch 6/20
88/88 [==============================] - 67s 765ms/step - loss: 0.2840 - accuracy: 0.9018 - val_loss: 14.8277 - val_accuracy: 0.1347
Epoch 7/20
88/88 [==============================] - 68s 768ms/step - loss: 0.2186 - accuracy: 0.9291 - val_loss: 17.0672 - val_accuracy: 0.1343
Epoch 8/20
88/88 [==============================] - 67s 764ms/step - loss: 0.1593 - accuracy: 0.9526 - val_loss: 19.2113 - val_accuracy: 0.1265
Epoch 9/20
88/88 [==============================] - 70s 793ms/step - loss: 0.1264 - accuracy: 0.9622 - val_loss: 19.8053 - val_accuracy: 0.1322
Epoch 10/20
88/88 [==============================] - 67s 763ms/step - loss: 0.0823 - accuracy: 0.9788 - val_loss: 22.0214 - val_accuracy: 0.1297
Epoch 11/20
88/88 [==============================] - 67s 763ms/step - loss: 0.0618 - accuracy: 0.9860 - val_loss: 22.9300 - val_accuracy: 0.1307
Epoch 12/20
88/88 [==============================] - 68s 773ms/step - loss: 0.0474 - accuracy: 0.9890 - val_loss: 25.3947 - val_accuracy: 0.1364
Epoch 13/20
88/88 [==============================] - 67s 761ms/step - loss: 0.0393 - accuracy: 0.9923 - val_loss: 26.5541 - val_accuracy: 0.1372
Epoch 14/20
88/88 [==============================] - 66s 752ms/step - loss: 0.0374 - accuracy: 0.9905 - val_loss: 25.3729 - val_accuracy: 0.1389
Epoch 15/20
88/88 [==============================] - 68s 772ms/step - loss: 0.0465 - accuracy: 0.9886 - val_loss: 26.4231 - val_accuracy: 0.1247
Epoch 16/20
88/88 [==============================] - 67s 763ms/step - loss: 0.0238 - accuracy: 0.9955 - val_loss: 27.9500 - val_accuracy: 0.1286
Epoch 17/20
88/88 [==============================] - 67s 757ms/step - loss: 0.0170 - accuracy: 0.9971 - val_loss: 31.1309 - val_accuracy: 0.1311
Epoch 18/20
88/88 [==============================] - 67s 766ms/step - loss: 0.0133 - accuracy: 0.9983 - val_loss: 32.0680 - val_accuracy: 0.1389
Epoch 19/20
88/88 [==============================] - 68s 775ms/step - loss: 0.0166 - accuracy: 0.9975 - val_loss: 31.9217 - val_accuracy: 0.1354
Epoch 20/20
88/88 [==============================] - 69s 782ms/step - loss: 0.0145 - accuracy: 0.9973 - val_loss: 31.0987 - val_accuracy: 0.1325
```

Evaluate on the test data

```
cnn_score = cnn.evaluate(test_images, test_labels)
print('Test loss:', cnn_score[0])
print('Test accuracy:', cnn_score[1])
```

```
94/94 [==============================] - 5s 48ms/step - loss: 7.4469 - accuracy: 0.6597
Test loss: 7.446874141693115
Test accuracy: 0.6596666574478149
```

Using a Pre_Trained model for Feature Extraction

Create the Base Model

```
IMG_SHAPE = IMAGE_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape = IMG_SHAPE, include_top = False, weights = 'imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_top.h5
9412608/9406464 [==============================] - 1s 0us/step
9420800/9406464 [==============================] - 1s 0us/step
```

Freeze the convolutional base

```
base_model.trainable = False
```

Add a classification head

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
prediction_layer = tf.keras.layers.Dense(6)
```

Build the model

```
inputs = tf.keras.Input(shape=(150, 150, 3))
x = base_model(inputs, training = False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model_1 = tf.keras.Model(inputs, outputs)
```

Compile and Train the model

```
base_learning_rate = 0.0001
model_1.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = base_learning_rate),
                loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
                metrics=['accuracy'])
history = model_1.fit(train_images, train_labels, batch_size = 128, epochs = 20, validation_split = 0.2)
```

```
Epoch 1/20
88/88 [==============================] - 72s 782ms/step - loss: 1.4069 - accuracy: 0.4515 - val_loss: 4.9871 - val_accuracy: 0.1218
Epoch 2/20
88/88 [==============================] - 66s 756ms/step - loss: 0.6850 - accuracy: 0.7615 - val_loss: 5.8157 - val_accuracy: 0.1350
Epoch 3/20
88/88 [==============================] - 66s 757ms/step - loss: 0.4505 - accuracy: 0.8566 - val_loss: 6.3380 - val_accuracy: 0.1382
Epoch 4/20
88/88 [==============================] - 67s 765ms/step - loss: 0.3554 - accuracy: 0.8855 - val_loss: 6.7601 - val_accuracy: 0.1389
Epoch 5/20
88/88 [==============================] - 67s 763ms/step - loss: 0.3055 - accuracy: 0.8984 - val_loss: 7.0627 - val_accuracy: 0.1389
Epoch 6/20
88/88 [==============================] - 68s 769ms/step - loss: 0.2737 - accuracy: 0.9070 - val_loss: 7.3511 - val_accuracy: 0.1389
Epoch 7/20
88/88 [==============================] - 67s 761ms/step - loss: 0.2534 - accuracy: 0.9146 - val_loss: 7.5434 - val_accuracy: 0.1389
Epoch 8/20
88/88 [==============================] - 67s 763ms/step - loss: 0.2335 - accuracy: 0.9203 - val_loss: 7.7478 - val_accuracy: 0.1389
Epoch 9/20
88/88 [==============================] - 67s 760ms/step - loss: 0.2209 - accuracy: 0.9257 - val_loss: 7.9245 - val_accuracy: 0.1389
Epoch 10/20
88/88 [==============================] - 67s 764ms/step - loss: 0.2120 - accuracy: 0.9274 - val_loss: 8.0626 - val_accuracy: 0.1393
Epoch 11/20
88/88 [==============================] - 67s 759ms/step - loss: 0.2040 - accuracy: 0.9290 - val_loss: 8.2271 - val_accuracy: 0.1397
Epoch 12/20
88/88 [==============================] - 67s 765ms/step - loss: 0.1997 - accuracy: 0.9301 - val_loss: 8.3951 - val_accuracy: 0.1397
Epoch 13/20
88/88 [==============================] - 67s 761ms/step - loss: 0.1901 - accuracy: 0.9350 - val_loss: 8.4148 - val_accuracy: 0.1400
Epoch 14/20
88/88 [==============================] - 67s 766ms/step - loss: 0.1851 - accuracy: 0.9371 - val_loss: 8.6579 - val_accuracy: 0.1397
Epoch 15/20
88/88 [==============================] - 67s 761ms/step - loss: 0.1842 - accuracy: 0.9353 - val_loss: 8.7176 - val_accuracy: 0.1397
Epoch 16/20
88/88 [==============================] - 67s 762ms/step - loss: 0.1792 - accuracy: 0.9370 - val_loss: 8.7823 - val_accuracy: 0.1404
Epoch 17/20
88/88 [==============================] - 67s 759ms/step - loss: 0.1742 - accuracy: 0.9389 - val_loss: 8.8692 - val_accuracy: 0.1411
Epoch 18/20
88/88 [==============================] - 67s 765ms/step - loss: 0.1719 - accuracy: 0.9405 - val_loss: 8.9589 - val_accuracy: 0.1414
Epoch 19/20
88/88 [==============================] - 67s 761ms/step - loss: 0.1708 - accuracy: 0.9417 - val_loss: 9.0680 - val_accuracy: 0.1407
Epoch 20/20
88/88 [==============================] - 67s 758ms/step - loss: 0.1647 - accuracy: 0.9434 - val_loss: 9.1029 - val_accuracy: 0.1411
```

Evaluate on the test data

```
model_1_score = model_1.evaluate(test_images, test_labels)
print('Test loss:', model_1_score[0])
print('Test accuracy:', model_1_score[1])
```

```
94/94 [==============================] - 16s 168ms/step - loss: 2.0620 - accuracy: 0.7710
Test loss: 2.062046766281128
Test accuracy: 0.7710000276565552
```

Fine Tuning

Un-freeze the top layers of the model

```
base_model.trainable = True
```

Freeze all the layers before the "fine_tune_at" layer

```
fine_tune_at = 100

for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False
```

Compile and Train the model

```
model_1.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = base_learning_rate / 10),
                loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
                metrics=['accuracy'])
history = model_1.fit(train_images, train_labels, batch_size = 128, epochs = 20, validation_split = 0.2)
```

```
Epoch 1/20
88/88 [==============================] - 121s 1s/step - loss: 0.1549 - accuracy: 0.9458 - val_loss: 10.1669 - val_accuracy: 0.1404
Epoch 2/20
88/88 [==============================] - 117s 1s/step - loss: 0.1263 - accuracy: 0.9573 - val_loss: 10.2568 - val_accuracy: 0.1411
Epoch 3/20
88/88 [==============================] - 117s 1s/step - loss: 0.1077 - accuracy: 0.9623 - val_loss: 11.1053 - val_accuracy: 0.1425
Epoch 4/20
88/88 [==============================] - 116s 1s/step - loss: 0.0922 - accuracy: 0.9670 - val_loss: 11.8964 - val_accuracy: 0.1418
```

```
Epoch 5/20
88/88 [==============================] - 116s 1s/step - loss: 0.0828 - accuracy: 0.9703 - val_loss: 11.4409 - val_accuracy: 0.1429
Epoch 6/20
88/88 [==============================] - 118s 1s/step - loss: 0.0691 - accuracy: 0.9757 - val_loss: 12.6421 - val_accuracy: 0.1425
Epoch 7/20
88/88 [==============================] - 116s 1s/step - loss: 0.0599 - accuracy: 0.9808 - val_loss: 13.0803 - val_accuracy: 0.1425
Epoch 8/20
88/88 [==============================] - 116s 1s/step - loss: 0.0519 - accuracy: 0.9821 - val_loss: 13.8961 - val_accuracy: 0.1425
Epoch 9/20
88/88 [==============================] - 116s 1s/step - loss: 0.0470 - accuracy: 0.9847 - val_loss: 13.9560 - val_accuracy: 0.1421
Epoch 10/20
88/88 [==============================] - 116s 1s/step - loss: 0.0426 - accuracy: 0.9850 - val_loss: 13.8344 - val_accuracy: 0.1421
Epoch 11/20
88/88 [==============================] - 116s 1s/step - loss: 0.0328 - accuracy: 0.9894 - val_loss: 14.7367 - val_accuracy: 0.1429
Epoch 12/20
88/88 [==============================] - 116s 1s/step - loss: 0.0289 - accuracy: 0.9917 - val_loss: 15.3487 - val_accuracy: 0.1425
Epoch 13/20
88/88 [==============================] - 116s 1s/step - loss: 0.0250 - accuracy: 0.9931 - val_loss: 15.9648 - val_accuracy: 0.1418
Epoch 14/20
88/88 [==============================] - 117s 1s/step - loss: 0.0212 - accuracy: 0.9943 - val_loss: 16.4197 - val_accuracy: 0.1421
Epoch 15/20
88/88 [==============================] - 116s 1s/step - loss: 0.0203 - accuracy: 0.9948 - val_loss: 15.7190 - val_accuracy: 0.1425
Epoch 16/20
88/88 [==============================] - 116s 1s/step - loss: 0.0190 - accuracy: 0.9950 - val_loss: 16.3944 - val_accuracy: 0.1421
Epoch 17/20
88/88 [==============================] - 116s 1s/step - loss: 0.0144 - accuracy: 0.9965 - val_loss: 16.7963 - val_accuracy: 0.1429
Epoch 18/20
88/88 [==============================] - 117s 1s/step - loss: 0.0136 - accuracy: 0.9971 - val_loss: 16.8694 - val_accuracy: 0.1429
Epoch 19/20
88/88 [==============================] - 117s 1s/step - loss: 0.0120 - accuracy: 0.9972 - val_loss: 17.5019 - val_accuracy: 0.1414
Epoch 20/20
88/88 [==============================] - 116s 1s/step - loss: 0.0100 - accuracy: 0.9980 - val_loss: 17.0241 - val_accuracy: 0.1425
```

Evaluate on the test data

```
model_2_score = model_1.evaluate(test_images, test_labels)
print('Test loss:', model_2_score[0])
print('Test accuracy:', model_2_score[1])
```

```
94/94 [==============================] - 17s 177ms/step - loss: 3.7476 - accuracy: 0.7873
Test loss: 3.7476298809051514
Test accuracy: 0.7873333096504211
```

Analysis of the performance of various approaches

The first model I created was a sequential model with two hidden layers: the first hidden layer had 512 nodes, and the second hidden layer had 1028 nodes. Both hidden layers had an activation of type 'relu', and the output layer had an activation of type 'softmax'. I ran this model for 20 epochs with a batch size of 128. This model only gave an accuracy of 49.7 percent.

For the second model, I decided to build a CNN architecture. In this model, I added 2 Cov2D layers with 32 filters and a kernel size of 3x3. After that, I flattened the data and passed it through a hidden dense layer of 64 nodes. Lastly, I added the same output layer that I made for the previous sequential model. I ran this model with the same configuration as I did the previous model. This model gave me an accuracy of 65.9 percent, which is an increase of 16.1 percent from the previous model.

Next, I performed feature extraction on a pre-trained model from MobileNetV2. I froze the convolutional base, added a classification head, and compiled the model with a learning rate of 0.0001. This model gave me an accuracy of 77.1, which is an increase of 11.2 from the previous CNN model.

Lastly, I performed fine tuning on the same pre-trained model as the previous step. I froze all the layers except the layers from 100 all the way to the top layer. I compiled this model with a learning rate of 0.00001. This model gave me an acccuracy of 78.7, which is only an increase of 1.6 percent from the previous model, but an increase of 29 percent from the first model.

These different models and their accuracies show why it is important to train your model using a large dataset because the pre-trained models gave a higher accuracy than the models that were only trained on the imported data.