title

Introduction

In this journal, a machine learning model is used to identify flowers through image classification. This multi-classification model is trained with thousands of images which are randomly augmented to achieve better generalization. The model serves to be a proof of concept in battling invasive plant species that damage the fragile ecosystems in which they reside. This journal will contain the steps taken to arrive at a similar model. Analysis of the results will be mentioned in the later sections, followed by what the results mean for the stakeholder.

Business Understanding

Every year, the U.S. is estimated to lose $120B due to the impact of invasive plant species. Invasive plant species can reduce yield in nearby agriculture, kill existing plants, increase the risk of forest fires, and much more. While the spread of invasive plant species can be slowed through customs, the plants that reside in U.S. right now need to be found and removed. Programs to remove these plants are already in place, but can be accelerated through the use of machine learning and scouting tools like drones. Drones can quickly scout dangerous terrain and machine learning can rapidly review photos to identify an invasive plant species. This proposal will help reduce cost spent in manual labor and remove the risk of workers traversing dangerous terrain.

Imports

In [1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Global
from tensorflow.keras.preprocessing.image import ImageDataGenerator, array_to_im
#from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
#from tensorflow.keras.applications import Xception, DenseNet201
from tensorflow.keras.callbacks import LearningRateScheduler
#import random
from PIL import Image
import shutil
import h5py
```

Global Variables

In [2]:
```
# image size options: 192
IMAGE_DIMENSION = 192
VECTOR_LEN = IMAGE_DIMENSION**2
```

```
NUM_CLASS = 96

train_dir = f'data/jpeg-{IMAGE_DIMENSION}x{IMAGE_DIMENSION}/train'
val_dir = f'data/jpeg-{IMAGE_DIMENSION}x{IMAGE_DIMENSION}/val'
test_dir = f'data/jpeg-{IMAGE_DIMENSION}x{IMAGE_DIMENSION}/test'

BATCH_SIZE = 64
TRAIN_BATCH_SIZE = BATCH_SIZE
VAL_BATCH_SIZE = BATCH_SIZE
TEST_BATCH_SIZE = BATCH_SIZE
```

Data Understanding

Functions

In [3]:

```
def get_subfolder_list(source_folder):
    # returns a list of all the subfolders in the source_folder
    class_list_dir = []

    for file in os.listdir(source_folder):
        d = os.path.join(source_folder, file)
        if os.path.isdir(d):
            class_list_dir.append(d)

    return class_list_dir

def map_classes(subfolder_list):
    # returns a dict with the flower as the key, and (count, directory) as the v
    class_dict = {}

    for class_folder in subfolder_list:
        file_count = sum(len(files) for _, _, files in os.walk(class_folder))
        class_dict[class_folder[24:]] = file_count, class_folder

    return class_dict

def get_flower_count(class_dict):
    # returns a list of the number of flowers found in the dict
    flower_count = []

    for flower in list(class_dict.values()):
        flower_count.append(flower[0])

    return flower_count

def get_metrics(flower_count):
    # returns basic metrics when given a list of flower value count
    class_std = np.std(flower_count)
    class_max = max(flower_count)
    class_min = min(flower_count)
    class_mean = np.mean(flower_count)
    class_first_quartile = np.percentile(flower_count, 25)
    class_third_quartile = np.percentile(flower_count, 75)
    class_tenth_percentile = np.percentile(flower_count, 10)
    class_fifth_percentile = np.percentile(flower_count, 5)
```

```
    print(f'0. standard deviation: {class_std}')
    print(f'1. max: {class_max}')
    print(f'2. min: {class_min}')
    print(f'3. mean: {class_mean}')
    print(f'4. 25%: {class_first_quartile}')
    print(f'5. 75%: {class_third_quartile}')
    print(f'6. 10%: {class_tenth_percentile}')
    print(f'7. 5%: {class_fifth_percentile}')

    return (class_std, class_max, class_min, class_mean, class_first_quartile,
class_third_quartile, class_tenth_percentile, class_fifth_percentile)

def plot_distribution(keys, values):
    list1 = list(train_dict.keys())
    list2 = train_flower_count

    dict_list = {}

    for i in range(len(list1)):
        dict_list[list1[i]] = list2[i]

    dict_list

    df = pd.DataFrame.from_dict(dict_list, orient='index')

    df_sorted = df.sort_values(0, ascending=False)

    flower_name = list(df_sorted.index)
    value_count = list(df_sorted[0])

    fig, ax = plt.subplots(figsize=(15, 20))

    ax.barh(flower_name, value_count);
    ax.set_xlabel('# of images')
```

In [4]:

```
train_subfolders = get_subfolder_list(train_dir)
train_dict = map_classes(train_subfolders)
train_flower_count = get_flower_count(train_dict)

train_subfolders[:5]
```

Out[4]:

```
['data/jpeg-192x192/train/toad lily',
 'data/jpeg-192x192/train/love in the mist',
 'data/jpeg-192x192/train/monkshood',
 'data/jpeg-192x192/train/azalea',
 'data/jpeg-192x192/train/fritillary']
```

In [5]:

```
get_metrics(train_flower_count)
```

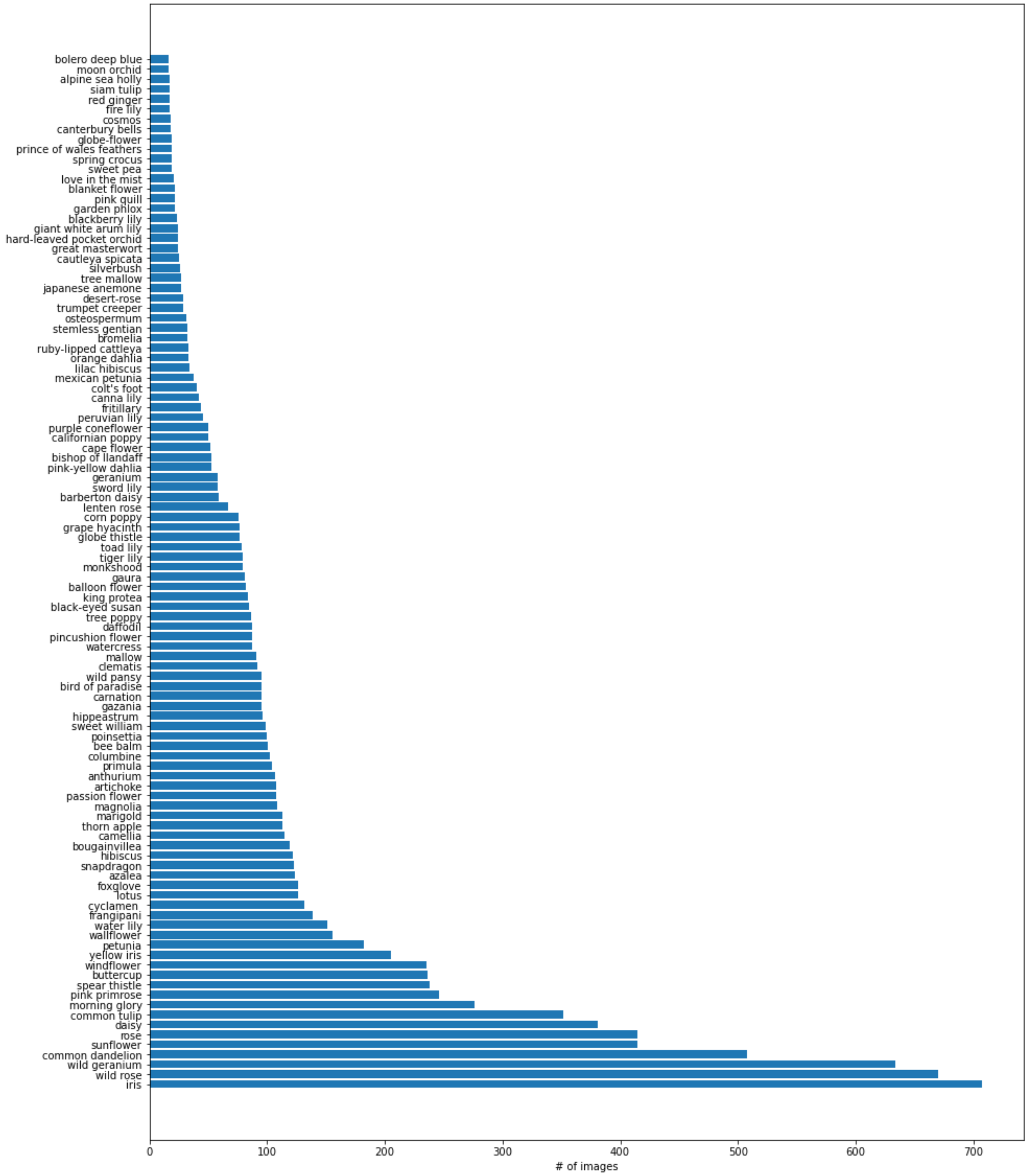
```
0. standard deviation: 132.99130714962862
1. max: 707
2. min: 16
3. mean: 111.1826923076923
4. 25%: 30.5
5. 75%: 113.5
6. 10%: 19.0
7. 5%: 17.15
(132.99130714962862, 707, 16, 111.1826923076923, 30.5, 113.5, 19.0, 17.15)
```

Out[5]:

In [6]:

```
list1 = list(train_dict.keys())
list2 = train_flower_count

plot_distribution(list1, list2)
```



There are 104 different flower species provided in this dataset, 16,463 images with different compositions. An image can be of a distant rose bush, a macro photo of a sunflower bud, or a portrait with a hibiscus tucked behind the ear. The distribution of these flowers is not equal and would result in some preprocessing before models should be trained.

Data Preparation

More functions

In [7]:
```
def copy_subfolder(source):
    # copies source folder to a new directory with '_new' attached to the end of
    prev_dir_index = source.rfind('/')
    destination = source[:prev_dir_index] + '_new' + source[prev_dir_index:]


    if not os.path.exists(destination):
        result = shutil.copytree(source, destination, symlinks=False, ignore=Non
                                 copy_function=shutil.copy2, ignore_dangling_sym
                                 dirs_exist_ok=False)
    else:
        print(f'{destination} already exists')

    return result

def item_count(folder):
    # helper function to identify which folders to remove
    file_count = sum(len(files) for _, _, files in os.walk(folder))

    return file_count

def get_short_list(subfolder_list, n):
    # find a list of classes that are divided by the specified n value
    temp_dict = map_classes(subfolder_list)
    temp_flower_count = get_flower_count(temp_dict)

    move_list = []
    ignore_list = []

    percent_cutoff = np.percentile(temp_flower_count, n)

    remove_count = 0

    for subfolder in subfolder_list:
        if item_count(subfolder) >= percent_cutoff:
            move_list.append(subfolder)
        else:
            ignore_list.append(subfolder)
            remove_count += item_count(subfolder)

    print(f'removed {remove_count} images')

    return move_list, ignore_list

def trim(subfolder_list):
    # main function used to copy classes into new train, val, test
    for subfolder in subfolder_list:
        copy_subfolder(subfolder)

        val_subfolder = subfolder.replace('/train/', '/val/')
        copy_subfolder(val_subfolder)

        test_subfolder = subfolder.replace('/train/', '/test/')
        copy_subfolder(test_subfolder)
```

The structure of the directory has class labels for the train and validation folder, but the test folder contains images without any labels. To create a typical train, validate, and test structure, the images in the test folder will be ignored from this point on. 10% of the images found in the train folder will be moved into a new test folder, resulting in three folders with labeled images.

In [8]:

```
move_list, ignore_list = get_short_list(train_subfolders, 10)
```

```
removed 136 images
```

In [9]:

```
# only needs to be run once if the direectory is not yet set up
trim(move_list)
```

In [10]:

```
# updating the directories as new folders are made
train_dir = f'data/jpeg-{IMAGE_DIMENSION}x{IMAGE_DIMENSION}/train_new'
val_dir = f'data/jpeg-{IMAGE_DIMENSION}x{IMAGE_DIMENSION}/val_new'
test_dir = f'data/jpeg-{IMAGE_DIMENSION}x{IMAGE_DIMENSION}/test_new'
```

resulting directory

With this directory set up, classes of flowers that do not have enough information on which to train a model can be removed. Functions have been created to select these folders to avoid manually separating the folders by hand. In the models shown in the notebook, the 10th percentile of the number of images found in the training are removed from the scope. This leaves 96 classes with 16,268 images. On average, there should be 168 images per class that can be divided into train, validation, and test sets.

With nearly 100 different classes, there are not enough images to effectively train a model. Data augmentation will be used to make the model more generalizable. Each image will randomly flip, change in brightness, crop, and many other transformations. Also, with these images being RGB, the images will need to be standardized from 0 and 255 to the range of 0 and 1.

In [11]:

```
train_generator = ImageDataGenerator(rescale=1./255,
                                     horizontal_flip=True,
                                     rotation_range=45,
                                     vertical_flip=False,
                                     brightness_range=[0.75,1.25],
                                     zoom_range=0.2,
                                     shear_range=0.2
                                    ).flow_from_directory(
    train_dir,
    target_size=(IMAGE_DIMENSION, IMAGE_DIMENSION),
    batch_size=TRAIN_BATCH_SIZE,
    shuffle=True
)

val_generator = ImageDataGenerator().flow_from_directory(
    val_dir,
    target_size=(IMAGE_DIMENSION, IMAGE_DIMENSION),
    batch_size=VAL_BATCH_SIZE,
    shuffle=True
)
```

```
test_generator = ImageDataGenerator().flow_from_directory(
    test_dir,
    target_size=(IMAGE_DIMENSION, IMAGE_DIMENSION),
    batch_size=TEST_BATCH_SIZE,
    shuffle=False
)
```

```
Found 11331 images belonging to 96 classes.
Found 3666 images belonging to 96 classes.
Found 1271 images belonging to 96 classes.
```

Modeling

The first model made is a simple convoluational neural network (CNN) which has two hidden layers.

In [12]:

```
model = models.Sequential()
model.add(layers.Conv2D(filters=64,
                        kernel_size=(2,2),
                        activation='relu',
                        padding = 'same',
                        input_shape=(IMAGE_DIMENSION, IMAGE_DIMENSION, 3),
                        data_format = 'channels_last'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Flatten())
model.add(layers.Dense(NUM_CLASS)) # output layer
model.add(layers.Activation('sigmoid'))

model.summary()

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

# training begins
model.fit(train_generator, steps_per_epoch=len(train_generator)*10 // BATCH_SIZE
          validation_data=val_generator, use_multiprocessing=False)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 192, 192, 64)      832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 96, 96, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 589824)            0
_________________________________________________________________
dense (Dense)                (None, 96)                56623200
_________________________________________________________________
activation (Activation)      (None, 96)                0
=================================================================
Total params: 56,624,032
Trainable params: 56,624,032
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/20
WARNING:tensorflow:AutoGraph could not transform <function Model.make_train_func
tion.<locals>.train_function at 0x28bf04550> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
WARNING: AutoGraph could not transform <function Model.make_train_function.<loca
ls>.train_function at 0x28bf04550> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
```

```
2021-12-06 23:02:35.384451: I tensorflow/compiler/mlir/mlir_graph_optimization_p
ass.cc:116] None of the MLIR optimization passes are enabled (registered 2)
2021-12-06 23:02:35.384566: W tensorflow/core/platform/profile_utils/cpu_utils.c
c:126] Failed to get CPU frequency: 0 Hz
27/27 [==============================] - ETA: 0s - loss: 34.9568 - accuracy: 0.0
324WARNING:tensorflow:AutoGraph could not transform <function Model.make_test_fu
nction.<locals>.test_function at 0x28a1b45e0> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
WARNING: AutoGraph could not transform <function Model.make_test_function.<local
s>.test_function at 0x28a1b45e0> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
27/27 [==============================] - 31s 1s/step - loss: 34.8496 - accuracy:
0.0325 - val_loss: 3934.7927 - val_accuracy: 0.0786
Epoch 2/20
27/27 [==============================] - 28s 1s/step - loss: 8.0935 - accuracy:
0.0645 - val_loss: 880.8830 - val_accuracy: 0.0955
Epoch 3/20
27/27 [==============================] - 27s 1s/step - loss: 3.8095 - accuracy:
0.1317 - val_loss: 905.4819 - val_accuracy: 0.0998
Epoch 4/20
27/27 [==============================] - 28s 1s/step - loss: 3.5433 - accuracy:
0.1671 - val_loss: 858.9821 - val_accuracy: 0.1247
Epoch 5/20
27/27 [==============================] - 26s 982ms/step - loss: 3.4312 - accurac
y: 0.1897 - val_loss: 835.2484 - val_accuracy: 0.1375
Epoch 6/20
27/27 [==============================] - 27s 998ms/step - loss: 3.2235 - accurac
y: 0.2028 - val_loss: 793.8884 - val_accuracy: 0.1563
Epoch 7/20
27/27 [==============================] - 27s 993ms/step - loss: 3.1676 - accurac
y: 0.2272 - val_loss: 714.5256 - val_accuracy: 0.1522
Epoch 8/20
27/27 [==============================] - 27s 989ms/step - loss: 3.0855 - accurac
y: 0.2137 - val_loss: 773.2524 - val_accuracy: 0.1593
Epoch 9/20
27/27 [==============================] - 27s 996ms/step - loss: 3.1416 - accurac
y: 0.2453 - val_loss: 696.3582 - val_accuracy: 0.1560
```

```
Epoch 10/20
27/27 [==============================] - 27s 989ms/step - loss: 3.1102 - accurac
y: 0.2346 - val_loss: 682.4738 - val_accuracy: 0.1639
Epoch 11/20
27/27 [==============================] - 28s 1s/step - loss: 3.0299 - accuracy:
0.2285 - val_loss: 718.7316 - val_accuracy: 0.1721
Epoch 12/20
27/27 [==============================] - 26s 972ms/step - loss: 2.9787 - accurac
y: 0.2498 - val_loss: 684.7418 - val_accuracy: 0.1863
Epoch 13/20
27/27 [==============================] - 27s 988ms/step - loss: 2.9454 - accurac
y: 0.2750 - val_loss: 716.7882 - val_accuracy: 0.1869
Epoch 14/20
27/27 [==============================] - 26s 958ms/step - loss: 2.9597 - accurac
y: 0.2663 - val_loss: 634.2930 - val_accuracy: 0.1874
Epoch 15/20
27/27 [==============================] - 27s 986ms/step - loss: 2.8714 - accurac
y: 0.2841 - val_loss: 665.1017 - val_accuracy: 0.1945
Epoch 16/20
27/27 [==============================] - 26s 969ms/step - loss: 2.8808 - accurac
y: 0.2640 - val_loss: 655.2601 - val_accuracy: 0.2035
Epoch 17/20
27/27 [==============================] - 26s 960ms/step - loss: 2.8156 - accurac
y: 0.2984 - val_loss: 590.0357 - val_accuracy: 0.2158
Epoch 18/20
27/27 [==============================] - 26s 967ms/step - loss: 2.8243 - accurac
y: 0.2951 - val_loss: 741.7135 - val_accuracy: 0.1727
Epoch 19/20
27/27 [==============================] - 26s 957ms/step - loss: 2.7992 - accurac
y: 0.2824 - val_loss: 681.8632 - val_accuracy: 0.1972
Epoch 20/20
27/27 [==============================] - 27s 986ms/step - loss: 2.7180 - accurac
y: 0.2809 - val_loss: 648.1423 - val_accuracy: 0.2160
```

Out[12]:
```
<tensorflow.python.keras.callbacks.History at 0x28beffdc0>
```

The second model uses a different layer, GlobalAveragePooling2D, instead of the common Conv2D and MaxPooling2D layers.

In [13]:
```
model = models.Sequential()
model.add(layers.Conv2D(filters=64,
                        kernel_size=(2,2),
                        activation='relu',
                        padding = 'same',
                        input_shape=(IMAGE_DIMENSION, IMAGE_DIMENSION, 3),
                        data_format = 'channels_last'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.7))
model.add(layers.Flatten())
model.add(layers.Dense(NUM_CLASS)) # output layer
model.add(layers.Activation('sigmoid'))

model.summary()

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

```
# training begins
model.fit(train_generator, steps_per_epoch=len(train_generator)*10 // BATCH_SIZE
          validation_data=val_generator, use_multiprocessing=False)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 192, 192, 64)      832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 96, 96, 64)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 96, 96, 64)        256
_________________________________________________________________
dropout (Dropout)            (None, 96, 96, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 589824)            0
_________________________________________________________________
dense_1 (Dense)              (None, 96)                56623200
_________________________________________________________________
activation_1 (Activation)    (None, 96)                0
=================================================================
Total params: 56,624,288
Trainable params: 56,624,160
Non-trainable params: 128
_________________________________________________________________
Epoch 1/20
WARNING:tensorflow:AutoGraph could not transform <function Model.make_train_func
tion.<locals>.train_function at 0x293db0670> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
WARNING: AutoGraph could not transform <function Model.make_train_function.<loca
ls>.train_function at 0x293db0670> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
27/27 [==============================] - ETA: 0s - loss: 46.8643 - accuracy: 0.0
728WARNING:tensorflow:AutoGraph could not transform <function Model.make_test_fu
nction.<locals>.test_function at 0x2965b5c10> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
WARNING: AutoGraph could not transform <function Model.make_test_function.<local
s>.test_function at 0x2965b5c10> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
27/27 [==============================] - 36s 1s/step - loss: 46.6907 - accuracy:
0.0730 - val_loss: 584.2486 - val_accuracy: 0.0177
Epoch 2/20
27/27 [==============================] - 34s 1s/step - loss: 10.5702 - accuracy:
```

```
0.0841 - val_loss: 69.9108 - val_accuracy: 0.1170
Epoch 3/20
27/27 [==============================] - 32s 1s/step - loss: 4.4192 - accuracy:
0.0611 - val_loss: 64.6247 - val_accuracy: 0.1072
Epoch 4/20
27/27 [==============================] - 36s 1s/step - loss: 4.0225 - accuracy:
0.1000 - val_loss: 71.0697 - val_accuracy: 0.1167
Epoch 5/20
27/27 [==============================] - 34s 1s/step - loss: 3.9234 - accuracy:
0.1330 - val_loss: 81.0931 - val_accuracy: 0.1129
Epoch 6/20
27/27 [==============================] - 32s 1s/step - loss: 3.8450 - accuracy:
0.1150 - val_loss: 89.2336 - val_accuracy: 0.1184
Epoch 7/20
27/27 [==============================] - 32s 1s/step - loss: 3.7532 - accuracy:
0.1319 - val_loss: 102.6124 - val_accuracy: 0.1195
Epoch 8/20
27/27 [==============================] - 31s 1s/step - loss: 3.7086 - accuracy:
0.1280 - val_loss: 116.9508 - val_accuracy: 0.1173
Epoch 9/20
27/27 [==============================] - 33s 1s/step - loss: 3.7971 - accuracy:
0.1206 - val_loss: 131.6573 - val_accuracy: 0.1154
Epoch 10/20
27/27 [==============================] - 32s 1s/step - loss: 3.6376 - accuracy:
0.1444 - val_loss: 168.0217 - val_accuracy: 0.1102
Epoch 11/20
27/27 [==============================] - 33s 1s/step - loss: 3.6349 - accuracy:
0.1553 - val_loss: 196.1830 - val_accuracy: 0.1045
Epoch 12/20
27/27 [==============================] - 34s 1s/step - loss: 3.7505 - accuracy:
0.1255 - val_loss: 213.2345 - val_accuracy: 0.1116
Epoch 13/20
27/27 [==============================] - 32s 1s/step - loss: 3.6565 - accuracy:
0.1363 - val_loss: 222.3793 - val_accuracy: 0.1154
Epoch 14/20
27/27 [==============================] - 31s 1s/step - loss: 3.6798 - accuracy:
0.1364 - val_loss: 248.8666 - val_accuracy: 0.1241
Epoch 15/20
27/27 [==============================] - 32s 1s/step - loss: 3.7489 - accuracy:
0.1260 - val_loss: 285.6989 - val_accuracy: 0.1233
Epoch 16/20
27/27 [==============================] - 31s 1s/step - loss: 3.6297 - accuracy:
0.1256 - val_loss: 346.1637 - val_accuracy: 0.1206
Epoch 17/20
27/27 [==============================] - 32s 1s/step - loss: 3.5647 - accuracy:
0.1623 - val_loss: 371.9765 - val_accuracy: 0.1157
Epoch 18/20
27/27 [==============================] - 32s 1s/step - loss: 3.5948 - accuracy:
0.1536 - val_loss: 420.1517 - val_accuracy: 0.1195
Epoch 19/20
27/27 [==============================] - 31s 1s/step - loss: 3.7530 - accuracy:
0.1417 - val_loss: 532.9658 - val_accuracy: 0.1075
Epoch 20/20
27/27 [==============================] - 32s 1s/step - loss: 3.5961 - accuracy:
0.1411 - val_loss: 620.6290 - val_accuracy: 0.0938
```

Out[13]:

```
<tensorflow.python.keras.callbacks.History at 0x293daed90>
```

Through many iterations, the final model comes out to be a convolutional neural network (CNN) with several layers. The model looks through each image and run for 20 epochs. There is also a learning rate that reduces the rate at which the model trains. The goal of the learning rate is to prevent training loss to get ahead of the validation loss.

In [14]:

```
def scheduler(epoch, lr):
    if epoch < 10:
        return lr
    else:
        return 0.00001

lr_callback = LearningRateScheduler(scheduler)
```

There were some attempts at bettering the model during training but metrics were not improved. The final model faced much overfitting issues but did not use BatchNormalization nor Dropout layers. Adding these layers did not help improve the cross-validation accuracy metric. When attempting to introduce transfer learning, the results instead reduced the cross-validation accuracy metric and increased validation loss. With shorter run times and better metrics, these layers were removed from the final model.

In [15]:

```
model = models.Sequential()
model.add(layers.Conv2D(filters=32,
                        kernel_size=(2,2),
                        activation='relu',
                        padding = 'same',
                        input_shape=(IMAGE_DIMENSION, IMAGE_DIMENSION, 3),
                        data_format = 'channels_last'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(16, (3,3), activation='relu'))
model.add(layers.MaxPooling2D((3,3)))
model.add(layers.Flatten())
model.add(layers.Dense(64))
model.add(layers.Dense(NUM_CLASS)) # output layer
model.add(layers.Activation('sigmoid'))

model.summary()

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

# training begins
model.fit(train_generator, steps_per_epoch=len(train_generator)*10 // BATCH_SIZE
          validation_data=val_generator, callbacks=lr_callback, use_multiprocess
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 192, 192, 32)      416
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 96, 96, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 94, 94, 16)        4624
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 31, 31, 16)        0
_________________________________________________________________
```

```
flatten_2 (Flatten)          (None, 15376)             0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                984128
_________________________________________________________________
dense_3 (Dense)              (None, 96)                6240
_________________________________________________________________
activation_2 (Activation)    (None, 96)                0
=================================================================
Total params: 995,408
Trainable params: 995,408
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
WARNING:tensorflow:AutoGraph could not transform <function Model.make_train_func
tion.<locals>.train_function at 0x12f2ee790> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
WARNING: AutoGraph could not transform <function Model.make_train_function.<loca
ls>.train_function at 0x12f2ee790> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
27/27 [==============================] - ETA: 0s - loss: 4.4151 - accuracy: 0.05
41WARNING:tensorflow:AutoGraph could not transform <function Model.make_test_fun
ction.<locals>.test_function at 0x2910acd30> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
WARNING: AutoGraph could not transform <function Model.make_test_function.<local
s>.test_function at 0x2910acd30> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosit
y to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: unsupported operand type(s) for -: 'NoneType' and 'int'
To silence this warning, decorate the function with @tf.autograph.experimental.d
o_not_convert
27/27 [==============================] - 15s 551ms/step - loss: 4.4090 - accurac
y: 0.0547 - val_loss: 376.9553 - val_accuracy: 0.0968
Epoch 2/20
27/27 [==============================] - 14s 532ms/step - loss: 3.8834 - accurac
y: 0.1053 - val_loss: 375.8888 - val_accuracy: 0.1315
Epoch 3/20
27/27 [==============================] - 15s 535ms/step - loss: 3.6784 - accurac
y: 0.1417 - val_loss: 339.8662 - val_accuracy: 0.1558
Epoch 4/20
27/27 [==============================] - 14s 532ms/step - loss: 3.5180 - accurac
y: 0.1628 - val_loss: 340.0695 - val_accuracy: 0.1451
Epoch 5/20
27/27 [==============================] - 15s 534ms/step - loss: 3.5152 - accurac
y: 0.1441 - val_loss: 366.4016 - val_accuracy: 0.1571
Epoch 6/20
27/27 [==============================] - 15s 536ms/step - loss: 3.4247 - accurac
y: 0.1581 - val_loss: 335.3971 - val_accuracy: 0.1669
Epoch 7/20
```

```
27/27 [==============================] - 14s 529ms/step - loss: 3.3087 - accurac
y: 0.1756 - val_loss: 389.9234 - val_accuracy: 0.1623
Epoch 8/20
27/27 [==============================] - 15s 536ms/step - loss: 3.3106 - accurac
y: 0.1906 - val_loss: 363.2328 - val_accuracy: 0.1620
Epoch 9/20
27/27 [==============================] - 15s 538ms/step - loss: 3.1708 - accurac
y: 0.2018 - val_loss: 369.5406 - val_accuracy: 0.1650
Epoch 10/20
27/27 [==============================] - 15s 534ms/step - loss: 3.1984 - accurac
y: 0.1736 - val_loss: 376.2228 - val_accuracy: 0.1800
Epoch 11/20
27/27 [==============================] - 14s 532ms/step - loss: 3.0356 - accurac
y: 0.2113 - val_loss: 369.8881 - val_accuracy: 0.1787
Epoch 12/20
27/27 [==============================] - 14s 533ms/step - loss: 3.0361 - accurac
y: 0.2265 - val_loss: 362.6716 - val_accuracy: 0.1789
Epoch 13/20
27/27 [==============================] - 14s 533ms/step - loss: 3.0274 - accurac
y: 0.2291 - val_loss: 356.3421 - val_accuracy: 0.1822
Epoch 14/20
27/27 [==============================] - 15s 534ms/step - loss: 3.0748 - accurac
y: 0.2324 - val_loss: 349.5383 - val_accuracy: 0.1836
Epoch 15/20
27/27 [==============================] - 14s 529ms/step - loss: 2.9851 - accurac
y: 0.2205 - val_loss: 345.9561 - val_accuracy: 0.1879
Epoch 16/20
27/27 [==============================] - 14s 532ms/step - loss: 3.1350 - accurac
y: 0.2150 - val_loss: 341.2249 - val_accuracy: 0.1882
Epoch 17/20
27/27 [==============================] - 15s 534ms/step - loss: 3.0401 - accurac
y: 0.2384 - val_loss: 339.7192 - val_accuracy: 0.1882
Epoch 18/20
27/27 [==============================] - 14s 533ms/step - loss: 3.0742 - accurac
y: 0.2105 - val_loss: 338.8625 - val_accuracy: 0.1899
Epoch 19/20
27/27 [==============================] - 14s 533ms/step - loss: 3.0196 - accurac
y: 0.2336 - val_loss: 335.8646 - val_accuracy: 0.1915
Epoch 20/20
27/27 [==============================] - 15s 543ms/step - loss: 3.0448 - accurac
y: 0.2372 - val_loss: 336.3132 - val_accuracy: 0.1939
```

Out[15]:

```
<tensorflow.python.keras.callbacks.History at 0x293e75850>
```

Analysis

With the model fully trained, the best accuracy of the model was 31% cross-validation accuracy. The same model is evaluated against the test set and performed at 28% test accuracy. The result is a model that can make classifications better than random guess, which would be 1/96. This is a significant improvement over random guess as random guessing would be correct ~1% of the time.

In [16]:

```
model.evaluate_generator(test_generator, use_multiprocessing=False)
```

```
/Users/nobletang/mambaforge/envs/apple_tensorflow/lib/python3.8/site-packages/te
nsorflow/python/keras/engine/training.py:1877: UserWarning: `Model.evaluate_gene
```

```
rator` is deprecated and will be removed in a future version. Please use `Model.
evaluate`, which supports generators.
  warnings.warn('`Model.evaluate_generator` is deprecated and '
```

Out[16]:
```
[337.1643981933594, 0.18961447477340698]
```

The model could be improved by having more images of flowers to train on. More images means the model could identify just the important features. Also, perhaps the model could improve if more resources were available, like RAM or GPU.

Conclusion

Image classification of flowers is possible with what was gathered for the model. 31% of the images looked at by this model are correctly identified. While the model could be improved, this is a proof of concept for the U.S. Department of Agriculture. The model has much room for improvement given more time and resources, but for the scope of the project, a model has been trained on images to identify the species of a flower.

Future Research

There is room to grow for the model. With more images to even out the class imbalance, the model could drastically improve in accuracy. More computational power would also more epochs and different batch sizes to be run. Lastly, proper installation of dependencies would grant access to different transfer learning applications, possibly increasing the model metrics.