DEV RANGERA CS-B5 · 1M AGO · PRIVATE

Agricultural crops image classification

Python · Agricultural crops image classification

Run
3.7s

Version 1 of 1

Add Tags

Import Packages

In [6]:
```
import os
import pandas as pd
import numpy as np
import random
import shutil
from shutil import copyfile
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from matplotlib.offsetbox import (TextArea, DrawingArea, OffsetImage,
                                  AnnotationBbox)
import matplotlib.patches as mpatches
import warnings
```

In [7]:
```
base_dir  = '/kaggle/input/agricultural-crops-image-classification/Agricultural-crops/'
os.chdir(base_dir)
```

In [8]:
```
# to list every directory name (label name)
directories_list = tf.io.gfile.listdir(base_dir)

# get number of labels
len_labels = len(directories_list)
print(f"Total Class Labels = {len_labels}")

vis_images = []; vis_labels =[]
length_file_list = []; label_list = []

for item in directories_list:

    # get each label directory
    item_dir = os.path.join(base_dir, item)
    # get list of images of each label
    item_files = os.listdir(item)
    # number of images per label
    len_per_label = len(os.listdir(item))

    length_file_list.append(len_per_label)
    label_list.append(item)

    # get first image of each label (for visualisation purpose)
    vis_images.append(os.path.join(item_dir, item_files[0]))
    # get respective label name (for visualisation purpose)
    vis_labels.append(item)

df_temp = pd.DataFrame({'Labels':label_list, 'Number of Images':length_file_list}).\
sort_values(by='Number of Images', ascending=False)
df_temp
```

```
Total Class Labels = 30
```

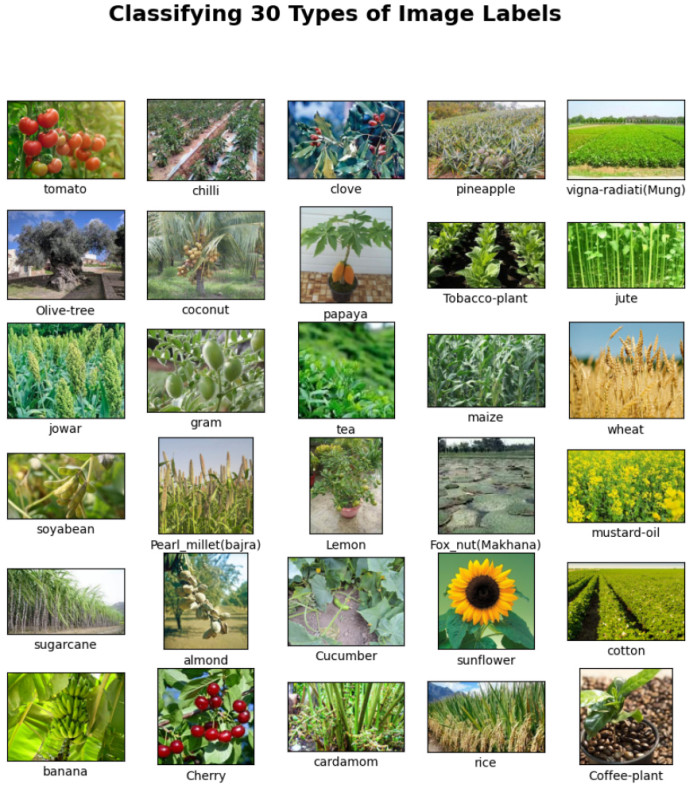
Out[8]:

	Labels	Number of Images
16	Pearl_millet(bajra)	39
8	Tobacco-plant	33
26	Cherry	32
24	cotton	32
13	maize	31
25	banana	31
22	Cucumber	31
14	wheat	31
15	soyabean	30
2	clove	30
5	Olive-tree	30
10	jowar	30
28	rice	29

29	Coffee-plant	29
17	Lemon	28
19	mustard-oil	28
4	vigna-radiati(Mung)	27
0	tomato	26
11	gram	25
20	sugarcane	25
6	coconut	25
3	pineapple	25
23	sunflower	24
12	tea	23
1	chilli	23
18	Fox_nut(Makhana)	23
9	jute	23
7	papaya	23
27	cardamom	22
21	almond	21

In [9]:

```
plt.figure(figsize=(10,10))
for i in range(len(vis_labels)):
    plt.subplot(6,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    img = mpimg.imread(vis_images[i])
    plt.imshow(img)
    plt.xlabel(vis_labels[i])
    plt.suptitle(f"Classifying {len_labels} Types of Image Labels",fontsize=18, fontweight='bold')
plt.show()
```



In [10]:

```
def split_data(SOURCE_DIR, TRAINING_DIR, VALIDATION_DIR, SPLIT_SIZE):

    selected_file_names = []
    all_file_names = os.listdir(SOURCE_DIR)
    for file_name in all_file_names:
        file_path = os.path.join(SOURCE_DIR, file_name)
        size = os.path.getsize(file_path)
        if size != 0:
              selected_file_names.append(file_name)
        else:
              print(f"{file_name} is zero length, so ignoring.")

    random.seed(42)
    selected_train_files = random.sample(selected_file_names, int(SPLIT_SIZE * len(selected_file_nam
es)))
    selected_val_files = [x for x in selected_file_names if x not in selected_train_files]

    for file_name in selected_train_files:
        source = os.path.join(SOURCE_DIR, file_name)
        destination = os.path.join(TRAINING_DIR, file_name)
        copyfile(source, destination)
```

```
    for file_name in selected_val_files:
        source = os.path.join(SOURCE_DIR, file_name)
        destination = os.path.join(VALIDATION_DIR, file_name)
        copyfile(source, destination)
```

In [11]:

```
def create_train_val_dirs(root_path, split_size = 0.9):
    for item in directories_list:
        source_dir = os.path.join(base_dir, item)
        training_dir = os.path.join(root_path, f'_MODELLING/training/{item}')
        validation_dir = os.path.join(root_path, f'_MODELLING/validation/{item}')

        # Create EMPTY directory
        os.makedirs(training_dir)
        os.makedirs(validation_dir)

        split_data(source_dir, training_dir, validation_dir, split_size)
    print(f"Created training and validation directories containing images at split size of {split_si
ze}")
```

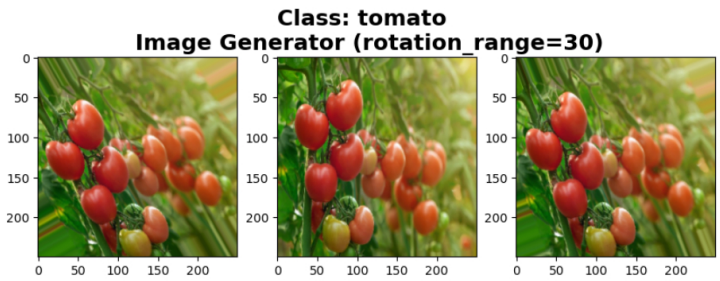
In [12]:

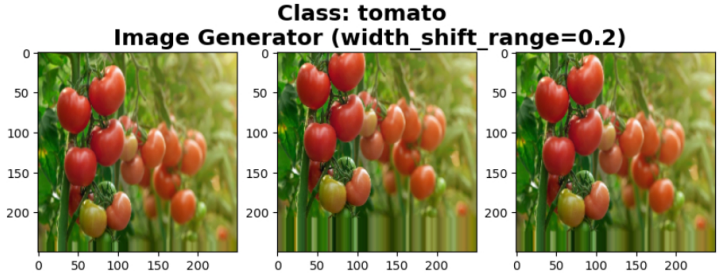
```
create_train_val_dirs('/kaggle/working', split_size = 0.9)
```

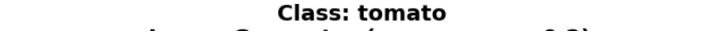
```
Created training and validation directories containing images at split size of 0.9
```

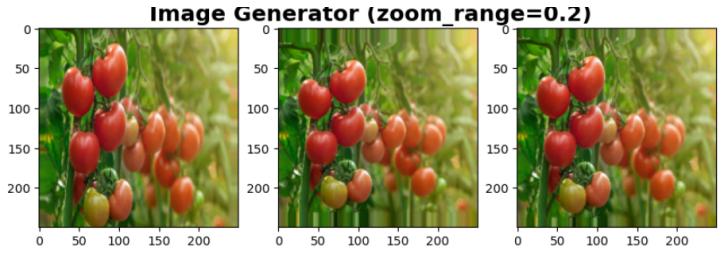
In [13]:

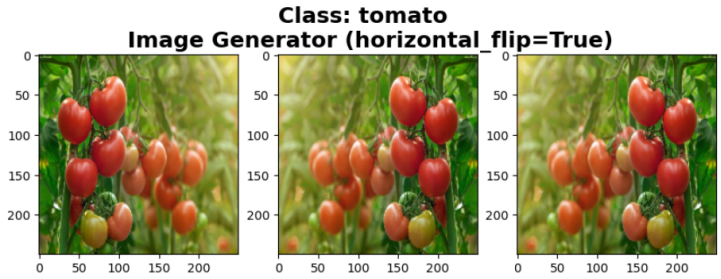
```
def show_ImageDataGenerator(vis_images, vis_labels, image_index):
    #Loads image in from the set image path
    class_label = vis_labels[image_index]
    img = tf.keras.preprocessing.image.load_img(vis_images[image_index], target_size= (250,250))
    img_tensor = tf.keras.preprocessing.image.img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)

    #Creates our batch of one image
    def show_image(datagen, param):
        pic = datagen.flow(img_tensor, batch_size =1)
        plt.figure(figsize=(10,3.5))
        #Plots our figures
        for i in range(1,4):
            plt.subplot(1, 3, i)
            batch = pic.next()
            image_ = batch[0].astype('uint8')
            plt.imshow(image_)
        plt.suptitle(f"Class: {class_label} \n Image Generator ({param})",fontsize=18, fontweight='b
old')

        plt.show()

    datagen = ImageDataGenerator(rotation_range=30)
    show_image(datagen, "rotation_range=30")

    datagen = ImageDataGenerator(width_shift_range=0.2)
    show_image(datagen, "width_shift_range=0.2")

    datagen = ImageDataGenerator(zoom_range=0.2)
    show_image(datagen, "zoom_range=0.2")

    datagen = ImageDataGenerator(horizontal_flip=True)
    show_image(datagen, "horizontal_flip=True")

show_ImageDataGenerator(vis_images, vis_labels, image_index = 0)
```



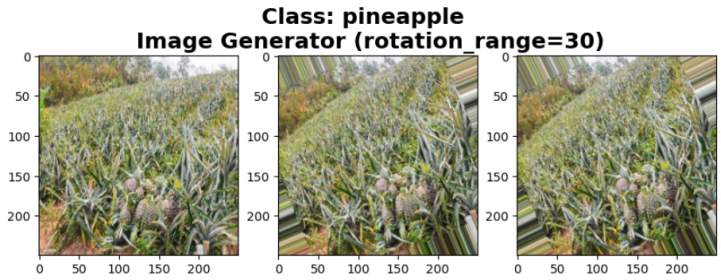


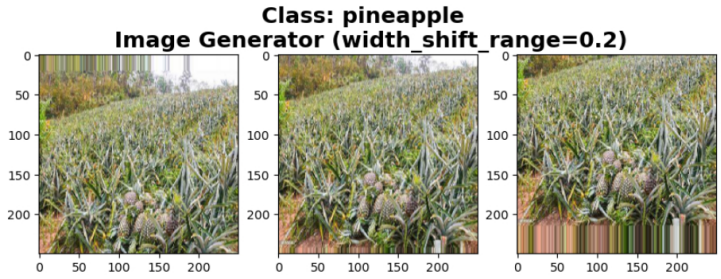


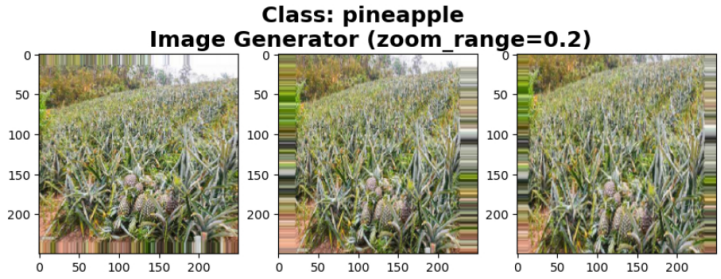


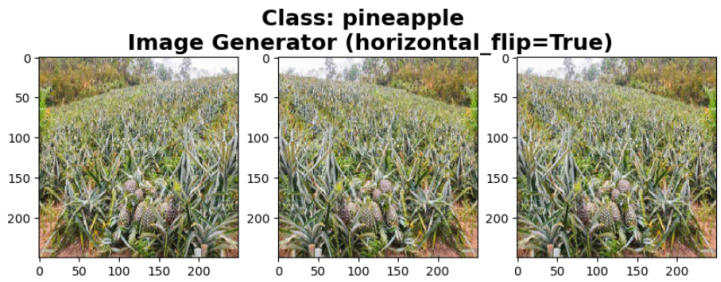


In [14]:
```
show_ImageDataGenerator(vis_images, vis_labels, image_index = 3)
```









In [15]:
```
def train_val_generators(TRAINING_DIR, VALIDATION_DIR):
```

```
    # Instantiate the ImageDataGenerator class (don't forget to set the arguments to augment the image
s)
    train_datagen = ImageDataGenerator(rescale=1./255,
                                       rotation_range=30,
                                       width_shift_range=0.2,
                                       height_shift_range=0.2,
                                       shear_range=0.2,
                                       zoom_range=0.2,
                                       horizontal_flip=True,
                                       fill_mode='nearest')

    # Pass in the appropriate arguments to the flow_from_directory method
    train_generator = train_datagen.flow_from_directory(directory=TRAINING_DIR,
                                                        batch_size=32,
                                                        class_mode='binary',
                                                        target_size=(150, 150))

    # Instantiate the ImageDataGenerator class (don't forget to set the rescale argument)
    validation_datagen = ImageDataGenerator(rescale=1./255)

    # Pass in the appropriate arguments to the flow_from_directory method
    validation_generator = validation_datagen.flow_from_directory(directory=VALIDATION_DIR,
                                                                  batch_size=32,
                                                                  class_mode='binary',
                                                                  target_size=(150, 150))
    return train_generator, validation_generator
```

In [16]:

```
training_dir = os.path.join('/kaggle/working', '_MODELLING', 'training')
validation_dir = os.path.join('/kaggle/working', '_MODELLING', 'validation')

print(validation_dir)
```

```
/kaggle/working/_MODELLING/validation
```

In [17]:

```
train_generator, validation_generator = train_val_generators(training_dir, validation_dir)
```

```
Found 731 images belonging to 30 classes.
Found 98 images belonging to 30 classes.
```

In [18]:

```
model_1 = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 150x150 with 3 bytes color
    # This is the first convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    # The second convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.Dense(len_labels, activation='softmax')
])

# Print the model summary
model_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 64)      1792

 max_pooling2d (MaxPooling2D  (None, 74, 74, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 64)       0
 2D)

 flatten (Flatten)           (None, 82944)             0

 dropout (Dropout)           (None, 82944)             0

 dense (Dense)               (None, 1024)              84935680

 dense_1 (Dense)             (None, 30)                30750

=================================================================
Total params: 85,005,150
Trainable params: 85,005,150
Non-trainable params: 0
_________________________________________________________________
```

In [19]:

```
# Define a Callback class that stops training once accuracy reaches 80%
class myCallback(tf.keras.callbacks.Callback):
```

```
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('accuracy')>0.8):
            print("\nReached 80% accuracy so cancelling training!")
            self.model.stop_training = True
callbacks = myCallback()
```

In [20]:

```
model_1.compile(optimizer = tf.keras.optimizers.Adam(learning_rate=0.001),
            loss = 'sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

In [21]:

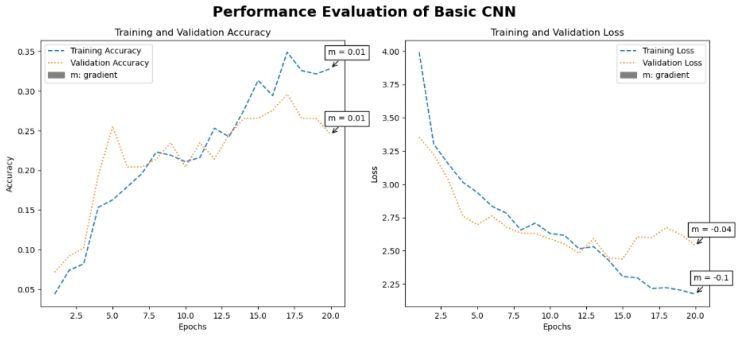
```
history_1 = model_1.fit(train_generator,
                    epochs=20,
                    validation_data=validation_generator,
                    callbacks=callbacks)
```

```
Epoch 1/20
23/23 [==============================] - 35s 1s/step - loss: 3.9921 - accuracy: 0.0438 - val_los
s: 3.3531 - val_accuracy: 0.0714
Epoch 2/20
23/23 [==============================] - 33s 1s/step - loss: 3.3009 - accuracy: 0.0739 - val_los
s: 3.2236 - val_accuracy: 0.0918
Epoch 3/20
23/23 [==============================] - 33s 1s/step - loss: 3.1528 - accuracy: 0.0821 - val_los
s: 3.0329 - val_accuracy: 0.1020
Epoch 4/20
23/23 [==============================] - 33s 1s/step - loss: 3.0155 - accuracy: 0.1532 - val_los
s: 2.7611 - val_accuracy: 0.1939
Epoch 5/20
23/23 [==============================] - 33s 1s/step - loss: 2.9379 - accuracy: 0.1628 - val_los
s: 2.6953 - val_accuracy: 0.2551
Epoch 6/20
23/23 [==============================] - 33s 1s/step - loss: 2.8351 - accuracy: 0.1792 - val_los
s: 2.7623 - val_accuracy: 0.2041
Epoch 7/20
23/23 [==============================] - 32s 1s/step - loss: 2.7814 - accuracy: 0.1956 - val_los
s: 2.6782 - val_accuracy: 0.2041
Epoch 8/20
23/23 [==============================] - 33s 1s/step - loss: 2.6563 - accuracy: 0.2230 - val_los
s: 2.6326 - val_accuracy: 0.2143
Epoch 9/20
23/23 [==============================] - 33s 1s/step - loss: 2.7081 - accuracy: 0.2189 - val_los
s: 2.6293 - val_accuracy: 0.2347
Epoch 10/20
23/23 [==============================] - 32s 1s/step - loss: 2.6302 - accuracy: 0.2107 - val_los
s: 2.5894 - val_accuracy: 0.2041
Epoch 11/20
23/23 [==============================] - 33s 1s/step - loss: 2.6155 - accuracy: 0.2161 - val_los
s: 2.5504 - val_accuracy: 0.2347
Epoch 12/20
23/23 [==============================] - 32s 1s/step - loss: 2.5146 - accuracy: 0.2531 - val_los
s: 2.4815 - val_accuracy: 0.2143
Epoch 13/20
23/23 [==============================] - 33s 1s/step - loss: 2.5305 - accuracy: 0.2421 - val_los
s: 2.5920 - val_accuracy: 0.2449
Epoch 14/20
23/23 [==============================] - 33s 1s/step - loss: 2.4325 - accuracy: 0.2750 - val_los
s: 2.4473 - val_accuracy: 0.2653
Epoch 15/20
23/23 [==============================] - 33s 1s/step - loss: 2.3060 - accuracy: 0.3133 - val_los
s: 2.4372 - val_accuracy: 0.2653
Epoch 16/20
23/23 [==============================] - 33s 1s/step - loss: 2.2970 - accuracy: 0.2941 - val_los
s: 2.6034 - val_accuracy: 0.2755
Epoch 17/20
23/23 [==============================] - 33s 1s/step - loss: 2.2158 - accuracy: 0.3488 - val_los
s: 2.5996 - val_accuracy: 0.2959
Epoch 18/20
23/23 [==============================] - 33s 1s/step - loss: 2.2223 - accuracy: 0.3256 - val_los
s: 2.6738 - val_accuracy: 0.2653
Epoch 19/20
23/23 [==============================] - 33s 1s/step - loss: 2.2035 - accuracy: 0.3215 - val_los
s: 2.6227 - val_accuracy: 0.2653
Epoch 20/20
23/23 [==============================] - 33s 1s/step - loss: 2.1734 - accuracy: 0.3283 - val_los
s: 2.5389 - val_accuracy: 0.2449
```

In [22]:

```
def vis_evaluation(history_dict, model_name):
    fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(15, 6))
    epochs = range(1, len(history_dict['accuracy'])+1)

    def get_gradient(y_arr, epochs):
        return round((y_arr[-1] - y_arr[0]) / (epochs[-1] - epochs[0]),2)

    def vis_sub_evaluation(n, Accuracy, train_acc, val_acc, epochs):
        axs[n].plot(epochs, train_acc, label=f'Training {Accuracy}', ls='--')
        axs[n].plot(epochs, val_acc, label=f'Validation {Accuracy}', ls='dotted')

        axs[n].set_title(f'Training and Validation {Accuracy}')
        axs[n].set_xlabel('Epochs')
        axs[n].set_ylabel(Accuracy)

        handles, labels = axs[n].get_legend_handles_labels()
```

```
        m_patch = mpatches.Patch(color='grey',label='m: gradient')
        handles.append(m_patch)
        axs[n].legend(handles=handles)

        def annotate_box(train_acc):
            return AnnotationBbox(TextArea(f"m = {get_gradient(train_acc, epochs)}"), (epochs[-1], t
rain_acc[-1]),
                                  xybox=(20, 20),
                                  xycoords='data',
                                  boxcoords="offset points",
                                  arrowprops=dict(arrowstyle="->"))
        axs[n].add_artist(annotate_box(train_acc))
        axs[n].add_artist(annotate_box(val_acc))

    train_acc = history_dict['accuracy']
    val_acc = history_dict['val_accuracy']
    vis_sub_evaluation(0, 'Accuracy', train_acc, val_acc, epochs)

    train_loss = history_dict['loss']
    val_loss = history_dict['val_loss']
    vis_sub_evaluation(1, 'Loss', train_loss, val_loss, epochs)

    plt.suptitle(f"Performance Evaluation of {model_name}",fontsize=18, fontweight='bold')
    plt.show()

history_dict_1 = history_1.history
vis_evaluation(history_dict_1, 'Basic CNN')
```



In [ ]:

Continue exploring

Input
1 file →

Output
0 files →

Logs
3.7 second run - successful →

Comments
0 comments →