Flower Classification of Petals to the Metal - Flower Classification on TPU dataset

Assignment goal is to build an Image Classifier . Regarding this dataset we have to explore diverse species of flowers and identify various types and understand popular preferences .

We have to utilize the deep learning classification models which help us to study accurately categorizes distinct flower species . Although the dataset was initially intended for training on a Tensor Processing Unit (TPU), due to its unavailability, we have adapted our processes to effectively utilize a Graphics Processing Unit (GPU) instead.

In Petals to Metal - Flower Classification on TPU dataset is a machine learning challenge competition to classify 104 types of flowers based on their images .

Load The dataset : Ryan Holdbrook , member of competition community post a sample notebook for loading the dataset . I use that notebook for loading the dataset to further process . Since we are not use TPU therefor we configure image size as [224,224] .

we're talking about using different powerful engines to handle different tasks. In this case, the engines are TPUs (Tensor Processing Units) and GPUs (Graphics Processing Units), and we will use GPU only .

When we use TPUs, it's like having a supercharged engine. TPUs are great for handling really big models and large amounts of data at once. They have a lot of memory, which is like having a huge space to work with. In our situation, we're dealing with images that are pretty big - each image is 512 by 512 pixels.

We often use GPUs when we're dealing with smaller images - like 224 by 224 pixels.

```
IMAGE_SIZE = [224, 224]
EPOCHS = 10
SEED = 100
BATCH_SIZE = 64
AUTO = tf.data.experimental.AUTOTUNE
```

```
GCS_DS_PATH = KaggleDatasets().get_gcs_path('tpu-getting-started')
GCS_PATH = GCS_DS_PATH + '/tfrecords-jpeg-224x224'

TRAINING_FILENAMES = tf.io.gfile.glob(GCS_PATH + '/train/*.tfrec')
VALIDATION_FILENAMES = tf.io.gfile.glob(GCS_PATH + '/val/*.tfrec')
TEST_FILENAMES = tf.io.gfile.glob(GCS_PATH + '/test/*.tfrec')
```

Data Augmentation : The most important part in machine learning is Data augmentation . In particularly in the context of training deep learning models like Convolutional Neural network(CNNs) for tasks such as image classification is a crucial technique . This is the step by step process of data augmentation .

- Resizing and Cropping : Resizes the images to a larger size (512x512) and then randomly crops them back to a smaller size (224x224). This technique is basically helps the model to learn to recognize the model in the broader sense .

- Adjusting Brightness : Here we randomly changes the brightness of the image . The main reason is to handle the problem that images are captured under different lighting conditions .

- Adjusting Contrast : Here also we randomly changes the contrast of the image . That is also common for different lighting conditions .

- Blurring : Here we apply a mean filter to blur the image . It’s a straightforward approach to help the model learn to identify features that are important for classification even when the image isn't perfectly sharp.

- Flipping Images : Same as before randomly flips the image horizontally and vertically . This is particularly useful for classes where orientation is not a defining characteristic.

Basically data Augmentation enhance the Data diversity , improving model robustness and generalization and simulating Real-World conditions .

```
def data_augment(image, label):
    # Set seed for data augmentation
    seed = 100
    # Define wrapper functions for each transformation
    def resize_and_crop(image):
        resized_image = tf.image.resize(image, [720, 720])
        return tf.image.random_crop(resized_image, [224, 224, 3], seed=seed)

    def adjust_brightness(image):
        return tf.image.random_brightness(image, 0.6, seed=seed)

    def adjust_saturation(image):
        return tf.image.random_saturation(image, 3, 5, seed=seed)

    def adjust_contrast(image):
        return tf.image.random_contrast(image, 0.3, 0.5, seed=seed)

    def blur_image(image):
        return tfa.image.mean_filter2d(image, filter_shape=10)

    def flip_image(image):
        flipped_image = tf.image.random_flip_left_right(image, seed=seed)
        return tf.image.random_flip_up_down(flipped_image, seed=seed)
```

```
# Apply transformations
image = resize_and_crop(image)
image = adjust_brightness(image)
image = adjust_saturation(image)
image = adjust_contrast(image)
image = blur_image(image)
image = flip_image(image)

return image, label
```

Data Pipelines :

Now the task is acquiring the training dataset , validation dataset and testing dataset after augmentation and also visualize it .

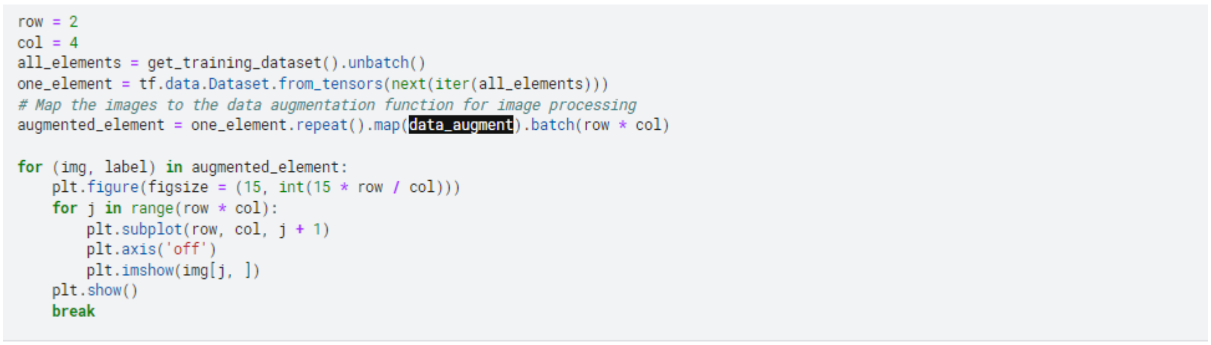
```
def get_training_dataset():

    train = load_dataset(TRAINING_FILENAMES, labeled = True)
    train = train.map(lambda image, label, idnum: [image, label])
    train = train.repeat()
    train = train.shuffle(2048)
    train = train.batch(BATCH_SIZE)
    train = train.prefetch(AUTO)

    return train

# This function is for data visualization
def get_training_dataset_preview(ordered = True):

    train = load_dataset(TRAINING_FILENAMES, labeled = True, ordered = ordered)
    train = train.batch(BATCH_SIZE)
    train = train.cache()
    train = train.prefetch(AUTO)

    return train
```

```
def get_validation_dataset(ordered = False):

    validation = load_dataset(VALIDATION_FILENAMES, labeled = True, ordered = ordered)
    validation = validation.map(lambda image, label, idnum: [image, label])
    validation = validation.batch(BATCH_SIZE)
    validation = validation.cache()
    # Prefetch next batch while training (autotune prefetch buffer size)
    validation = validation.prefetch(AUTO)

    return validation
```

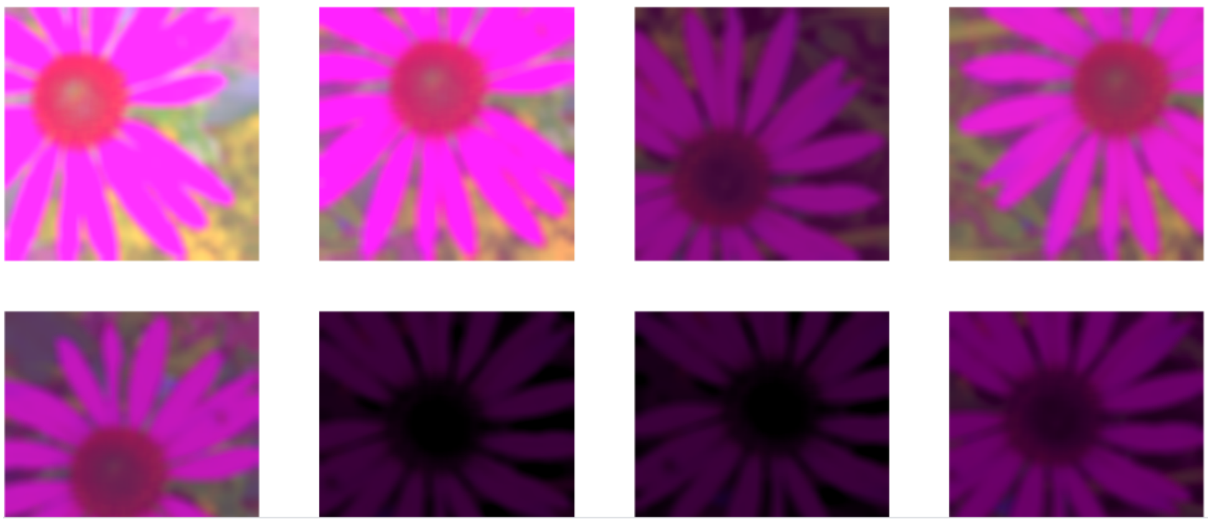
```
def get_test_dataset(ordered = False):

    test = load_dataset(TEST_FILENAMES, labeled = False, ordered = ordered)
    test = test.batch(BATCH_SIZE)
    test = test.prefetch(AUTO)

    return test
```

<u>**Dataset Exploration :**</u> Let’s visualize some dataset before and after augmentation .

Now let’s see the example of augmentation .



```
row = 2
col = 4
all_elements = get_training_dataset().unbatch()
one_element = tf.data.Dataset.from_tensors(next(iter(all_elements)))
# Map the images to the data augmentation function for image processing
augmented_element = one_element.repeat().map(data_augment).batch(row * col)

for (img, label) in augmented_element:
    plt.figure(figsize = (15, int(15 * row / col)))
    for j in range(row * col):
        plt.subplot(row, col, j + 1)
        plt.axis('off')
        plt.imshow(img[j, ])
    plt.show()
    break
```



Handling the Class Imbalanced Dataset : Since in this dataset classes are not balanced that makes irregular problem on model accuracy .To handle this problem different strategies are employed, and one of them is to assign different weights to classes during training.

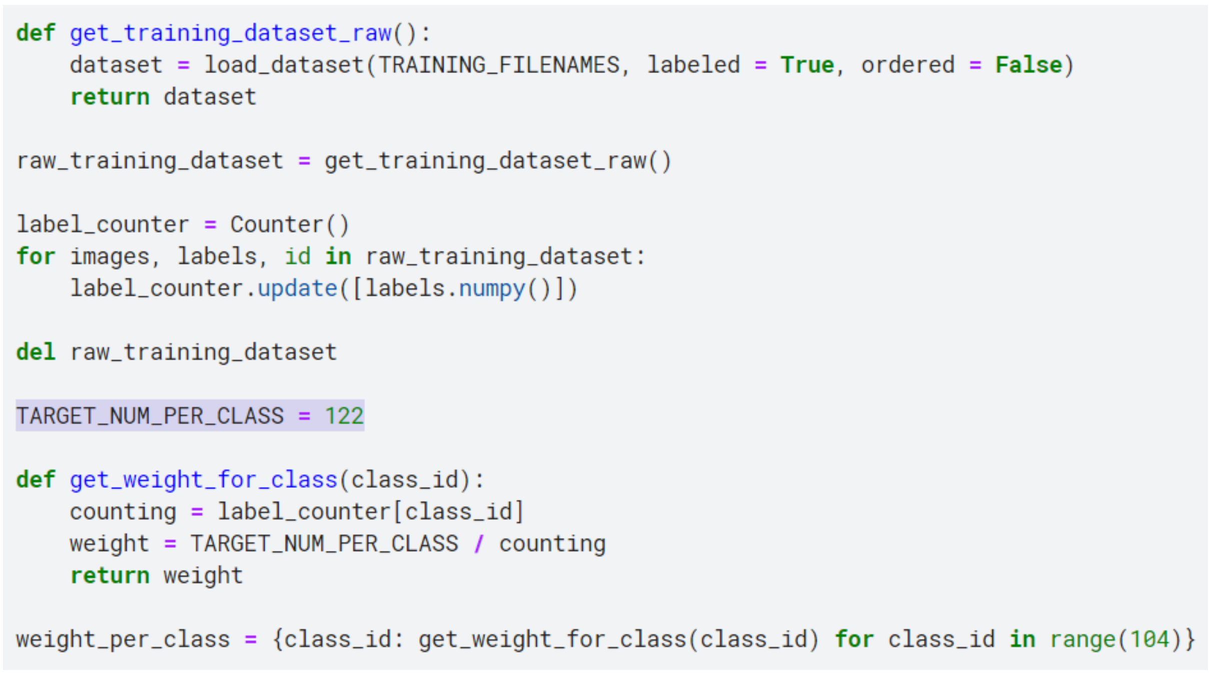
- **Step 1** : Identifying Class Imbalance
- **Step 2** : Setting a Target Number Per Class → A variable TARGET_NUM_PER_CLASS is set, indicating the desired number of samples per class. This number is a hypothetical ideal and does not change the actual dataset.
- **Step 3** : Calculating Class Weights →

  Process :

  1. For each class, the code calculates a weight. This weight is inversely proportional to the class frequency in the dataset.

2. Then weight for a class is computed by dividing the TARGET_NUM_PER_CLASS by the actual number of samples in that class (‘counting’).

- **Step 4** : Creating a Weight Map.
- **Step 5** : Applying weights during the train .



```
def get_training_dataset_raw():
    dataset = load_dataset(TRAINING_FILENAMES, labeled = True, ordered = False)
    return dataset

raw_training_dataset = get_training_dataset_raw()

label_counter = Counter()
for images, labels, id in raw_training_dataset:
    label_counter.update([labels.numpy()])

del raw_training_dataset

TARGET_NUM_PER_CLASS = 122

def get_weight_for_class(class_id):
    counting = label_counter[class_id]
    weight = TARGET_NUM_PER_CLASS / counting
    return weight

weight_per_class = {class_id: get_weight_for_class(class_id) for class_id in range(104)}
```

Model Building : This is the crucial part of the project . we experiment with different model like some small model from scratch and also use transfer learning and then use different layer to enhance the accuracy .

We first use a well-known model called ‘ DenseNet201’ as the foundation of our learning process . we first adopt it as pretrained model which basically learn from a huge collection of images called ‘ImageNet’ . Then customized it to make the model trainable and adding a final layer and as usual use the ‘softmax’ function for classification .

We use ‘Adam’ optimizer and changing the learning rate using ‘LearningrateScheduler’ and took epoch as 30 for model compilation .

```python
with strategy.scope():
    # Create DenseNet(201) model
    rnet = DenseNet201(
        input_shape = (224, 224, 3),
        weights = 'imagenet',  # Use the preset parameters of ImageNet
        include_top = False  # Drop the fully connected network on the top
    )

    rnet.trainable = True
    model = tf.keras.Sequential([
        rnet,
        tf.keras.layers.GlobalAveragePooling2D(),
        tf.keras.layers.Dense(len(CLASSES), activation = 'softmax')
    ])

    model.compile(
        optimizer=tf.keras.optimizers.Adam(),
        loss = 'sparse_categorical_crossentropy',
        metrics = ['sparse_categorical_accuracy']
    )

    model.summary()
    # Save the model
    model.save('DenseNet.h5')
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 densenet201 (Functional)    (None, 7, 7, 1920)        18321984

 global_average_pooling2d (  (None, 1920)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 104)               199784

=================================================================
Total params: 18521768 (70.65 MB)
Trainable params: 18292712 (69.78 MB)
Non-trainable params: 229056 (894.75 KB)
```

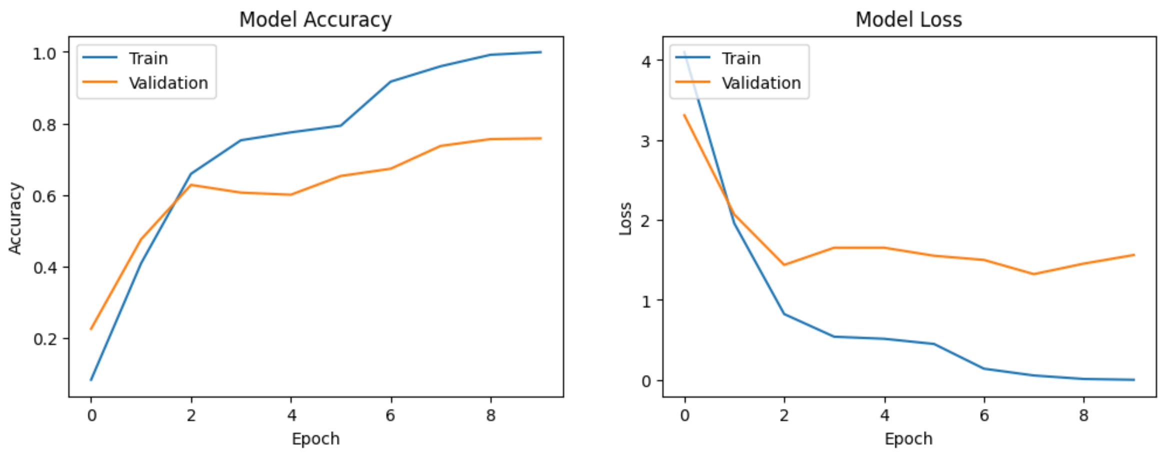
Then we use 2 simple model named as model6 and model3 which gives very very bad accuracy .

```python
with strategy.scope():
    model6 = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape = (224, 224, 3)),
        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.LayerNormalization(),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(256, activation='relu'),
        tf.keras.layers.LayerNormalization(),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.LayerNormalization(),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(len(CLASSES), activation='softmax')
    ])

    model6.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    model6.summary()
    # Save the model
    model6.save('model6.h5')
```

```python
with strategy.scope():
    model2 = tf.keras.Sequential([
        tf.keras.layers.Flatten(input_shape = (224, 224, 3)),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.LayerNormalization(),
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(len(CLASSES), activation='softmax')
    ])

    model2.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['sparse_categorical_accuracy'])
    model2.summary()
    # Save the model
    model2.save('model2.h5')
```

Then we use VGG16 pretrained mode and then train it with extra layers which gives quite good accuracy .

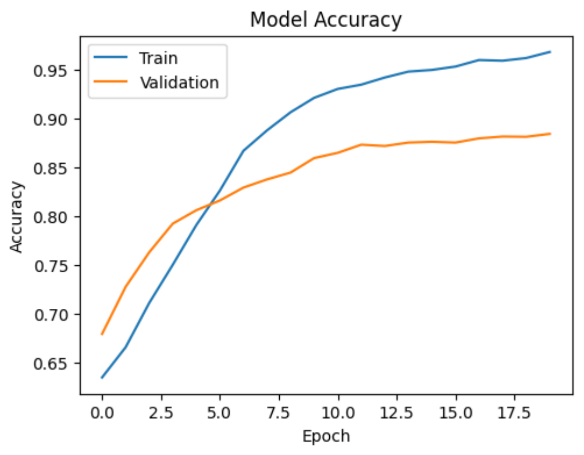
```
with strategy.scope():
  ## Model Load
    base_model_vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    base_model_vgg16.trainable = True

    x = Flatten()(base_model_vgg16.output)
    x = Dense(1024, activation='relu')(x))
    x = Dense(512, activation='relu')(x)
    output = Dense(len(CLASSES), activation='softmax')(x)

    model_vgg16 = Model(base_model_vgg16.input, output)

    model_vgg16.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['sparse_categorical_accuracy'])
    model_vgg16.summary()
```

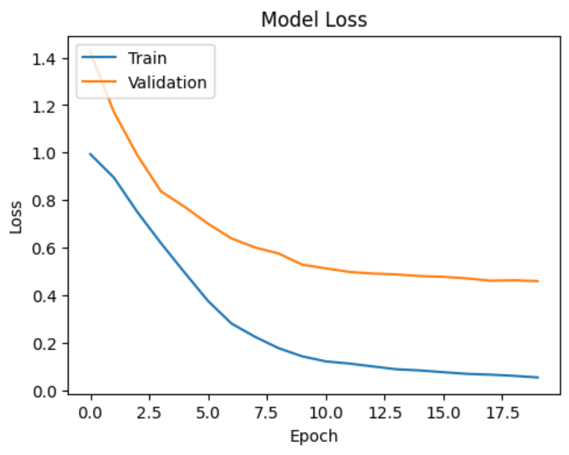


Then use pretrained MobilenetV2 model and add one dense layer with 512 neurons and trained on this dataset and gives better result than last one .

```
from tensorflow.keras.applications import MobileNetV2
with strategy.scope():
# Model Loading
    base_model_mobilenetv2 = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))


    base_model_mobilenetv2.trainable = True

    x = Flatten()(base_model_mobilenetv2.output)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.5)(x)
    output = Dense(len(CLASSES), activation='softmax')(x)

    model_mobilenetv2 = Model(base_model_mobilenetv2.input, output)

    model_mobilenetv2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['sparse_categorical_accuracy'])
    model_mobilenetv2.summary()
```

```
=================================================================================================
Total params: 2967208 (11.32 MB)
Trainable params: 2933096 (11.19 MB)
Non-trainable params: 34112 (133.25 KB)
_________________________________________________________________________________________________
```





From the accuracy and loss visualization we may conclude that model is facing some overfitting issues . then to tackle overfitting issued we have used ‘Dropout(), Layernormalisatio()’ but no one give promising result .

In conclusion we can say that Densenet201 compare to all others. We also conclude that to get better accuracy we need more complex models which have more parameters and may take a huge amount of time on the GPU .

Reference :

1. Create Your First Submission : Petals to Metal competition community .
2. Rotation Augmentation GPU/TPU : Techniques to increase model accuracy and using GPU/TPU : CHRIS DEOTTE

3.