Technical Foundation of A.I

Petals to the Metal

Final Project Report

MUMA COLLEGE OF BUSINESS

UNIVERSITY OF SOUTH FLORIDA

Group 6

Venkata Hima Rohith Reddy Thammuneni

Haritha Karna

About the Kaggle Competition:

This competition narrows its focus to the world of botany. By harnessing the power of TPUs to build a deep learning model to classify over 100 flowers, participants aim to enhance the accuracy and efficiency of flower classification in this extensive dataset. Notably, participants can access TPU quotas on Kaggle at no cost, which opens a world of possibilities for optimizing deep learning models. This competition offers an engaging platform to experiment, innovate, and explore the capabilities of TPUs, ultimately contributing to the advancement of machine learning in the context of biodiversity and botanical recognition. In this report, we will delve into our approach, methodology, and results in tackling this challenging task using TPUs, showcasing the potential of this specialized hardware for image classification tasks.

Interestingly, TPU quotas are freely available on Kaggle For 3 Times, opening a Plenty of opportunities for deep learning model optimization. In the end, this challenge provides a stimulating environment for experimenting, innovating, and exploring TPU capabilities, ultimately advancing machine learning within the context of biodiversity and botanical recognition.

We will examine our strategy, process, and outcomes in this study as we tackle this difficult problem with TPUs, demonstrating the usefulness of this specialized hardware and pre trained models such as Efficient Net for image classification applications.

Understanding TPU:

Tensor Processing Units (TPUs) are hardware accelerators that are specifically designed for deep learning, and Google is the company that invented them. They are essential to this Kaggle challenge, as we explore their deep usefulness in floral classification.

TPUs are renowned for their prowess in parallel processing, surpassing both conventional CPUs and GPUs in handling matrix operations—a crucial task for deep learning. This innate parallelism provides a competitive advantage by speeding up complicated operations. This allows for the efficient execution of deep learning operations, such as matrix multiplications and convolutions.

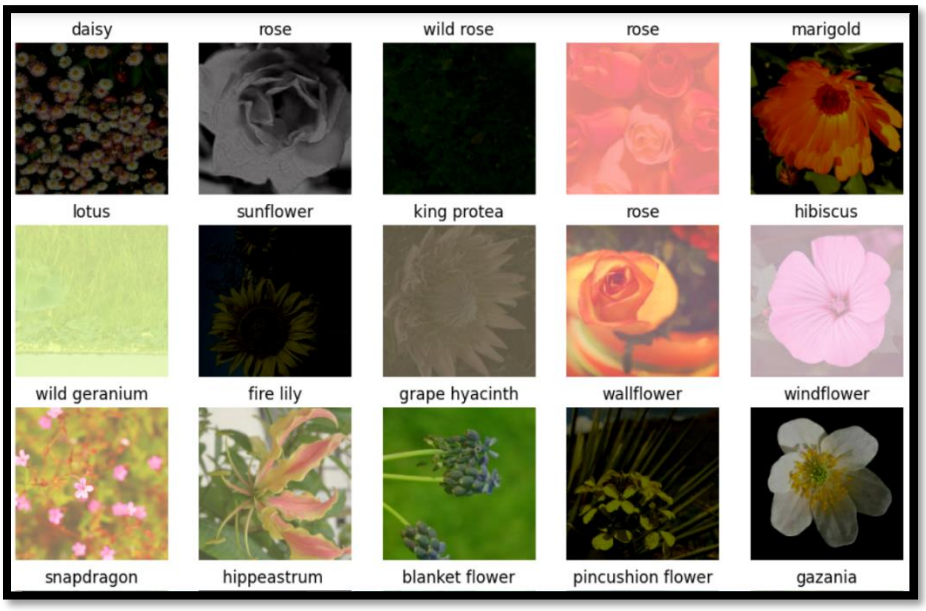
Understanding Efficient Net:

Efficient Net is a novel deep learning architecture that has become well-known for its exceptional neural network design efficiency. It was unveiled by the research team at Google and is designed to provide outstanding performance with a smaller computational footprint. Efficient Net maximizes the model's depth, width, and resolution all at once by using a unique compound scaling technique. It makes sure the network stays computationally economical while offering cutting-edge performance on a variety of tasks, such as object detection and image classification, by carefully balancing these characteristics.

Efficient Net’s engineering essentially affects the profound learning local area, as it represents how to figure out some kind of harmony between model size and precision. It has turned into a go-to decision for building proficient yet strong brain organizations, making it a significant resource in present day AI and PC vision applications.

Data Exploration Insights:

- Data given by Kaggle was stored in Data stored in TFRecord format, a common container format in TensorFlow for efficient storage and retrieval.
- The Dataset Consisted of 104 labels in total. A few of them were type of flowers and others were subtypes.



- We have received the dataset in 3 chunks
  1) Training set: 12,753 images
  2) Validation set: 3,712 images
  3) Testing set: 7,382 images

```
num_training_images = count_data_items(TRAINING_FILENAMES)
num_validation_images = count_data_items(VALIDATION_FILENAMES)
num_test_images = count_data_items(TEST_FILENAMES)

print(f"Number of training images:   {num_training_images:,d}.")
print(f"Number of validation images: {num_validation_images:,d}.")
print(f"Number of testing images:    {num_test_images:,d}.")

train_dataset = get_training_dataset()
val_dataset = get_validation_dataset()
test_dataset = get_test_dataset(ordered=True)
```

```
Number of training images:   12,753.
Number of validation images: 3,712.
Number of testing images:    7,382.
```

- Resolution and pixel sizes variations required preprocessing for standardization and optimal model performance. Non- uniform data with varying resolution and pixel sizes across different images

Data Preprocessing (Augmentation):

Data Augmentation expansion is a method utilized in Machine learning to improve the variety of prepared information by applying irregular changes to enter tests. It incorporates tasks like flips, turns, brilliance, and differentiation changes, mimicking certifiable varieties.

We have used data augmentation making the model to model more robust to unseen data or real-world scenario. It also helps Overfitting by ensuring that the model doesn't memorize the training data set but instead learns the captures the patterns.

```
def data_augment(image, label):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)
    image = tf.image.random_saturation(image, 0, 2)
    image = tf.image.random_brightness(image, max_delta=0.5)
    image = tf.image.random_contrast(image, lower=0.1, upper=0.9)
    image = tf.image.rot90(image, k=tf.random.uniform([], 0, 4, dtype=tf.int32))
    return image, label
```

Model Architecture:

The model architecture employed in this study utilizes the EfficientNetB0 pre-trained convolutional neural network (CNN) available through TensorFlow's Keras applications. This architecture capitalizes on transfer learning, with the pre-trained model serving as the foundation. The top layers are customized, featuring a global average pooling layer for feature aggregation, followed by a dense layer for categorical class predictions. The model is compiled using the Adam optimizer, categorical cross-entropy loss function, and accuracy as the primary performance metric. This configuration optimizes the balance between computational efficiency and classification capability, making it a robust choice for image classification tasks in the competition.

```python
def get_model():
    backbone = tf.keras.applications.EfficientNetB0(
        include_top=False,
        weights="imagenet",
        input_shape=[IMAGE_SIZE, IMAGE_SIZE, 3],
    )
    model = tf.keras.models.Sequential(
        [
            backbone,
            tf.keras.layers.GlobalAveragePooling2D(),
            tf.keras.layers.Dense(len(CLASSES), activation="softmax"),
        ]
    )
    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
        loss="categorical_crossentropy",
        metrics=["accuracy"],
    )
    return model
```

```python
model = get_model()
model.summary()
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb0_notop.h5
16705208/16705208 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb0 (Functional)  (None, 6, 6, 1280)       4049571

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 104)               133224

=================================================================
Total params: 4,182,795
Trainable params: 4,140,772
Non-trainable params: 42,023
```

Model Training:

The model uses the Efficient Net design for a total of 30 epochs. Preparing is performed on the training dataset. Also, two fundamental callbacks are carried out:

ModelCheckpoint:

The Model Checkpoint callback is a crucial component of model training. It saves the best-performing model based on a specified monitoring metric, such as validation accuracy, ensuring that the model's progress is preserved. This safeguard helps prevent the loss of valuable training progress in case the training process is interrupted, and it is especially useful in long training sessions.

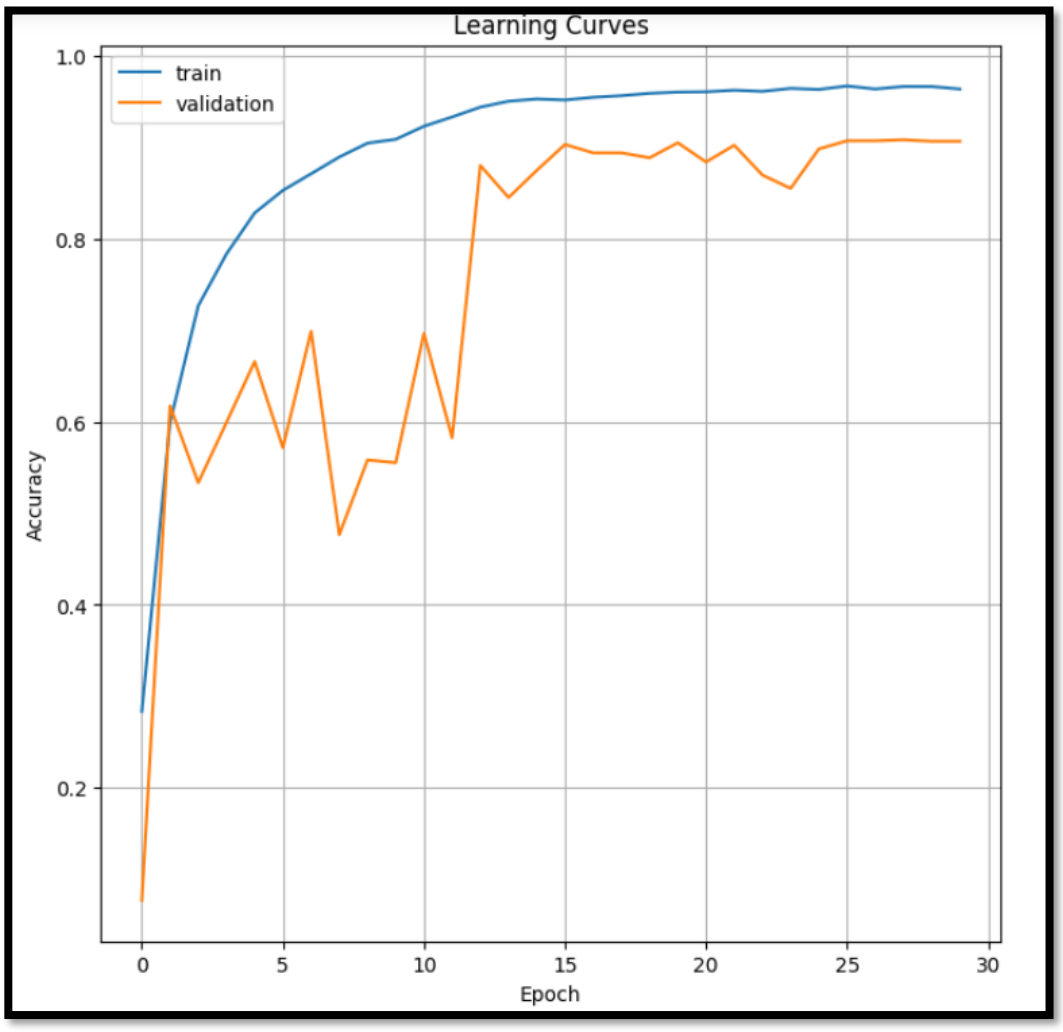
ReduceLROnPlateau:

The ReduceLROnPlateau callback is employed to dynamically adjust the learning rate during training. It monitors a specified metric, such as validation accuracy, and reduces the learning rate when the metric's improvement plateaus. This adjustment helps the model fine-tune its parameters and overcome training plateaus.

```
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=30,
    steps_per_epoch=num_training_images // BATCH_SIZE,
    validation_steps=num_validation_images // BATCH_SIZE,
    callbacks=[
        tf.keras.callbacks.ModelCheckpoint(
            "model.h5",
            monitor="val_accuracy",
            mode="max",
            save_best_only=True,
            save_weights_only=True,
            verbose=1,
        ),
        tf.keras.callbacks.ReduceLROnPlateau(
            monitor='val_accuracy',
            mode='max',
            patience=5,
            min_lr=1e-6,
            verbose=2,
        ),
    ],
    verbose=1 if os.environ["KAGGLE_KERNEL_RUN_TYPE"] == "Interactive" else 2,
).history
```

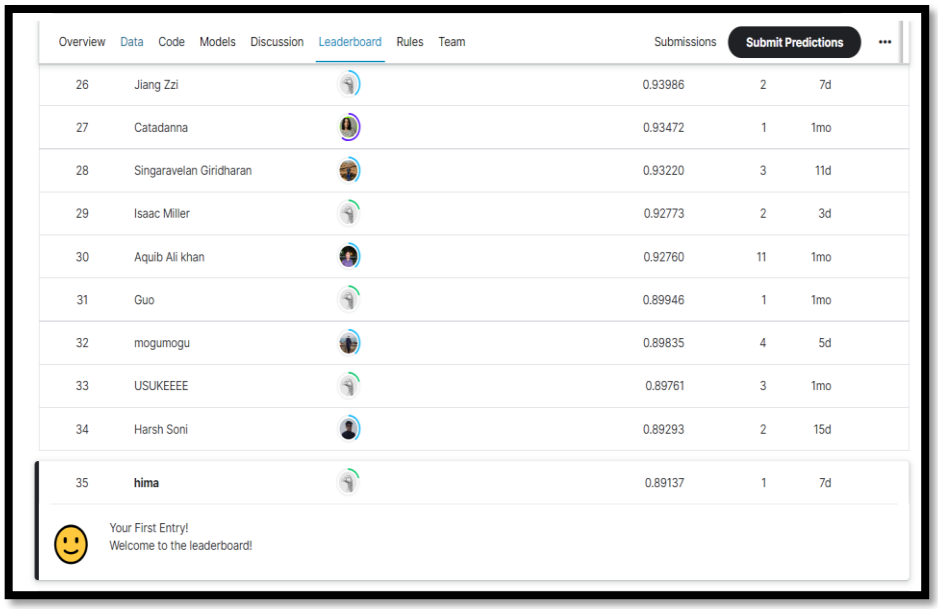
Results and Learning Curves:

After 30 epochs the **final model accuracy was 96.3%** and loss of 0.13 for the training data set. While the validation accuracy was 90% and validation loss 0.38. To check for any over fitting of the model we have plot between the training accuracy and validation accuracy. And could confirm that the model neither overfitting nor underfitting. We have also checked the model performance by validating random samples from the data.



Kaggle Ranking:

After Submitting the final file on the Kaggle. We have been ranked 35 among 140 participants.



Challenges with the current Project:

- Dealing with a dataset of **104 labels** and different types and subtypes of flowers was challenging in ensuring the quality and consistency of the data.
- The **4-attempt** constraint on Kaggle was challening, limiting iterations and fine-tuning. Addressing misclassifications is challenging without multiple runs for in-depth analysis and root cause identification, constraining improvements.
- Determining the optimal hyperparameters for training the model was **time-consuming**. Balancing between model complexity and avoiding overfitting while achieving high accuracy required thorough experimentation.
- The limited timeframe of only **10 hours** for this Kaggle competition posed a significant challenge, requiring careful planning and prioritization of tasks. Meeting the deadline while experimenting with different approaches and fine-tuning the model was challenging.
- Working with **TPUs** is powerful, but it also comes with challenges in terms of understanding and efficiently utilizing computational resources. Managing TPU quotas and optimizing the model for TPU architecture was challenging.

Future Scope:

- Experiment with advanced data augmentation techniques to further enhance the model's robustness to variations in flower images, potentially improving generalization.
- Implement explainability techniques to interpret model decisions, providing insights into the features influencing flower classification and increasing model transparency.
- Experimenting with alternative pre-trained models, such as VGG-16 or ResNet, to assess their performance in comparison to

EfficientNet. Exploring different architectures may uncover nuances in how each model captures and generalizes botanical features, providing insights for further model optimization and potentially improving overall classification accuracy.

- Collaboration with botanists and ecologists can open avenues for domain-specific adaptations. Fine-tuning the model based on expert knowledge can lead to a more nuanced understanding of botanical features, enabling the system to identify subtle differences crucial for accurate classification in the wild.
- Additionally, exploring the integration of other modalities, such as environmental data or multispectral imaging, could enhance the model's capabilities for comprehensive ecological studies. This multidisciplinary approach could contribute significantly to the intersection of machine learning and botanical sciences, fostering advancements in biodiversity research and conservation efforts.

References:

- https://www.youtube.com/playlist?list=PLqFaTIg4myu-1c3ygYzakW8-hNzQG59-5
- https://www.kaggle.com/docs/tpu
- https://www.kaggle.com/competitions/tpu-getting-started/discussion
- https://www.kaggle.com/code/ryanholbrook/
- https://arxiv.org/abs/1905.11946
- https://medium.com/mlearning-ai/understanding-efficientnet-the-most-powerful-cnn-architecture-eaeb40386fad
- https://huggingface.co/docs/transformers/model_doc/efficientnet