Project Report: Animal Classification using Fine-Tuned EfficientNet-B0

1. Introduction

This project report outlines the development of an image classification system designed to recognize 15 different animal species `[Bear, Bird, Cat, Cow, Deer, Dog, Dolphin, Elephant, Giraffe, Horse, Kangaroo, Lion, Panda, Tiger, Zebra]`. The project utilizes a fine-tuned `EfficientNet-B0` model, a convolutional neural network architecture known for its efficiency and performance. The primary goal of this project is to demonstrate the application of fine-tune transfer learning principles and the process of building a machine learning model for a specific task like image classification.

2. Project Objectives

- Develop a multi-class image classification model capable of identifying 15 distinct animal species.
- Apply transfer learning techniques to leverage a pre-trained `EfficientNet-B0` model, shortening training time and resources.
- Fine-tune the `EfficientNet-B0` model by unfreezing its top 10 layers, allowing for adaptation to the specific animal dataset.
- Implement distributed training with PyTorch for multi-GPU usage, increasing performance and reducing training time.
- Thoroughly evaluate the model's performance using a variety of metrics and visualizations.
- Use **MLFlow** via **Dagshub** to enable experiment tracking and management.
- Finally, deploy the model as a web application using `Streamlit` for user-friendly interface and real-time predictions.

3. Methodology

The project follows these steps:

3.1. Data Acquisition and Preparation (Section 1)

- **GitHub Repository Cloning:** The project begins by cloning a GitHub repository containing the required dataset and supporting scripts. This is a standard practice to manage and version control the project's components.

```bash
!git clone https://github.com/PranayJagtap06/UFM_Animal_Classification.git
```

- **Dataset Extraction:** A zipped archive (`animal_classification.zip`) containing the image dataset is extracted to the local working directory. The dataset is extracted from a zip file which can be downloaded from Kaggle, and the code includes the options to download from kaggle or colab.

```python
 import zipfile
zip_ref = zipfile.ZipFile("/content/UFM_Animal_Classification/animal_classification.zip",
'r')
zip_ref.extractall("/content")
zip_ref.close()
```

- **Directory Creation:** The images, initially stored in a single directory, are reorganized into separate 'train' and 'test' directories. This division is crucial for training and evaluating the model appropriately. The code uses conditional checks to create the directories if they don't exist.

```python
 import os
import shutil
import random
# Define the dataset folder and the split ratio
dataset_folder = '/content/Animal Classification/dataset'
train_ratio = 0.8
# Create the training and testing folders if they don't exist
train_folder = os.path.join(dataset_folder, 'train')
test_folder = os.path.join(dataset_folder, 'test')
if not os.path.exists(train_folder):
  os.makedirs(train_folder)
if not os.path.exists(test_folder):
  os.makedirs(test_folder)
```

- **Data Splitting:** Images are split with a 80:20 ratio between train and test datasets. The files from each animal category are shuffled and split based on the ratio. The split process is printed to confirm the number of training and test images per animal category.

3.2. Data Loading and Preprocessing (Section 2)

- `data_loader.py` **Script:** The `data_loader.py` script contains the dataset loading and preprocessing logic. The script dynamically creates training and validation datasets, including distributed training. The script first loads weights, applies appropriate image transformation as per the downloaded weights, downloads the `EfficientNet-B0`, and recreates the classifier for a 15 class output.
- **Pre-Trained Weights Loading:** Pre-trained weights of `EfficientNet-B0` are loaded. These weights are essential as they allow the model to leverage the existing knowledge of image features instead of learning from scratch. The model can be then used for fine tuning.

```python
 weights = torchvision.models.EfficientNet_B0_Weights.DEFAULT
model = torchvision.models.efficientnet_b0(weights=weights)
```

- **Image Transformations:** A transformation pipeline is defined for image preprocessing, including resizing, normalization, and conversion to tensors.

```python
 auto_transforms = weights.transforms()
torch.save(auto_transforms, "effnetb0_transform.pt")
```

- **Custom Model Classifier:** A new classifier layer is created and placed at the end of `EfficientNet-B0` to output 15 classes of animal images.

```python
 model.classifier = nn.Sequential(
    nn.Dropout(p=0.2, inplace=True),
    nn.Linear(
    in_features=model.classifier[1].in_features,
    out_features=len(class_names), bias=True
    ))
```

- **Data Loaders:** PyTorch's `ImageFolder` and `DataLoader` classes are used to create datasets for training and validation. The training dataset is configured with an option for distributed sampling via the `DistributedSampler` if multiple GPUs are used.

3.3. PyTorch Engine (Section 3)

- `pt_engine.py` **Script:** This script contains the code for the training loop, evaluation loop, early stopping, and data saving.
- `CustomTrainer` **Class:** This class encapsulates the entire training process. It is initialized with all the required elements for the training including `model`, `dataloaders`, `criterion`, `optimizer`, `gpu-id`, path, `patience`, `max_epochs`, `world_size`, and a s `cheduler`.
- **Distributed Data Parallel:** The model is wrapped using Distributed Data Parallel if GPU is enabled.
- **Training Loop:** A training loop using batches of data, is defined which runs for every batch of training data. The loss is calculated and backpropagated.
- **Evaluation Loop:** Similarly, an evaluation loop is defined for every epoch and calculates the performance metrics.
- **Early Stopping:** The training process includes a custom implementation of early stopping to prevent overfitting by observing the validation loss. The model is only saved at the end of the epoch, if the performance on validation data has improved.
- **Metric Calculation:** `torchmetrics` library provides calculation of validation metrics, including accuracy and f1 score.
- **Tensor Gathering:** The results such as training losses, validation losses, accuracies, and f1 scores from all available GPUs are gathered to calculate overall performance of the training.

3.4. Training Script (Section 4)

- `pt_train.py` **Script:** The training script integrates all components and performs the overall model training.
- **Argument Parsing:** The script uses `argparse` to specify parameters like total epochs, batch size, GPU usage, model save path, training and validation paths, learning rate and a learning rate scheduler.
- **Distributed Training:** Implements distributed training using PyTorch's DDP (DistributedDataParallel) if GPU option is selected. It also takes in consideration the world_size, which corresponds to how many gpus to use. The script initializes the process group, sets the device, and handles the cleanup of the process group.
- **Free Port:** The script also uses the `find_free_port()` function, which helps to locate and free port for the distributed training communication.
- `set_seed` **:** The script uses the set seed function to control randomness to make the model training results reproducible.
- **Main Training Loop:** The training script uses `mp.spawn` to handle multi-gpu training and executes the main function. The model is then trained using the custom trainer class.
- **Data Handling:** The script loads the data by using the custom data loader.

3.5. Model Training (Section 5)

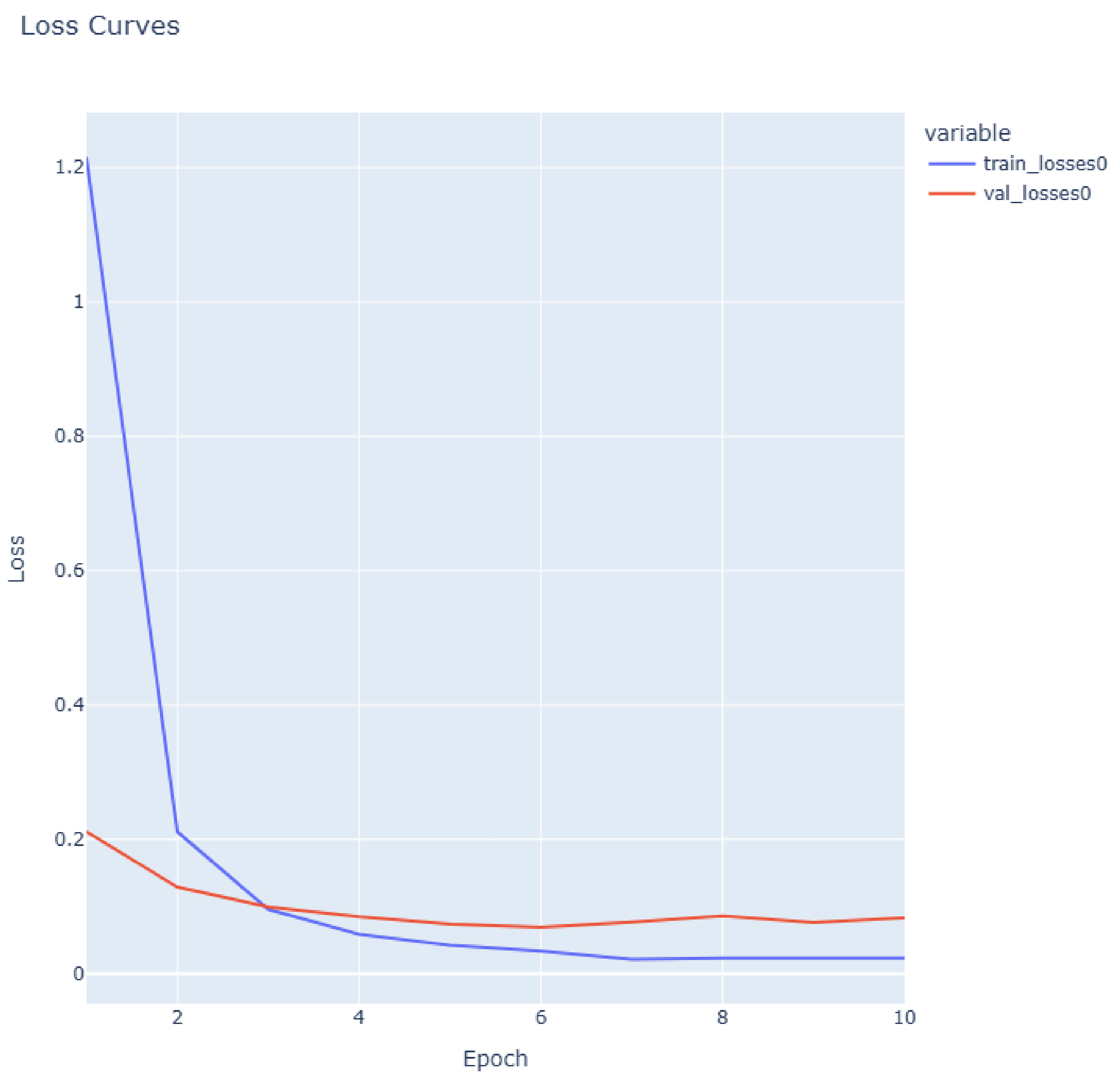
- **Command-line Execution:** The `pt_train.py` script is executed from the notebook. The command line interface (CLI) options control various parameters, such as which data to use for training and validation, learning rate, batch size and so on.

```bash
 !python pt_train.py --total_epochs 10 --batch_size 64 --gpu --xtrain_path
'/content/Animal Classification/dataset/train' --xval_path '/content/Animal
Classification/dataset/test' --learning_rate 0.001 --world_size 1
```

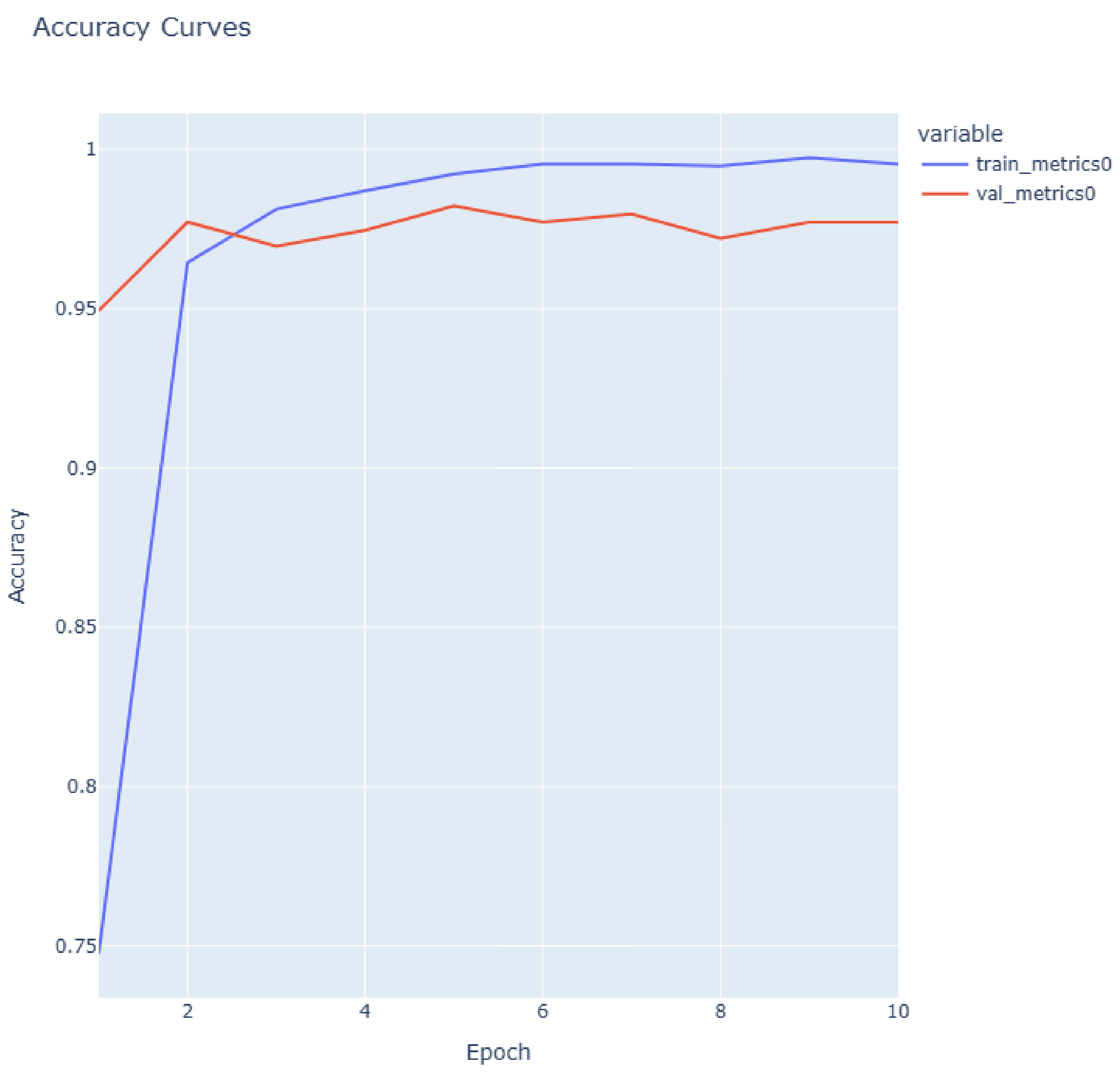
- **Output:** During training, the code displays real-time updates on training and validation metrics, epochs, batch processing speed and also saves the best performing model according to the validation results.

3.6. Model Evaluation (Section 6)

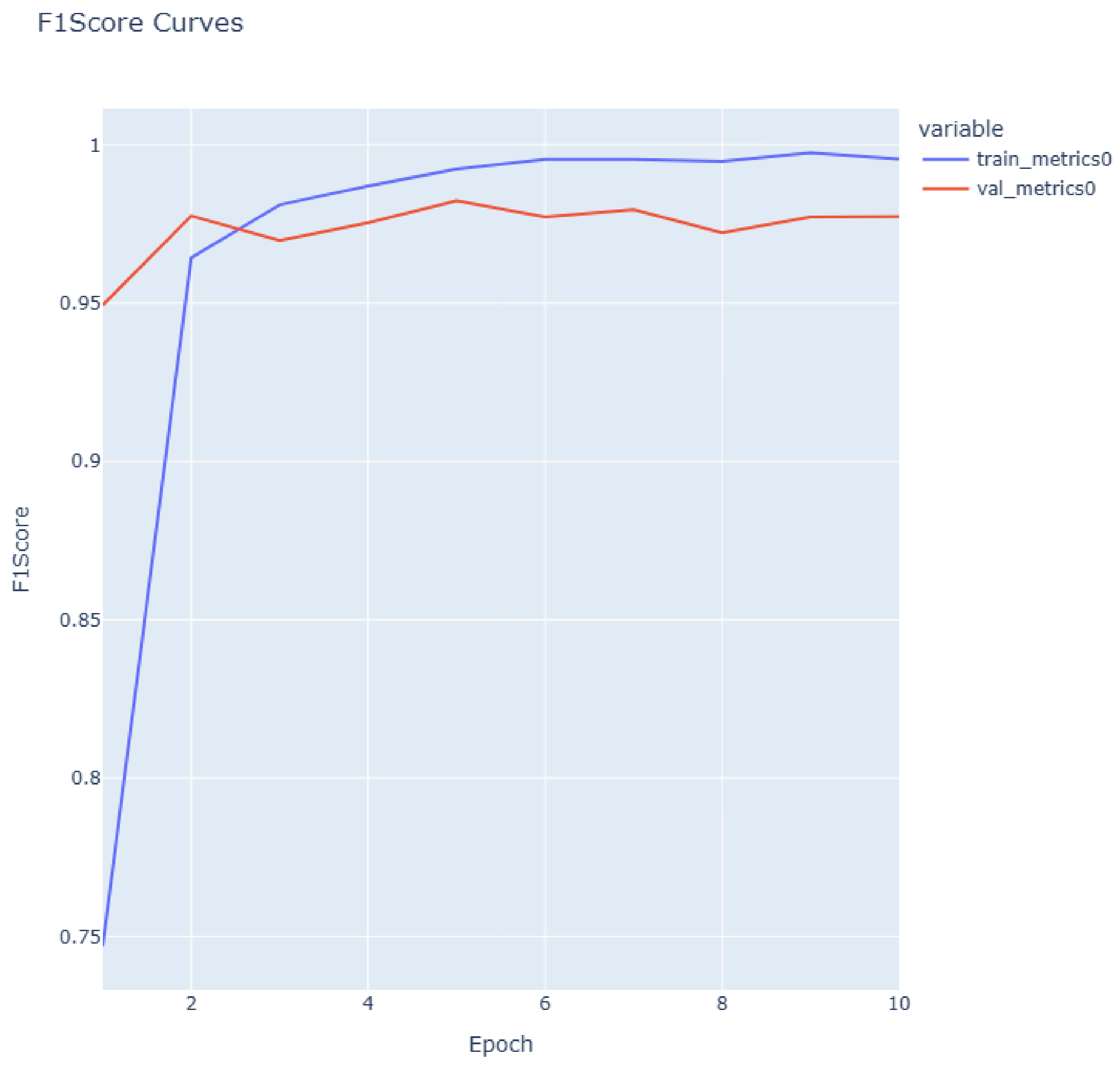
- **Metric Visualization:** The notebook loads the tracked losses, accuracies and f1 scores from files that are created by the model trainer. The loaded performance results are plotted using plotly for further analysis.



Loss Curves



Accuracy Curves



F1Score Curves

- Both accuracy and F1-score curves show very similar patterns, which suggests consistent performance across classes. The model achieves impressive final metrics, with training accuracy/F1-score around 99% and validation accuracy/F1-score around 97%. Let's break down the learning progression:
  - *Initial Learning Phase (Epochs 1-4):* The model shows rapid improvement in both metrics, with training performance starting from around 77% and quickly climbing to nearly 99%. This steep learning curve indicates that the model is effectively capturing the distinguishing features of different animal classes during early training.
  - *Convergence Phase (Epochs 4-10):* After epoch 4, both metrics stabilize, with training metrics hovering around 99% and validation metrics around 97%. This consistent gap between training and validation performance (about 2%) indicates a small but acceptable level of overfitting.
  - *Initial Loss Reduction:* Training loss starts quite high (around 1.2) and drops dramatically in the first two epochs, while validation loss starts lower (around 0.3). This pattern is typical when using transfer learning with a pre-trained model like EfficientNetB0.
  - *Loss Convergence:* After epoch 4, both training and validation losses stabilize, with training loss slightly lower than validation loss. The final values (approximately 0.05 for training and 0.1 for validation) indicate good model convergence.
  - *Overall Assessment:* The high F1-scores suggest good performance across all animal classes, indicating balanced learning. The model reaches stability relatively quickly (by epoch 4), suggesting efficient

learning. The moderate gap between training and validation metrics indicates the model should generalize well to new images.

- **Performance Report:** The script calculates and displays a detailed classification report, which includes metrics like precision, recall, F1-score, and support for each animal class. This is an important step to see if any specific class is suffering during the training process.

    - *Accuracy:* 97.72% - This signifies that the model correctly classified nearly 98% of the animals in the validation/test set. This is a very high accuracy, indicating strong overall performance.
    - *Macro Average:* This averages the precision, recall, and F1-score across all classes. A macro average of 0.98 is excellent and shows that the model performs well across all animal categories.
    - *Weighted Average:* This is similar to macro average, but it weights the scores based on the number of samples in each class. The 0.98 weighted average confirms the model's strong performance, especially considering class imbalances.
    - *Precision:* Out of all the animals predicted to be a certain class, what proportion was actually correct?
    - *Recall:* Out of all the animals that truly belong to a certain class, what proportion did the model correctly identify?
    - *F1-Score:* The harmonic mean of precision and recall, providing a balanced measure of the model's performance.
    - *High Precision and Recall:* Most classes have precision and recall values above 0.90, implying that the model is accurate in identifying each animal and rarely misclassifies them.
    - *Perfect Scores:* Some animals (Lion, Panda, Tiger, Zebra, Deer, Bear, Giraffe) achieved perfect or near-perfect scores (1.00 or 0.98+). This highlights the model's exceptional ability to distinguish these animals.
    - *Slight Variations:* While most classes performed exceptionally well, a few (Cow, Horse) show slightly lower scores (around 0.90). This suggests there might be some confusion in distinguishing these animals from others, possibly due to visual similarities or fewer training samples.

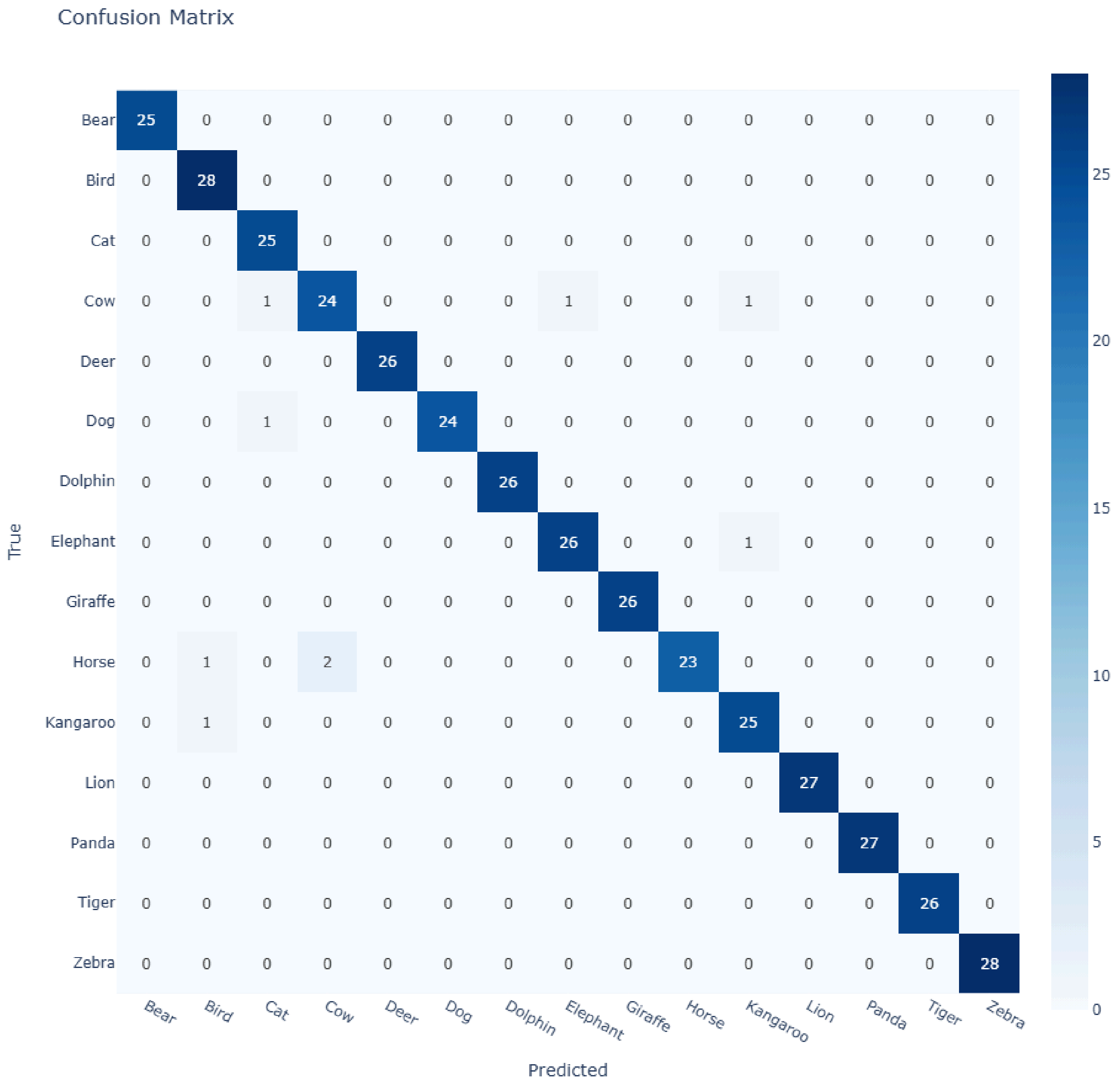
- **Confusion Matrix:** Confusion matrices are plotted to check how the model is behaving when identifying images across the categories on both train and validation data. This is an important step to identify if any classes are getting confused with each other.

    -
      ```
      Classification Report:
                    precision    recall  f1-score   support


              Bear       1.00      1.00      1.00        25
              Bird       0.93      1.00      0.97        28
               Cat       0.93      1.00      0.96        25
               Cow       0.92      0.89      0.91        27
              Deer       1.00      1.00      1.00        26
               Dog       1.00      0.96      0.98        25
           Dolphin       1.00      1.00      1.00        26
          Elephant       0.96      0.96      0.96        27
           Giraffe       1.00      1.00      1.00        26
             Horse       1.00      0.88      0.94        26
          Kangaroo       0.93      0.96      0.94        26
              Lion       1.00      1.00      1.00        27
             Panda       1.00      1.00      1.00        27
             Tiger       1.00      1.00      1.00        26
             Zebra       1.00      1.00      1.00        28


          accuracy                           0.98       395
      ```

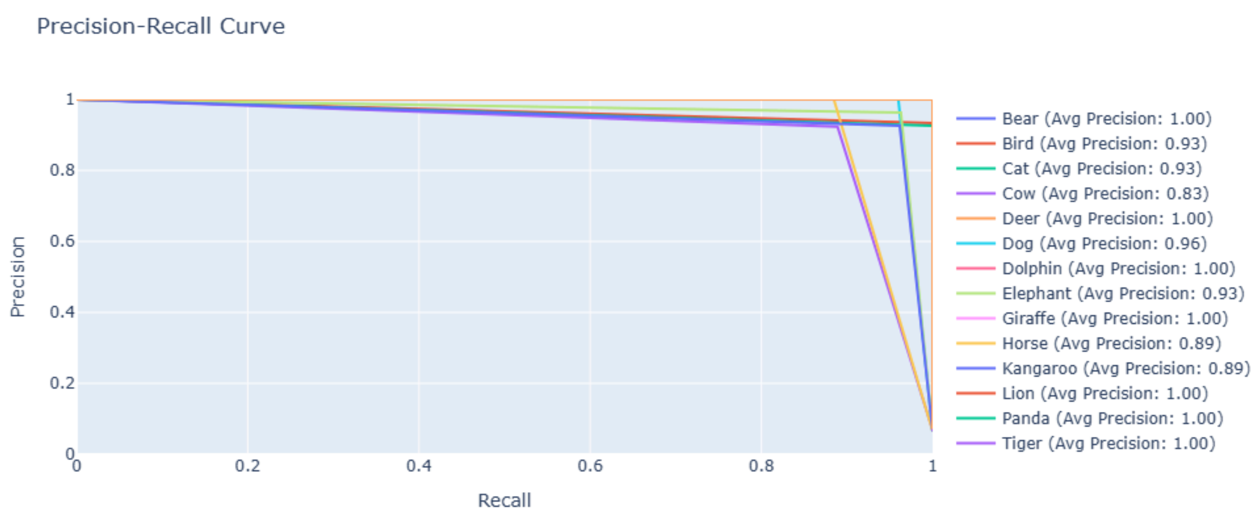
```
   macro avg       0.98      0.98      0.98       395
weighted avg       0.98      0.98      0.98       395
```



True \ Predicted	Bear	Bird	Cat	Cow	Deer	Dog	Dolphin	Elephant	Giraffe	Horse	Kangaroo	Lion	Panda	Tiger	Zebra
Bear	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bird	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0
Cat	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0
Cow	0	0	1	24	0	0	0	1	0	0	1	0	0	0	0
Deer	0	0	0	0	26	0	0	0	0	0	0	0	0	0	0
Dog	0	0	1	0	0	24	0	0	0	0	0	0	0	0	0
Dolphin	0	0	0	0	0	0	26	0	0	0	0	0	0	0	0
Elephant	0	0	0	0	0	0	0	26	0	0	1	0	0	0	0
Giraffe	0	0	0	0	0	0	0	0	26	0	0	0	0	0	0
Horse	0	1	0	2	0	0	0	0	0	23	0	0	0	0	0
Kangaroo	0	1	0	0	0	0	0	0	0	0	25	0	0	0	0
Lion	0	0	0	0	0	0	0	0	0	0	0	27	0	0	0
Panda	0	0	0	0	0	0	0	0	0	0	0	0	27	0	0
Tiger	0	0	0	0	0	0	0	0	0	0	0	0	0	26	0
Zebra	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28

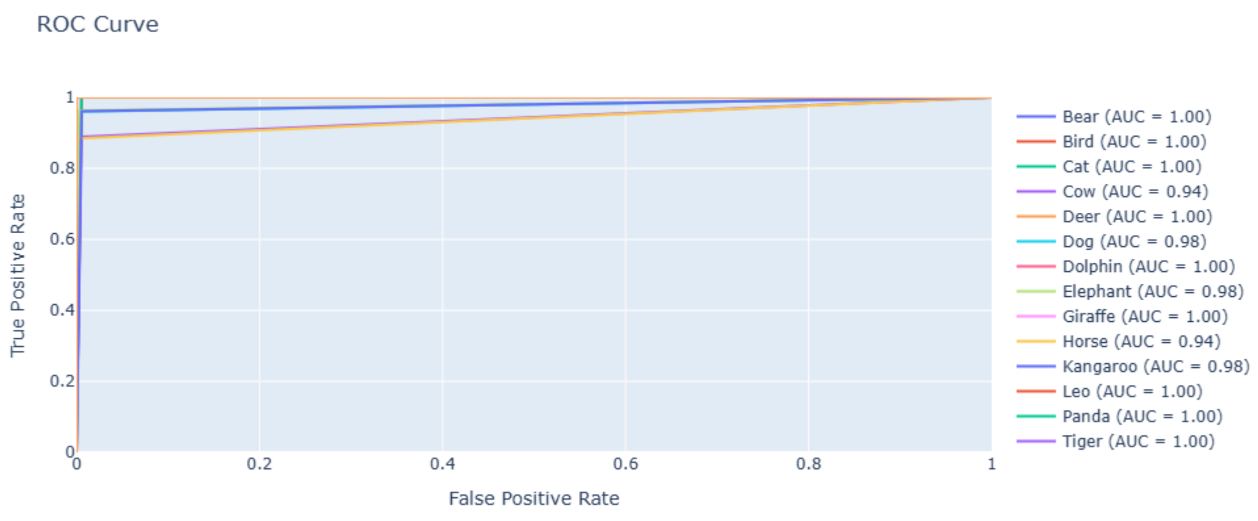
Validation Confusion Matrix

- **Precision Recall Curve:** A precision-recall curve is plotted for every class, this allows to further examine the classification results. This plot visually justifies the high precision and recall values obtained in the classification report.



Validation Precision Recall Curve

- **ROC Curve:** ROC curves are also plotted to better evaluate the models ability to identify and separate the categories on the train and validation data.



Validation ROC Curve

3.7. Model and Artifact Logging (Section 7)

- **Dagshub Integration:** Dagshub is used to track the experiment via the API.
- **MLflow Tracking:** The project uses MLflow for tracking experiment parameters, training metrics, and artifacts. MLflow is initialized to use dagshub tracking url, which can be obtained from the Dagshub interface by opening the remote tab on the left, then clicking on the experiment section, which will open up the tracking uri.
- **Experiment Logging:** After the connection to the MLflow is established, the experiments are logged using a function called `create_experiment`. This function also receives metrics, model, artifacts, run parameters, and tags as part of the log. The tracked information can then be seen in dagshub.

3.8. Identifying Animals (Section 8)

- **Prediction Function:** A function `pred_and_plot_image` is used, to showcase how to load the trained model, make a prediction and display the result with the actual image.
- **Image Sampling:** Uses `random.sample` to randomly select images from the validation set and classify the images using the loaded trained model.

4. Results and Observations

- The model achieves excellent overall performance with a validation accuracy of around 98%.
- The F1-scores are very good, hovering around 98% too, indicating a good balance of precision and recall across all classes.
- The training process converges quickly, becoming stable by epoch 4.
- The learning curves show the loss to reduce quickly and the accuracy and F1-scores to go up quickly in the initial training epochs, and all metrics to stabilize after epoch 4.
- Some animal categories, like 'Cow' and 'Horse', may have slightly lower classification metrics compared to others, suggesting a slight confusion of these categories by the model. However, this is a very small margin and can be accepted.

5. Conclusion

This project successfully developed an animal classification system using transfer learning and fine-tuning a pre-trained `EfficientNet-B0` model. The model achieved strong performance across all 15 animal classes, indicating the effectiveness of the approach. The use of distributed training accelerated the training process, and integration with Dagshub and MLflow provided detailed experiment tracking and management. In next step this trained model was deployed on streamlit so users can upload their own images to the application and identify the class of the image. Link to project's streamlit app is here