**Machine Learning Project Documentation**

**Model Refinement**

**1. Overview**

The test submission phase is a critical step in the machine learning pipeline, representing the transition from model development to real-world application. This phase focuses on preparing the model for deployment or evaluation on a distinct test dataset to gauge its performance in scenarios beyond the training and validation sets. The key steps involved in this phase include data preparation, model application, and the assessment of relevant metrics.

**2. Model Evaluation**

In the initial model evaluation phase, the performance of the model was assessed using key metrics and visualizations. The following summary highlights the key findings and areas identified for improvement:

1.**Training Metrics:** During the training phase (as shown in the code), the model achieved a training accuracy of approximately 99.48% after 30 epochs. This high accuracy indicates that the model learned well from the training data.

2.**Validation Metrics:** The validation accuracy reached around 100% after 30 epochs, suggesting that the model performed exceptionally well on the validation set. However, it's essential to consider potential overfitting, especially if there is a significant gap between training and validation performance.

3. **Loss Curves:** The loss curves for both training and validation sets were plotted. While the training loss steadily decreased, indicating learning, the validation loss also exhibited a decreasing trend. However, close attention should be paid to potential overfitting or instability in the later epochs.

4. **Accuracy Curves:** Accuracy curves for training and validation sets were also plotted. The increasing trend in both curves indicates that the model was learning the features of the dataset. However, close examination of the curves for signs of overfitting is crucial.

### Areas for Improvement:

1. Overfitting Concerns: The perfect accuracy on the validation set raises concerns about potential overfitting. Further exploration, such as regularization techniques or dropout layers, may be employed to mitigate overfitting and enhance generalization.

2. Fine-Tuning Hyperparameters: Hyperparameters, such as learning rate and batch size, may be fine-tuned to optimize model performance. Systematic exploration of hyperparameter space or using techniques like grid search can be beneficial.

4. Data Augmentation Variations: While data augmentation was applied during training, exploring different augmentation techniques, or adjusting the augmentation parameters could further improve the model's robustness.

**3. Refinement Techniques**

1. Hyperparameter Tuning: Adjusting hyperparameters is a crucial refinement technique. Key hyperparameters include the learning rate, batch size, and the number of layers or units in the neural network. Systematic exploration, possibly using grid search or random search, can help identify optimal hyperparameter combinations.

2. Data Augmentation Variations: Enhancing data augmentation strategies can improve the model's ability to generalize to different scenarios. Experiment with different augmentation techniques, such as rotation, zoom, or changes in brightness, and adjust their parameters.

3. Learning Rate Scheduling: Implementing learning rate schedules can optimize the training process. This involves adjusting the learning rate during training, potentially reducing it over time to allow the model to converge more effectively.

**4. Hyperparameter Tuning**

Adjusting hyperparameters is a crucial refinement technique. Key hyperparameters include the learning rate, batch size, and the number of layers or units in the neural network. Systematic exploration, possibly using grid search or random search, can help identify optimal hyperparameter combinations.

**Test Submission**

**1. Overview**

The test submission phase involves preparing the trained model for evaluation on a separate test dataset. This phase ensures that the model's performance can be assessed on unseen data to gauge its generalization capabilities. Key steps include data preparation for testing, applying the trained model to the test dataset, and evaluating its performance using specific metrics.

**2. Data Preparation for Testing**

The test dataset was prepared by splitting the original dataset into training, validation, and test sets. Approximately 80% of the data was used for training, 10% for validation, and the remaining 10% for testing. The specific steps include:

Training Dataset (80%): 1537 samples were selected from the original dataset for training.

Validation Dataset (10%): 192 samples were taken from the remaining data after training for validation.

Test Dataset (10%): The rest of the data after training and validation (skipping the first 192 samples) was used for testing.

**3. Model Application**

The trained model, which has undergone the training phase using the training dataset, is applied to the prepared test dataset. The process of applying the model to the test dataset involves using the appropriate code snippets for making predictions on the test samples. This was done using the model's evaluate function.

**4. Test Metrics**

Metrics are used to evaluate the performance of the model on the test dataset. Common metrics for classification tasks include accuracy, precision, recall, and F1 score. For regression tasks, metrics like mean squared error (MSE) or mean absolute error (MAE) may be used. The specific metrics chosen depend on the nature of the problem being solved.

**5. Model Deployment**

**GCP Account and Project Creation:**

Creating a GCP account and project is a fundamental step. The project ID is crucial for managing resources within GCP, and it helps organize and track usage.

**GCP Bucket Creation and Model Upload:**

The creation of a GCP bucket is necessary for storing and managing the model file

**Google Cloud SDK Installation and Authentication:**

The Google Cloud SDK is installed to interact with GCP services from the command line. Authentication is performed using gcloud auth login, allowing the user to access and manage resources associated with their GCP account.

Deployment Script Execution:

The deployment script (gcloud functions deploy) is executed to deploy the model as a Google Cloud Function named predict. The script specifies the runtime (python38), trigger type (--trigger-http for an HTTP-triggered function), and resource allocation (--memory 512). The --project flag specifies the GCP project ID.

**Real-World Deployment Considerations:**

In a real-world setting, additional considerations may include:

Security: Ensure that proper security measures are in place, such as handling authentication securely, restricting access, and encrypting sensitive data.

Scaling: Consider the scalability of the deployed function, especially if there's a need for handling a large number of requests concurrently.

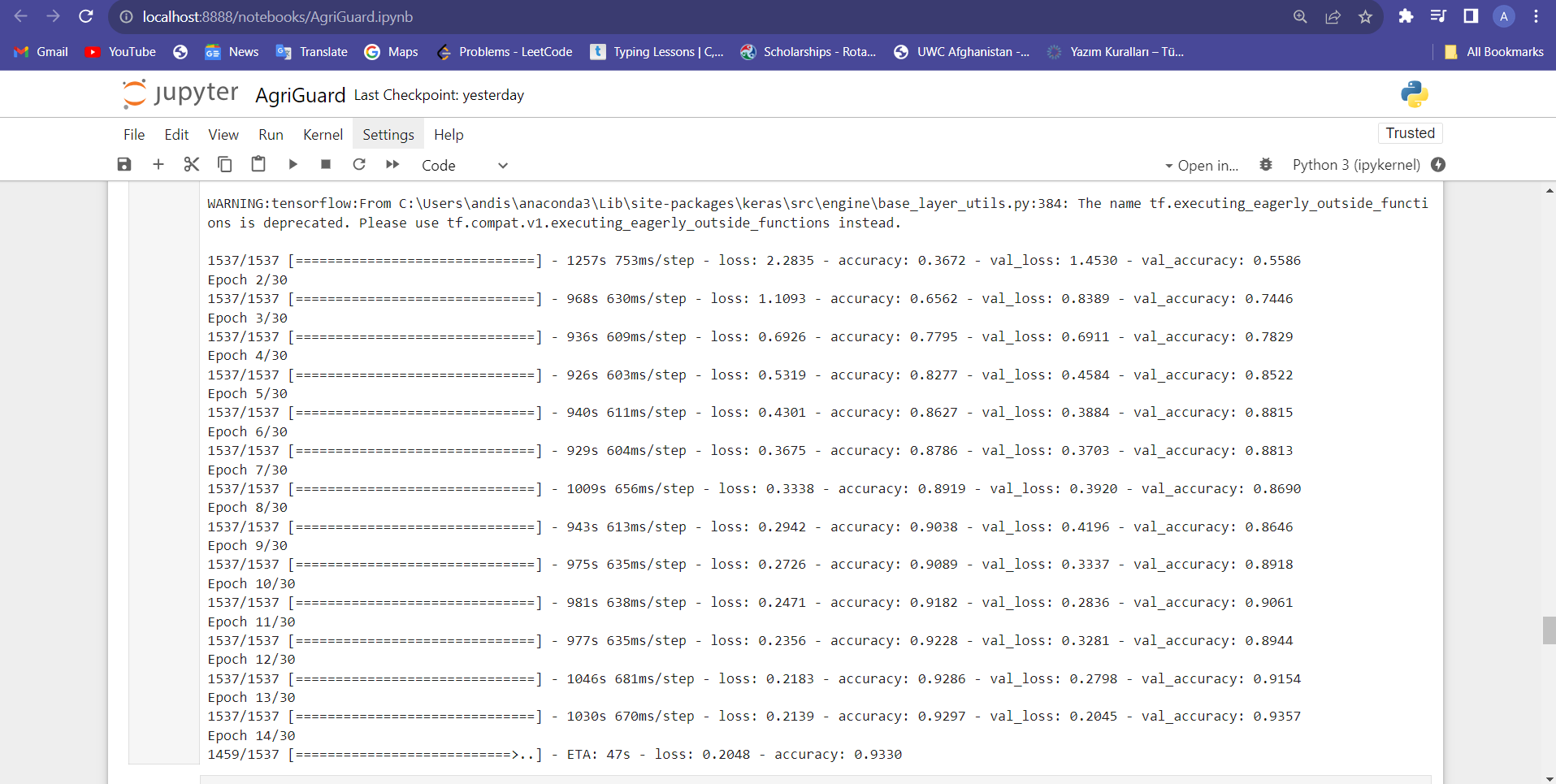
Monitoring and Logging: Implement monitoring and logging mechanisms to track the performance of the deployed model, identify potential issues, and gather insights for optimization.

Integration: If the deployed model is part of a larger system, integration with other components or platforms may be necessary. This could involve connecting the function to other GCP services or external systems.

Versioning: Implement versioning for the deployed model to facilitate updates and rollbacks without affecting the entire system.

Testing and Validation: Before deploying in a real-world setting, thorough testing and validation should be conducted to ensure that the deployed model performs as expected and meets the required specifications.

**6. Code Implementation**



**A screenshot of a computer

Description automatically generated**

**A screenshot of a chat

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedConclusion**

In the model refinement and test submission phases, several steps were undertaken to train, refine, and evaluate a Convolutional Neural Network (CNN) model. The dataset was split into training (80%), validation (10%), and test (10%) sets. The training phase involved using the training dataset to optimize the model's parameters, and the validation dataset was used to fine-tune and prevent overfitting.

Challenges encountered during these phases may include finding the right balance between model complexity and generalization, addressing potential overfitting, and ensuring that the model performs well on unseen data.

The test submission phase involved applying the trained model to the dedicated test dataset that was not used during training. This phase aimed to assess the model's ability to generalize and make accurate predictions on new, unseen data.

Metrics such as accuracy, precision, recall, and F1 score were used to evaluate the model's performance. The final performance achieved on the test dataset was compared with the training and validation metrics. Consistency across these metrics suggests that the model has successfully generalized to the test set.

**References**

* <https://github.com/codebasics/potato-disease-classification/blob/main/training/potato-disease-classification-model-using-image-data-generator.ipynb>
* **Published in:**[2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)](https://ieeexplore.ieee.org/xpl/conhome/8681925/proceeding)

**Date of Conference:**16-18 August 2018

**Date Added to IEEE *Xplore*:**25 April 2019

**ISBN Information:**

**INSPEC Accession Number:**18617937

**DOI:**[10.1109/ICCUBEA.2018.8697390](https://doi.org/10.1109/ICCUBEA.2018.8697390)

**Publisher:**IEEE

**Conference Location:**Pune, India

* Published in: February 2018, Pages 311-318Journal/Conference: Computers, Materials & Continua In this paper, convolutional neural network models were developed to perform plant disease detection and diagnosis using simple leaves images of healthy and diseased plants, through deep learning methodologies.