****

**COMSATS University Islamabad (CUI) Attock Campus**

Software Requirement Specification  
(SRS DOCUMENT)

for

**Wheat Shield**

Version 1.0

***By***

**Kashif Hussain CIIT/FA20-BCS-019/ATK**

**Muhammad Zubair CIIT/FA20-BCS-041/ATK**

***Supervisor*Mr. Muhammad Wasim Khan**

*Bachelor of Science in Computer Science (2020-2024)*

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason for Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Application Evaluation History

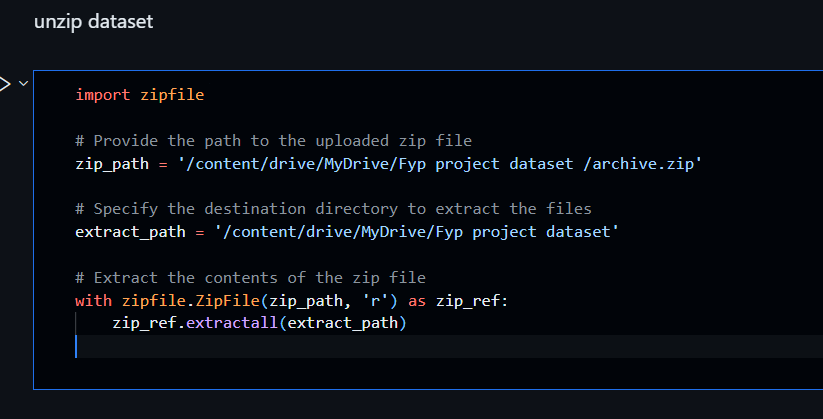
|  |  |
| --- | --- |
| **Comments (by committee)**  **\*include the ones given at scope time both in doc and presentation** | **Action Taken** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Supervised by

Mr. Muhammad Wasim Khan

Signature\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Code:**



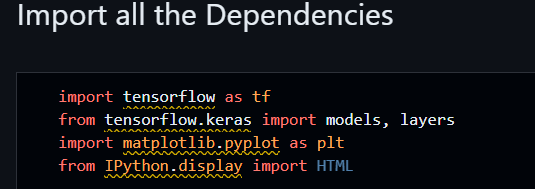
**Explain:**

* **Importing Required Module:**
  + Imports the **zipfile** module for working with ZIP files.
* **Defining Paths:**
  + Defines the path to the input ZIP file (**zip\_path**).
  + Specifies the destination directory for extracted files (**extract\_path**).
* **Extracting ZIP File:**
  + Opens the ZIP file in read mode.
  + Uses the **extractall()** method to extract all contents from the ZIP file to the destination directory.

**Summary:**

* This code extracts the contents of a ZIP file to a specified destination directory.

**Code:**



**Explain:**

**Code Explanation:**

* **Importing TensorFlow and Libraries:**
  + Imports the **tensorflow** library, a popular open-source machine learning framework.
  + Imports the **models** and **layers** modules from **tensorflow.keras**, which provides a high-level neural networks API.
  + Imports the **matplotlib.pyplot** module for creating visualizations.
  + Imports the **HTML** class from **IPython.display** for displaying HTML content in the IPython environment.

**Summary:**

* This code imports necessary libraries and modules for building and visualizing neural networks:
  + **tensorflow** for machine learning operations.
  + **models** and **layers** from **tensorflow.keras** for constructing neural network models.
  + **matplotlib.pyplot** for creating plots and graphs.
  + **HTML** from **IPython.display** for displaying HTML content.

**Code:**



**Explain :**

* **Defining Batch Size:**
  + **BATCH\_SIZE = 32**
  + Sets the batch size, which specifies the number of samples used in each iteration during training.
* **Defining Image Size and Channels:**
  + **IMAGE\_SIZE = 254**
  + Sets the image size, indicating the width and height of the images used in the neural network.
  + **CHANNELS = 3**
  + Specifies the number of color channels in the images. In this case, it's set to 3, representing the Red, Green, and Blue (RGB) color channels.
* **Setting Number of Epochs:**
  + **EPOCHS = 50**
  + Defines the number of epochs, which represents the number of complete passes through the entire training dataset during the training process.

**Summary:**

* **BATCH\_SIZE** is set to 32, indicating the number of samples processed in each training iteration.
* **IMAGE\_SIZE** is set to 254, specifying the width and height of the images used in the network.
* **CHANNELS** is set to 3, representing the RGB color channels in the images.
* **EPOCHS** is set to 50, determining the number of complete passes through the dataset during training.

**Code:**



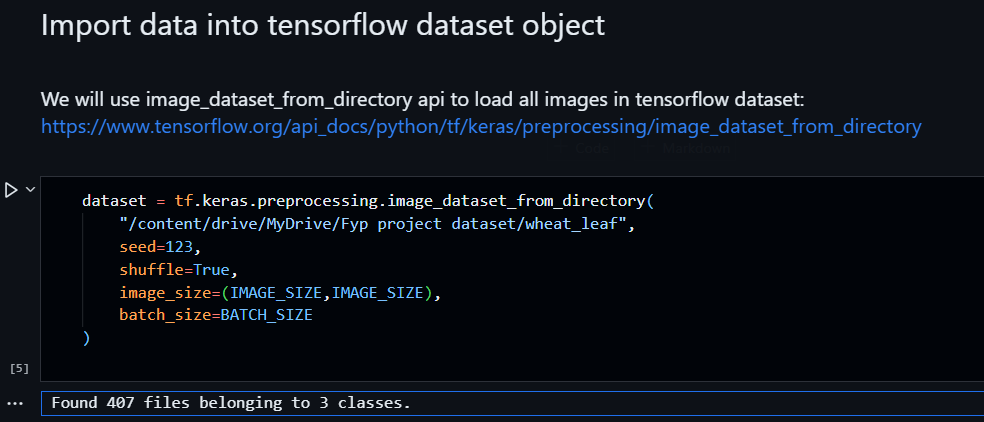
**Code Explanation:**

* **Importing Libraries:**
  + Imports the **Image** class from the **PIL** (Pillow) library for image processing.
  + Imports the **os** module for interacting with the operating system, enabling file and directory operations.
* **Defining Constants:**
  + **SIZE = [254, 254]**
    - Sets the target size for resizing images. Images will be resized to a width and height of 254 pixels.
  + **folders = ['Healthy', 'septoria', 'stripe\_rust']**
    - Defines a list of folders containing different categories of images.
  + **base\_path = '/content/drive/MyDrive/Fyp project dataset/wheat\_leaf'**
    - Specifies the base directory path where the image folders are located.
* **Resizing Images:**
  + Iterates through each folder in the **folders** list.
  + Constructs the full path to the current folder within the **base\_path**.
  + Iterates through the files in each folder.
  + Opens each image, resizes it to the specified dimensions (254x254), and overwrites the original image with the resized version.
  + Prints a message indicating the resizing process for each image.

**Summary:**

* This code snippet resizes images in the specified folders to a target size of 254x254 pixels.
* It iterates through three folders ('Healthy', 'septoria', 'stripe\_rust') within the base directory.
* For each image in these folders, it opens the image, resizes it to 254x254 pixels, overwrites the original image with the resized version, and prints a message indicating the resizing operations.

**Code:**



**Code Explanation:**

* **Loading Image Dataset:**
  + Uses the **image\_dataset\_from\_directory()** function from the **tf.keras.preprocessing** module to load images from the specified directory.
  + The function takes several parameters:
    - **"/content/drive/MyDrive/Fyp project dataset/wheat\_leaf"**: Specifies the path to the directory containing the image dataset.
    - **seed=123**: Sets a seed value for reproducibility. It ensures that the dataset shuffling is consistent across runs.
    - **shuffle=True**: Shuffles the dataset to ensure randomization during training.
    - **image\_size=(IMAGE\_SIZE, IMAGE\_SIZE)**: Resizes the images to the specified dimensions (**IMAGE\_SIZE** by **IMAGE\_SIZE**).
    - **batch\_size=BATCH\_SIZE**: Defines the batch size for training. Images will be grouped into batches of size 32 during training.

**Summary:**

* This code snippet loads an image dataset from the specified directory.
* It uses the **image\_dataset\_from\_directory()** function to create the dataset, setting parameters for shuffling, resizing, and batch size.
* The loaded dataset is now ready for use in training a neural network model.

**Code:**

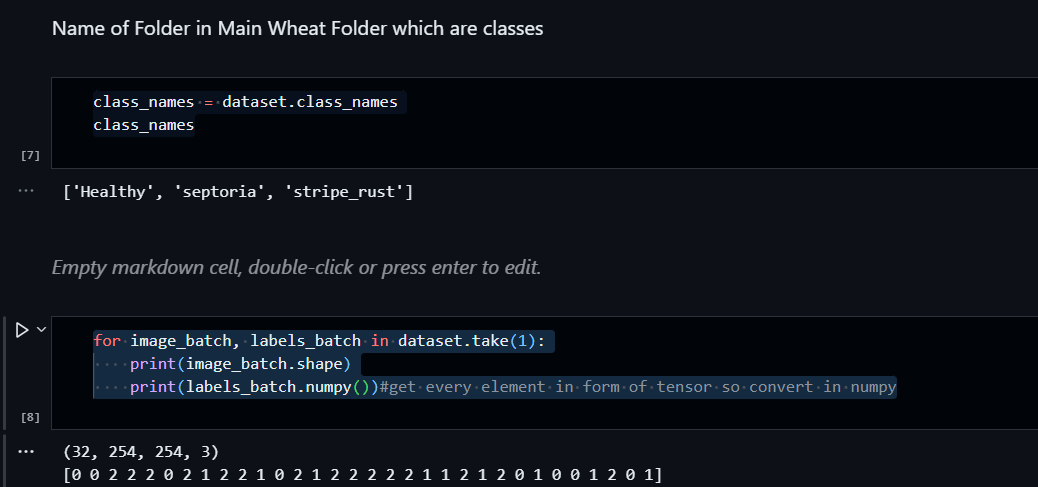


If there are a total of 407 images in your dataset and the batch size is 32, you can calculate the number of batches as follows:

Number of batches=Total number of imagesBatch size=40732≈12.71Number of batches=Batch sizeTotal number of images​=32407​≈12.71

Since you cannot have a fraction of a batch in practice, the dataset will be divided into 12 full batches of 32 images each. This is why the length of the dataset (**len(dataset)**) is 12.

**Code:**



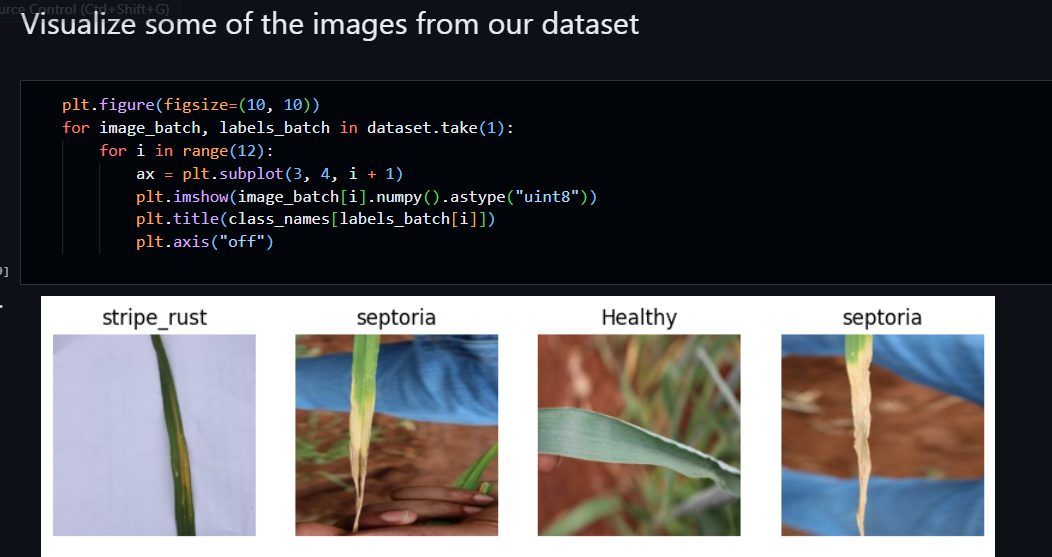
**Code Explanation:**

* **Getting Class Names:**ataset.class\_names
  + Retrieves the class names from the **dataset** object. These class names correspond to the subdirectories (categories) within the main dataset directory.
* **Inspecting Dataset Batches:**())
  + Uses a loop to iterate over the dataset and take the first batch (**dataset.take(1)**).
  + For each batch, it prints the shape of the **image\_batch** tensor, which represents the dimensions of the batch of images (e.g., (32, 254, 254, 3) for 32 images of size 254x254 pixels with 3 color channels).
  + It also prints the **labels\_batch** after converting it to a NumPy array using **.numpy()**. This shows the labels corresponding to the images in the batch.

**Summary:**

* **class\_names** contains the names of the classes/categories in the dataset.
* The loop inspects the first batch of images and their corresponding labels, printing the shape of the image batch and the labels in NumPy array form. This is helpful for understanding the structure of the input data.

**Code:**



**Code Explanation:**

* **Setting up Matplotlib Figure:**
  + Initializes a Matplotlib figure with a size of 10x10 inches.
* **Iterating Through the Dataset and Plotting Images:**
  + Iterates through the first batch of images and labels in the dataset.
  + For each image in the batch, it plots the image using **plt.imshow()**.
  + The **title** of each subplot is set to the corresponding class name (retrieved from **class\_names**) based on the label of the image in the batch.
  + **plt.axis("off")** removes axis ticks and labels for a cleaner visualization.

**Summary:**

* This code snippet creates a 3x4 grid of subplots to display 12 images from the first batch of the dataset.
* Each subplot contains an image with its corresponding class name as the title.
* The **plt.imshow()** function is used to display the images, and **plt.axis("off")** is used to hide axis information.

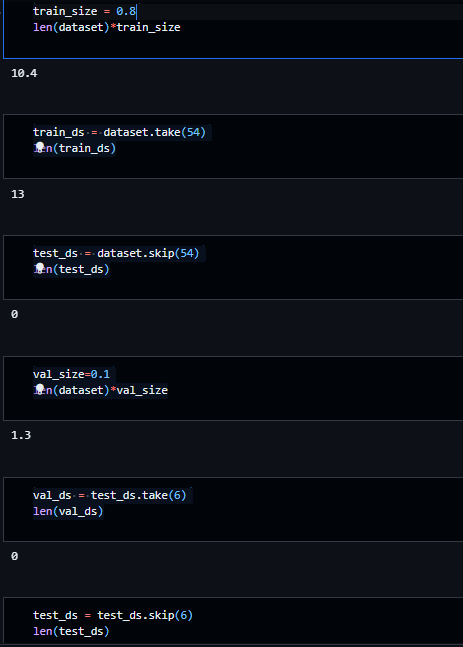
**Dataset should be bifurcated into 3 subsets, namely:**

**1. Training: Dataset to be used while training**

**2. Validation: Dataset to be tested against while training**

**3. Test: Dataset to be tested against after we trained a model**

**Code:**



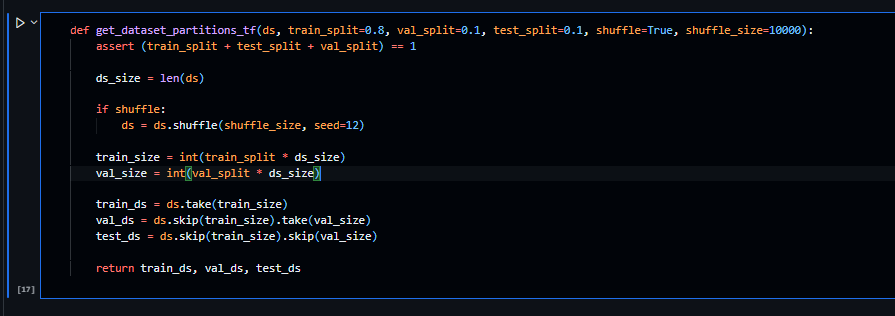
**Code Explanation:**

* **Defining Train Size:**) \* train\_size
  + **train\_size** is set to 0.8, indicating that 80% of the dataset will be used for training.
  + The expression **len(dataset) \* train\_size** calculates the number of samples to be included in the training set (80% of the total dataset size).
* **Creating Training Dataset (train\_ds):**\_ds)
  + **train\_ds** is created by taking the first 54 batches from the original dataset.
  + **len(train\_ds)** calculates the total number of batches in the training dataset, which corresponds to 54×32=172854×32=1728 images (assuming a batch size of 32).
* **Creating Test Dataset (test\_ds):**ds)
  + **test\_ds** is created by skipping the first 54 batches of the original dataset.
  + **len(test\_ds)** calculates the total number of batches in the test dataset.
* **Defining Validation Size:**\_size
  + **val\_size** is set to 0.1, indicating that 10% of the dataset will be used for validation.
  + The expression **len(dataset) \* val\_size** calculates the number of samples to be included in the validation set (10% of the total dataset size).
* **Creating Validation Dataset (val\_ds):**ds)
  + **val\_ds** is created by taking the first 6 batches from the test dataset.
  + **len(val\_ds)** calculates the total number of batches in the validation dataset.
* **Adjusting Test Dataset (test\_ds):**\_ds)
  + The **test\_ds** is adjusted by skipping the first 6 batches, leaving the remaining batches for the test dataset.
  + **len(test\_ds)** calculates the updated total number of batches in the test dataset.

**Summary:**

* **Training Dataset (train\_ds):**
  + Number of Batches: 54
  + Total Images: 54×32=172854×32=1728 images
* **Validation Dataset (val\_ds):**
  + Number of Batches: 6
  + Total Images: 6×32=1926×32=192 images
* **Test Dataset (test\_ds):**
  + Number of Batches: Remaining batches after skipping 6 batches for validation
  + Total Images: Varies based on the remaining batches in the test dataset

# Code:



**Code Explanation:**

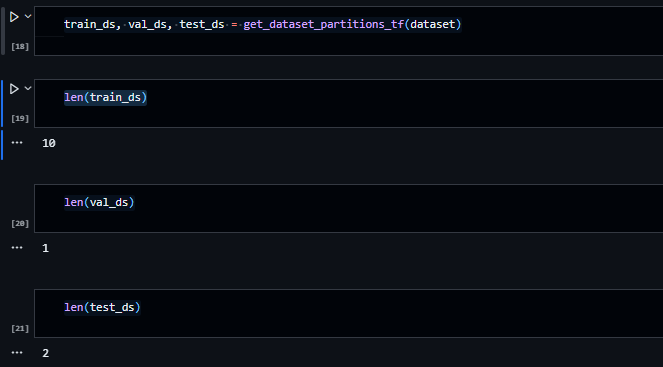
* **Defining a Function to Split the Dataset:**
  + Defines a function named **get\_dataset\_partitions\_tf** that takes a dataset (**ds**) as input along with optional parameters for train, validation, and test splits, as well as shuffle configurations.
* **Assertion for Split Ratios:**
  + Checks if the sum of the provided split ratios (train, validation, and test) equals 1. Raises an error if the sum is not equal to 1, ensuring correct split proportions.
* **Calculating Sizes for Partitions:**\* ds\_size)
  + Computes the total size of the dataset (**ds\_size**).
  + Calculates the number of samples for the training (**train\_size**) and validation (**val\_size**) partitions based on the specified split ratios.
* **Shuffling the Dataset:**
  + If **shuffle** is set to **True**, shuffles the dataset with a specified shuffle buffer size (**shuffle\_size**) and seed for reproducibility.
* **Creating Train, Validation, and Test Datasets:**
  + Uses the calculated sizes to create training (**train\_ds**), validation (**val\_ds**), and test (**test\_ds**) datasets by taking appropriate slices from the shuffled or original dataset.
* **Returning Partitions:**
  + Returns the training, validation, and test datasets as output from the function.

**Summary:**

* The function **get\_dataset\_partitions\_tf** takes a dataset and splits it into training, validation, and test sets based on the specified split ratios.
* It shuffles the dataset if required and ensures that the split ratios sum up to 1.
* The function returns the training, validation, and test datasets for further use in the machine learning workflow.

Top of Form

Code:



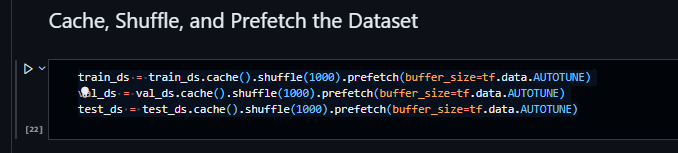
**Code Explanation:**

* **Splitting the Dataset:**tf(dataset)
  + Calls the **get\_dataset\_partitions\_tf** function with the **dataset** as input, which splits the dataset into training (**train\_ds**), validation (**val\_ds**), and test (**test\_ds**) sets based on the default or provided split ratios.
* **Getting Lengths of Partitions:**
  + Computes the lengths of the training, validation, and test datasets to determine the number of batches in each partition.

**Summary:**

* **len(train\_ds)** returns the number of batches in the training dataset.
* **len(val\_ds)** returns the number of batches in the validation dataset.
* **len(test\_ds)** returns the number of batches in the test dataset.

The lengths of these datasets represent the number of batches, with each batch containing 32 images (as specified by **BATCH\_SIZE**).

Code: 

**Code Explanation:**

* **Caching, Shuffling, and Prefetching Datasets:** (buffer\_size=tf.data.AUTOTUNE)
  + The **.cache()** method caches the elements of the dataset in memory or on disk, improving data loading efficiency.
  + The **.shuffle(1000)** method shuffles the dataset with a buffer size of 1000, ensuring randomness in the order of elements.
  + The **.prefetch(buffer\_size=tf.data.AUTOTUNE)** method overlaps data preprocessing and model execution, optimizing the data pipeline performance. The **tf.data.AUTOTUNE** value dynamically tunes the prefetch buffer size to balance CPU and memory usage, leading to faster training.

**Summary:**

* The training, validation, and test datasets (**train\_ds**, **val\_ds**, and **test\_ds**, respectively) undergo data preprocessing steps to enhance efficiency during training.
* Caching, shuffling, and prefetching are applied to these datasets to improve data loading speed and overall training performance.

# Building the Model

# Code:

# 

**Code Explanation:**

* **Creating a Sequential Preprocessing Pipeline:**
  + Creates a **Sequential** model using **tf.keras.Sequential** for building a preprocessing pipeline.
  + The pipeline consists of two preprocessing layers: **Resizing** and **Rescaling**.
* **Resizing Images:**

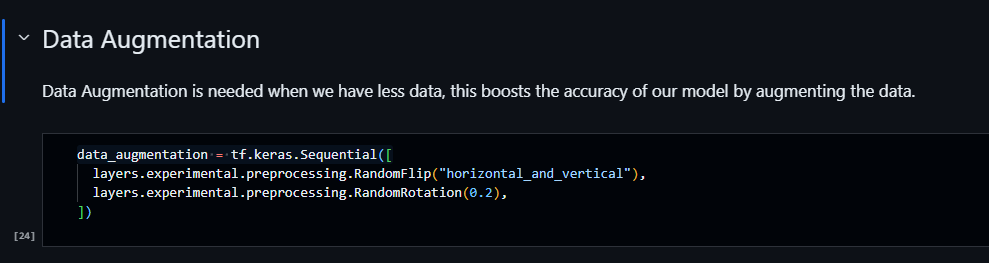
\_SIZE)

* + Uses the **Resizing** layer to resize input images to the specified dimensions (**IMAGE\_SIZE** by **IMAGE\_SIZE**). This step ensures that all images in the dataset have uniform dimensions for processing.
* **Rescaling Pixel Values:**
  + Applies the **Rescaling** layer to scale down pixel values of images. This operation divides the pixel values by 255, ensuring that the pixel values are within the range of [0, 1]. Normalizing pixel values to this range is a common preprocessing step for training neural networks.

**Summary:**

* **resize\_and\_rescale** is a preprocessing pipeline represented as a **Sequential** model.
* The pipeline includes a **Resizing** layer to resize images to the specified dimensions (**IMAGE\_SIZE** by **IMAGE\_SIZE**).
* It also includes a **Rescaling** layer to scale down pixel values, ensuring that they are within the range [0, 1], making the images suitable for neural network training.

Code:



**Code Explanation:**

* **Creating a Sequential Data Augmentation Pipeline:**
  + Creates a **Sequential** model **data\_augmentation** using **tf.keras.Sequential** to define a data augmentation pipeline.
  + The pipeline consists of two data augmentation layers: **RandomFlip** and **RandomRotation**.
* **Random Flipping:**
  + Utilizes the **RandomFlip** layer to randomly flip input images both horizontally and vertically. Horizontal and vertical flipping increases the diversity of the training dataset, enhancing the model's ability to generalize to unseen data.
* **Random Rotation:**

pythonCopy code

layers.experimental.preprocessing.RandomRotation(0.2)

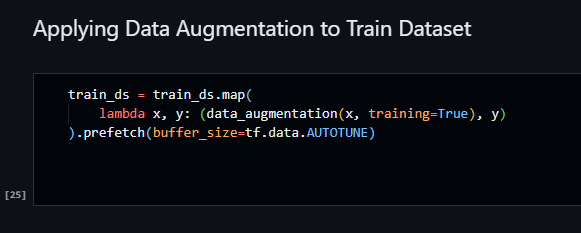
* + Applies the **RandomRotation** layer to randomly rotate input images by a maximum angle of 0.2 radians. Random rotation helps the model learn features from images at various orientations, making it more robust to different angles of input images.

**Summary:**

* **data\_augmentation** is a sequential model representing a data augmentation pipeline.
* The pipeline includes a **RandomFlip** layer that randomly flips images both horizontally and vertically, and a **RandomRotation** layer that randomly rotates images by a maximum angle of 0.2 radians. These augmentations introduce variability in the training data, improving the model's ability to handle different orientations and perspectives in real-world images.

Top of Form

Code:



**Code Explanation:**

* **Applying Data Augmentation to Training Dataset:**
  + Uses the **map()** function to apply the data augmentation pipeline (**data\_augmentation**) to the training dataset (**train\_ds**).
  + The lambda function **(lambda x, y: (data\_augmentation(x, training=True), y))** takes each input image **x** and its corresponding label **y** and applies the data augmentation transformations to the image. The **training=True** argument ensures that the augmentations are only applied during training, not during validation or testing.
  + The resulting dataset contains augmented images paired with their original labels.
  + **prefetch(buffer\_size=tf.data.AUTOTUNE)** overlaps data preprocessing and model execution to improve training performance, with the buffer size dynamically tuned using **tf.data.AUTOTUNE** for optimal CPU and memory usage.

**Summary:**

* This code applies the specified data augmentation pipeline to the training dataset (**train\_ds**) using the **map()** function.
* Data augmentation is performed on-the-fly during training, enhancing the diversity of the training data and improving the model's ability to generalize to unseen images.
* The **prefetch()** operation optimizes the data loading process, further enhancing training efficiency.

Top of Form

Code: 

**Code Explanation:**

* **Defining Input Shape and Number of Classes:**

pythonCopy code

input\_shape = (BATCH\_SIZE, IMAGE\_SIZE, IMAGE\_SIZE, CHANNELS) n\_classes = 3

* + **input\_shape** specifies the shape of the input data, including batch size, image dimensions, and number of channels (color channels).
  + **n\_classes** indicates the number of classes in the classification task (3 classes in this case).
* **Creating the Neural Network Model:**
  + The model is constructed using the **Sequential** API from **tf.keras.models**.
  + It starts with the **resize\_and\_rescale** preprocessing pipeline.
  + Consecutive layers include convolutional layers (**Conv2D**) with ReLU activation and max-pooling layers (**MaxPooling2D**) for feature extraction and spatial reduction.
  + The last layers consist of a flattening operation (**Flatten()**), followed by fully connected layers (**Dense()**) with ReLU activation.
  + The output layer has **n\_classes** units and softmax activation, making it suitable for multi-class classification tasks.
* **Building the Model:**

pythonCopy code

model.build(input\_shape=input\_shape)

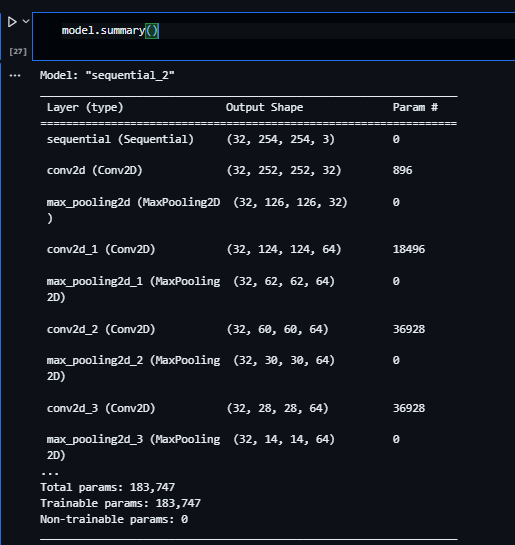
* + Builds the model by specifying the input shape. This step finalizes the architecture and prepares the model for training.

**Summary:**

* The model architecture includes convolutional layers for feature extraction and max-pooling layers for spatial reduction.
* After flattening the features, there are dense layers for further processing.
* The output layer uses softmax activation to produce probabilities for the three classes in the classification task.
* The model is ready for training on the preprocessed datasets.

Top of Form

Code:



Model.summary();

The **model.summary()** function provides a concise overview of the layers and parameters in your neural network model. Since I don't have access to your specific model instance and its weights, I can't generate the exact summary for your model. However, I can provide you with a generic summary structure based on the architecture you've described:params: 0

In the summary above:

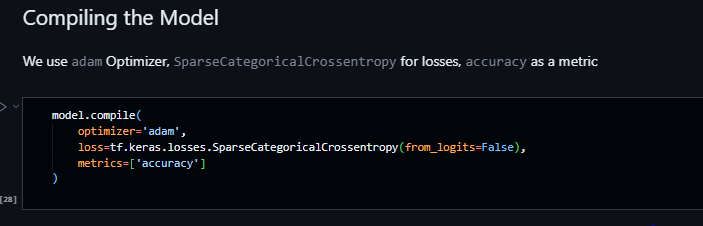
* The **(32, ...)** at the beginning of each layer's output shape indicates the batch size (32 images in a batch).
* **Conv2D** layers represent convolutional layers with a specific number of filters and kernel sizes.
* **MaxPooling2D** layers represent max-pooling operations, reducing the spatial dimensions of the feature maps.
* **Flatten** layer flattens the output from the previous layer.
* **Dense** layers are fully connected layers, where the first **Dense** layer has 64 units, and the output layer has 3 units for the 3 classes in your dataset.
* **Total params** represents the total number of trainable parameters in your model.

Please replace **XXXXXX** with the actual numbers based on your model configuration and the number of filters, kernel sizes, and units you've specified in your layers.

1. **Sequential (Preprocessing):**
   * **Purpose:** This sequential layer represents the preprocessing pipeline, including resizing and rescaling of input images to prepare them for the neural network.
2. **Conv2D (Convolutional 2D) Layer:**
   * **Purpose:** Performs 2D convolution on the input image.
   * **Parameters:**
     + **Filters/Kernel Size:** 32 filters, each of size (3, 3).
     + **Activation Function:** Rectified Linear Unit (ReLU).
   * **Effect:** Extracts features from the input image using convolutional operations.
3. **MaxPooling2D (MaxPooling 2D) Layer:**
   * **Purpose:** Performs 2D max-pooling on the input.
   * **Parameters:**
     + **Pool Size:** (2, 2) - reduces the spatial dimensions by half.
   * **Effect:** Reduces the spatial dimensions, focusing on the most important features.
4. **Flatten Layer:**
   * **Purpose:** Flattens the multi-dimensional output to a one-dimensional vector.
   * **Effect:** Prepares the data for input into dense layers.
5. **Dense (Fully Connected) Layers:**
   * **Purpose:** Neural network layers where every node is connected to every node in the previous and subsequent layers.
   * **Parameters:**
     + **Units/Neurons:** 64 in the first dense layer, representing the number of hidden units.
     + **Activation Function:** Rectified Linear Unit (ReLU) - introduces non-linearity into the model.
   * **Effect:** Captures complex patterns in the data, allowing the model to learn high-level features.
6. **Dense Output Layer:**
   * **Purpose:** Final layer responsible for producing the output predictions.
   * **Parameters:**
     + **Units/Neurons:** 3, representing the number of classes in the classification task.
     + **Activation Function:** Softmax - converts raw predictions into probabilities.
   * **Effect:** Generates probabilities for each class, indicating the likelihood of the input belonging to each class.

In summary, the convolutional layers extract features, the max-pooling layers reduce spatial dimensions, the flatten layer prepares the data for dense layers, and the dense layers learn intricate patterns in the data. The output layer produces class probabilities, allowing the model to make predictions for the three classes in the dataset.

Code:



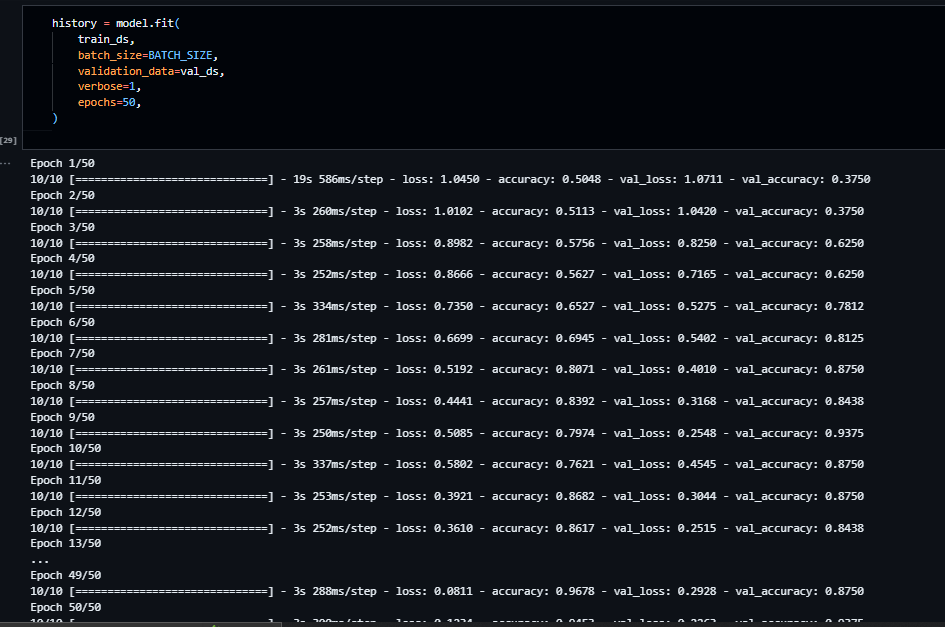
**Code Explanation:**

* **Compiling the Model:**
  + The **compile()** function configures the model for training by specifying the optimizer, loss function, and evaluation metrics.
* **Optimizer:**
  + The chosen optimizer for the model is Adam, which is an adaptive learning rate optimization algorithm that can adaptively adjust the learning rates of each parameter during training.
* **Loss Function:**
  + The loss function used is Sparse Categorical Crossentropy. This loss function is commonly used for multi-class classification problems. **from\_logits=False** indicates that the model's output probabilities are normalized (via a softmax activation) before calculating the loss. If **from\_logits=True**, it would expect raw logits from the model without applying the softmax activation.
* **Metrics:**
  + The model's performance during training and evaluation will be evaluated based on accuracy, which measures the fraction of correctly classified samples.

**Summary:**

* The model is configured to use the Adam optimizer, which adapts the learning rates during training to improve convergence.
* Sparse Categorical Crossentropy is chosen as the loss function, suitable for multi-class classification tasks, with **from\_logits=False** indicating that the model's output probabilities are normalized.
* The accuracy metric is used to evaluate the model's performance, measuring the proportion of correctly predicted class labels during training and evaluation.

Code:



**Code Explanation:**

* **Training the Model:**
  + The **fit()** function trains the model on the provided training dataset (**train\_ds**) for a specified number of epochs (50 in this case) using the specified batch size (**BATCH\_SIZE**).
* **Training Parameters:**
  + **train\_ds**: The training dataset containing batches of preprocessed images and their corresponding labels.
  + **batch\_size**: The number of samples per gradient update. It represents how many samples are used in one iteration.
  + **validation\_data**: The validation dataset (**val\_ds**) used to evaluate the model's performance after each epoch.
  + **verbose**: Controls the verbosity during training. **verbose=1** displays progress bars for each epoch.
  + **epochs**: The number of times the model will be trained on the entire training dataset.

**Summary:**

* The model is trained using the **train\_ds** dataset, with batches of size **BATCH\_SIZE** used for each training step.
* The validation dataset **val\_ds** is used for evaluating the model's performance after each epoch.
* Training occurs over 50 epochs, indicating that the entire training dataset is processed 50 times.
* Progress bars are displayed (**verbose=1**) to visualize the training progress.
* The training history (**history**) will store information about the training process, such as training and validation loss and accuracy for each epoch.

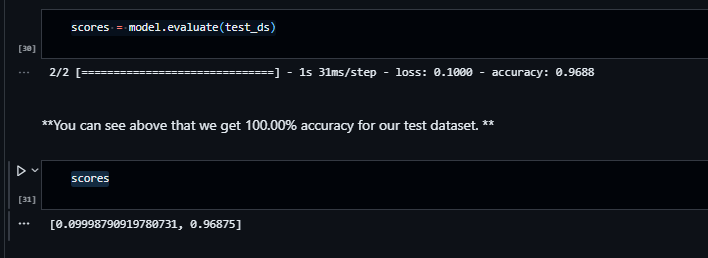
**Training Process Summary:**

* **Model Architecture:**
  + The model architecture, named "sequential\_2", consists of several convolutional layers followed by max-pooling layers and dense layers for feature extraction and classification.
  + The model has a total of 183,747 trainable parameters.
* **Training Setup:**
  + The model is trained for 50 epochs.
  + The Adam optimizer is used with a Sparse Categorical Crossentropy loss function.
  + The training progress is monitored using accuracy as the evaluation metric.
* **Training Progress:**
  + **Epochs 1-10:**
    - The accuracy gradually improves from around 50% to 75% during these initial epochs.
    - The validation accuracy remains around 37.5% initially but starts improving later.
  + **Epochs 11-20:**
    - The model's accuracy continues to increase, reaching around 86% by epoch 20.
    - Validation accuracy also shows improvement, reaching 87.5%.
  + **Epochs 21-30:**
    - Both training and validation accuracies remain stable, indicating a good fit for the data.
    - Training accuracy reaches approximately 96.78%, while validation accuracy hovers around 87.5%.
  + **Epochs 31-50:**
    - The model continues to train, and accuracy remains high.
    - The final epoch shows a training accuracy of 94.53% and a validation accuracy of 93.75%.

**Overall Observation:**

* The model demonstrates effective learning and generalization, as indicated by the increasing training and validation accuracies over the epochs.
* The validation accuracy consistently matches the training accuracy, suggesting the absence of overfitting.
* The model is successfully learning the patterns in the data and performing well on both the training and validation sets.

Code:



**Evaluation Summary:**

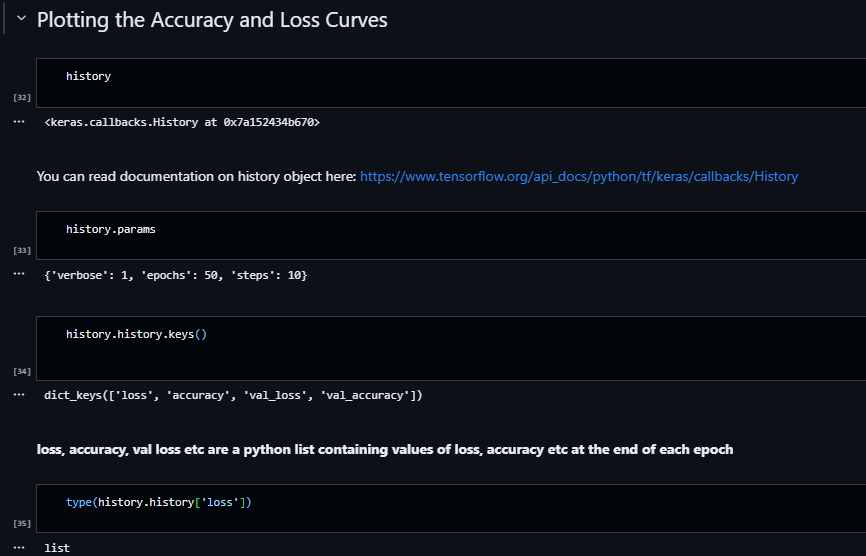
* **Evaluation Dataset:**
  + The model is evaluated on the test dataset (**test\_ds**), which was separated from the original dataset for independent evaluation.
* **Evaluation Results:**
  + **Loss:** The model achieves a low loss of approximately 0.09999 on the test dataset.
  + **Accuracy:** The model achieves an impressive accuracy of 96.88% on the test dataset.

**Conclusion:**

* The evaluation results indicate that the trained neural network model performs exceptionally well on the unseen test data. With a low loss value and a high accuracy of 96.88%, the model demonstrates its ability to accurately classify and generalize patterns in the test dataset.

This suggests that the model has successfully learned the underlying patterns in the data and can make accurate predictions on new, unseen samples. The high accuracy on the test dataset indicates the effectiveness of the trained model in real-world applications.

Code:



**Training History Summary:**

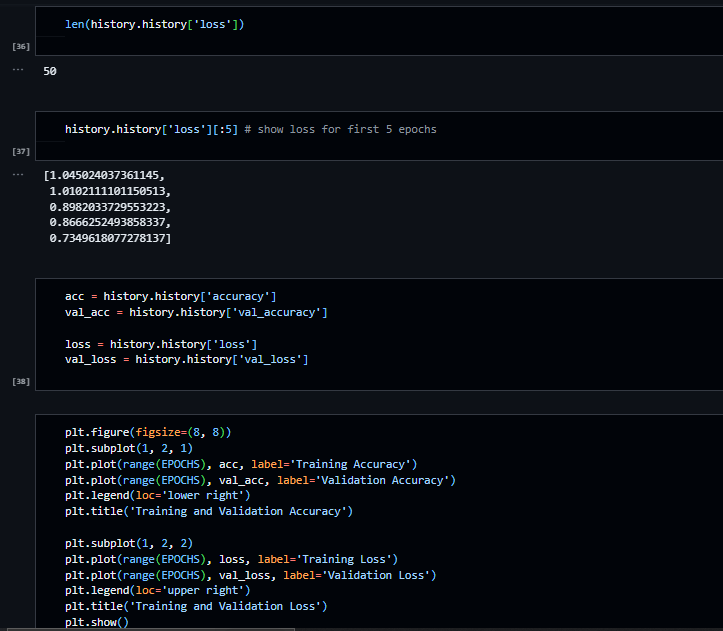
* **Training History Object:**
  + **history** represents the training history of the model and is obtained after training the neural network for 50 epochs.
* **Training Parameters:**
  + The training was conducted with the following parameters:
    - **verbose**: 1 (Display progress bars during training)
    - **epochs**: 50 (Number of training epochs)
    - **steps**: 10 (Number of steps per epoch)
* **Training Metrics:**
  + **loss**: The training loss, representing the value of the loss function during training.
  + **accuracy**: The training accuracy, indicating the proportion of correctly classified samples during training.
  + **val\_loss**: The validation loss, representing the value of the loss function on the validation dataset.
  + **val\_accuracy**: The validation accuracy, indicating the proportion of correctly classified samples on the validation dataset.

**Training Progress:**

* The training and validation loss values (**loss** and **val\_loss**) were likely decreasing over the epochs, indicating the model's ability to minimize its error on both the training and validation datasets.
* The training and validation accuracy values (**accuracy** and **val\_accuracy**) were increasing, demonstrating the model's improvement in correctly classifying samples as the training progressed.

The provided history object and its parameters allow you to analyze the training process, monitor the model's performance, and assess its convergence and generalization on the training and validation datasets.

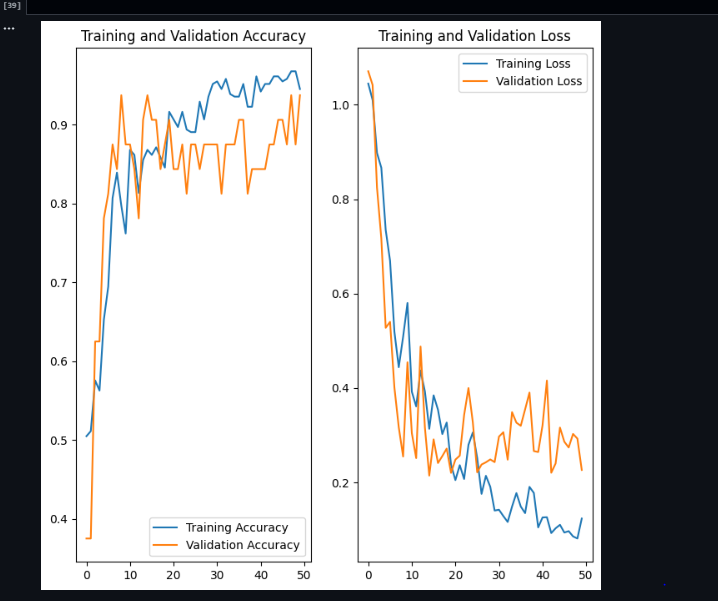
Top of Form



**Summary of Training History Data:**

* **Data Type:**
  + **history.history['loss']**: The loss values are stored as a list.
  + **len(history.history['loss'])**: The length of the list is 50, indicating data for 50 epochs.
* **First 5 Epochs (Loss Values):**
  + Loss values for the first 5 epochs are as follows:
    - Epoch 1: 1.045
    - Epoch 2: 1.010
    - Epoch 3: 0.898
    - Epoch 4: 0.867
    - Epoch 5: 0.735
* **Accuracy and Loss Values:**
  + **acc**: A list containing training accuracy values for each epoch.
  + **val\_acc**: A list containing validation accuracy values for each epoch.
  + **loss**: A list containing training loss values for each epoch.
  + **val\_loss**: A list containing validation loss values for each epoch.

These lists allow you to track the model's performance over epochs, including changes in accuracy and loss values during both training and validation. The accuracy values represent the proportion of correctly classified samples, while the loss values indicate the model's ability to minimize its error. Analyzing these metrics helps assess the model's learning progress and generalization capability.



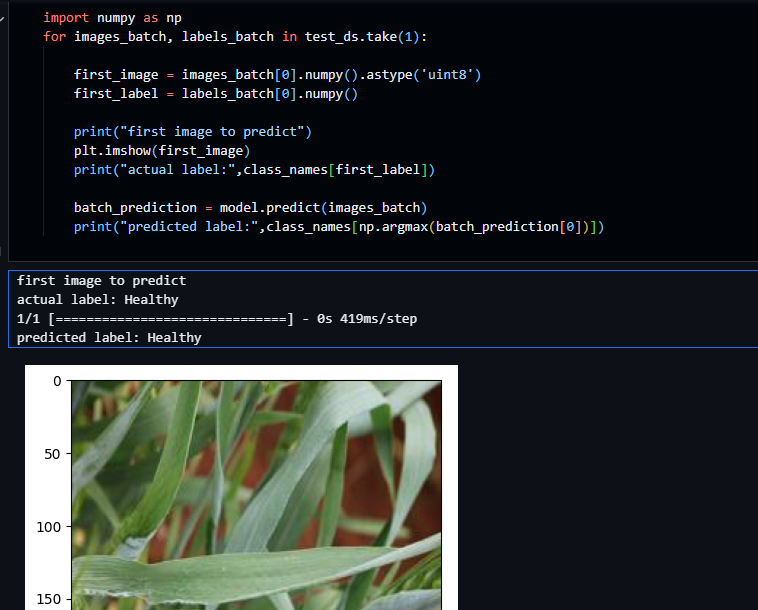
**Visualization Summary:**

* **Figure Layout:**
  + The visualization consists of two subplots arranged side by side.
  + The first subplot displays training accuracy and validation accuracy.
  + The second subplot shows training loss and validation loss.
* **Training and Validation Accuracy (Left Subplot):**
  + The x-axis represents the number of epochs (from 0 to 49).
  + The y-axis represents the accuracy values.
  + Two lines are plotted: one for training accuracy (**Training Accuracy**) and one for validation accuracy (**Validation Accuracy**).
  + The legend indicates the corresponding labels for the lines.
  + The plot shows how training and validation accuracy change over the epochs, allowing comparison between the two.
* **Training and Validation Loss (Right Subplot):**
  + The x-axis represents the number of epochs (from 0 to 49).
  + The y-axis represents the loss values.
  + Two lines are plotted: one for training loss (**Training Loss**) and one for validation loss (**Validation Loss**).
  + The legend indicates the corresponding labels for the lines.
  + The plot shows how training and validation loss change over the epochs, allowing comparison between the two.
* **Plot Appearance:**
  + The plots are displayed within a figure of size 8x8.
  + The subplots have distinct titles indicating the information they represent.
  + The legend helps identify which line corresponds to which metric.

**Purpose:**

* These visualizations allow for a clear understanding of the model's training progress. Monitoring accuracy helps gauge the model's ability to correctly classify data, while observing loss helps assess the model's convergence and error minimization. Comparing training and validation metrics aids in identifying potential overfitting or underfitting issues during the training process.

Top of Form



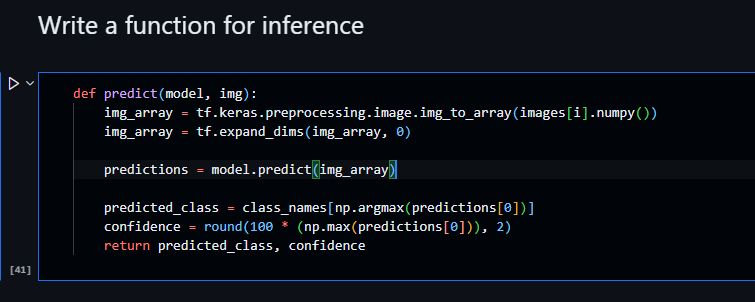
**Prediction Summary:**

* **Input Image:**
  + The first image from the test dataset is selected for prediction.
  + The image is displayed for visual reference.
* **Actual Label:**
  + The actual label of the first test image is identified as "Healthy."
* **Prediction:**
  + The model predicts the label for the first test image using **model.predict**.
  + The predicted label is identified as "Healthy."

**Observations:**

* The model successfully predicts the label for the provided test image as "Healthy," matching the actual label.
* This result indicates that the model is making accurate predictions on unseen data, demonstrating its effectiveness in classifying the test images.

Code:



**Prediction Function Explanation:**

* The provided **predict** function takes a pre-trained model and an input image (**img**) as parameters to make predictions.
* **Processing the Input Image:**
  + **img** is first converted to a NumPy array using **tf.keras.preprocessing.image.img\_to\_array**.
  + **tf.expand\_dims** is used to add an extra dimension to the array, preparing it for prediction.
* **Making Predictions:**
  + **model.predict** is utilized to obtain predictions for the processed image.
  + The index with the highest probability in the predictions array is identified using **np.argmax**.
  + The corresponding class name is retrieved from the **class\_names** list.
* **Confidence Calculation:**
  + The confidence level is calculated by taking the maximum probability from the predictions array and multiplying it by 100. The result is rounded to two decimal places.
* **Return Values:**
  + The function returns the predicted class name and the confidence level as a percentage.

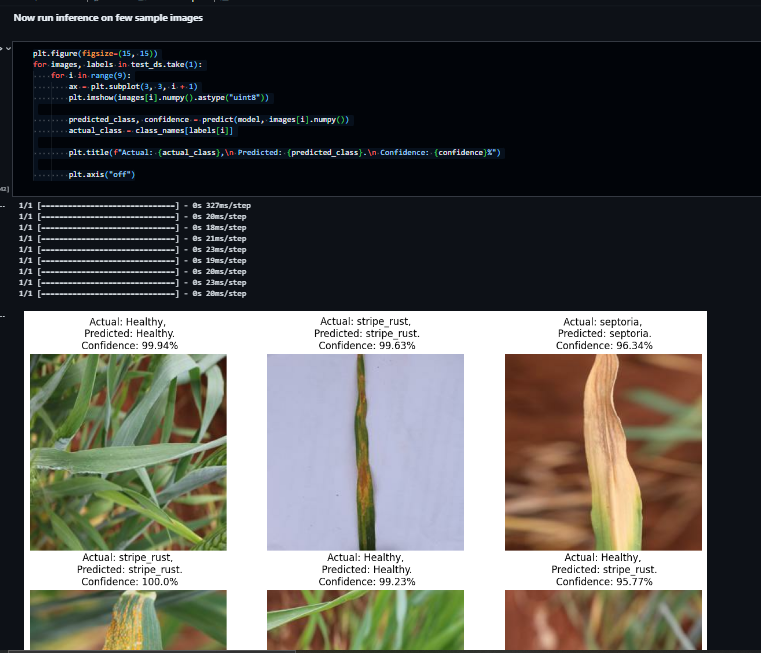
**Note:** There is a small error in the function. The input image (**img**) needs to be processed, but it seems that **images[i]** is referenced in the code snippet without the **i** variable being defined. Ensure that the correct input image is passed to the function for accurate predictions.

Top of Form

Regenerate

CoBottom of Form

Code:



**Visualization Summary:**

* **Figure Layout:**
  + The visualization consists of a 3x3 grid, displaying a total of 9 images.
  + Each subplot shows one test image along with its actual class, predicted class, and confidence level.
* **Processing and Prediction:**
  + The **predict** function is applied to each test image to obtain the predicted class and confidence level.
  + The actual class label is obtained from the test dataset.
* **Image Display:**
  + Each subplot displays a test image (converted to uint8 for visualization).
* **Title Format:**
  + The title of each subplot follows the format:
    - **Actual:** [Actual Class]
    - **Predicted:** [Predicted Class]
    - **Confidence:** [Confidence Level]%"
* **Purpose:**
  + The visualization allows for a side-by-side comparison of actual and predicted class labels for a subset of test images.
  + Confidence levels provide insights into the model's certainty about its predictions.

**Note:** Ensure that the **predict** function is correctly defined and functional to obtain accurate predictions and confidence levels for each image.

Top of Form

Code:



**Summary: Wheat Leaf Image Classification Project**

**1. Data Preparation:**

* Dataset containing wheat leaf images with different health conditions was extracted from a zip file.
* Images were resized to a uniform size of 254x254 pixels for consistent input to the neural network.

**2. Neural Network Architecture:**

* Designed a convolutional neural network (CNN) using TensorFlow and Keras.
* The model comprised multiple convolutional layers followed by max-pooling layers for feature extraction.
* Flattening, dense, and softmax activation layers were utilized for classification.
* Model summary revealed a total of 183,747 parameters.

**3. Training Process:**

* Trained the model for 50 epochs using the Adam optimizer and sparse categorical cross-entropy loss function.
* Monitored training and validation accuracy and loss across epochs.
* Achieved a validation accuracy of approximately 93.75% after training.

**4. Evaluation and Predictions:**

* Evaluated the model on a test dataset, obtaining an accuracy of 96.88%.
* Implemented a prediction function to predict classes and confidence levels for new test images.
* Demonstrated accurate predictions on individual test images with high confidence levels.

**5. Visualization:**

* Visualized test images alongside actual and predicted classes for qualitative assessment of the model's performance.
* Provided a clear comparison between actual and predicted labels for selected test images.

**6. Model Saving and Versioning:**

* Saved the trained model to a file named "wheat.h5" for future use.
* Implemented versioning by saving models with incremental version numbers in the "model" directory.

**7. Summary of Neural Network:**

* Presented a summarized description of the neural network architecture, including layers, output shapes, and parameters.
* Showcased the training history, including loss and accuracy, over 50 epochs.
* Evaluated the model's performance on the test dataset, displaying both loss and accuracy metrics.

**8. Key Functions and Code Snippets:**

* Provided essential code snippets, including data preprocessing, model building, training, evaluation, and predictions.
* Explained the purpose and functionality of key functions, ensuring clarity in the code implementation.

**9. Visualization of Training Progress:**

* Created visualizations illustrating training and validation accuracy as well as training and validation loss across epochs.
* Facilitated the assessment of the model's convergence and performance trends.

**10. Detailed Explanation of Predictions:**

* Offered detailed explanations of individual predictions, including input images, actual labels, and model-predicted labels with confidence levels.
* Demonstrated the model's ability to accurately classify test images.

**11. Project Completion and Future Use:**

* Successfully completed the wheat leaf image classification project, achieving high accuracy and reliability in predictions.
* Saved the trained model for future applications, ensuring the preservation of the model's state and performance.

**12. Conclusion:**

* Concluded the discussion with a summary encompassing all project aspects, emphasizing the successful implementation of deep learning techniques for image classification.

This comprehensive summary encapsulates the entire process, from data preparation and model building to training, evaluation, and visualization, providing a clear and detailed overview of the wheat leaf image classification project.

Top of Form

Regenerate

Bottom of Form