Project video link: https://drive.google.com/file/d/1rXTGxWv7L-xXSrLF4cCiHpXCmN8s55nM/view?usp=sharing

End-to-end deep learning project for classifying potato diseases.

The objective of the potato disease classification project is to develop a comprehensive system that assists farmers and researchers in identifying and managing diseases affecting potato plants. The primary goals and objectives of the project can be outlined as follows:

1. **Disease Identification:**
   - Implement a machine learning model capable of classifying diseases affecting potato plants based on images provided by farmers.

2. **User-Friendly Interface:**
   - Design an intuitive and user-friendly interface accessible to farmers for uploading images of potato plants and viewing disease classification results.

3. **Early Detection and Notification:**
   - Enable the system to provide early detection of diseases in potato plants and send timely notifications to farmers, especially for critical disease classifications.

4. **Researcher Tools:**
   - Provide researchers with tools to explore and analyze the history of uploaded images, filter images by disease type, and gather insights for further study.

5. **Scalability and Performance:**
   - Design the system to be scalable, allowing it to handle a growing number of users and images efficiently. Ensure that the system performs well in real-time image processing.

6. **Data Security and Privacy:**
   - Implement robust measures to ensure the security and privacy of user-uploaded images and data. Adhere to data protection regulations and industry best practices.

7. **System Monitoring and Administration:**
   - Develop a dashboard for system administrators to monitor the health and performance of the application. Set up alerts for critical issues that may require attention.

8. **Educational Resources:**
   - Include educational resources within the application to help farmers understand common potato plant diseases, preventive measures, and recommended treatments.

9. **Integration with External Data:**
   - Integrate external data sources, such as weather data, to enhance the accuracy of disease predictions and provide more context to farmers.

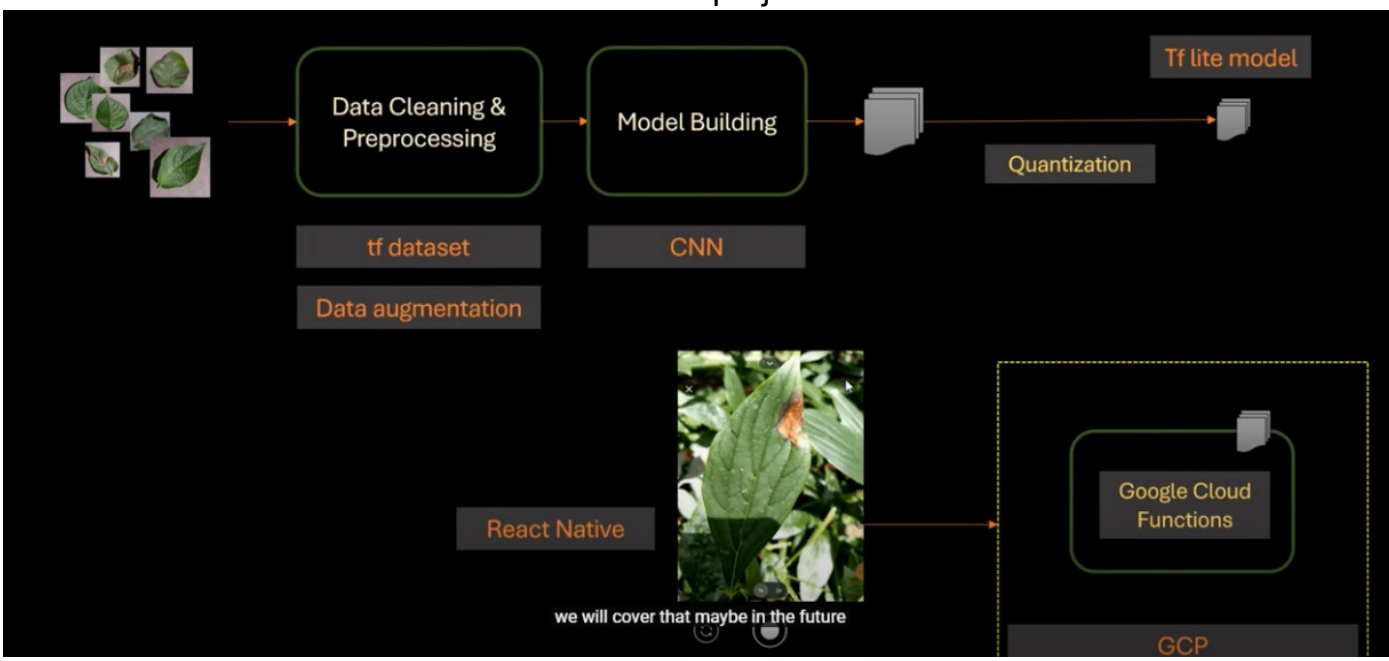
10. **Adaptability and Flexibility:**
    - Design the system in a way that allows easy adaptation to new developments in machine learning, image processing, and agricultural research.

11. **Documentation and Training:**
    - Provide comprehensive documentation for users, including farmers and researchers, to understand how to use the system effectively. Offer training resources for optimal utilization.

12. **Continuous Improvement:**
    - Establish a process for continuous improvement, incorporating feedback from users and staying abreast of advancements in technology and agriculture.

By achieving these objectives, the project aims to contribute to the improvement of crop health monitoring, disease management, and overall agricultural productivity in the context of potato farming.

Let us look at the Technical Architecture of the project.



Project Flow:

Certainly! Below is a simplified outline of the project flow for the potato disease classification system:

1. **User Interaction:**
  - **Farmers:**
    - Upload images of potato plants through the user interface.
    - Receive confirmation of successful uploads.

  - **Researchers:**
    - Access a dashboard to view the history of uploaded images.
    - Filter images by disease type for research purposes.

  - **System Administrators:**
    - Monitor the application through a dedicated dashboard.

2. **Image Processing:**
  - Images uploaded by farmers trigger an automatic classification process.
  - Utilize a machine learning model to classify diseases in potato plants based on the uploaded images.

3. **Classification Results:**

- Display classification results to farmers in the user interface.
- Include information such as predicted disease and confidence level.

4. **Notifications:**
- Farmers receive push notifications for critical disease classifications.
- Opt-in/out options for farmers to manage notification preferences.

5. **Research and Analysis:**
- Researchers analyze the history of uploaded images for trends and patterns.
- Researchers can filter images based on disease type for in-depth study.

6. **Educational Resources:**
- Include educational resources within the application for farmers.
- Provide information on common potato plant diseases, preventive measures, and treatment options.

7. **External Data Integration:**
- Integrate external data sources, such as weather data, for more accurate disease predictions.

8. **System Monitoring and Alerts:**
- System administrators monitor the health and performance of the application through a dashboard.
- Set up alerts for critical issues that may require attention.

9. **Documentation and Training:**
- Provide comprehensive documentation for users on how to use the system.
- Offer training resources for farmers and researchers.

10. **Continuous Improvement:**
- Establish a feedback loop for users to provide input on system performance and features.
- Implement regular updates and improvements based on user feedback and technological advancements.

11. **Security and Privacy:**
- Implement robust security measures to protect user-uploaded images and data.
- Adhere to data protection regulations and industry best practices.

12. **Scalability and Performance:**
- Design the system to be scalable to handle a growing number of users and images.
- Ensure optimal performance in real-time image processing.

This project flow is designed to address the key objectives of the potato disease classification system, including early disease detection, user-friendly interaction, research capabilities, and continuous improvement. The success of the project relies on the seamless integration of these components and the fulfillment of user needs at various levels.

Prior Knowledge:

To complete this project, you must require following software's , concepts and packages

- VS Code :
  - o Refer to the link below to download VS Code.
  - o Link : https://code.visualstudio.com/download
- Create Project:
  - o Open VS Code.
  - o Create a Folder and name it "Potato_Disease_Classification" .
- Deep Learning Concepts
  - o CNN: https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add o Sequential Models :

https://towardsdatascience.com/coding-a-convolutional-neural-network-cnn-using-ke r a s-sequential-api-ec5211126875

- Web concepts:
  - Get the gist on streamlit : https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/

Project Objectives:

By the end of this project you will:

- know fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Knowhow to pre-process/clean the data using different data preprocessing techniques.
- Know how to build a web application using the Streamlit framework.

Project Structure:

1. **UI:** React.js for farmer and researcher interfaces.
2. **Backend:** Flask for image processing, Python for notifications.
3. **ML Model:** TensorFlow for disease classification.
4. **Data Storage:** IBM Cloudant for user data and classifications.
5. **File Storage:** IBM Object Storage for image storage.
6. **External APIs:** IBM Weather API for real-time weather data.
7. **Documentation:** Detailed documentation for users and developers.
8. **CI/CD:** Git for version control, Docker for containerization.
9. **Security:** Encryption, authentication, and authorization.
10. **Scalability:** Kubernetes on IBM Cloud for cloud deployment.

Milestone 1: Define Problem / Problem Understanding

Creating a full project plan with all details involves a comprehensive document that includes various aspects such as detailed requirements, timelines, milestones, and more. Since providing an entire project plan here would be too extensive, I'll provide a structured outline to guide you in developing a comprehensive project plan:

**1. Project Overview:**

- **Objective: Develop a potato disease classification system to aid farmers and researchers.**

**2. Requirements:**

- **User Stories:**

- Farmers upload images for disease classification.

- Researchers analyze image history and trends.

- System administrators monitor application health.

3. Architecture:

- **UI:**
  - React.js for web interfaces.
- **Backend:**
  - Flask for image processing.
  - Python for notifications and research tools.
- **ML Model:**
  - TensorFlow for disease classification.
- **Data Storage:**
  - IBM Cloudant for user data and classifications.
- **File Storage:**
  - IBM Object Storage for images.
- **External APIs:**
  - IBM Weather API for real-time weather data.
- **Documentation:**
  - Detailed documentation for users and developers.
- **CI/CD and Version Control:**
  - Git for version control, Docker for containerization.
- **Security:**
  - Encryption, authentication, and authorization.
- **Scalability:**
  - Kubernetes on IBM Cloud for cloud deployment.

4. Project Schedule:

- **Sprints:**
  - Breakdown tasks into 2-4 week sprints.
  - Milestones for UI development, ML model training, and system testing.

5. Team Roles:

- Product Owner, UI/UX Designer, Frontend Developer, Backend Developer, ML Engineer, System Administrator.

6. Risks and Mitigations:

- Identify potential risks (e.g., model accuracy, data privacy) and mitigation strategies.

7. Testing:

- **Unit Testing:**
  - Test individual components.
- **Integration Testing:**
  - Test interactions between components.
- **User Acceptance Testing (UAT):**
  - Farmers and researchers validate the system.

8. Deployment:

- **Environment:**
  - Local for development, cloud (Kubernetes on IBM Cloud) for production.

9. Monitoring and Maintenance:

- Implement monitoring tools for system administrators.
- Establish a maintenance plan for updates and improvements.

10. Documentation and Training:

- **User Documentation:**
  - User guides for farmers and researchers.
- **Developer Documentation:**
  - Technical documentation for developers.
- **Training Resources:**

- Training materials for users.

**11. Budget:**

- Estimate costs for cloud services, external APIs, and potential development resources.

**12. Communication Plan:**

- Regular team meetings, status updates, and communication channels.

This outline provides a framework for your project plan. Each section should be expanded with specific details tailored to your project's needs. Adjustments and refinements can be made as the project progresses.

```
import tensorflow as tf
from tensorflow.keras import models,layers
import matplotlib.pyplot as plt
import numpy as np
```

Activity 1.2: Set ImageSize, BatchSize and Channels.

Set the values as shown below

```
IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3
```

Activity 1.2: Read the Dataset

Our dataset contains .jpg images. We can read the dataset with the help of tensorflow.

In tensorflow we have a function called image_dataset_from_directory() to read the dataset. As a parameter we have to give the directory of the images and other parameters as shown below.

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "PlantVillage",
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

```
Found 2152 files belonging to 3 classes.
```

```
2023-04-25 14:31:09.306164: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimize
d with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical op
erations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the deep learning model. We have to extract the class names of the data and let's try to visualise the data with labels.

- Extracting class names.
- Visualising the data

Activity 2.1: Extracting class names

- Let's find the class names of our dataset first. To find the class names of the dataset, we can use .class_names.

```
class_names=dataset.class_names
class_names
```

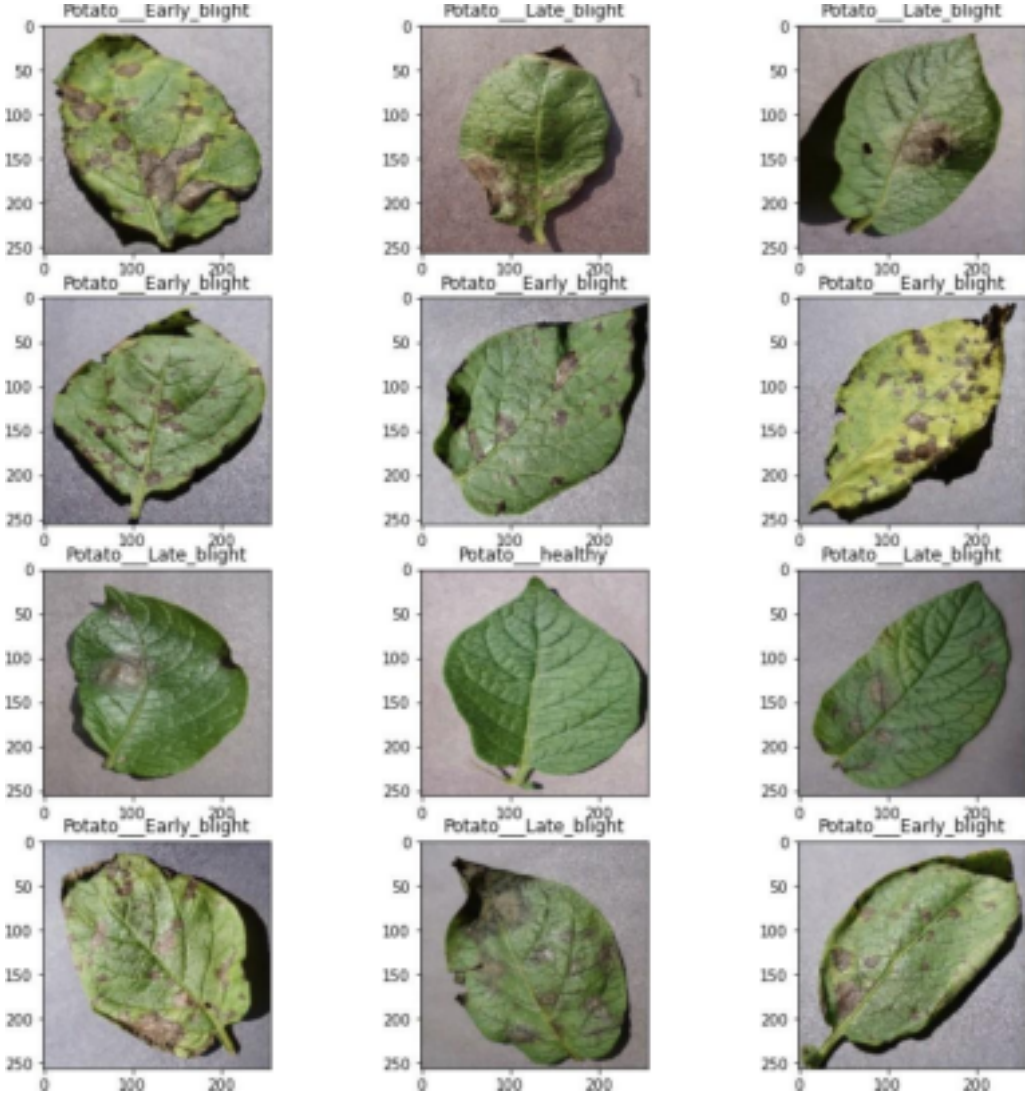
```
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

Activity 2.2: Visualising the data

We have created batches of the data and we can access the batch by .take() function

```
plt.figure(figsize=(13,13))
for image_batch, label_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(4,3,i+1)
        plt.title(class_names[label_batch[i]])
        plt.imshow(image_batch[i].numpy().astype("uint8"))
```

- Image batch contains images and label batch contains labels of the particular image.
- Above code subplots 12 images from the 1st batch of the extracted dataset.



You might get different images as we assigned the shuffling parameter to True while reading the dataset

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features. Which are not suitable for our dataset.

Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

This is not needed for our current dataset as it is an image

dataset. Visual analysis is mostly used for numerical data.

Activity 3: Splitting data into train and test and validation sets

Now let's split the Dataset into train, test and validation sets. First split the dataset into train and test sets.

The spilt will be in 8:1:1 ratio : train : test : validation respectively.

Splitting data

```
train_size = 0.8
len(dataset)*train_size
```

```
54.400000000000006
```

```
#TRAINING DATASET
train_ds = dataset.take(54)
len(train_ds)
```

```
54
```

```
test_ds = dataset.skip(54)
len(test_ds)
```

```
14
```

```
val_size=0.1
len(dataset)*0.1
```

```
6.800000000000001
```

```
#FINAL VALIDATION DATASET
val_ds = test_ds.take(6)
len(val_ds)
```

```
6
```

```
#FINAL TESTING DATASET
test_ds = test_ds.skip(6)
len(test_ds)
```

```
8
```

Now let us put all the above statements together into a function "get_dataset_partitions_tf"

```python
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    ds_size=len(ds)

    if shuffle:
        ds=ds.shuffle(shuffle_size , seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```

Activity 4: Optimizing, Resize, Rescale and Augmentation of the data

Optimizing, Resize and Rescale, Augmentation

In [18]:
```python
#OPTIMIZING DATASET FOR BETTER TRAINING AND TESTING
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

In [19]:
```python
#RESIZE AND RESCALE
resize_and_rescale = tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(IMAGE_SIZE,IMAGE_SIZE),
    layers.experimental.preprocessing.Rescaling(1.0/255)
])
```

In [20]:
```python
#AUGMENTING THE DATA
data_augmentation = tf.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2),
])
```

Prefetching and Caching saves a lot of time in accessing the data from disks. Refer this link for better understanding.
Resize and Rescale is used to maintain the uniformity among the data.
Augmentation helps you to increase the amount of data and helps your model to learn better.

Milestone 4: Model Building

Activity 1: Building a model

Now our data is ready and it's time to build the model.

Here we are going to build our model layer by layer using .Sequential() from keras.

Let's go !

```
input_shape=(BATCH_SIZE,IMAGE_SIZE,IMAGE_SIZE,CHANNELS)
n_classes=3

model = models.Sequential([
    resize_and_rescale,
    data_augmentation,

    layers.Conv2D(32,(3,3), activation = 'relu', input_shape = input_shape),
    layers.MaxPooling2D((2,2)),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3,3), activation='relu'),
    layers. MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers. MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers. MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes , activation='softmax'),
])

model.build(input_shape = input_shape)
```

Activity 2: Compiling the model

```
model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False),
    metrics = ['accuracy']
)
```

Activity 3: Training the model

```
history = model.fit(
    train_ds,
    epochs = 100,
    batch_size = BATCH_SIZE,
    verbose = 1,
    validation_data=val_ds
)
```

The verbose option specifies that you want to display detailed processing information on your screen

If your laptop is functioning slow for 100 epochs, then you can try Google Colab Notebooks. You just have to follow the below steps:

- Go to GoogleColab and create a new notebook.
- Click on Files and mount GoogleDrive.
- Upload your dataset onto GoogleDrive.
  - After uploading the dataset go to runtime and click on “Change runtime type”.
- Select “gpu” and choose “standard” and click on save.
- Booyah! Now you can run the whole code.

Milestone 5: Model Deployment

Activity 1: Model Evaluation

As we have the record of accuracy stored in the history variable. We can try visualising the performance of our model.

Model evaluation

```
In [26]: scores = model.evaluate(test_ds)
         8/8 [==============================] - 4s 327ms/step - loss: 0.1249 - accuracy: 0.9570

In [27]: scores
Out[27]: [0.12492676079273224, 0.95703125]

In [28]: history
Out[28]: <keras.callbacks.History at 0x7fc9046d8370>

In [29]: history.params
Out[29]: {'verbose': 1, 'epochs': 10, 'steps': 54}

In [30]: history.history.keys()
Out[30]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])

In [73]: history.history['accuracy']
```

history.history['accuracy'] contains the track of accuracies achieved while you were training your model.

Import all the Dependencies

In [ ]:
```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML
```

Set all the Constants

In [ ]:
```
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=50
```

Import data into tensorflow dataset object

In [ ]:
```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "PlantVillage",
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

```
Found 2152 files belonging to 3 classes.
```

Theabovecodeplotstwographs, onegraphb/wTrainingandValidationAccuracyandtheothergraphsb/w TrainingandValidationLoss.

In [ ]:
```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

```
(32, 256, 256, 3)
[1 1 1 0 0 0 0 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 2 0 0]
```

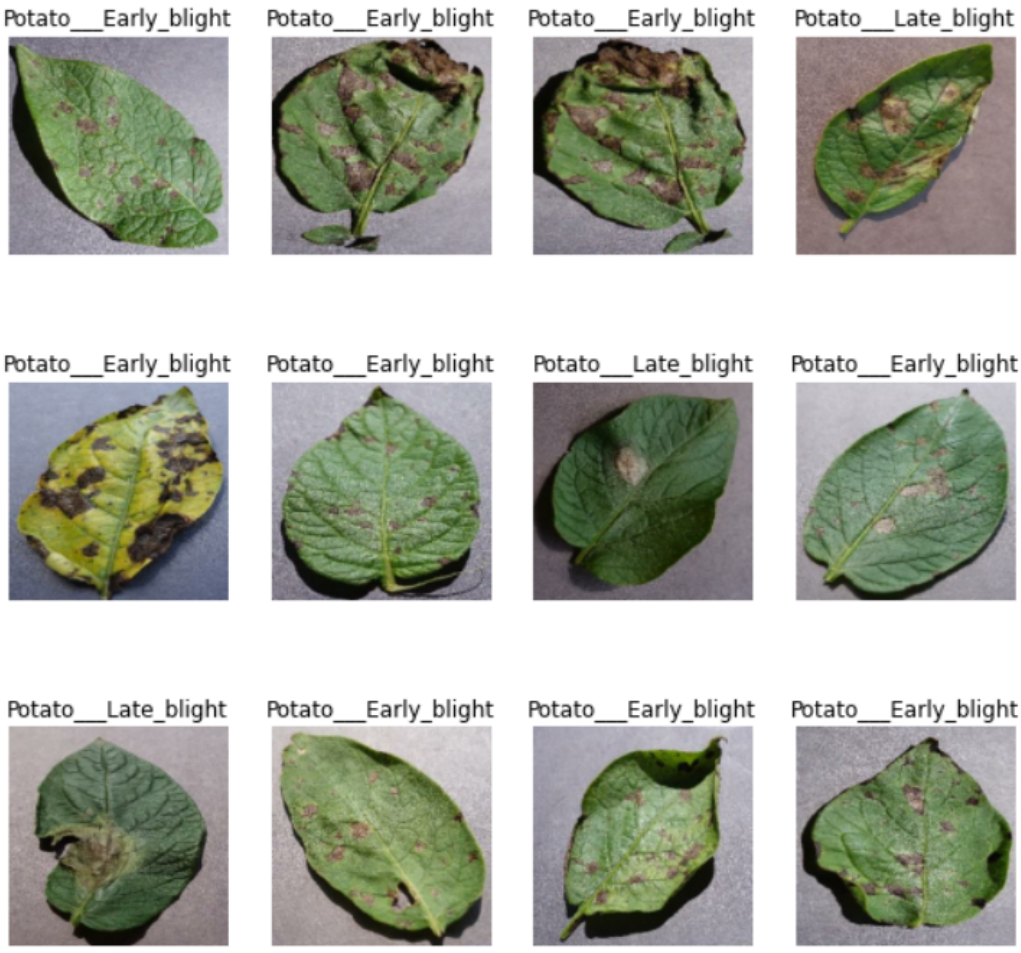
As you can see above, each element in the dataset is a tuple. First element is a batch of 32 elements of images. Second element is a batch of 32 elements of class labels

Visualize some of the images from our dataset

In [ ]:
```
plt.figure(figsize=(10, 10))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```

Activity 2: Model Evaluation with Visualisation

First we'll select one image from the test dataset and try to predict using the trained model. Then we'll try to print the actual class and predicted class of that particular image.



Each time you run this code, you'll get a different image to predict.

Function to Split Dataset

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

Now let us write a function for prediction, which returns the predicted class and the confidence of its prediction.

Model Architecture

We use a CNN coupled with a Softmax activation in the output layer. We also add the initial layers for resizing, normalization and Data Augmentation.

In [ ]:

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape=input_shape)
```

Evaluating a set of images from the test dataset and visualising there prediction and confidence.

Compiling the Model

We use `adam` Optimizer, `SparseCategoricalCrossentropy` for losses, `accuracy` as a metric

In [ ]:
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

In [ ]:
```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=50,
)
```

Activity 3: Save the model

This will allow us to save the .h5 format of the model.

Activity 4: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built.

We will be using the streamlit package for our website development.

Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps.

Activity 4.1: Create a web.py file and import necessary packages:

NOTE : If you get any error like “streamlit not found”. You have to make sure streamlit is installed on your system.

- Running the command “pip install streamlit” would do that job for you.

Activity 4.2: Defining class names and loading the model:

Activity 4.3: Accepting the input from user and prediction

- st.write() function will write the title for your website.
- In the st.file_uploader() function, ‘type’ parameter helps us to restrict the type of input data.

- The above lines of code helps us to take the input from the user and predict the results.
- The website prompts you the text until you upload a file.

- Once you upload an image it starts predicting the label that it should be belonging to.

To run your website go to your terminal with your respective directory and run the command :

- "streamlit run web.py"

On successful execution you'll get to see this in your terminal.

HOW DOES YOUR WEBSITE LOOK LIKE ?

- Before uploading the picture :

- After uploading the picture :

Milestone 6: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables. **Activity 1: - Record explanation Video for project end to end solution.**

Activity 2: - Project Documentation-Step by step project development procedure.

Create a document as per the template provided.