POTATO LEAF DISEASE

IMAGE CLASSIFICATION

PROJECT

Team Members:

1. SUVARAPU SREEVISHNU
2. Thippannagari Indu
3. Bikki Anusha
4. K Jathin Kumar Reddy

Table Of Contents:

1. Introduction

1.1 Introduction

As we know, potato is one of the most extensively consumed crops. If we want that the crop production continues well, then it is very important to identify the reasons which may cause problems in crop production. Phytophthora infestans and Alternaria solani are two pathogens that cause significant production loss in potato crop, causing diseases named late blight and early blight respectively. If somehow, we can detect these diseases early, then it will allow us for the implementation of preventive measures as well as the reduction of financial and yield losses. The most common method for detecting and identifying plant diseases in recent decades has been expert naked eye monitoring. However, in many circumstances, this strategy is impractical due to scarcity of experts on farms in remote places and long processing times. As a result, the use of image processing technologies has proven to be an excellent way for continuously checking plant health and early disease diagnosis. Hence, the objective of this project is to create classification methodology using simple convolutional neural network to detect disease by giving input as potato leaf.

1.2 Project Objectives

As we know that Indian economy has a huge dependency on agriculture sector. Agriculture sector contributes 18% to India's GDP and about 60% of Indian population work in this sector. So, there is need of new technologies to help in growth of crop production. If we can detect diseases early, we can apply various methods to prevent disease and ultimately production of crops will increase as well as revenue will also increase. If we try to monitor with our naked eyes then it will be very time consuming and it also requires expertise in the field. So, we can apply image processing techniques to correctly classify plant whether it is healthy or not which will save time as well as resources. So, the objective of this project is to use CNN for image classification and create a web application which will take an image as an input and will classify it according to the saved model.

1.3 **Methodology**

This problem is an image classification problem. We are using convolutional neural network to train our model as it provides better resource utilization and better accuracy as compare to normal neural networks. It normally contains sequentially number of convolution layers, max pooling layers, activation function followed by normal dense layers. These extra layers provide better accuracy as compare to normal dense layers. ReLU activation function has been used and also adam optimizer has been used as it provides both dynamically changing learning rate and exponential weighted average concept to reduce noise.

2.Problem Statement

In India's vast agricultural landscape, where more than 70% of the population depends on farming for their livelihood, early identification of plant diseases is crucial to preserving crop harvests. Traditional disease diagnosis techniques, which frequently rely on visual inspection by qualified experts, are labor and time intensive as well as requiring a high level of plant pathology knowledge. The revolutionary potential of image processing and machine learning approaches in revolutionizing the field of plant disease identification is made possible by this inherent constraint. In order to identify plant diseases, our work explores the fields of image processing and machine learning. We are able to extract important features from leaf photos and detect the minute details that indicate the presence of disease by utilizing the capabilities of image processing tools. Subsequently, these retrieved features are fed into advanced machine learning algorithms, which allow them to create a strong correlation with particular plant illnesses. With careful consideration of variables like leaf color, damage severity, position, and surface texture, the model produces precise disease identifications. Because of its improved computing efficiency and decreased dependence on assumptions, our suggested method differs from other machine learning-based techniques. Our strategy achieves outstanding accuracy in disease identification without the associated computational loads by utilizing measurable artificial intelligence and image processing techniques, in contrast to existing approaches that frequently use deep learning architectures. This methodological change has many strong advantages over conventional approaches. First of all, it drastically lowers labor expenses, which can 3 add up in large-scale farming operations. Second, it improves scalability, making it possible to monitor large areas effectively and guarantee early disease outbreak identification. Lastly, disease diagnosis can be done more simply and affordably by directly observing leaf symptoms rather than requiring specialized tools or in-depth training.

3.Abstract

Potato is one of the most extensively consumed staple foods, ranking fourth on the global food pyramid. Furthermore, due to the global pandemic coronavirus, global potato consumption is expanding dramatically. Potato diseases, on the other hand, are the primary cause of harvest quality and quantity decline. Plant conditions will be dramatically worsened by incorrect disease classification and late identification due to which Potato farmers are experiencing significant financial losses due to numerous diseases. If farmers can identify these diseases early and treat them appropriately, they can save a lot of money. Leaf conditions can help identify various illnesses in potato plants. In this project, with the help of convolution neural network we tried to classify a potato plant by its leaf whether it is healthy or not. Mainly there are three classes Late Blight, Early Blight and Healthy. According to the confidence each potato leaf has been classified. A simple CNN architecture is used to train and test the model. Main objective was to create a web api which will take leaf image as input and will classify according to the trained model.

4.Data Collection and Preprocessing

4.1 IMAGE ACQUISITION

The initial step in any supervised machine learning project is the data collection procedure. To gather datasets, there are essentially three steps involved:

1) Gather and annotate data independently.

2) Create web scraping scripts to gather online photo content.

3) Purchase data from independent suppliers or utilize open repositories like Kaggle.

4) The dataset was obtained from the Kaggle database. The images in the dataset are divided into three classes. The dataset link is provided below: https://www.kaggle.com/arjuntejaswi/plant-village .

Dataset contains three types of images:

Potato_healthy

Potato_Early_blight

Potato_Late_blight

4.2 IMAGE PROCESSING

Since the images were taken in the actual field, there may be water stains, spores, and residue as commotion. Pre-handling of information is motivated by the desire to alter pixel values and remove disturbance from the image. It enhances the image's character.

Pre-handling is primarily used to eliminate unwanted picture information and enhance some important picture highlights. It combines middle sifting, picture resizing, and RGB to dim change. In order to make the picture device autonomous, the variety picture is completely replaced with a dim scale image. After that, the image is resized to 256 by 256 pixels.

5.FEATURE EXTRACTION

Feature extraction in the Convolutional Neural Network (CNN) process of classifying potato diseases entails methodically recognizing and enumerating unique patterns from input images of potato leaves.

While the pooling layers lower dimensionality, the convolutional layers of the CNN analyze the images to identify edges, textures, and shapes. The network is then able to comprehend complex relationships and patterns by feeding the flattened features into dense layers. For various disease categories, probabilities are provided by the output layer. By means of training on labeled datasets, the CNN gains the ability to optimize its parameters, thereby augmenting its precision in classifying potato leaves as either disease-free or impacted by early or late blight. The model is able to automatically identify 19 pertinent patterns thanks to this hierarchical feature extraction approach, which aids in the efficient classification of diseases.

What are Convolutional Neural Networks?

Convolutional Neural Networks are a special type of Neural Networks that has shown to be extremely effective in image categorization. As they provide extra layers such as convolution, pooling these networks can be used in computer vision applications and they are very useful in face recognition, object detection. Generally, this special type of networks consists of more than one convolutional layer. In the end layers they are followed by dense layers which are fully connected such as a simple neural network. CNN is made in such a way it takes the advantage of Two-dimensional structure of an image. In this, tied weights and local connections has been used. Then they are followed by max or mean pooling layers which are used to extract the most prominent features. Main advantage CNNs have over normal neural networks are that instead of having same number of hidden layers, they will have less parameters to train as compared to neural networks. They are easier to train because of having less parameters.

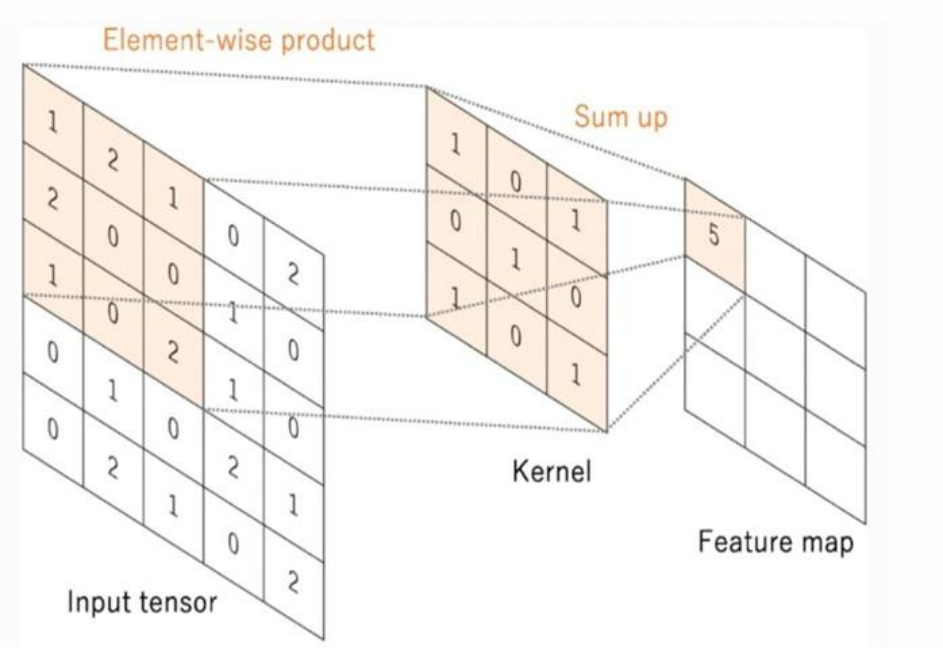
Overview of different operations in CNN:

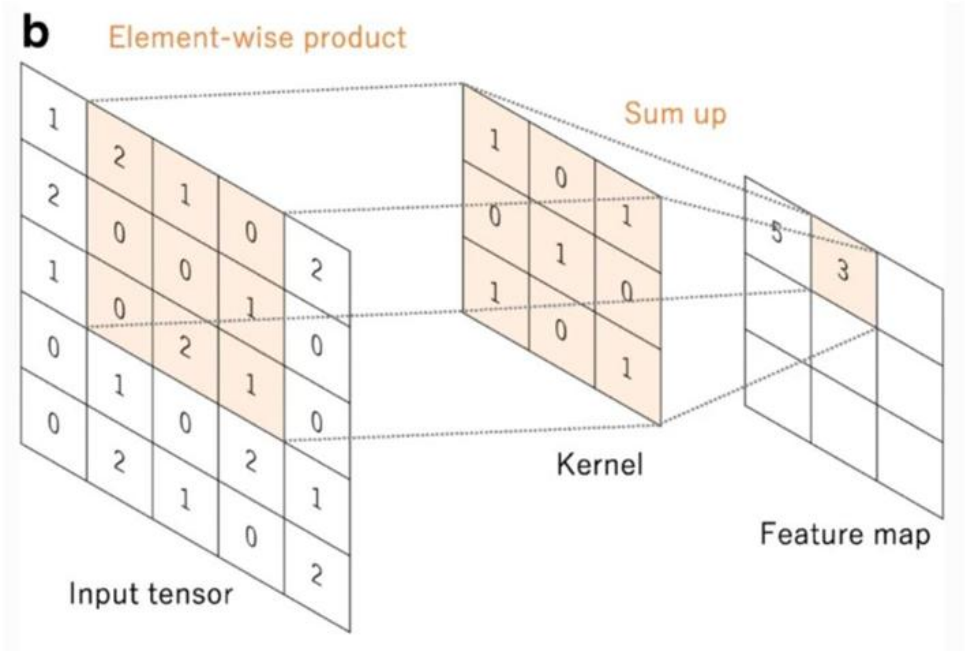
As shown in figure above, it is a CNN architecture which contains some sequential operations. This simple CNN consists of convolutional layer in which a kernel or filter is applied which is used to detect the vertical or horizontal edges and then it is followed by Relu activation function. Next layer is max pooling layer in which a kernel is applied to extract the most prominent features from the image. Then followed by fully connected dense layers. Accuracy of the model is determined by the loss function in forward propagation and according to the loss value weights, filters and bias are updated in back propagation which may be through gradient descent or AdaDelta or adam.

Convolution

In this operation, a small tensor is applied across an input image which is called as a kernel or a filter. Its main task is to extract the important features from the image. After this operation a feature map is generated which is computed through element wise multiplication of each value in image and kernel. This feature map will contain the vertical and horizontal edges which are enhanced in next step.

Figure showing convolution operation





Above figure is showing a convolution operation on a 5x5 image. Filter size is 3x3 without padding and stride equal to 1. If there is no padding then it will reduce its size as shown in the figure. To maintain same image size padding is used which will add extra pixels across image to maintain its size.

Formula to calculate image size:

(N+ 2P – F)/S + 1

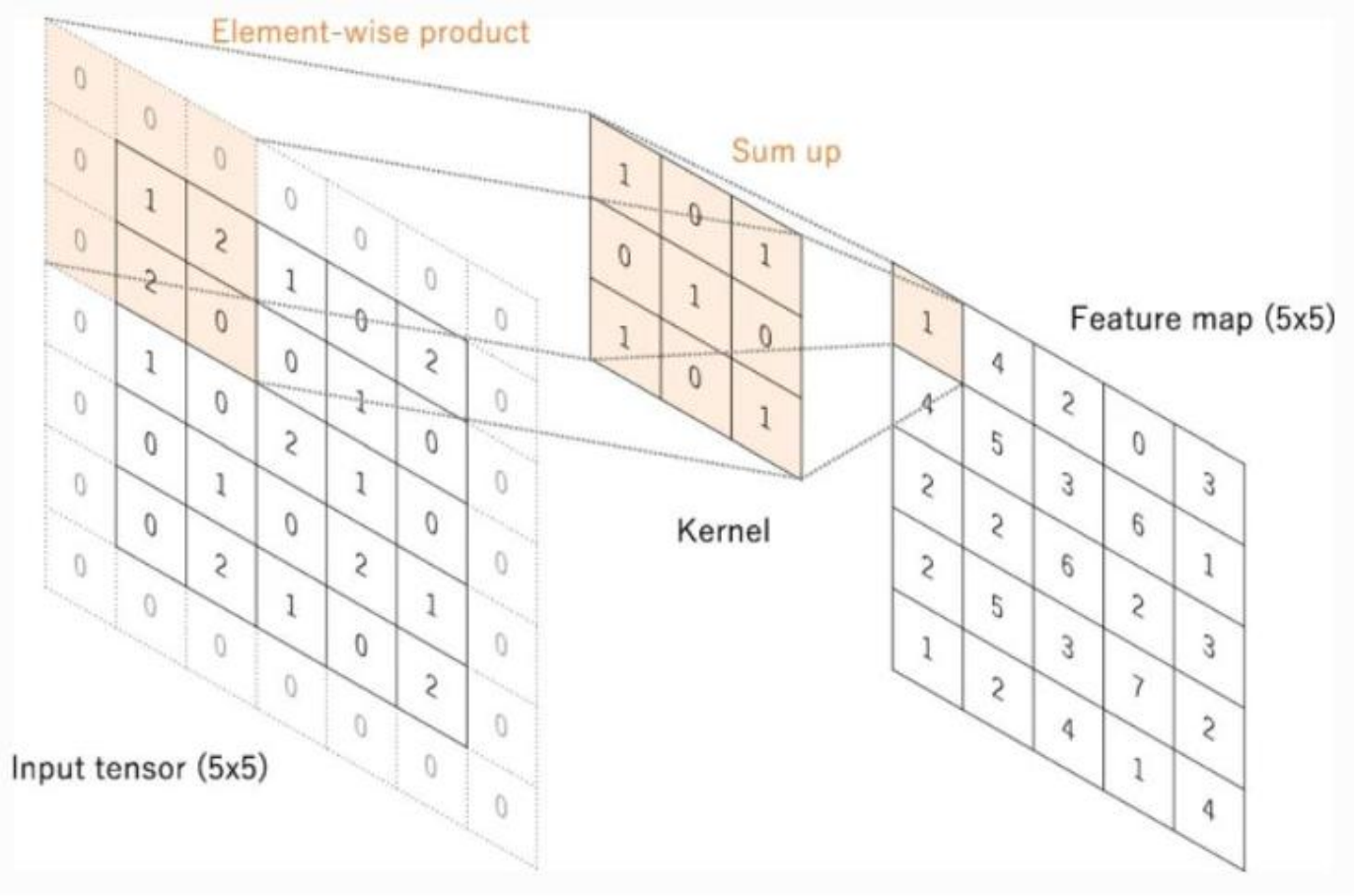
Where N = image size

P = padding

F = kernel size

S = stride Figure with padding:

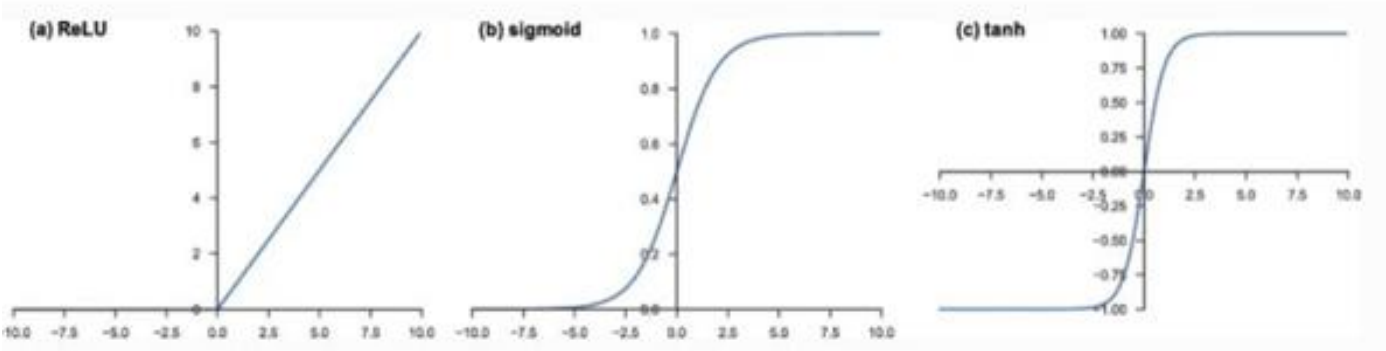
Figure with padding:



Activation Function:

Feature maps which are obtained after applying convolution are passed through an activation function. These activation functions are non-linear to introduce non-linearity in the network. Activation functions such as sigmoid, tangent are not used in hidden layers as it may result is vanishing gradient problem. Highly used activation function is Relu function.

F(x) = max(0,x)

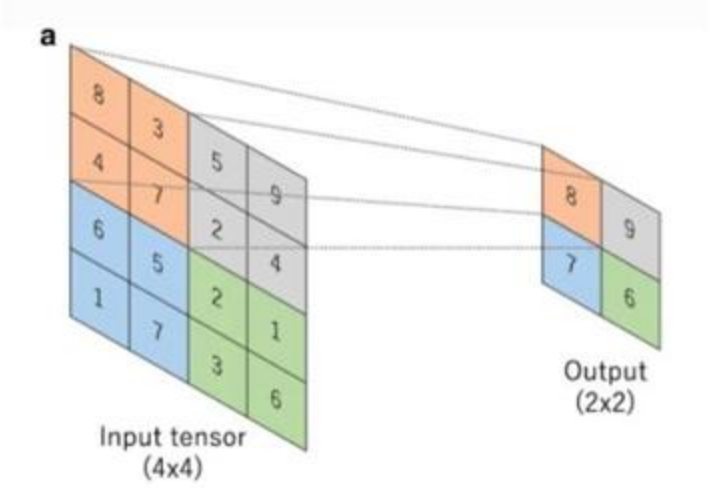


Pooling Layer

This layer is used to extract the most important features. Its main task is to reduce the learnable parameters. It reduces the height and width of feature map but depth remains same.

Max Pooling

In this layer a kernel is applied across an image which will extract maximum value from each patch. Mainly 2x2 filter size is used with stride equal to 2 over an image. Below is the figure showing max pooling operation.

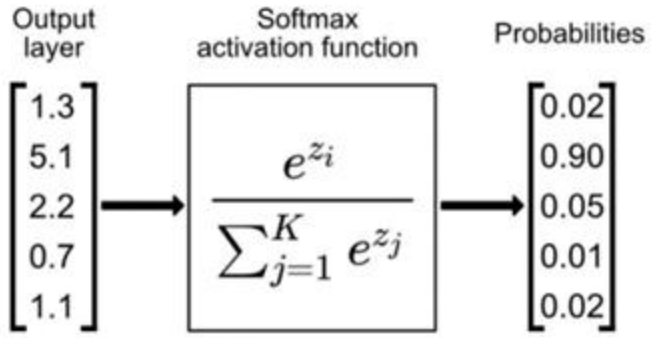


Fully Connected Dense Layer

After receiving feature maps from max pooling layer, these feature maps are flattened and passed to a fully connected dense layer. Feature maps are first converted to a one-dimensional array and passed to a dense layer. There can be many dense layers in-between. Final output layer mainly contains the number of output classes. Every hidden dense layer contains an activation function. Most commonly used activation function is Relu.

Activation Function at Last Layer

Activation function which is used at last output layer is different from the activation functions that has been used in hidden dense layers as their main task is to introduce non-linearity. Activation function at last layer is used to categorize that means it gives probability of each class. If there are two classes then sigmoid function is used. But in this project, there are more than two categories. So, in this situation softmax function is used which will normalize in such a way that each probability value lies in range of 0-1 and final sum will be 1. Below is the figure showing output of a sigmoid function.



In forward propagation all the weights, filters and bias are calculated. These are then validated by a cost function which will give error. Error is the difference between actual label and predicted label. In back propagation weights, filters and bias updation occurs. This updation is carried out by an optimizer which is an algorithm. Its task is to find values of weights, filters and bias is such a way that it will reduce its loss. There are many optimizers available. One which I am using is adam. Loss Function Loss function is used to validate our result. It gives the difference between actual label and predicted label. When records are passed in batches or all the records are passed then loss function is termed as cost function. There are many types of cost functions such as multi-class cross entropy and sparse categorical cross entropy. Different loss functions may give different results. So, we need to choose loss functions according to the use case.

Binary Cross Entropy

$$\text{Loss} = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

Multi Class Cross Entropy

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

Optimizers

Optimizers are algorithms which are used to minimize the loss by updating weights and bias. Optimizers are of different types. The optimizer which we are using is referred to as adam optimizer.

Adam Optimizer

Adam optimizer is state of the art algorithm. It increases the speed of gradient descent. It is very effective when using a data set which is very large in size. It uses both momentum concept i.e., exponential weighted average as well as dynamically changing learning rate. It is very fast and resource effective. It is a combination of both gradient descent with momentum and Root mean squared propagation.

Momentum Concept

$$w_{t+1} = w_t - \alpha m_t$$

where,

$$m_t = \beta m_{t-1} + (1 - \beta) \left[\frac{\delta L}{\delta w_t} \right]$$

m_t = aggregate of gradients at time t [current] (initially, $m_t = 0$)
m_{t-1} = aggregate of gradients at time t-1 [previous]
W_t = weights at time t
W_{t+1} = weights at time t+1
α_t = learning rate at time t
∂L = derivative of Loss Function
∂W_t = derivative of weights at time t
β = Moving average parameter (const, 0.9)

RMS Prop

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t+\varepsilon)^{1/2}} * \left[\frac{\delta L}{\delta w_t}\right]$$

where,

$$v_t = \beta v_{t-1} + (1-\beta) * \left[\frac{\delta L}{\delta w_t}\right]^2$$

W_t = weights at time t
W_{t+1} = weights at time t+1
α_t = learning rate at time t
∂L = derivative of Loss Function
∂W_t = derivative of weights at time t
V_t = sum of square of past gradients. [i.e sum($\partial L/\partial Wt-1$)] (initially, $V_t = 0$)
β = Moving average parameter (const, 0.9)
ϵ = A small positive constant (10^{-6})

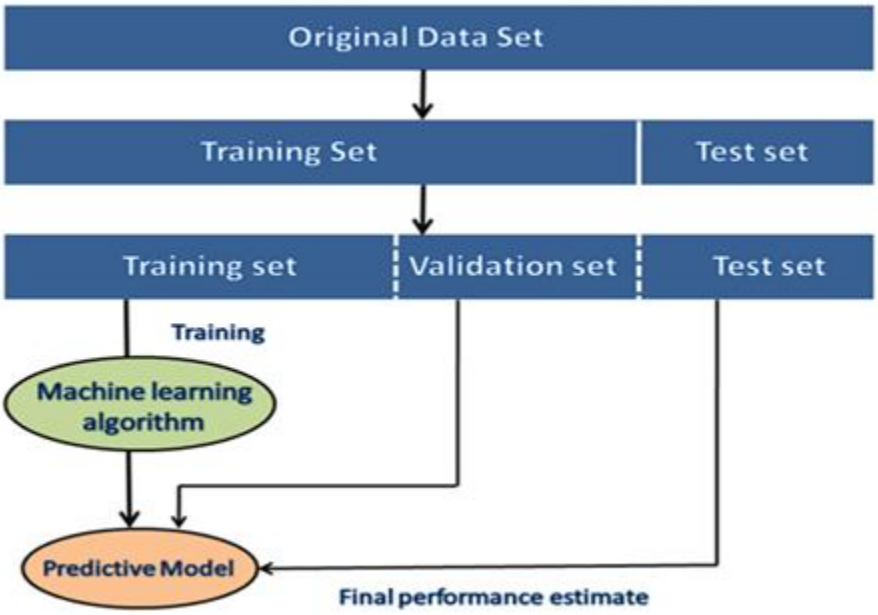
Combining both above equations

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right] v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2$$

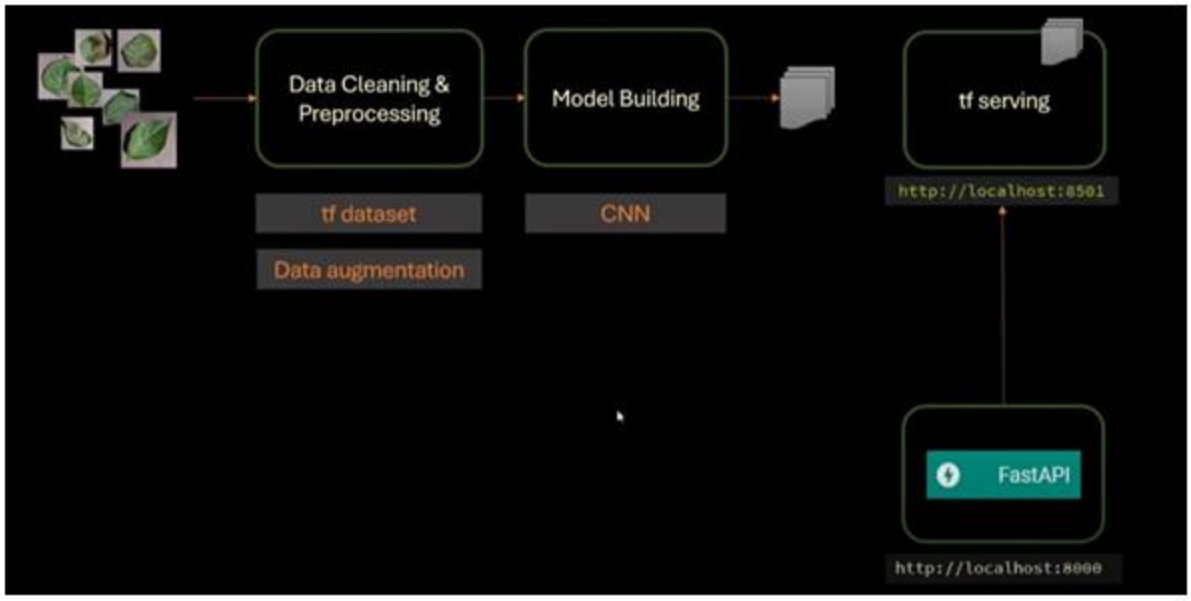
6.Modeling and Analysis

State Transition Diagram of the Major Project

State of the data changes every time it passes through a model. First, step is data pre-processing. Then data is splitted into two sets training set and test set. Training set is used for model building and training. Test set is used for model testing and checking the accuracy. Training set is further divided into a set which will be used for actual training and a validation set which is used for validation at back propagation. After model training, accuracy of the model is validated by using it on testing set. Then model is saved and deployed over an application.



After collecting the data, first step is data cleaning and pre-processing for which I am using tf dataset and data augmentation. Data augmentation used when dataset is not that large then we can just rotate or increase decrease the contrast of the image to create a new image which can be used as an input. Next step is model building using convolutional neural network. CNN is a standard way for image classification as it provides better accuracy as compare to normal neural network. Then saving the model for deployment. Then backend of the web application is designed by FastApi. Below is the figure showing state transition diagram.



Program Implementation

1. First of all, importing all the dependencies

2. Specifying batch size in which each image will be processed. Image size is 256x256. Image is in RGB format. Epochs specify that how many times forward and back propagation will occur.

3. We will use keras to create an input pipeline. In which we have to specify the directory in which images are there. There are three different folders separated according to their classes. Folder name will be considered as the class of each image present in that folder. As all of the images are of (256x256). So, image size is 256.

4. We also need to specify batch size and it will process the images in batches. Benefit of processing in batches will be that it will utilize the resources effectively. If there is a huge dataset that is greater in size as compare to ram of the system then system may crash.

5. There are three classes
   Potato_Early_Blight
   Potato_Late_Blight
   Potato_Healthy

6. First element is a batch of 32 elements of images. Second element is a batch of 32 elements of class labels.

7. Our dataset looks balanced now. We can create a model on top of this data.

8. Plotting the images with their class label using matplotlib

9. For splitting the data into training, testing and validation set a function is defined which takes dataset as an input. Other parameters have default values for train, validation and test split. There is also a shuffle parameter which will shuffle the images in the dataset to add randomness.

10. For resizing the input image if it is not of size (256x256) a layer has been defined using keras which will resize the image to (256x256) so that training can be done. It will also rescale the RGB values.

11. To reduce the chances of underfitting data augmentation has been done which will increase data. It will flip the pictures vertically or horizontally and will also change their contrast so that new input images could be created.

Experimental

After the data preprocessing steps, we trained a simple convolutional neural network. It contains a resizing and rescaling layer followed by number of convolution and max pooling layers. After these layers a fully connected layer is there with 64 neurons. At the end, there is output layer having three neurons as there are three different classes. In every hidden layer Relu activation function has been used. In output layer softmax function has been used. Below is the code:

Total trainable parameters is shown above in the figure.

There are different type of loss functions. In this project sparse categorical cross entropy loss function has been used. Adam optimizer has been used which will reduce the loss by updating wieghts and bias.

Accuracy metrics has been used for validating the loss in back propagation.

For prediction a function has been written which will take model and an input image as input parameters. Model will return an array which will contain the

probability corresponding to each class from which we will select the probability having highest value.

After that model was saved. So that it can be used for prediction in our web application.

7.Results and Findings

The architecture of the CNN model was created to efficiently recognize and understand complex patterns seen in potato pictures of diseases. The classification part of the model consists of fully connected layers after a number of convolutional and pooling layers. An overview of the model architecture is provided below:

Convolutional Layers: To extract hierarchical features, many layers with progressively larger filters and kernel sizes are used.

MaxPooling Layers: A technique for down sampling while keeping important details.

Flatten Layer: In order to make room for fully connected layers, flatten the output.

Dense Layers: Fully connected layers that record intricate relationships by activating ReLU.

Dense layer with SoftMax activation for multi-class classification is the output layer. The created CNN model attained an exceptional accuracy of 90.62% after 35 epochs on the test dataset, following rigorous training and validation. This high accuracy highlights the model's efficacy in correctly categorizing potato illnesses. The model's performance is further validated by the confusion matrix and classification report, which show how precisely it can distinguish between various disease groups.

8.Discussion

1.4 Comparison of Training vs Validation Loss and Accuracy

A comparison of different models and techniques was performed to evaluate their performance and determine the most suitable approach.

[41]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

[43]:
```
acc[0:5]
```

[43]:
```
[0.47476527094841003,
 0.6930751204490662,
 0.7881455421447754,
 0.8420138955116272,
 0.8644366264343262]
```

[45]:
```
len(acc), len(val_acc), len(loss), len(val_loss)
```

[45]:
```
(35, 35, 35, 35)
```

[47]:
```
model.save('model.h5')
```

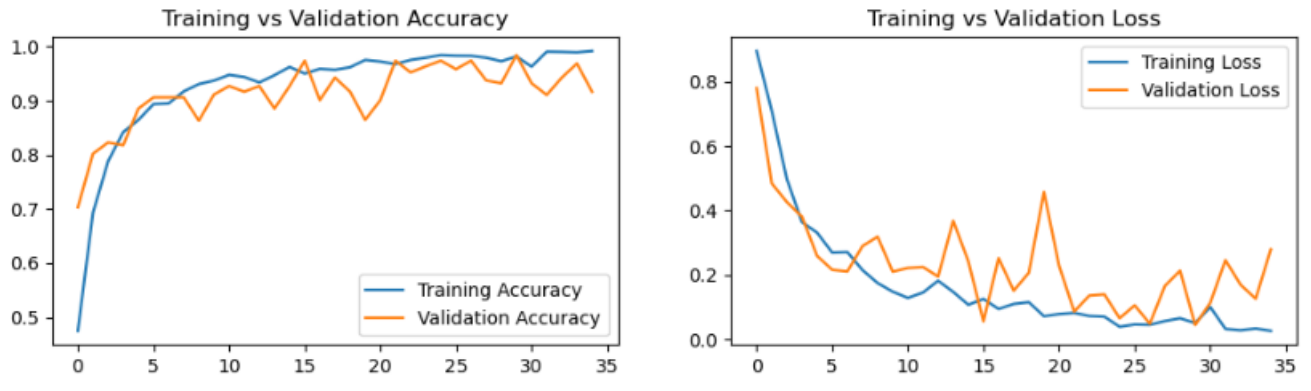
```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considere
recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```

[49]:
```
plt.figure(figsize=(12, 3))

plt.subplot(1, 2, 1)
plt.plot(range(len(acc)), acc, label='Training Accuracy')
plt.plot(range(len(val_acc)), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training vs Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(len(loss)), loss, label='Training Loss')
plt.plot(range(len(val_loss)), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training vs Validation Loss')
```

[49]:
```
Text(0.5, 1.0, 'Training vs Validation Loss')
```



1.5 Challenges and Limitations Encountered

During the project, certain challenges and limitations were encountered, such as data quality issues, limited availability of certain variables, or the presence of outliers. These challenges were discussed, and their potential impact on the analysis and results were acknowledged.

1.6 Ethical Considerations and Data Privacy

Ethical considerations, including data privacy and confidentiality, were taken into account throughout the project. Measures were implemented to ensure compliance with applicable data protection regulations and to protect the privacy of individuals involved.

9.Conclusion

1.7 Conclusions

In this project, convolutional neural network architecture is used in the model. So, that it can be used to classify for web application. Image of a potato leaf will be given as an input then it will be able to classify whether plant is healthy or effected by late blight and early blight. With little modifications, this project can prove very beneficial for farmers. Without monitoring on our own, we can easily identify the plant is infected or healthy. It will reduce the production cost as well as preventive measures can be taken early. This project can be modified so that it can be used for disease identification for other species as well. This can be done in real time and in a better way by using computer vision.

1.8 Future Scope

We will try out CNN architectures such as VGG, Res-Net, Inception Net and try to improve factors like accuracy. A mobile application can be created which will make a call to FastAPI using cross origin resource sharing and will give the results. Deploy this project on Heroku, AWS or various cloud platforms. So that it will be publicly available.

1.9 Applications

There are various applications of deep learning algorithms but talking specifically about this project given an image it can identify a plant whether it is healthy or infected. CNN networks can be trained to classify anything. With computer vision combined these architectures can produce amazing results. After some improvements, this project can be very useful in agriculture sector. Combined with mobile application, it can produce amazing results. Overall, this project may help not only in the agriculture sector but growth of overall country as we know that crops are one of the neediest things in the country.

10.References

https://en.wikipedia.org/wiki/Wiki
https://www.w3schools.com/python/ https://pandas.pydata.org
Matplotlib — Visualization with Python NumPy

11.Appendix

Python: Developed in the latter part of the 1980s and named after Monty Python, Python is a generally supportive programming language used by a vast number of people to do tasks ranging from testing central processor at Intel, to managing Instagram, to creating video games using PyGame library, in addition to completing the photo handling such in our project.

Open CV: A free and open-source computer vision and artificial intelligence program OpenCV (OpenSource PC Vision Library) is the name of the library. OpenCV was created to provide a uniform foundation for PC vision applications and expedite their use of AI in commercial products. Given that OpenCV is a BSD-supported product, institutions can surely make use of and modify the code.

CNN: Within Empirical Education, a Convolutional Brain Association or CNN is a type of affiliation that is typically used for images or objects. insisting and gathering. Significant advancement in this manner perceives things in an image by the use of a CNN. CNNs anticipate playing a huge role in several tasks/restrictions such as photo management problems, PC vision tasks like restriction and division, video assessment, and a genuine examination of oneself driving automobiles and conversing in a native tongue while operating it.

Code :

```
#importing packages and libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from PIL import Image
```

```
from warnings import filterwarnings
filterwarnings('ignore')
```

```
# Make a Dataset
batch_size = 32
image_size = 256
dataset = keras.preprocessing.image_dataset_from_directory(directory=os.path.join('dataset', 'PlantVillage'),
                                                           batch_size=batch_size,
                                                           image_size=(image_size, image_size),
                                                           shuffle=True)
dataset
```

```
Found 2152 files belonging to 3 classes.
<_PrefetchDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec
>
```

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

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

```
len(dataset)
```

```
68
```

```
93 * batch_size
```

```
2976
```

```
def train_validation_test_split(dataset, train_size=0.8, validation_size=0.1, test_size=0.1, shuffle=True, shuffle_size=10000):

    dataset_batch_count = len(dataset)
    train_batch_count = int(dataset_batch_count * train_size)
    validation_test_batch_count = int(dataset_batch_count * validation_size)

    if shuffle:
        dataset = dataset.shuffle(buffer_size=shuffle_size)

    train_ds = dataset.take(train_batch_count)
    validation_ds = dataset.skip(train_batch_count).take(validation_test_batch_count)
    test_ds = dataset.skip(train_batch_count).skip(validation_test_batch_count)

    return train_ds, validation_ds, test_ds
```

```
train_ds, validation_ds, test_ds = train_validation_test_split(dataset)
len(train_ds), len(validation_ds), len(test_ds)
```

```
(54, 6, 8)
```

```
for image_batch, label_batch in train_ds.take(1):
    print(image_batch.numpy()[0].shape)
    print(image_batch.numpy()[0])
    print()

    print(label_batch.numpy().shape)
    print(label_batch.numpy()[0])
    print()

    plt.imshow(image_batch.numpy()[0].astype('uint8'))
    plt.axis('off')
```

```
train_ds
```

```
<_TakeDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3), dtype=tf.float32, name=None), TensorSpec
```

```
channel = 3  # RGB
input_shape = (batch_size, image_size, image_size, channel)
target = 3  # len(class_names)

input_shape
```

```
(32, 256, 256, 3)
```

```
# Image Data Preprocessing

preprocessing = keras.Sequential([
                    keras.layers.Resizing(height=image_size, width=image_size),
                    keras.layers.Rescaling(scale=1./255),
                    keras.layers.RandomFlip(mode='horizontal_and_vertical'),
                    keras.layers.RandomRotation(factor=0.2)
                ])
preprocessing
```

```
# Build CNN Architecture

model = keras.Sequential([
            # CNN
            preprocessing,
            keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=input_shape),
            keras.layers.MaxPooling2D(pool_size=(2, 2)),

            keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
            keras.layers.MaxPooling2D(pool_size=(2, 2)),
            keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
            keras.layers.MaxPooling2D(pool_size=(2, 2)),
            keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
            keras.layers.MaxPooling2D(pool_size=(2, 2)),
            keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
            keras.layers.MaxPooling2D(pool_size=(2, 2)),
            keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
            keras.layers.MaxPooling2D(pool_size=(2, 2)),

            # ANN
            keras.layers.Flatten(),
            keras.layers.Dense(units=64, activation='relu'),
            keras.layers.Dense(units=target, activation='softmax')
        ])
model
```

```
<Sequential name=sequential_1, built=False>
```

```
model.build(input_shape)
model.summary()
```

```
model.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-3),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model
```

```
<Sequential name=sequential_1, built=True>
```

```
history = model.fit(train_ds,
                    batch_size=batch_size,
                    epochs=50,
                    verbose=1,
                    validation_data = validation_ds)
history
```

```
model.evaluate(test_ds)
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 184ms/step - accuracy: 1.0000 - loss: 0.0026
[0.0022487202659249306, 1.0]
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
acc[0:5]
```

```
[0.5082159638404846,
 0.73591548204422,
 0.862089216709137,
 0.8978873491287231,
 0.9409722089767456]
```

```
len(acc), len(val_acc), len(loss), len(val_loss)
```

```
(50, 50, 50, 50)
```

```
model.save('model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.sav
recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras
```

```
plt.figure(figsize=(12, 3))

plt.subplot(1, 2, 1)
plt.plot(range(len(acc)), acc, label='Training Accuracy')
plt.plot(range(len(val_acc)), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training vs Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(len(loss)), loss, label='Training Loss')
plt.plot(range(len(val_loss)), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training vs Validation Loss')
```

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

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

```
def prediction(image_path, class_names = dataset.class_names):

    img = Image.open(image_path).resize((256,256))
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)

    model = tf.keras.models.load_model('model.h5')
    prediction = model.predict(img_array)

    predicted_class = class_names[np.argmax(prediction)]
    confidence = round(np.max(prediction)*100, 2)

    print(f'Predicted Class : {predicted_class}')
    print(f'Confident : {confidence}%')
    print('')
    plt.imshow(img)
    plt.axis('off')
```

```
prediction(image_path='New folder/Healthy_27.jpg')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.c
the model.
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 584ms/step
Predicted Class : Potato___Early_blight
Confident : 91.0%
```

```
prediction(image_path='New folder/Late_Blight_21.jpg')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.com
the model.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 333ms/step
Predicted Class : Potato___Early_blight
Confident : 81.98%
```

Code for Streamlit:

```
import streamlit as st
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import io

# Load the pre-trained model
model = load_model(r'C:\Users\lenovo\Brain O Vision\Project\Final\model.h5')  # Update this path if needed

# Define the categories (for example, three leaf diseases)
categories = ['early blight', 'late blight', 'healthy']  # Adjust based on your model categories

# Streamlit UI layout
st.title("Potato Leaf Disease Classification")

st.write("""
    Upload an image of a potato leaf, and the model will predict the disease type with confidence scores.
""")

# Dropdown for disease selection (optional, for reference)
disease_option = st.selectbox(
    "Select Leaf Disease Type",
    categories
)

# Image upload for prediction
uploaded_image = st.file_uploader("Upload Potato Leaf Image", type=['jpg', 'png', 'jpeg'])

if uploaded_image is not None:
    # Display the uploaded image
    st.image(uploaded_image, caption="Uploaded Potato Leaf Image", width=100)

    # Convert image to the format suitable for the model
    img = image.load_img(uploaded_image, target_size=(224, 224))  # Adjust size based on your model
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    # Predict using the model
    predictions = model.predict(img_array)

    # Get the confidence scores
    confidence_scores = predictions[0]  # Assuming the model outputs a vector of class probabilities

    # Display the confidence scores as a bar chart
    fig, ax = plt.subplots()
    ax.bar(categories, confidence_scores, color='skyblue')
    ax.set_xlabel("Leaf Disease Categories")
    ax.set_ylabel("Confidence Score")
    ax.set_title("Model Confidence Scores")
    st.pyplot(fig)

    # Display the most likely disease prediction
    predicted_class = categories[np.argmax(confidence_scores)]
    confidence = np.max(confidence_scores)
    st.write(f"**Prediction:** {predicted_class} with **confidence:** {confidence*100:.2f}%")
```