Image Classification In Tomato leaf Disease Detection Using Convolutional Neural Network

Deep Learning !

#. Introduction!

- **To increase the agricultural productivity , enhance the performance of pesticides Evaluation.**
- **To ensuring the health and productivity of plants and maintain the growth.**
- **Facilitate of timely intervention , essential to prevent or mitigate negative outcomes.**

#. Motivation!

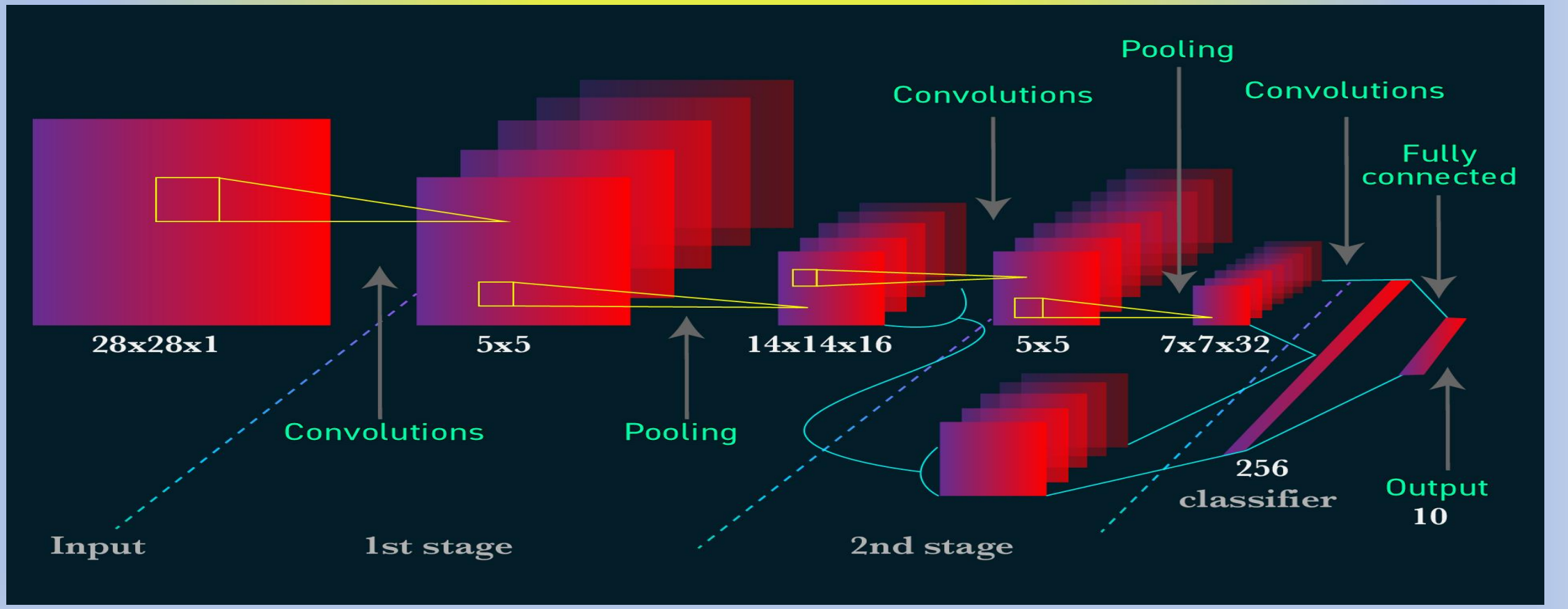
- **Timely pesticides for plants disease was provided by expert people in village , and its take time to give timely treatment.**
- **Now a days it become tedious to give fast and timely treatments for village formers.**
- **To provide the timely and fast treatment .**
- **we have designed the model which will detect disease in less time and provide satisfactory solution .**

#. Task !

- **Trained a custom CNN from scratch , obtained a better performance on our CNN MODEL.**

- **Used various pre-trained model for tomato leaf dataset , to find out the best accuracy model for our dataset.**

#. General Architecture:



#. Image Datasets !

Tomato Late Blight

Tomato Early Blight

Tomato Septoria

Tomato Leaf Mold

Tomato Septoria

Leaf Spot & Tomato Target Spot

Tomato Spider Mites Two Spotted Spider Mite

Tomato Yellow Leaf Curl Virus

Tomato Mosaic Virus

Tomato Healthy

Data Description:

1. **Precision = It is defined as what proportion of predicted positive is truly positives.**

   #Formula= True Positive/Total

2. **Recall = It is defined as what proportion of actual positives is correctly classified .**

   #Formula =True Positive/Actual Yes

3. **F1 Score = It is define as the average of Precision and Recall value.**

   #Formula = 2*(Precision*Recall) / (Precision + Recall)

Classes()	(Own Model) Precision, Recall, F1 Score	(VGG16) Precision, Recall,F1 Score	(VGG19) Precision,Recall,F1 Score	(ResNet50) Precision, Recall,F1 Score	(Xception) Precision, Recall,F1 Score	(MobileNet) Precision, Recall,F1 Score	(EffectiveNetB0) Precision, Recall,F1 Score
Tomato Bacterial spot	0.89, 0.90, 0.90	0.88 0.95 0 .91	0.90, 0.98, 0.94	0.97, 0.97, 0.97	0.78, 0.80, 0.79,	0.88 0.95 0.91	0.97 0.97 0.97
Tomato Early blight	0.90, 0.79, 0.84	0.65 0.83 0 .73	0.71, 0.86, 0.78	0.88, 0.76, 0.82	0.50, 0.78, 0.61	0.65 0.83 0.73	0.88 0.91 0.90
Tomato Late blight	0.84, 0.76, 0.80	0.90 0.83 0 .86	0.80, 0.88, 0.84	0.88, 0.95, 0.91	0.84, 0.54, 0.66	0.90 0.83 0.86	0.84 0.95 0.89
Tomato Leaf Mold	0.97, 0.88, 0.92	0.77 0.86 0 .82	0.71, 0.87, 0.78	0.94, 0.87, 0.90	0.84, 0.57, 0.68	0.77 0.86 0.82	0.93 0.93 0.93
Tomato Septoria leaf spot	0.77, 0.84, 0.80	0.77 0.69 0 .73	0.97, 0.75, 0.85	0.73, 0.94, 0.82	0.58, 0.59, 0.59	0.77 0.69 0.73	0.81 0.95 0.87

Tomato Spider mites Two-spotted spider mite	0.86, 0.80, 0.83	0.88 0.85 0.86	0.84, 0.81, 0.82	0.99, 0.79, 0.88	0.79, 0.60, 0.68	0.88 0.85 0.86	0.91 0.89 0.90
Tomato Target Spot	0.82, 0.91, 0.86	0.88 0.73 0.80	0.77, 0.76, 0.76	0.78, 0.82, 0.80	0.65, 0.66, 0.65	0.88 0.73 0.80	0.93 0.71 0.81
Tomato Yellow Leaf_ Curl Virus	0.90, 0.94, 0.92	0.99 0.96 0.97	1.00, 0.91, 0.95	0.99, 0.98, 0.98	0.70, 0.86, 0.77	0.99 0.96 0.97	0.97 0.97 0.97
Tomato mosaic virus	0.88, 1.00, 0.93	0.89 0.95 0.92	0.98, 0.79, 0.87	0.88, 0.99, 0.93	0.73, 0.85, 0.79	0.89 0.95 0.92	0.84 0.88 0.86

Tomato healthy	0.99, 0.97, 0.98	0.99 0.86 0.92	0.98, 0.90, 0.94	1.00, 0.88, 0.94	0.92, 0.85, 0.89	0.99 0.86 0.92	0.95 0.83 0.89

Why Pre-trained model:

- Time and Resource efficiency ,needs weeks & months to build a CNN form scratch , required less effort to train the parameter for future used.

- Data availability ,Deep learning is data hungry need a huge amount of data to train CNN network.

- We have leverage to used its pre-existing knowledge to perform well on a new, similar task, even with limited data also.

Pre-trained Network Used !

- **VGG-16**
- **ResNet50**
- **Mobile Net**
- **EfficientNetB0**
- **EfficientNetB1**

Models	Parameters	Size(MB)	Epoch	Acc(Train)	Loss(Train)	Acc(Test)	Loss(Test)
Own Model	Total params: 10,490 Trainable params: 10,490	-	50	0.9645000	0.1148474	0.9419999	0.2602567
VGG16	Total params: 21,140,042 Trainable params: 6,425,354	528	26	0.9697999	0.1047671	0.8510000	0.8213710
VGG19	Total params: 26,449,738 Trainable params: 6,425,354	549	22	0.9771999	0.0761569	0.85100001	0.8689888
ResNet50	Total params: 49,280,650 Trainable params: 25,692,938	098	21	0.9839000	0.080691	0.8949999	0.4781706
EfficientNetB1	Total params: 21,388,594 Trainable params:	088	13	0.8022000	0.5697292	0.7099999	0.9042072

MobileNet	**Total params: 16,076,746 Trainable params: 12,847,882**	**016**	**26**	**0.9771999**	**0.098631**	**0.7990000**	**0.7165103**
EfficientNetB0	**Total params: 20,108,717 Trainable params: 16,059,146**	**029**	**10**	**0.9968000**	**0.0098878**	**0.8989999**	**0.7075234**

Generalized Cluster Attached After Pre-trained Models…

Optimizer=Adom

Loss Function=Categorical Cross Entropy

Epochs=50

#. Formula of Categorical Cross Function !

$$L_{CE} = -\sum_{i=1}^{n} t_i \log(p_i), \text{ for n classes,}$$

where t_i is the truth label and p_i is the Softmax probability for the i^{th} class.

#. Adom's Formula !

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Both are come from Momentum and Adagrad concept.	Weight Updation formula !

1. VGG-16 (Visual Geometry Group)

Layer	Output Shape	Parameter

VGG-16	(None,7,7,512)	14714688
Flatten	(None,25088)	0
Dense layer	(None,256)	6422784
Dense output layer	(None,10)	2570

Total Parameters	26,449,738
Trainable Parameters	6,425,354
Non-trainable parameters	20,024,384

2. ResNet50 !

Layer	Output Shape	Parameters
Resnet50	**(None,7,7,2048)**	**23587712**
Flatten	(None,100352)	0
Dense	(None,256)	25690368
Dropout	(None,256)	0
Dense (Output layer)	(None,10)	2570

Total Parameter	**49,280,650**
Trainable Parameter	25,692,938
Non-Trainable Parameter	23,587,712

3. Mobile Net !

layer	Output Shape	Parameter

Mobile net	(None,7,7,1024)	3228864
Flatten	(None,50176)	0
Dense	(None,256)	12845312
Dropout	(None,256)	0
Dense (output layer)	(None,10)	2570

Total Param	16,076,746
Trainable Param	12,847,882
Non-Trainable Param	3,228,864

4. EfficientNetB0!

Layer	Output Shape	Param
Efficientnetb0	**(None,7,7,1280)**	**4049571**
Flatten	(None,62720)	0
Dense	(None,256)	16056576
Dropout	(None,256)	0
Dense(output layer)	(None,10)	2570

Total Parameter	20,108,717
Trainable Parameter	16,059,146
Non-Trainable Parameter	4,049,571

5. EfficientNetB1 !

Layer	Output Shape	Param
Efficientnetb1	**(None,7,7,1280)**	**6575239**
Flatten	(None,62720)	0
Dense	(None,256)	16056576
Dropout	(None,256)	0
Dense(output layer)	(None,10)	2570

Total Parameter	22,634,385
Trainable Parameter	16,059,146
Non-Trainable Parameter	6,575,239

#. Result:

Model	Top-1 Accuracy	Top-5 Accuracy	Correct Prediction	Total Prediction	In (%) Correct P.
VGG-16	73.3%	90.1%	841	1000	84.1
ResNet50	74.9%	92.1%	972	1000	97.2
Mobile Net	70.4%	89.5%	799	1000	79.9
Efficient NetB0	77.7%	93.3%	899	1000	89.9
Efficient NetB1	80.1%	94.9%	899	1000	89.9

#. Custom Network :

Created Models	Conv Layers	Max Pool layers	Global Average Pooling Layer	Dense (Output layer)	Accuracy in (%)	Loss in (%)
First Model	3	3	1	1	88	33
Second Model	2	2	1	1	79	63
Third Model	4	4	1	1	83	59

#. Custom Network Summary :

Created Models	Total Param	Trainable Param	Non-Trainable Param	Correct Prediction	Total Prediction	In (%) correct prediction
First Model	10,490	10,490	0	904	1000	90.4
Second Model	5,690	5,690	0	792	1000	79.2
Third Model	29,306	29,306	0	821	1000	82.1

#. Result !

- Custom network ,first model get better performance over the other two model.
- Our model classifying correctly at the highest probability which is belong to correct class but it is not exactly equal to truth value of that particular class.
- Loss is 33% ,reason is the gap between truth value and actual value of that particular class , After summation of whole sample ,it got maximize.

Experimental Setup:

- **Plate form – Kaggle**
- **Dataset - Plant Village**
- **GPU Version-NIVIDA GPU T4*2**
- **Batch Size – 32**
- **Optimizer – Adom**
- **Loss Function – Categorical Cross Entropy.**
- **Epoch- 50**
- **Train/Validation/Test – 70/20/10 (Ratio%)**

#. Conclusion !

- Trained custom CNN from scratch with an accuracy of 87% ,we have got after evaluation.

- Pre-trained model,EfficienNetB0 give the optimal accuracy with an 99% on our datasets.


Thank You
Dear
Friend