Weekly Report

Comparative Analysis of Image Classification Models for Efficient and Accurate Classification across Noisy Vegetable Images

Student: Yusif Mukhtarov

Date: 11/07/2023

Contents

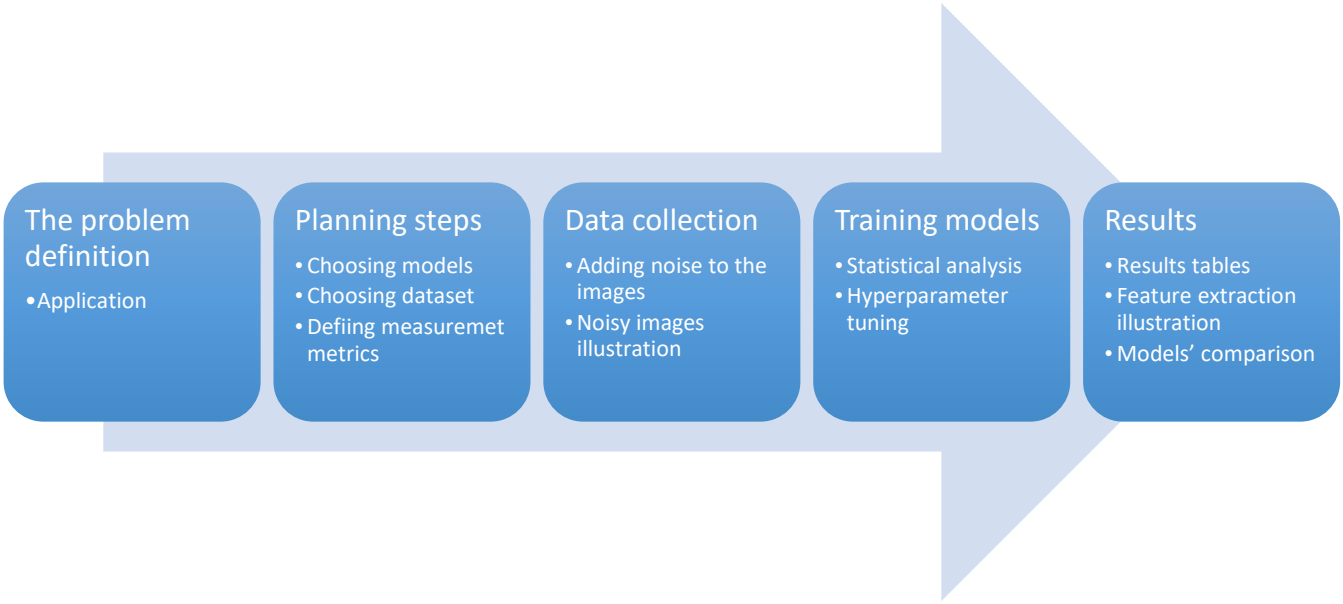
Abstract

This research delves into the in-depth analysis of popular deep learning models, including AlexNet, VGG16, ResNet, and EfficientNet, applied to vegetable custom datasets. The study aims to explore and compare the performance of these models, considering a range of performance metrics to ensure an unbiased evaluation. The chosen vegetable datasets are carefully preprocessed, encompassing required image conditions and added noise to simulate real-world scenarios.

To evaluate the models thoroughly, classification accuracy serves as the primary performance metric. Additionally, precision, recall, and F1-Score are incorporated to provide a comprehensive understanding of the models' strengths and weaknesses in handling false positives, false negatives, and overall balanced performance. Furthermore, the research records the time taken for training and prediction phases, along with the number of parameters, to assess the models' efficiency.

The implications of this research span across various industries. In agriculture, these findings can lead to smarter farming practices, crop health monitoring, and pest detection. The food processing industry can benefit from improved quality control, as the models can assist in sorting vegetables based on visual attributes. Moreover, retail and supply chain management can optimize inventory processes using accurate vegetable identification.

Road map



The problem definition

In the field of deep learning for recognizing images, we usually train models using very good and clear datasets without any errors. However, in real-life situations, the pictures we get might not be perfect. They could have problems like bad weather, issues with the camera, or other factors, making the images noisy and of lower quality. This difference between the training data and real-world images can greatly affect how well the deep learning models work in real situations.

The problem we want to solve is that there isn't enough research and study on how these popular deep learning models, such as AlexNet, VGG16, ResNet, and EfficientNet, perform when they have to deal with noisy and imperfect images. We need to look into how these models behave and adapt in these more challenging situations. By doing this research and studying how well these models work with realistic images, we can better understand how they can be used in practical applications and real-life scenarios.

Application areas

The research has wide-ranging impacts across various industries. In agriculture, it can improve farming practices, monitor crop health, and detect pests. Food processing can benefit from better quality control by sorting vegetables accurately. Retail and supply chain management can optimize inventory using precise vegetable identification.

Apart from agriculture and food, this research has potential applications in environmental monitoring, healthcare, nutrition, and smart kitchen appliances. It can aid in preserving biodiversity and promoting healthier eating habits. Smart kitchen appliances can use these models to improve user experiences and cooking processes.

Moreover, the research goes beyond vegetable recognition and can be applied to other image types, like noisy faces and cars. By using similar methods, it can guide deep learning models for various tasks like facial recognition, object detection, autonomous vehicles, surveillance, and medical imaging. The insights gained here serve as a helpful resource for researchers and practitioners dealing with different image challenges in AI-driven solutions.

Planning steps

At the beginning of the project, a plan was laid out to test six different models for image classification: LesNet, MobileNet, AlexNet, VGG16, EfficientNet, and ResNet. The dataset selection was intended to be from popular sources, and various augmentation techniques like flipping, rotation, occlusion, and noise addition were planned. However, after careful discussions with supervisors, it was realized that attempting all this within the 11-month time frame would be too ambitious.

Consequently, the project scope was narrowed down, and it was decided to focus on testing only four models: AlexNet, VGG16, EfficientNet, and ResNet. Moreover, the decision was made to evaluate these models using only noisy images.

Initially, the intention was to use well-known datasets such as ImageNet, Cifar-10, or Cifar-100. However, following discussions with supervisors, it was deemed better to opt for a random dataset. The reason behind this was that some of the previously listed models might have been trained on the mentioned popular datasets and using them could lead to ambiguity in the results. Therefore, the Vegetable dataset, considered neutral, was chosen for the image classification tasks. Consequently, the project's name was modified to "Comparative Analysis of Image Classification Models for Efficient and Accurate Classification across Noisy Vegetable Images ". To degrade the image quality, it was chosen to add Gaussian noise at different levels: 10%, 30%, 50%, 70%, and 100%. After introducing this noise, it was intended to evaluate the performance of the models on each of these percentage levels to understand how well they can handle images with varying degrees of noise.

In the early stages of the project, there was a consideration of training all the models from scratch. However, due to computational limitations, it was advised by the supervisor to use pretrained models instead. This approach would save computational resources and still allow for meaningful comparisons. For hyperparameter tuning, the focus is planned on optimizing learning rates, schedulers, and momentum values for the models. These parameters play a crucial role in the performance of deep learning models and tuning them would help achieve better results in our image classification tasks.

Data collection

As it was highlighted before, to ensure the best dataset selection, we were careful not to use datasets that had been previously used for pretrained models. After conducting a thorough analysis of various datasets, it was ultimately decided to go with the Vegetable Dataset available on Kaggle (https://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset).

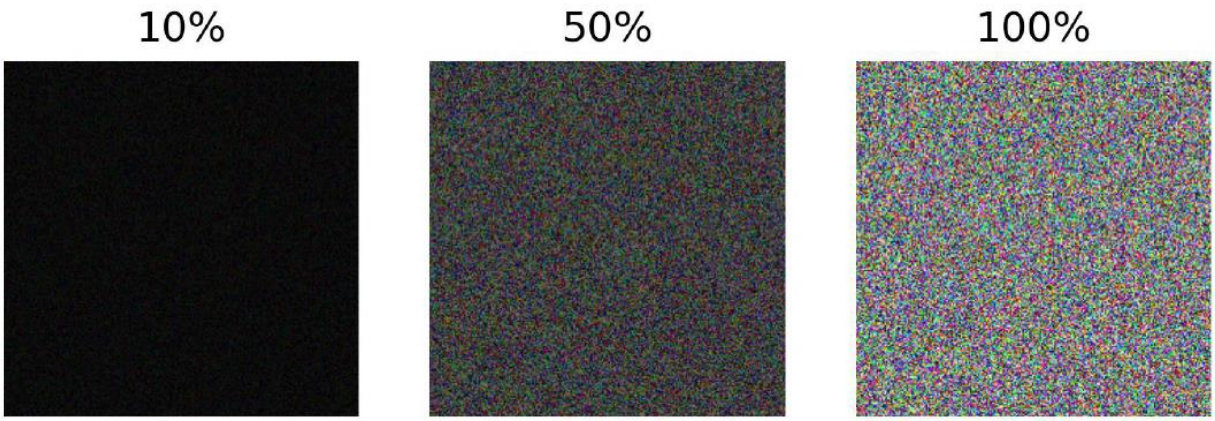
This particular dataset consists of 21,000 images belonging to 15 different classes, with each class containing a total of 1,400 images. The dataset is thoughtfully split into three parts for training, testing, and validation, with 70%, 15%, and 15% of the images allocated to each respective partition.

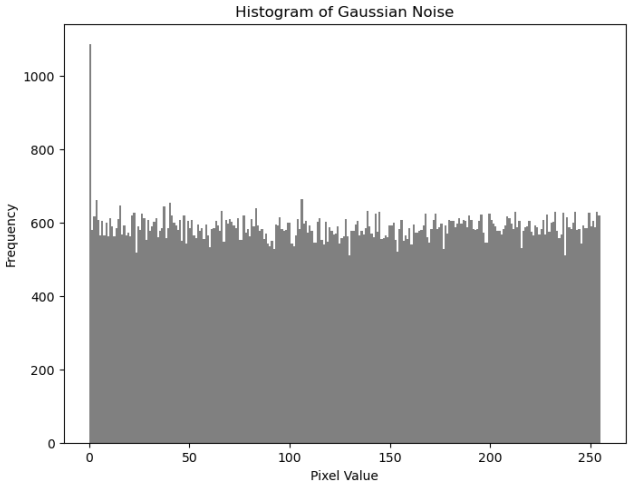
Initially, the dataset information on Kaggle indicated that all classes were proportionally represented and that the image resolution was standardized at 224x224 pixels in JPEG format. However, upon closer examination, we found that there were 9 images that deviated from the expected 224x224 resolution and had different aspect ratios.

```
 from train dataset:
1 0741.jpgwith shape(210, 224, 3)
2 0176.jpgwith shape(198, 224, 3)
3 0126.jpgwith shape(211, 224, 3)
4 0609.jpgwith shape(200, 224, 3)
5 0430.jpgwith shape(193, 224, 3)
6 0526.jpgwith shape(205, 224, 3)

 from validation dataset:
7 1138.jpgwith shape(187, 224, 3)
8 1150.jpgwith shape(223, 224, 3)

 from test dataset:
9 1246.jpgwith shape(207, 224, 3)
```

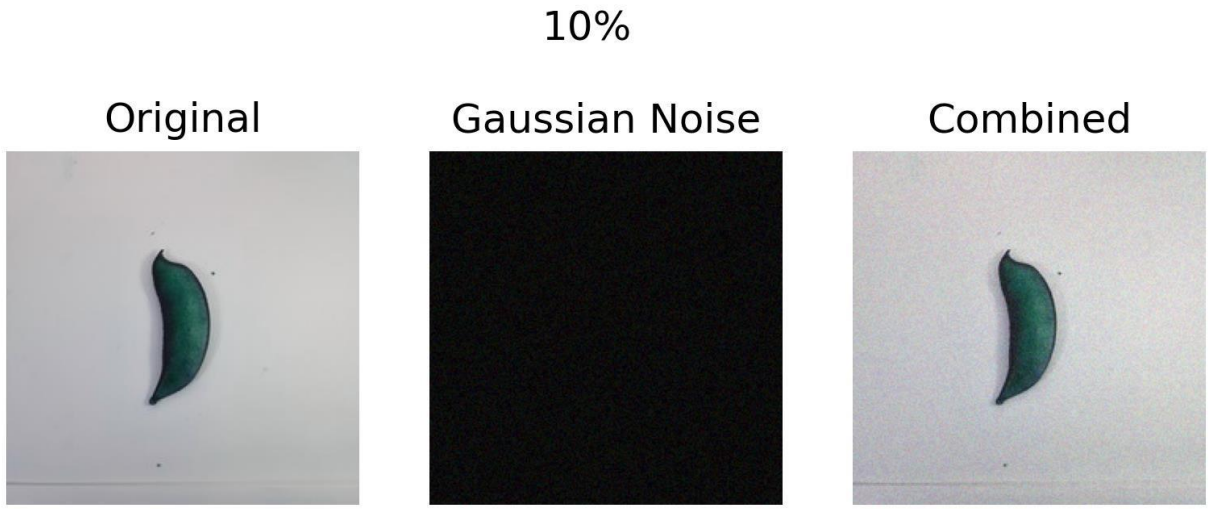
Considering that all images will be resized before training and testing it was decided that this anomaly is insignificant. Furthermore, images should be mixed with noise. For that Gaussian noise was opted to be used in 10, 30, 50, 70 and 100% percentages.

Gaussian noise is widely recognized as statistical noise, following a normal distribution probability density function. This noise is uniformly spread throughout the signal. In the case of a noisy image, the pixels consist of their original values added to a random Gaussian noise value. The figure below illustrates the Gaussian distribution function of Gaussian noise and the pixel representation of this type of noise.





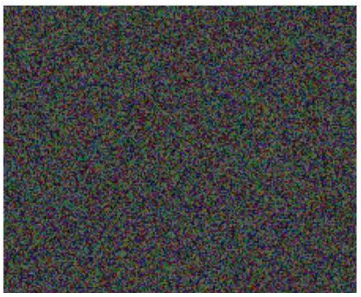
Representation of noisy images:



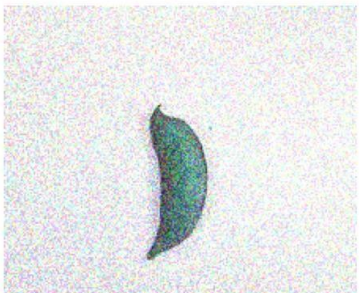
50%

Original

Gaussian Noise



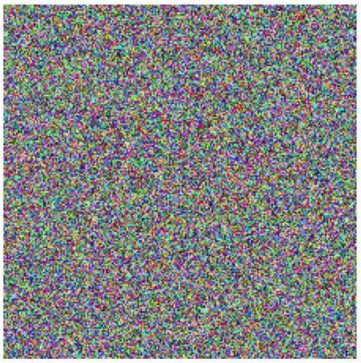
Combined



100%

Original

Gaussian Noise



Combined

Training models

For all 4 models a pixel normalization has been done using the same mean and standard deviation values. The mean values of three channels were calculated simply by taking mean values of pictures for each pixel, then mean values of pixels for three channels:

```
torch.Size([7552, 3, 51529])
```

```
[5]
1 mean = torch.mean(pixel_values, dim=0)
2 mean.shape
```

```
torch.Size([3, 51529])
```

```
[6]
1 mean_per_channel = mean.mean(dim=1)
2 mean_per_channel
```

```
tensor([0.4691, 0.4631, 0.3419])
```

Although the mean calculations were simple, the same strategy cannot be applied for calculation of standard deviation since the equation of standard deviation is not a linear function:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{N}}$$

As it is obvious from the formula above, the mean of standard deviations of pictures pixels will not be equal to true standard deviation. For that, firstly, the standard deviations of pictures' pixels should be raised to the power of two, and after computing their means the square root of the means should be found as a final result:

```
[7]
1 std = torch.std(pixel_values, dim=0)
2
```

```
[8]
1 std_sq = torch.square(std)
2 std_mean = std_sq.mean(dim = 1)
3 std_mean
```

```
tensor([0.0539, 0.0516, 0.0573])
```

```
[9]
1 std_mean = torch.sqrt(std_mean)
2 std_mean
```

```
tensor([0.2322, 0.2272, 0.2394])
```

These values have been used to normalize pictures before inputting into all four models. However, the size of images should be different for AlexNet since pretrained models have been utilized:

	Resizing
AlexNet	227x227
ResNet	224x224
EfficientNet	224x224
VGG16	224x224

The first run was experimental just to setup system, check how dataloading, training, and testing functions are working. For the first run it was decided to set batch size to 64, epoch number to 15, learning rate to 0.001, scheduler step size to 7 and scheduler gamma to 0.1. The training procedures are illustrated below:

```
criterion = nn.CrossEntropyLoss()


optimizer = optim.SGD(efficientnet.parameters(), lr=0.001)


step_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)

efficientnet = train_model(efficientnet, criterion, optimizer, step_lr_scheduler, num_epochs=15)
```

```
Epoch 15/15
----------
15 epoch, correct: 1751/1792 = 0.98
15 epoch, correct: 3500/3584 = 0.98
15 epoch, correct: 5253/5376 = 0.98
15 epoch, correct: 6997/7168 = 0.98
15 epoch, correct: 8746/8960 = 0.98
15 epoch, correct: 10499/10752 = 0.98
15 epoch, correct: 12252/12544 = 0.98
15 epoch, correct: 14001/14336 = 0.98

train Loss: 0.2960 Acc: 0.9766


val Loss: 0.2475 Acc: 0.9847

Training complete in 31m 40s
Best val Acc: 0.984667
```

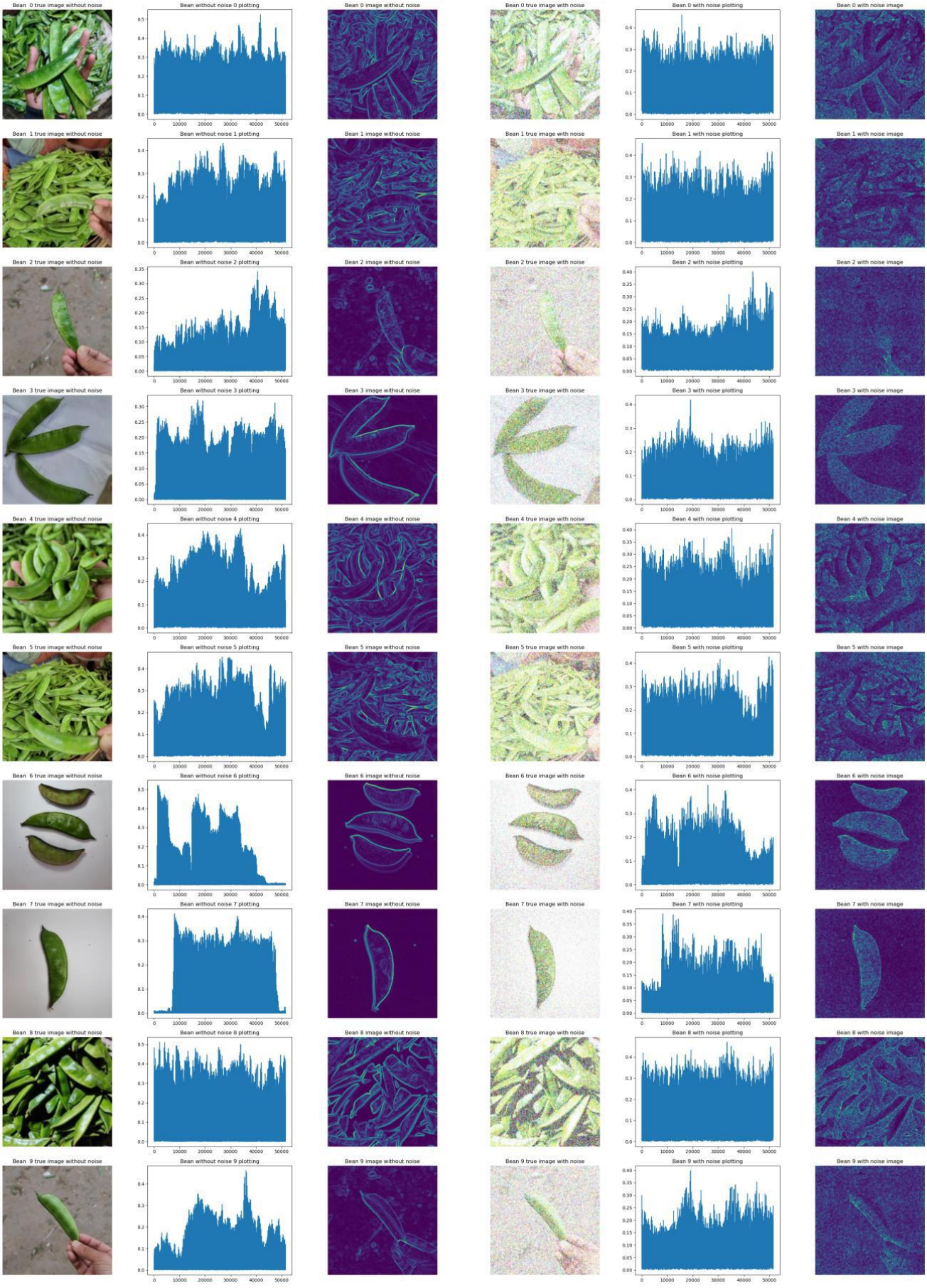
At the end of the first run the following results have been gotten:

	Training accuracy	Test accuracy on images with 10% noise
AlexNet	0.989667	0.96
ResNet	0.998	0.98
EfficientNet	0.984667	0.96
VGG16	0.994667	975

During the training and testing there was a lot of useful information gained. It is planned in the final presentation to show all of that information in charts, graphs, histograms and other types of plotting and take a lot of insights from them. Final result for each model will look like:

	Class	Accuracy	Recall	Precision	F_score
0	Bean	96.50	0.96	0.99	0.97
1	Bitter_Gourd	99.00	0.99	1.00	0.99
2	Bottle_Gourd	96.00	0.96	0.99	0.97
3	Brinjal	89.00	0.89	0.98	0.93
4	Broccoli	99.50	0.99	0.99	0.99
5	Cabbage	99.00	0.99	0.99	0.99
6	Capsicum	74.50	0.74	1.00	0.85
7	Carrot	99.50	0.99	0.99	0.99
8	Cauliflower	98.00	0.98	0.99	0.98
9	Cucumber	98.50	0.98	0.84	0.90
10	Papaya	98.50	0.98	0.87	0.92
11	Potato	100.00	1.00	0.88	0.94
12	Pumpkin	100.00	1.00	0.98	0.99
13	Radish	100.00	1.00	1.00	1.00
14	Tomato	94.00	0.94	0.99	0.96
15	Total	96.13	0.96	0.97	0.96

After gaining the first results feature extraction methods have been also developed to understand how well all 4 models extract features from noisy images:


Bean 0 true image without noise
Bean without noise 0 plotting
Bean 0 image without noise
Bean 0 true image with noise
Bean 0 with noise plotting
Bean 0 with noise image
Bean 1 true image without noise
Bean without noise 1 plotting
Bean 1 image without noise
Bean 1 true image with noise
Bean 1 with noise plotting
Bean 1 with noise image
Bean 2 true image without noise
Bean without noise 2 plotting
Bean 2 image without noise
Bean 2 true image with noise
Bean 2 with noise plotting
Bean 2 with noise image
Bean 3 true image without noise
Bean without noise 3 plotting
Bean 3 image without noise
Bean 3 true image with noise
Bean 3 with noise plotting
Bean 3 with noise image
Bean 4 true image without noise
Bean without noise 4 plotting
Bean 4 image without noise
Bean 4 true image with noise
Bean 4 with noise plotting
Bean 4 with noise image
Bean 5 true image without noise
Bean without noise 5 plotting
Bean 5 image without noise
Bean 5 true image with noise
Bean 5 with noise plotting
Bean 5 with noise image
Bean 6 true image without noise
Bean without noise 6 plotting
Bean 6 image without noise
Bean 6 true image with noise
Bean 6 with noise plotting
Bean 6 with noise image
Bean 7 true image without noise
Bean without noise 7 plotting
Bean 7 image without noise
Bean 7 true image with noise
Bean 7 with noise plotting
Bean 7 with noise image
Bean 8 true image without noise
Bean without noise 8 plotting
Bean 8 image without noise
Bean 8 true image with noise
Bean 8 with noise plotting
Bean 8 with noise image
Bean 9 true image without noise
Bean without noise 9 plotting
Bean 9 image without noise
Bean 9 true image with noise
Bean 9 with noise plotting
Bean 9 with noise image

The first row illustrates the original pictures, the second row its pixel distribution in histogram, the third row shows extracted features from the model, and the next 3 rows represent the same plotting for the noisy versions of the same images. In this plottings the change in pixel and feature extraction capabilities of models are obviously seen. This plotting will be illustrated for all models in the final presentation.

What is left to do?

1. Hyperparameter tuning for scheduler, momentum, and learning rate.
2. Selection of the best versions of models based on the test accuracies, precision, and F scores.
3. Statistical analysis of the feature extraction capabilities of those models.
4. Graphical illustrations of the results.