PlantVillage

By

Mr.Abhinav Lugun
Mr.Md. Sakib Bin Alam
Mr.Pongraphe kangsanarak
Mr.Saw Lar Ka Paw
Mr.Paramik Dasgupta
Mr.Set Khaing Oo

Presentation Outline

- Motivation
- About Dataset
- Experiment
- Extract, transform, load (ETL)
- Exploratory data analysis (EDA)
- Train, Val & Test Split
- Creating CNN Models
- Training CNN Models
- Test/Inference
- Examining Misclassified Labels
- Transfer learning
- Observation
- Conclusion

Motivation

Plant disease detection can be a crucial work in the agriculture domain. Early stage symptom detection will increase the productivity of crops. By this study we used deep convolutional neural network and transfer learning for classifying plant diseases.

About Dataset

- https://data.mendeley.com/datasets/tywbtsjrjv/1
- Contains images of healthy leaves and many types of unhealthy leaves
- has total class of 39
- Dataset which we used were already augmented
  - image flipping
  - Gamma correction
  - noise injection
  - PCA color augmentation
  - rotation
  - scaling

Classes in Dataset

```
['Apple___Apple_scab',
 'Apple___Black_rot',
 'Apple___Cedar_apple_rust',
 'Apple___healthy',
 'Background_without_leaves',
 'Blueberry___healthy',
 'Cherry___Powdery_mildew',
 'Cherry___healthy',
 'Corn___Cercospora_leaf_spot Gray_leaf_spot',
 'Corn___Common_rust',
 'Corn___Northern_Leaf_Blight',
 'Corn___healthy',
 'Grape___Black_rot',
 'Grape___Esca_(Black_Measles)',
 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
 'Grape___healthy',
 'Orange___Haunglongbing_(Citrus_greening)',
 'Peach___Bacterial_spot',
 'Peach___healthy',
 'Pepper,_bell___Bacterial_spot',
 'Pepper,_bell___healthy',
 'Potato___Early_blight',
 'Potato___Late_blight',
 'Potato___healthy',
 'Raspberry___healthy',
 'Soybean___healthy',
 'Squash___Powdery_mildew',
 'Strawberry___Leaf_scorch',
 'Strawberry___healthy',
 'Tomato___Bacterial_spot',
 'Tomato___Early_blight',
 'Tomato___Late_blight',
 'Tomato___Leaf_Mold',
 'Tomato___Septoria_leaf_spot',
 'Tomato___Spider_mites Two-spotted_spider_mite',
 'Tomato___Target_Spot',
 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
 'Tomato___Tomato_mosaic_virus',
 'Tomato___healthy']
```

Experiment

- To see the effect of increasing the number of layers in CNN models on plant health identification
- To see the effectiveness of transfer learning for this project

1) Extract, Transform, Load (ETL)

Extract, Transform, Load (ETL)

```
imageSize = 100
```

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize((imageSize, imageSize)),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
    ])
```

```
DIR = './Plant_leave_diseases_dataset_with_augmentation/'
plantVillDataset = datasets.ImageFolder(DIR, transform=transform)
plantVillDataset
```

```
Dataset ImageFolder
    Number of datapoints: 61486
    Root location: ./Plant_leave_diseases_dataset_with_augmentation/
    StandardTransform
Transform: Compose(
               ToTensor()
               Resize(size=(100, 100), interpolation=bilinear, max_size=None, antialias=None)
               Normalize(mean=(0.5, 0.5, 0.5), std=(0.5, 0.5, 0.5))
           )
```

2) Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA)

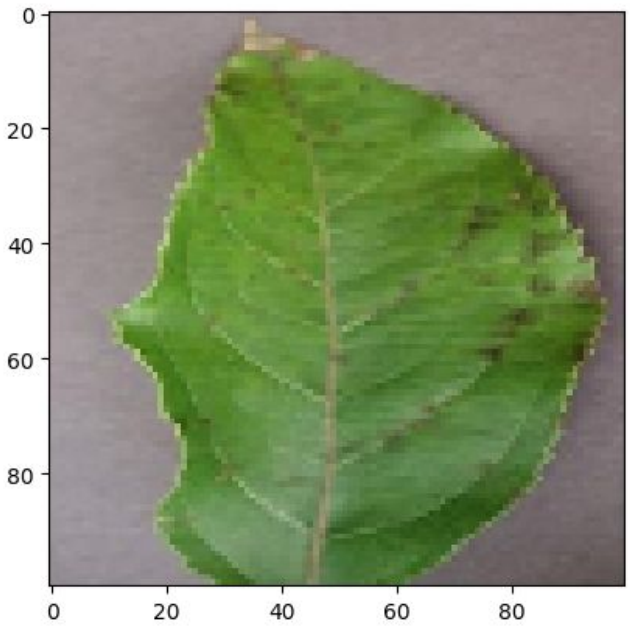
Showing a sample image

```
classNames = list(plantVillDataset.class_to_idx)
```

```
def displayImage(img):
    img = img / 2 + 0.5
    plt.imshow(transforms.ToPILImage()(img))
```

```
for temp in plantVillDataset:
    print(temp[0].shape)
    displayImage(temp[0])
    break
```

```
torch.Size([3, 100, 100])
```

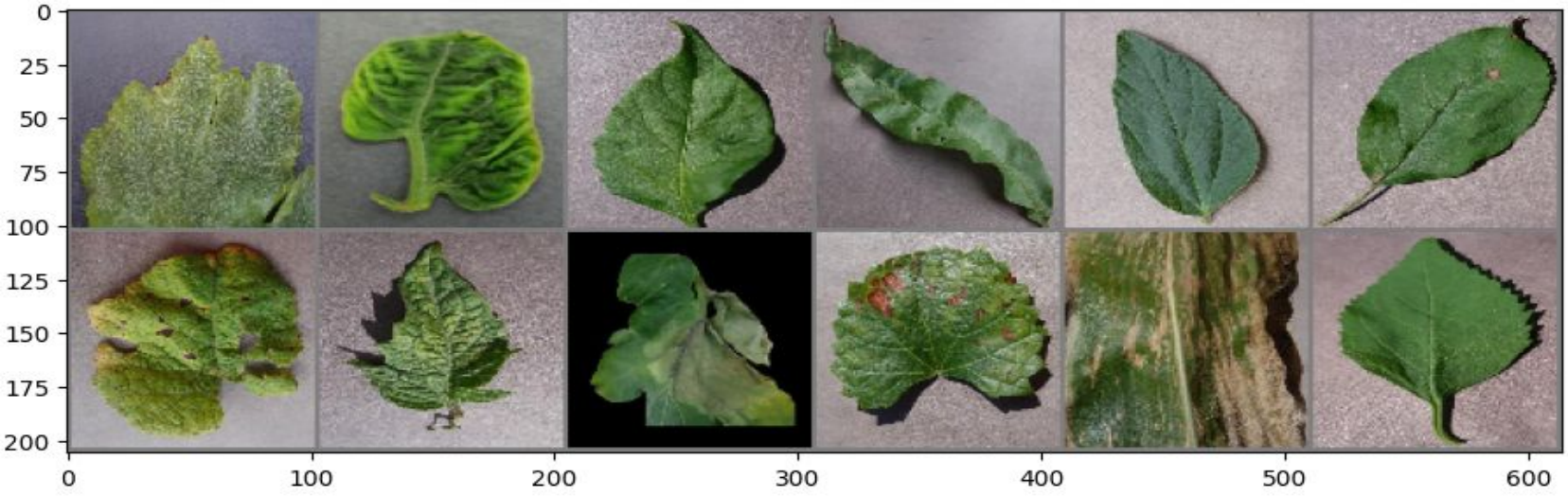


Showing images from different labels

```
def show_sample_images(dataset):
    loader = DataLoader(dataset, batch_size = 12, shuffle=True)
    batch = next(iter(loader))
    images, labels = batch

    grid = make_grid(images, nrow = 6)
    grid = grid / 2 + 0.5 # unnormalize image
    im_transpose = np.transpose(grid, (1, 2, 0))
    plt.figure(figsize=(11, 11))
    plt.imshow(im_transpose)
    print('Encoded Labels: ', labels)
    print('Labels:')
    for label in labels:
        print(classNames[label])
```

```
show_sample_images(plantVillDataset)
```

```
Encoded Labels:  tensor([26, 36, 20, 17, 25,  1, 14, 34, 31, 13,  8,  7])
```



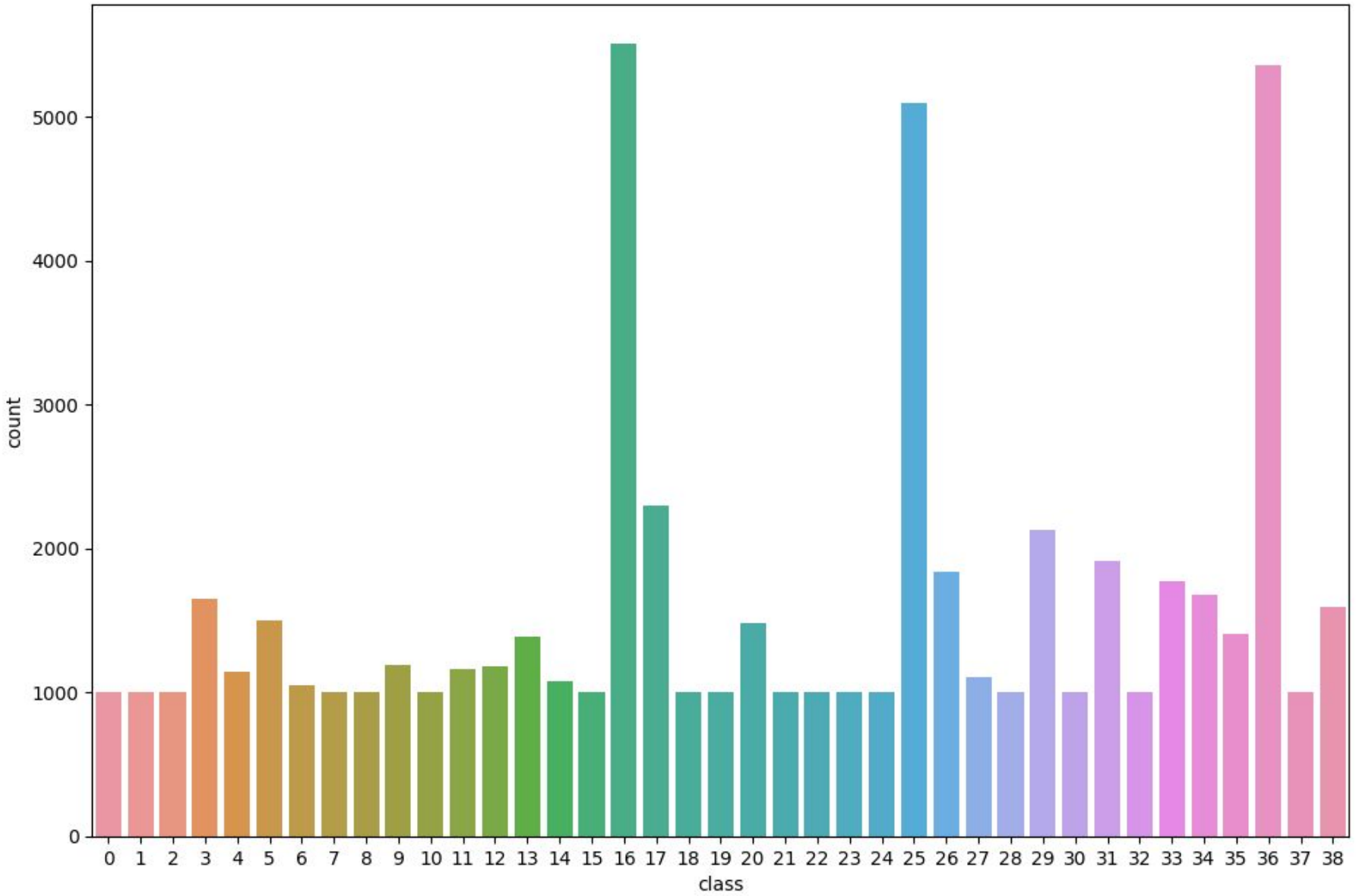
Check the balances of dataset

```
# counting no. of each label
labels, noOfImagesPerClass = torch.unique(torch.tensor(plantVillDataset.targets), return_counts=True)
```

```
classCount =  pd.DataFrame({'class': labels, 'count': noOfImagesPerClass})
```

```
plt.figure(figsize=(12, 8))
sns.barplot(x = 'class', y='count', data = classCount)
```

- The dataset has a total of 39 classes.
- Classes 16, 25, and 36 have much more samples than other classes making this an imbalanced dataset.



Reducing samples for label 16 25 36

```
# Mean of total samples for each label excluding 16, 25, and 36
meanCount = int(classCount.sort_values(by=['count'], ascending=False)[4:].mean()[1])
meanCount
```

```
1235
```

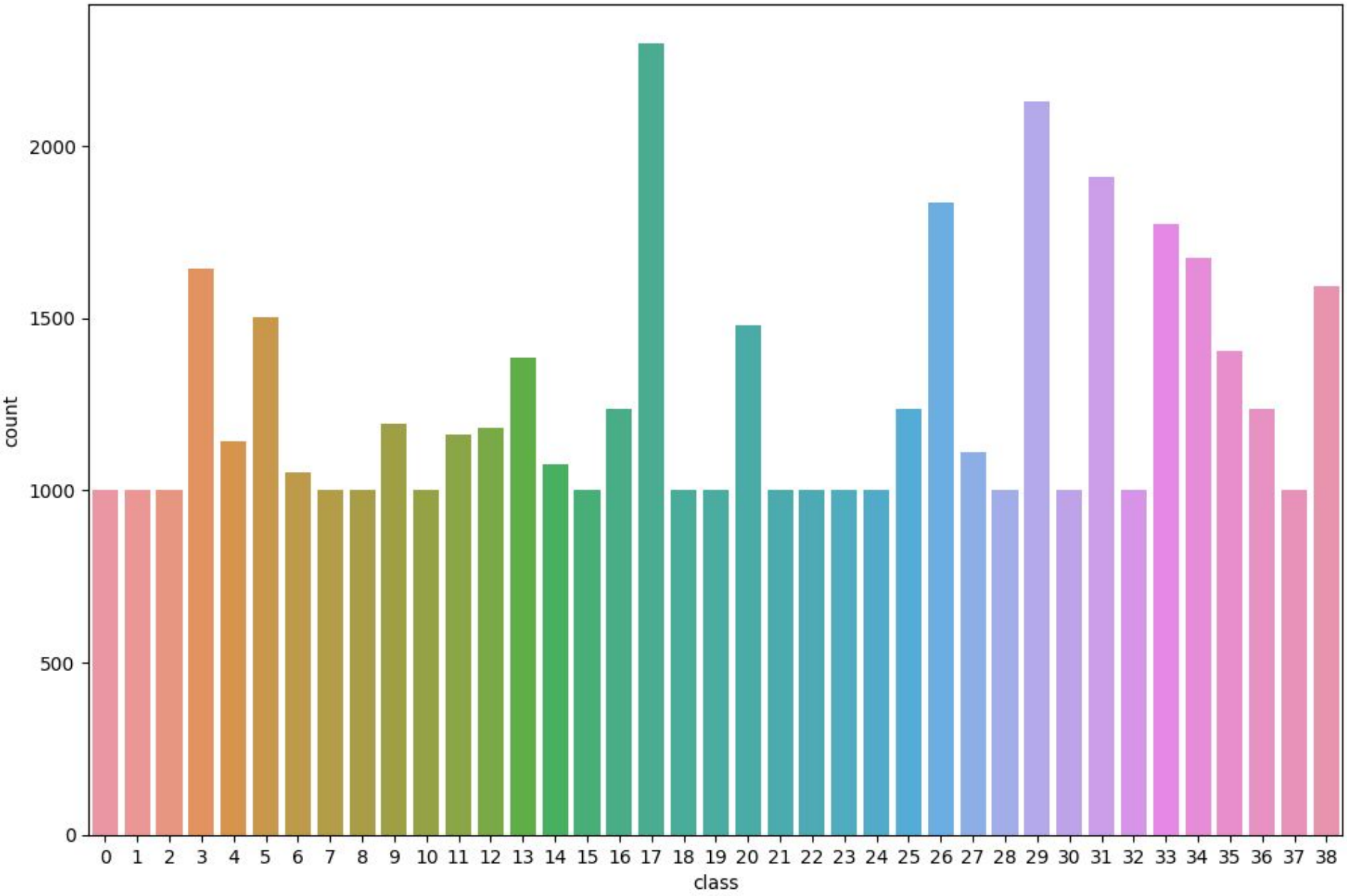
```
indices_to_keep = []
targets = torch.tensor(plantVillDataset.targets)
for label in labels:
    if label == 16 or label == 25 or label == 36:
        permIndex = torch.randperm(len(targets))
        samples = (targets == label).nonzero()
        indices_to_keep.extend(samples[:meanCount])
    else:
        indices_to_keep.extend((targets == label).nonzero())
```

```
plantVillDatasetBalanced = torch.utils.data.Subset(plantVillDataset, indices_to_keep)
```

https://discuss.pytorch.org/t/how-to-sample-images-belonging-to-particular-classes/43776/5

Result:

➔ Classes are more balanced now



3) Train, Test, Val Split

Train, Test, Val Split

```
# Total data samples after reducation
len(plantVillDatasetBalanced)
```

```
49237
```

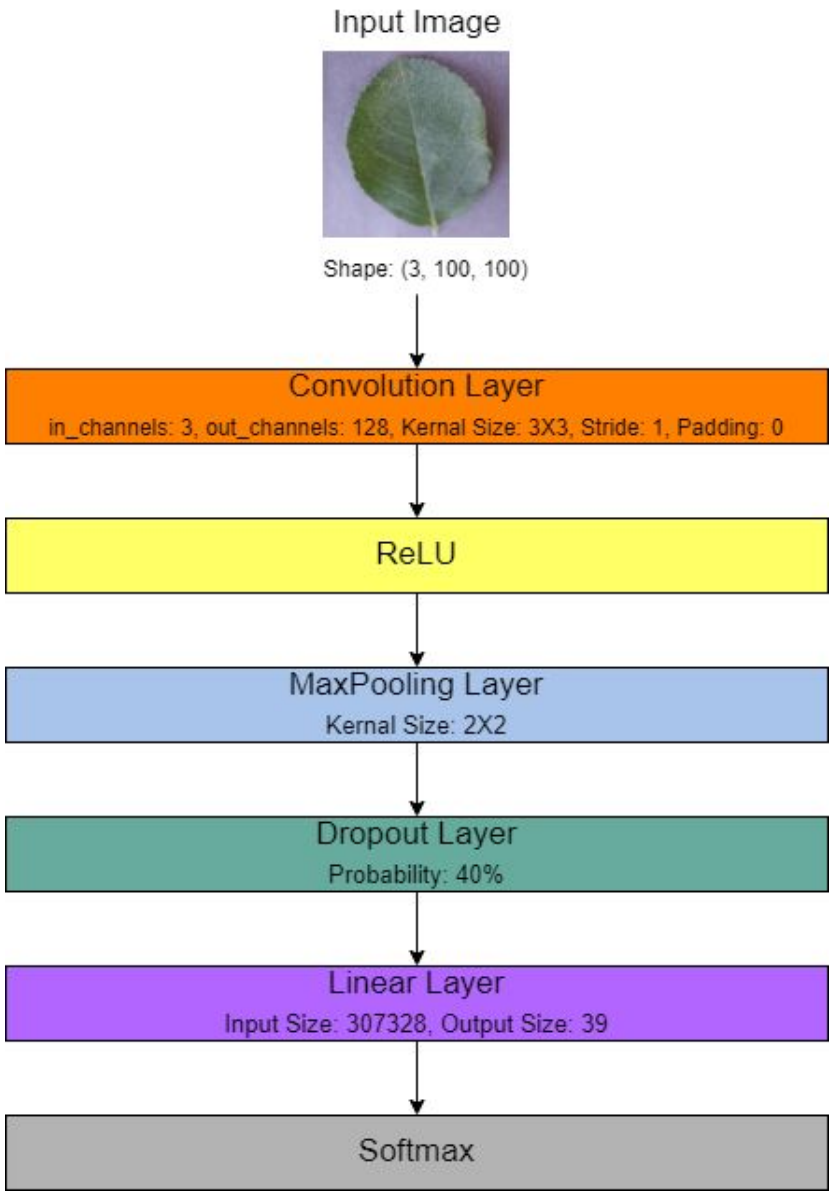
```
train_set, test_set = torch.utils.data.random_split(plantVillDatasetBalanced, [39237, 10000])
train_set, val_set = torch.utils.data.random_split(train_set, [30000, 9237])
```

```
batch_size = 50

train_loader = DataLoader(train_set, batch_size, shuffle=True, num_workers=2, pin_memory=True)
# val_loader = DataLoader(val_set, len(val_set), shuffle=True)
# test_loader = DataLoader(test_set, len(test_set), shuffle=True)
val_loader = DataLoader(val_set, batch_size, shuffle=True, num_workers=2, pin_memory=True)
test_loader = DataLoader(test_set, batch_size, shuffle=True, num_workers=2, pin_memory=True)
```

4) Creating CNN Models

Creating CNN Models

Architecture of **Model 1** :



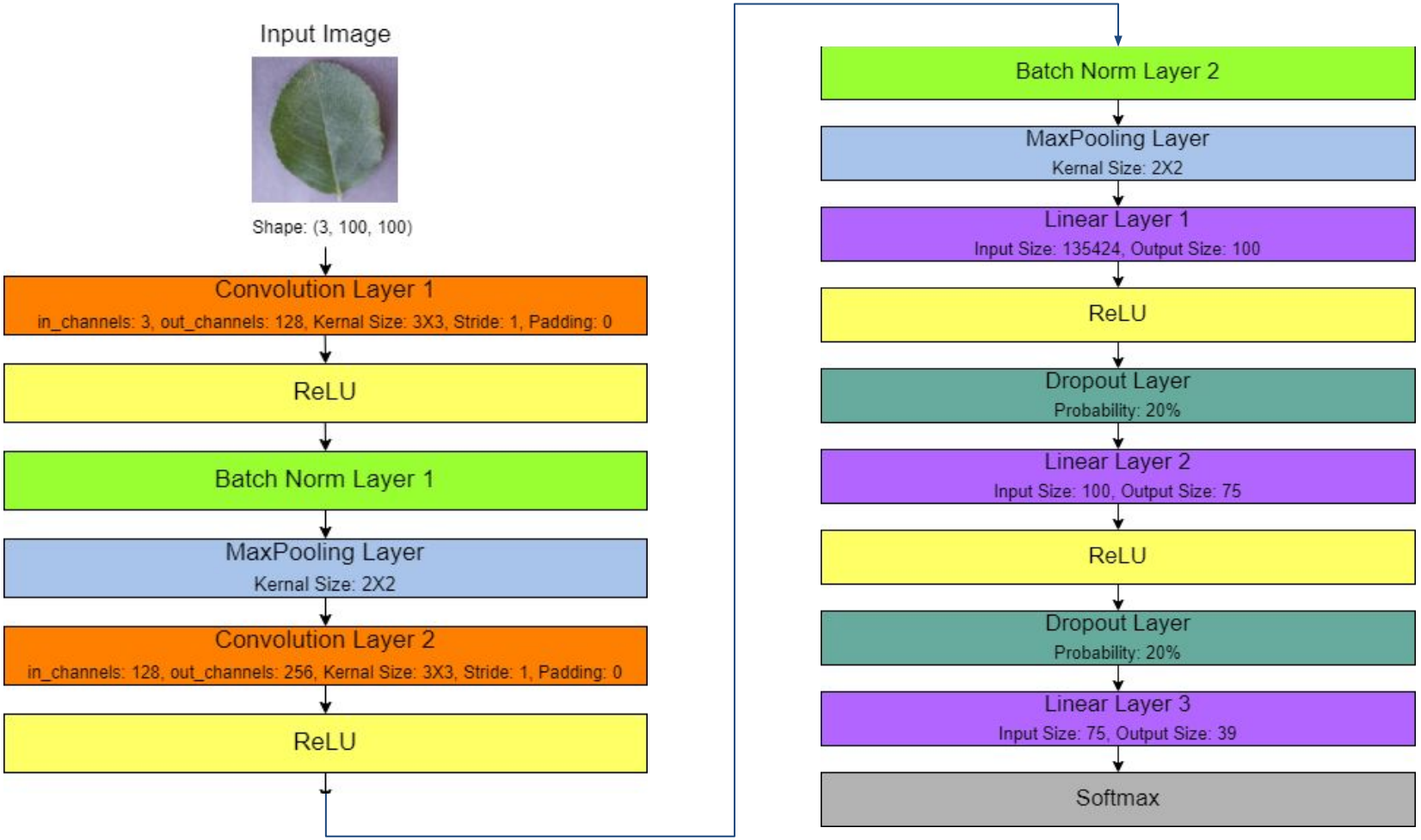
Creating CNN Models

Model 1: Base CNN Model

```
class basePlantClassifier(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(3, 128, 3, 1)
        self.maxpool = nn.MaxPool2d(2, 2)

        self.linear1 = nn.Linear(128*49*49, 39)
        self.dropoutLayer = nn.Dropout(p=0.4)
        self.relu    = nn.ReLU()

    def forward(self, image):
        out = self.conv1(image)
        out = self.relu(out)
        out = self.maxpool(out)

        out = out.reshape(-1, 128*49*49)
        out = self.dropoutLayer(out)
        out = self.linear1(out)
        return out
```

Creating CNN Models

Architecture of **Model 2** :



Creating CNN Models

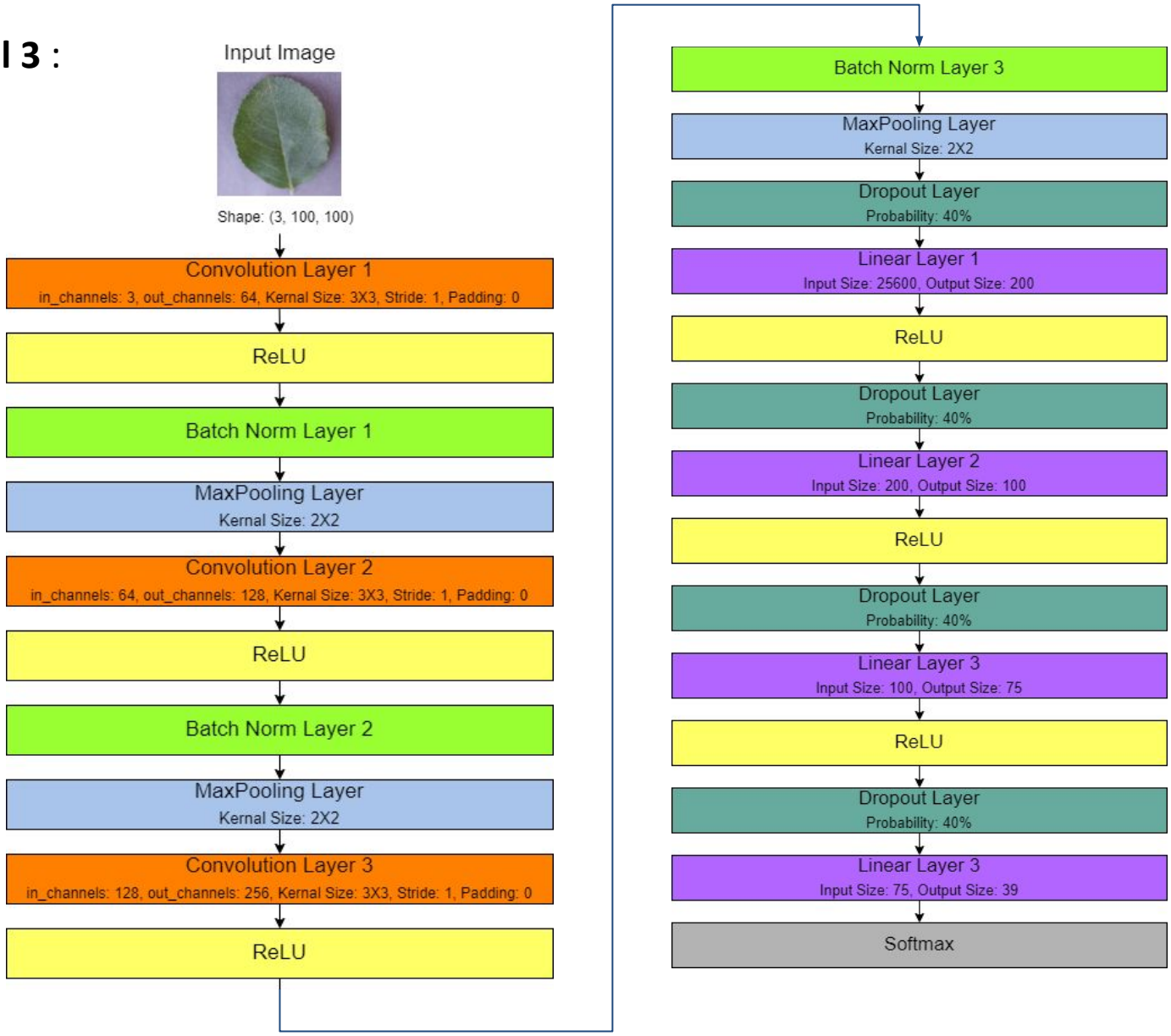
Model 2: Deeper CNN Model

```
class deeperPlantClassifier(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(3, 128, 3, 1, 0)
        self.batchNorm1 = nn.BatchNorm2d(num_features=128)
        self.maxpool = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(128, 256, 3, 1, 0)
        self.batchNorm2 = nn.BatchNorm2d(num_features=256)

        self.linear1 = nn.Linear(256*23*23, 100)
        self.linear2 = nn.Linear(100, 75)
        self.linear3 = nn.Linear(75, 39)
        self.relu    = nn.ReLU()
        self.dropoutLayer = nn.Dropout(p=0.2)
```

```
def forward(self, image):
    out = self.conv1(image)
    out = self.relu(out)
    out = self.batchNorm1(out)
    out = self.maxpool(out)

    out = self.conv2(out)
    out = self.relu(out)
    out = self.batchNorm2(out)
    out = self.maxpool(out)

    out = out.reshape(-1, 256*23*23)
    out = self.linear1(out)
    out = self.relu(out)
    out = self.dropoutLayer(out)

    out = self.linear2(out)
    out = self.relu(out)
    out = self.dropoutLayer(out)

    out = self.linear3(out)
    return out
```

Creating CNN Models

Architecture of **Model 3** :



Creating CNN Models

Model 3: More Deeper CNN Model

```
class moreDeeperPlantClassifier(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(3, 64, 3, 1, 0)
        self.batchNorm1 = nn.BatchNorm2d(num_features=64)
        self.maxpool = nn.MaxPool2d(2, 2)

        self.conv2 = nn.Conv2d(64, 128, 3, 1, 0)
        self.batchNorm2 = nn.BatchNorm2d(num_features=128)

        self.conv3 = nn.Conv2d(128, 256, 3, 1, 0)
        self.batchNorm3 = nn.BatchNorm2d(num_features=256)



        self.linear1 = nn.Linear(256*10*10, 200)
        self.linear2 = nn.Linear(200, 100)
        self.linear3 = nn.Linear(100, 75)
        self.linear4 = nn.Linear(75, 39)
        self.relu    = nn.ReLU()
        self.dropoutLayer = nn.Dropout(p=0.4)
```

```
def forward(self, image):
    out = self.conv1(image)
    out = self.relu(out)
    out = self.batchNorm1(out)
    out = self.maxpool(out)

    out = self.conv2(out)
    out = self.relu(out)
    out = self.batchNorm2(out)
    out = self.maxpool(out)

    out = self.conv3(out)
    out = self.relu(out)
    out = self.batchNorm3(out)
    out = self.maxpool(out)

    out = out.reshape(-1, 256*10*10)
    out = self.dropoutLayer(out)

    out = self.linear1(out)
    out = self.relu(out)
    out = self.dropoutLayer(out)

    out = self.linear2(out)
    out = self.relu(out)
    out = self.dropoutLayer(out)

    out = self.linear3(out)
    out = self.relu(out)
    out = self.dropoutLayer(out)

    out = self.linear4(out)
    return out
```

Model Summary

	Model 1	Model 2	Model 3
Total No. of Parameters	**11,989,415**	**13,852,559**	**5,522,551**
Total No. of Layers	**2**	**5**	**7**
Total No. of Conv. Layers	**1**	**2**	**3**
Total No. of Linear Layers	**1**	**3**	**4**

5) Training CNN Models

Training CNN Models

Basic Steps:

- ➔ Predict Class
- ➔ Calculate Loss
- ➔ Update Training Weights
- ➔ Repeat steps 1-3 until
  - ◆ until of epoch
  - ◆ early stopping due to overfitting

Validation:

- ➔ Monitor model performance during training
- ➔ For best val loss, model weights saved

Train Model

```
model1 = basePlantClassifier().to(device)
```

```
print(model1)
```

```
basePlantClassifier(
  (conv1): Conv2d(3, 128, kernel_size=(3, 3), stride=(1, 1))
  (maxpool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (linear1): Linear(in_features=307328, out_features=39, bias=True)
  (dropoutLayer): Dropout(p=0.4, inplace=False)
  (relu): ReLU()
)
```

Loss Function and Optimizer

```
J_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model1.parameters(), lr=0.001)
```

Train Model(cont…)

```
class EarlyStopping():
    def __init__(self, patience, threshold):
        self.patience      = 5
        self.threshold     = 0.1
        self.prev_val_loss = None
        self.patienceCount = 0

    def _checkPatience(self,):
        if self.patienceCount == self.patience:
            return True
        else:
            self.patienceCount += 1
            return False

    def checkCondition(self, val_loss):
        if self.prev_val_loss == None:
            self.prev_val_loss = val_loss
        elif val_loss - self.prev_val_loss > self.threshold:
            return self._checkPatience()

        self.patienceCount = 0
        return False
```

Train Model(cont…)

```
def train(model, optimizer, J_fn, filepath, epochs=10, patience = 5, threshold = 0.1):

    earlyStopping = EarlyStopping(patience, threshold)

    val_old_loss = float("Inf")
    train_losses = []
    train_accs   = []
    avg_train_losses_per_epoch = []
    avg_train_accs_per_epoch   = []
    val_losses   = []
    val_accs     = []
    training_time = []
    training_time_per_epoch = []

    for e in range(epochs):
        total_corr       = 0
        total_loss       = 0
        total_train_loss = 0
        total_time_epoch = 0
        for b, (image, label) in enumerate(train_loader):
            start_time = time.time()

            image = image.to(device)
            label = label.to(device)

            yhat = model(image)
            train_loss = J_fn(yhat, label)
            total_train_loss += train_loss

            train_losses.append(train_loss)
            predicted = torch.max(yhat, 1)[1]
            batch_corr = (predicted == label).sum()
            total_corr += batch_corr

            optimizer.zero_grad()
            train_loss.backward()
            optimizer.step()

            train_acc = (total_corr * 100) / (batch_size * (b + 1))
            train_accs.append(train_acc)
```

Train Model(cont…)

```
    total_time = time.time() - start_time
    total_time_epoch += total_time
    training_time.append(total_time)

    if (b+1) % 50 == 0:
        print(f"Epoch: {e + 1} - Batch: {b + 1} - Train Loss: {train_loss:.2f} - Train Acc: {train_acc:.2f} - Total Time: {total_time:.2f}s")
        # print("Loss Gradient for first Conv Layer: ", torch.mean(model.conv1.weight.grad) )

avg_train_time = total_time_epoch / len(train_loader)
training_time_per_epoch.append(avg_train_time)

avg_train_loss = total_train_loss / len(train_loader)
avg_train_acc  = (total_corr * 100) / len(train_set)

avg_train_losses_per_epoch.append(avg_train_loss)
avg_train_accs_per_epoch.append(avg_train_acc)

print(f"++++++End of Epoch {e + 1} ++++++ Avg Train Loss: {avg_train_loss:.2f} - Avg Train Acc: {avg_train_acc:.2f} - Avg Train Time: {avg_train_time:.
```

Train Model(cont…)

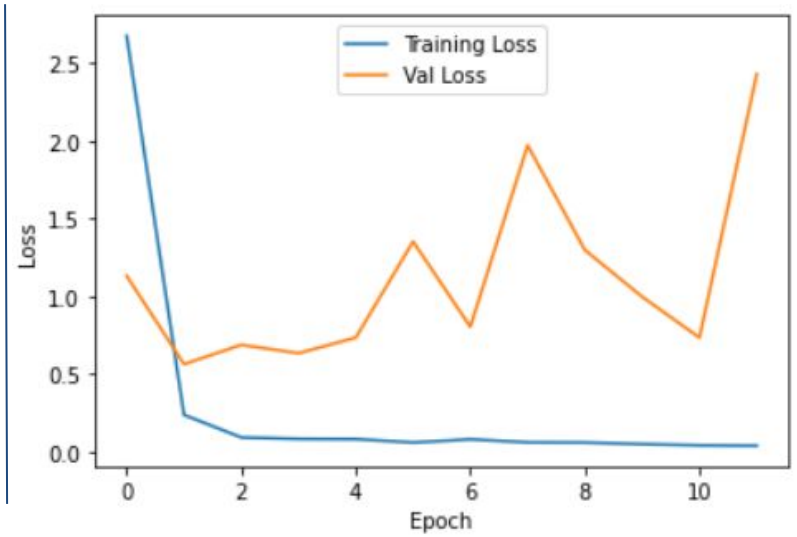
```
    with torch.no_grad():
        val_corr       = 0
        total_val_loss = 0
        for val_image, val_label in val_loader:
            val_image = val_image.to(device)
            val_label = val_label.to(device)
            val_yhat = model(val_image)
            val_loss = J_fn(val_yhat, val_label)
            total_val_loss += val_loss
            val_predicted = torch.max(val_yhat, 1)[1]
            val_corr += (val_predicted == val_label).sum()
        val_acc = (val_corr * 100) / len(val_set)
        avg_val_loss = total_val_loss / len(val_loader)
        val_accs.append(val_acc)
        val_losses.append(val_loss)

        if avg_val_loss < val_old_loss:
            val_old_loss = avg_val_loss
            saveObject = {
                'epoch': e + 1,
                'batch': b + 1,
                'train_loss': avg_train_loss,
                'train_acc': avg_train_acc,
                'val_loss': avg_val_loss,
                'val_acc': val_acc,
                'model_state_dict': model.state_dict(),
                'optimizer_state_dict': optimizer.state_dict(),
            }
            torch.save(saveObject, filepath)

    if earlyStopping.checkCondition(avg_val_loss):
        break

    print(f"++++++Validation++++++ Val Loss: {avg_val_loss:.2f} - Val Acc: {val_acc:.2f}")

train_losses               = torch.Tensor(train_losses).cpu()
avg_train_losses_per_epoch = torch.Tensor(avg_train_losses_per_epoch).cpu()
train_accs                 = torch.Tensor(train_accs).cpu()
avg_train_accs_per_epoch   = torch.Tensor(avg_train_accs_per_epoch).cpu()
val_losses                 = torch.Tensor(val_losses).cpu()
val_accs                   = torch.Tensor(val_accs).cpu()


return train_losses, avg_train_losses_per_epoch, train_accs, avg_train_accs_per_epoch, training_time, training_time_per_epoch, val_losses, val_accs
```

Training Observation

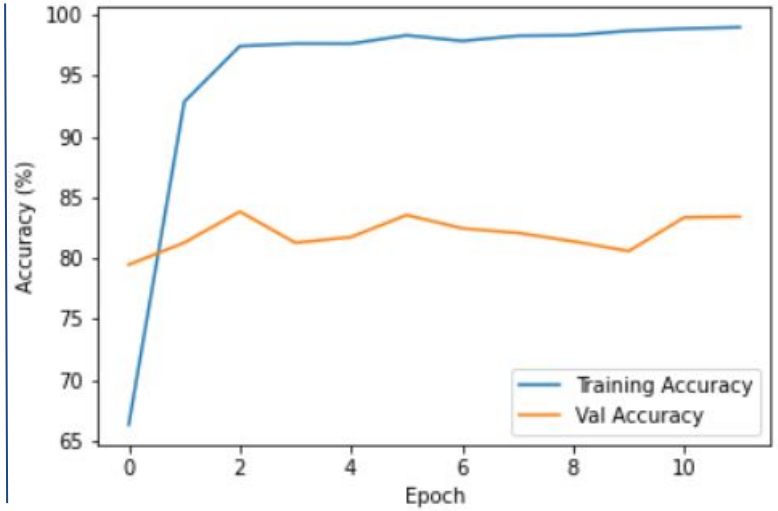
- **Model 1 and 3 overfitting early**
  - **Model 1 after 5 epochs**
  - **Model 3 after 15 epochs**
- **Higher value of dropout used which helped significantly with overfitting issue**

Plotting loss, accuracy and training time: **Model 1**

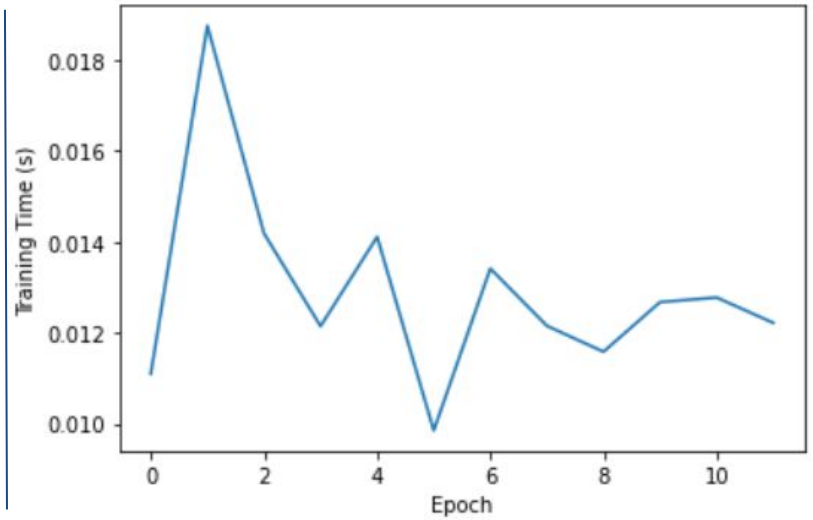
Loss Plot



Accuracy Plot



Training Time Plot



Training loss decreases rapidly
Validation loss increases quickly
Accuracy increases rapidly - network is learning fast
Training stopped by earl stopping callback

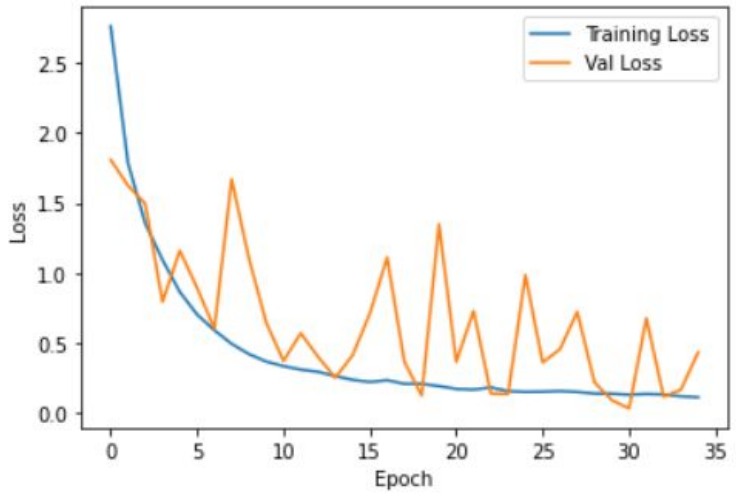
Plot Saved and Loaded

```
saveObject = {
    'train_losses': train_losses,
    'avg_train_losses_per_epoch': avg_train_losses_per_epoch,
    'train_accs': train_accs,
    'avg_train_accs_per_epoch': avg_train_accs_per_epoch,
    'training_time': training_time,
    'training_time_per_epoch': training_time_per_epoch,
    'val_losses': val_losses,
    'val_accs': val_accs
}

torch.save(saveObject, './ModelParameter/Model 1/baseModeltrainingHist.pt')
```

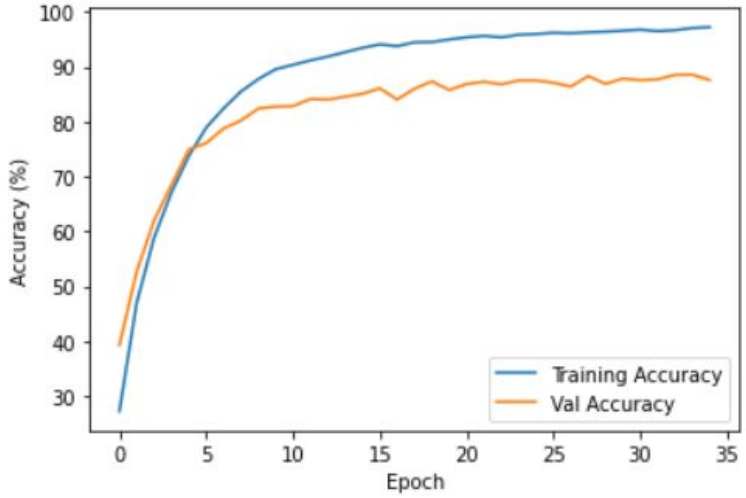
```
loadObject1 = torch.load('./ModelParameter/Model 1/baseModeltrainingHist.pt')
```

Plotting loss, accuracy and training time: **Model 2**

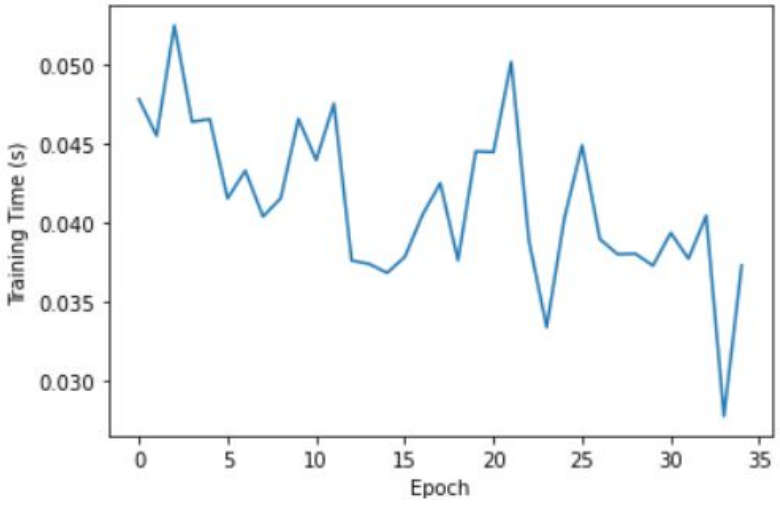
Loss Plot



Accuracy Plot



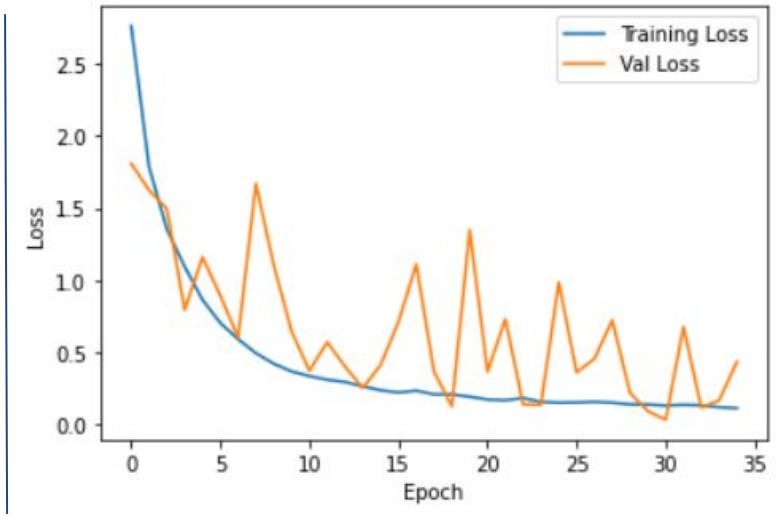
Training Time Plot



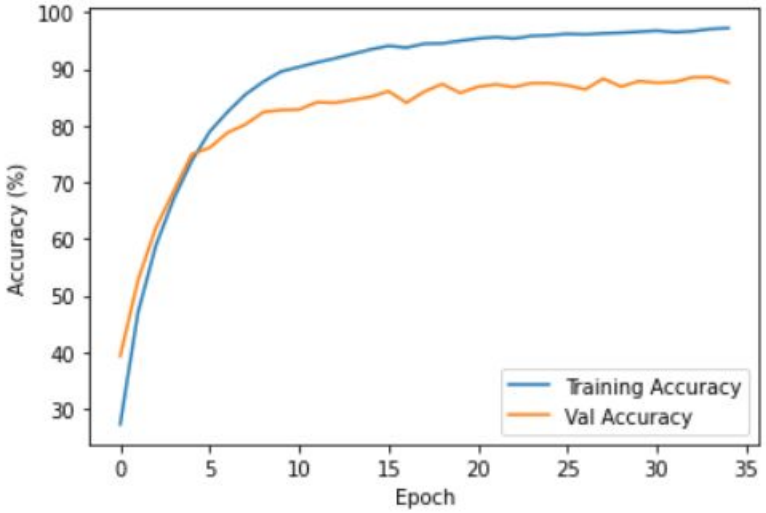
Validation loss increases
(but not very stably)
Validation accuracy rises too.

Plotting loss, accuracy and training time: **Model 3**

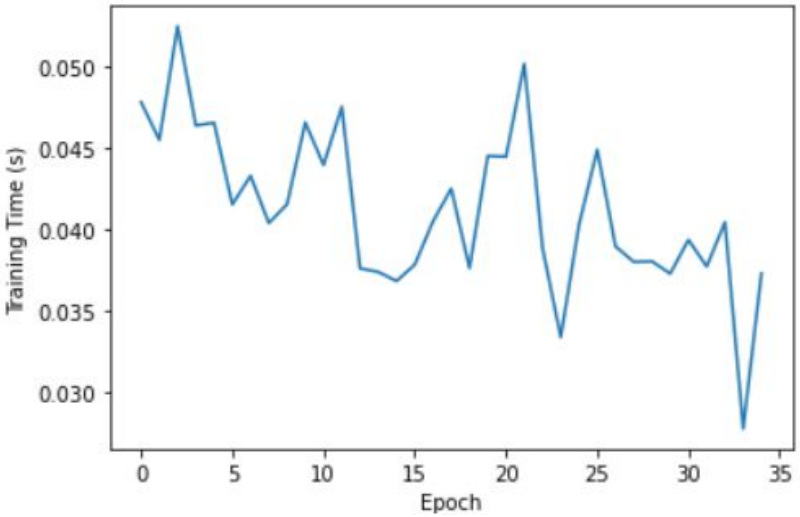
Loss Plot



Accuracy Plot



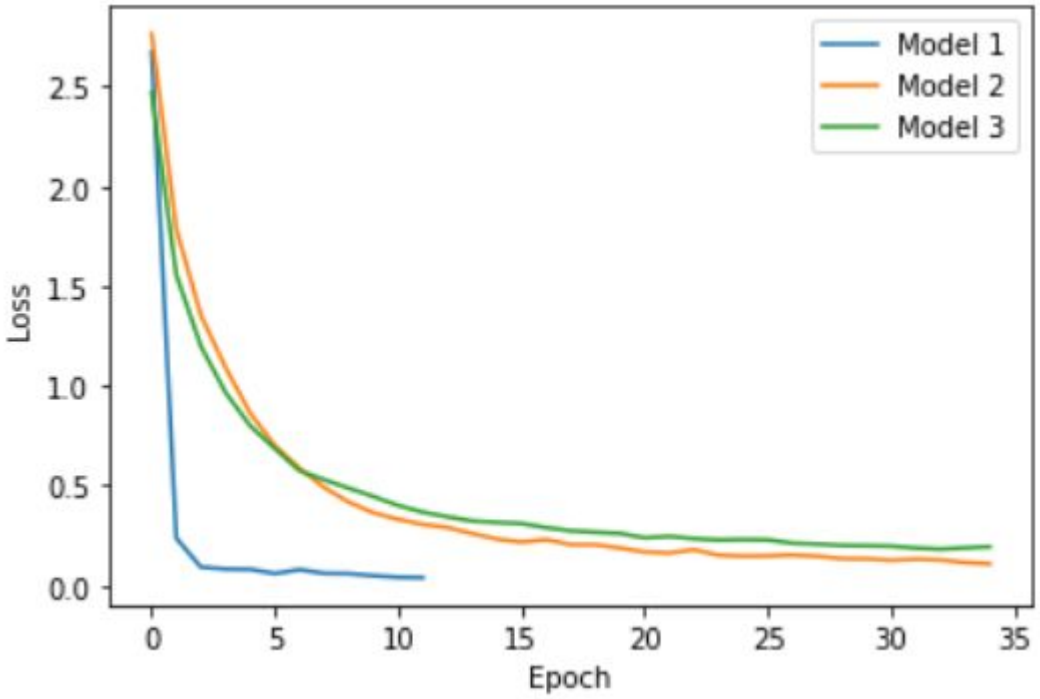
Training Time Plot



Compare 3 Models: Train loss

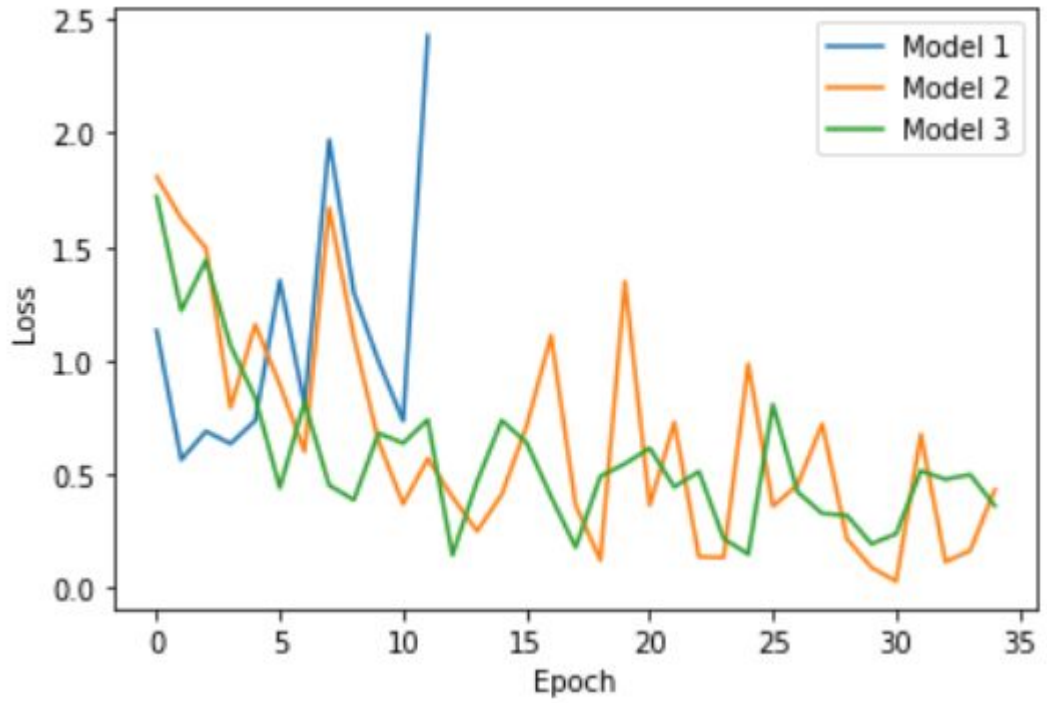
```
plt.plot(torch.Tensor(avg_train_losses_per_epoch).cpu())
plt.plot(torch.Tensor(avg_train_losses_per_epoch2).cpu())
plt.plot(torch.Tensor(avg_train_losses_per_epoch3).cpu())
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(["Model 1", "Model 2", "Model 3"])
```

Train Loss



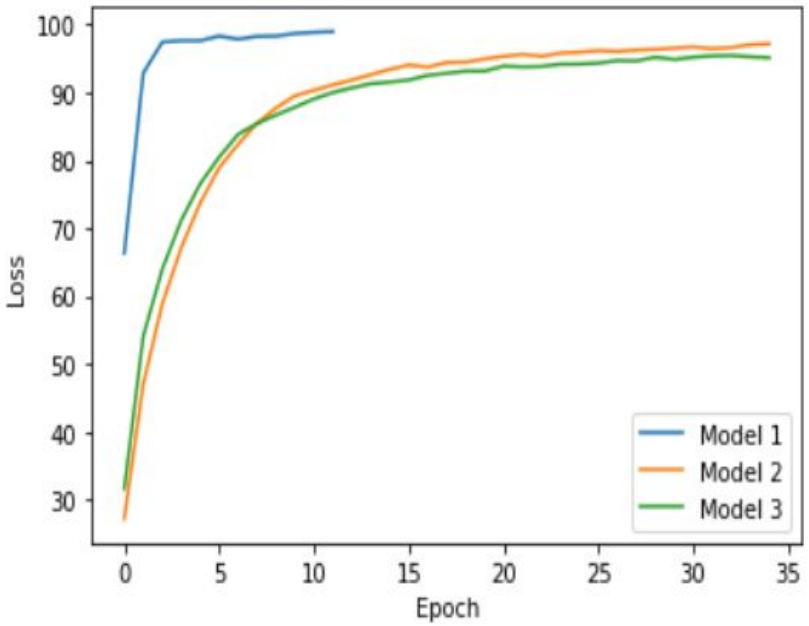
Compare 3 Models: Validation loss

Validation Loss

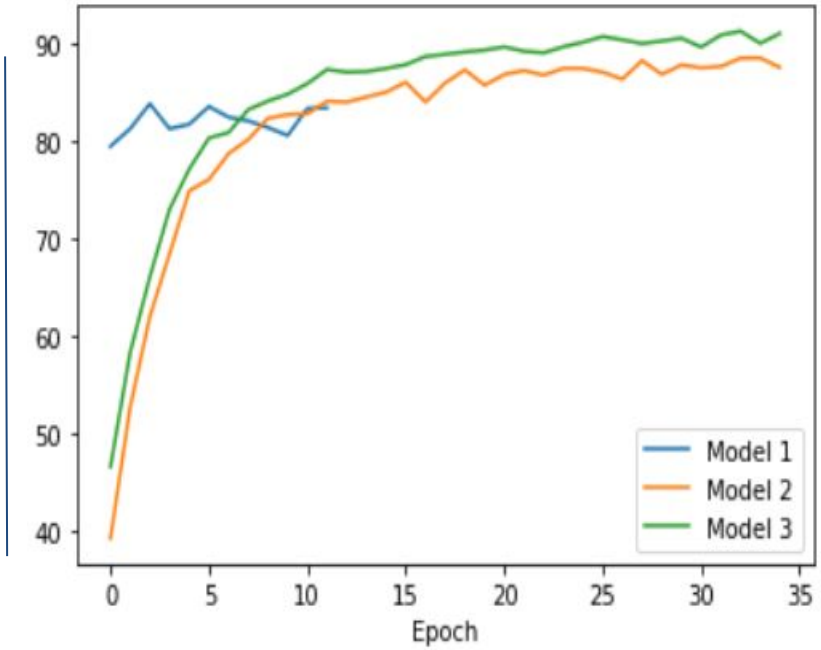


Compare 3 Models: Train accuracy and Validation accuracy

Training Accuracy



Validation Accuracy



Model 1 stop early
Good accuracy for training data.
Validation accuracy, model 3 perform the best
Training accuracy, model 2 perform the best

6) Testing/Inference

Test

```
checkpoint1 = torch.load(filepath)
checkpoint2 = torch.load(filepath2)
checkpoint3 = torch.load(filepath3)
```

```
bestModel1 = basePlantClassifier().to(device)
bestModel2 = deeperPlantClassifier().to(device)
bestModel3 = moreDeeperPlantClassifier().to(device)
```

```
bestModel1.load_state_dict(checkpoint1['model_state_dict'])
bestModel2.load_state_dict(checkpoint2['model_state_dict'])
bestModel3.load_state_dict(checkpoint3['model_state_dict'])
```

```
<All keys matched successfully>
```

```
bestModel1.eval()
bestModel2.eval()
bestModel3.eval()
```

Test (cont…)

```python
def testModel(model):
    with torch.no_grad():
        correct = 0
        acc     = 0
        total   = 0
        allPrediction       = []
        correctLabels       = []
        misclassifiedLabels = []

        for images, targets in test_loader:
            images  = images.to(device)
            targets = targets.to(device)

            yhat    = model(images)
            predicted = torch.max(yhat, 1)[1]
            batch_corr = (predicted == targets).sum()
            correct   += batch_corr
            total     += targets.shape[0]

            for i in range(len(predicted)):
                if predicted[i] != targets[i]:
                    misclassifiedLabels.append({
                        'correctLabel': targets[i],
                        'predicted': predicted[i],
                        'image': images[i]
                    })

            allPrediction.append(predicted)
            correctLabels.append(targets)

        acc = 100 * correct / total
        print(f"Accuracy: {acc:.2f}")

        return allPrediction, correctLabels, misclassifiedLabels
```

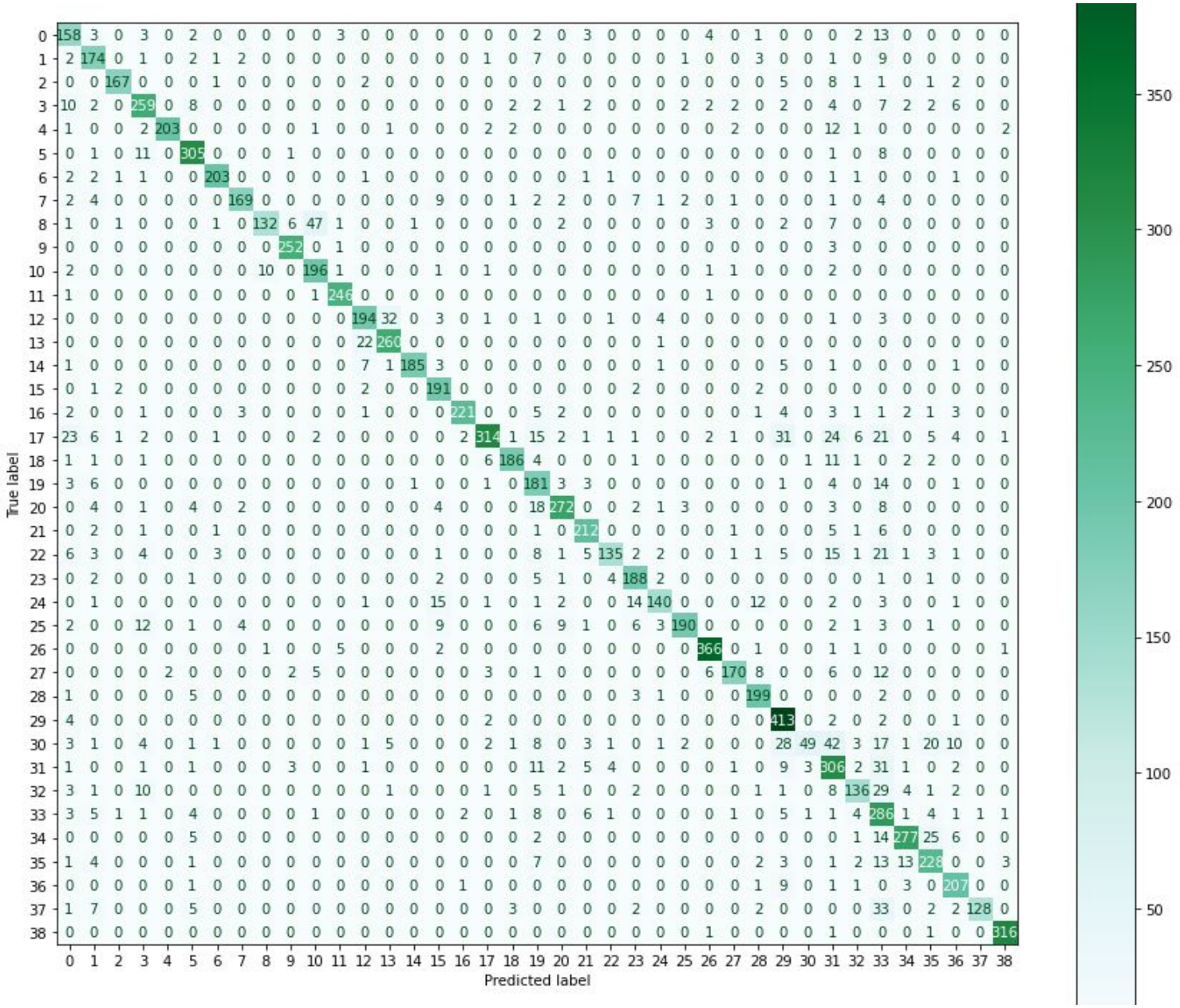
Accuracy

Model	Accuracy
Model 1	**84.14**
Model 2	**92.24**
Model 3	**96.66**

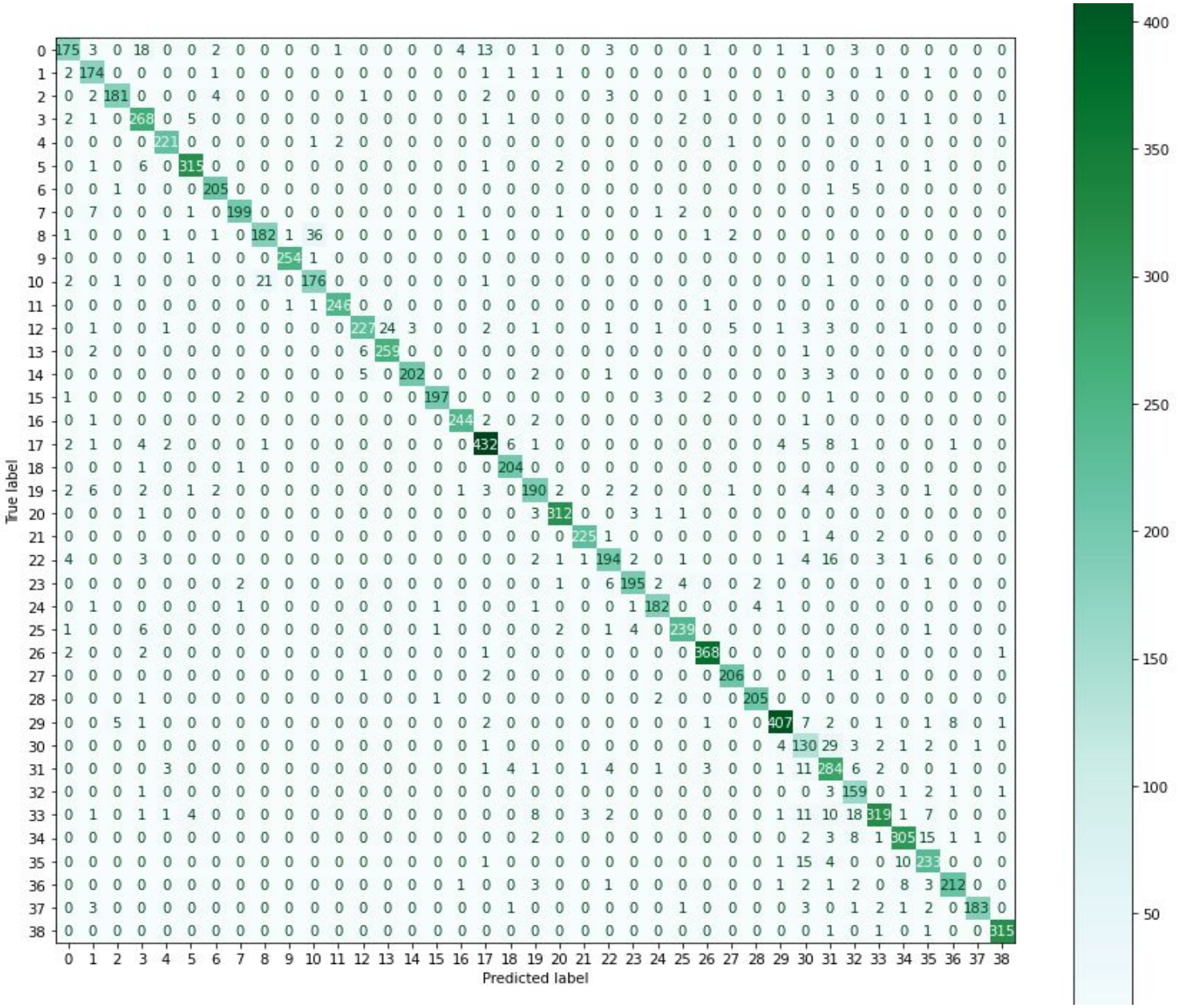
Total Number of Misclassification per Label

Model	Total Number of Misclassifications per Label
Model 1	**1586**
Model 2	**776**
Model 3	**334**

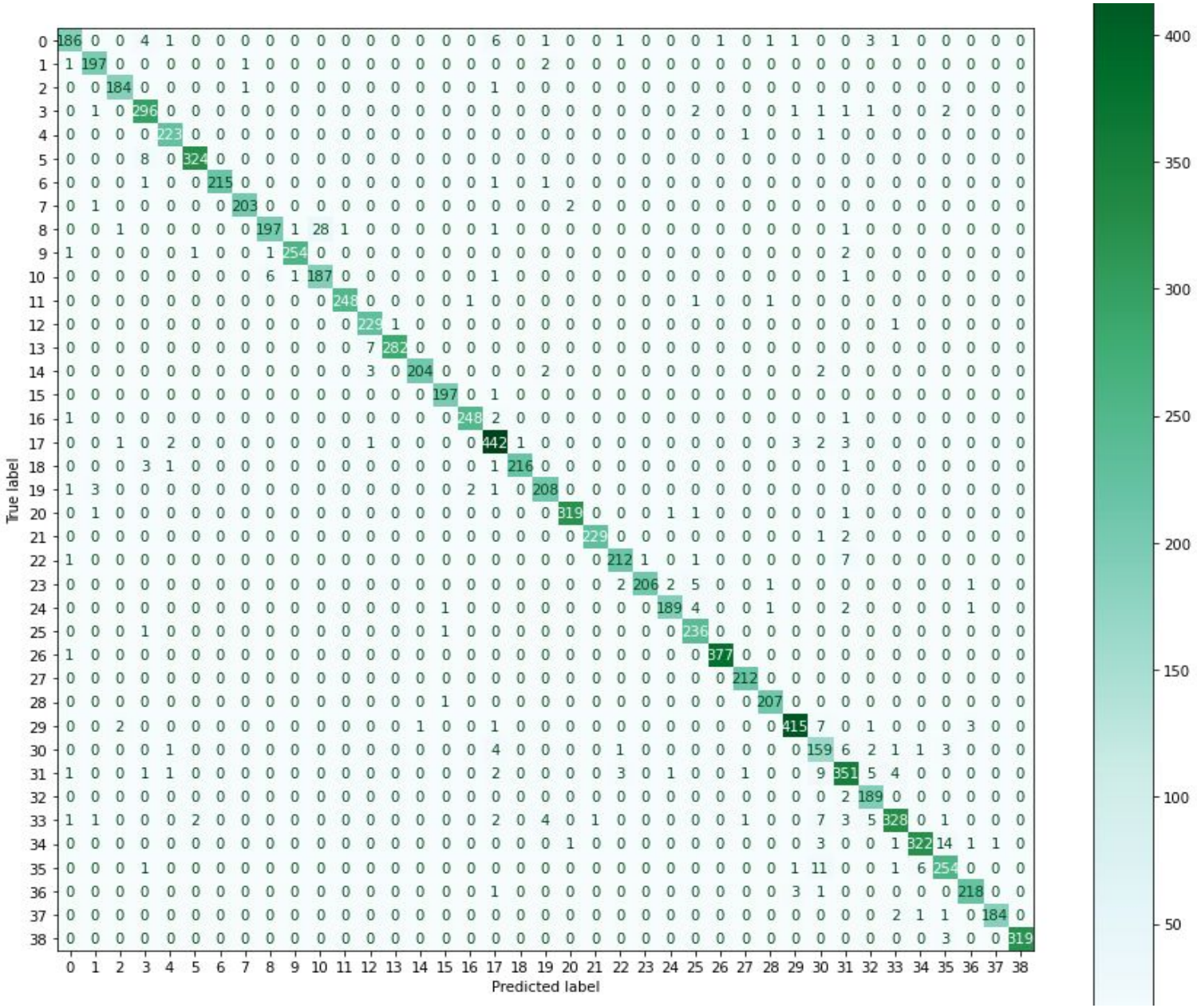
Confusion Matrix: **Model 1**



Confusion Matrix: **Model 2**

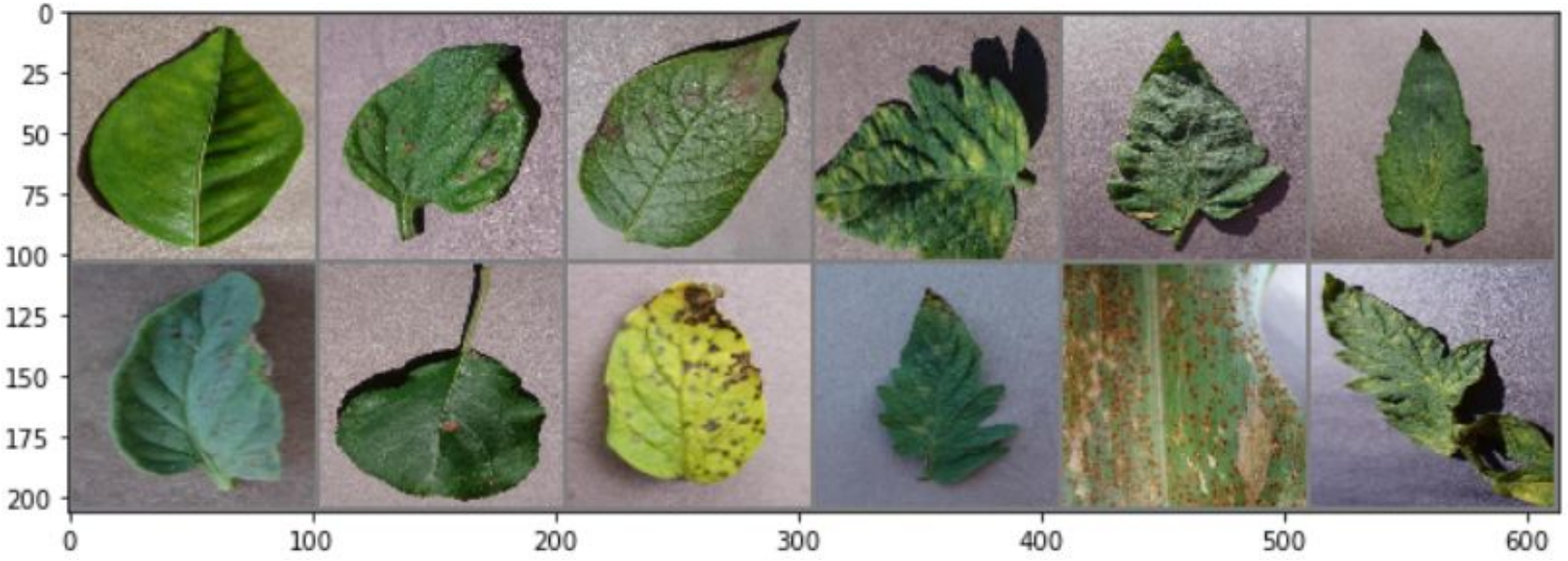


Confusion Matrix: **Model 3**



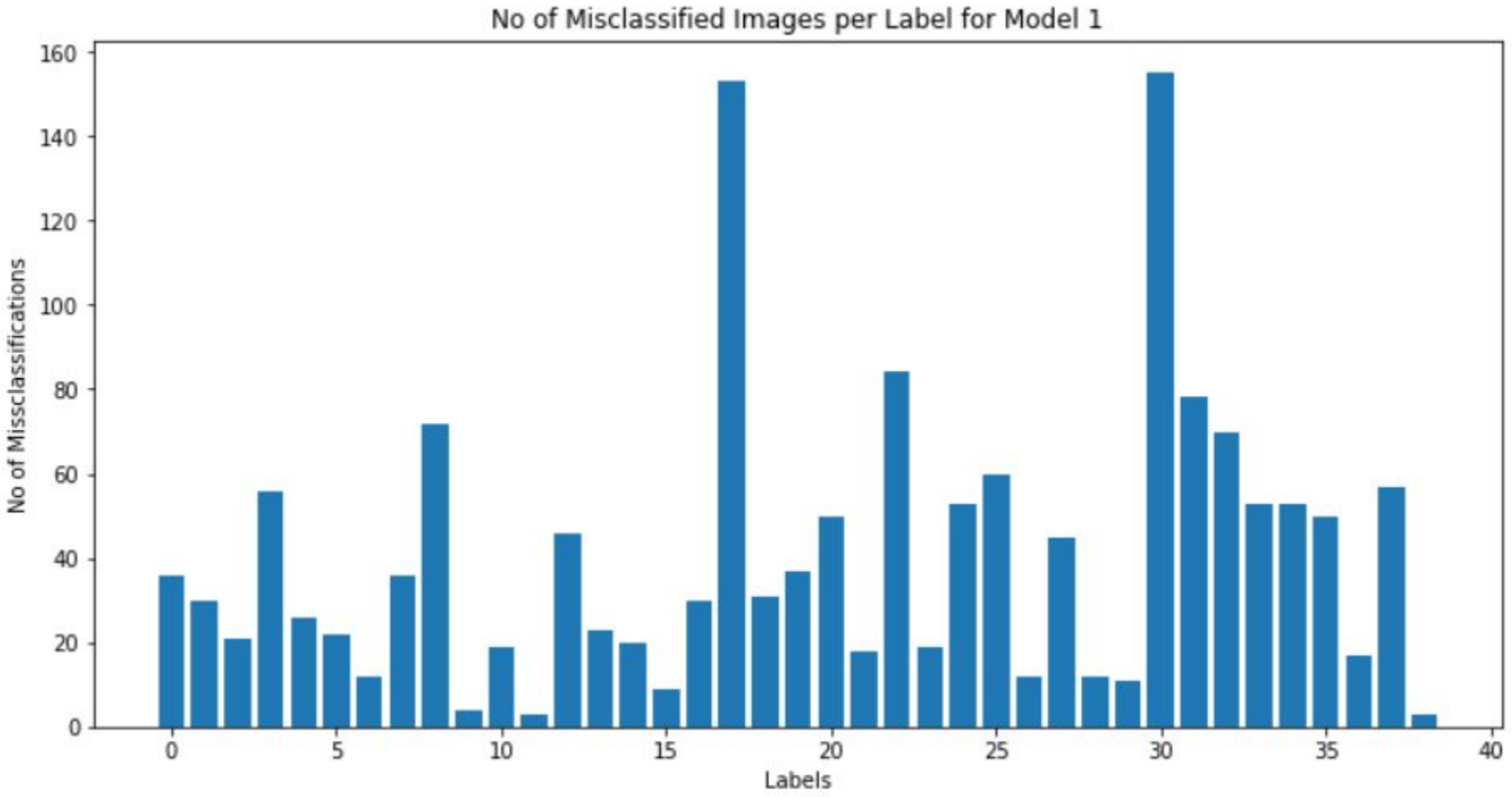
6.1) Examine misclassified images

Examine misclassified images: **Model 1**

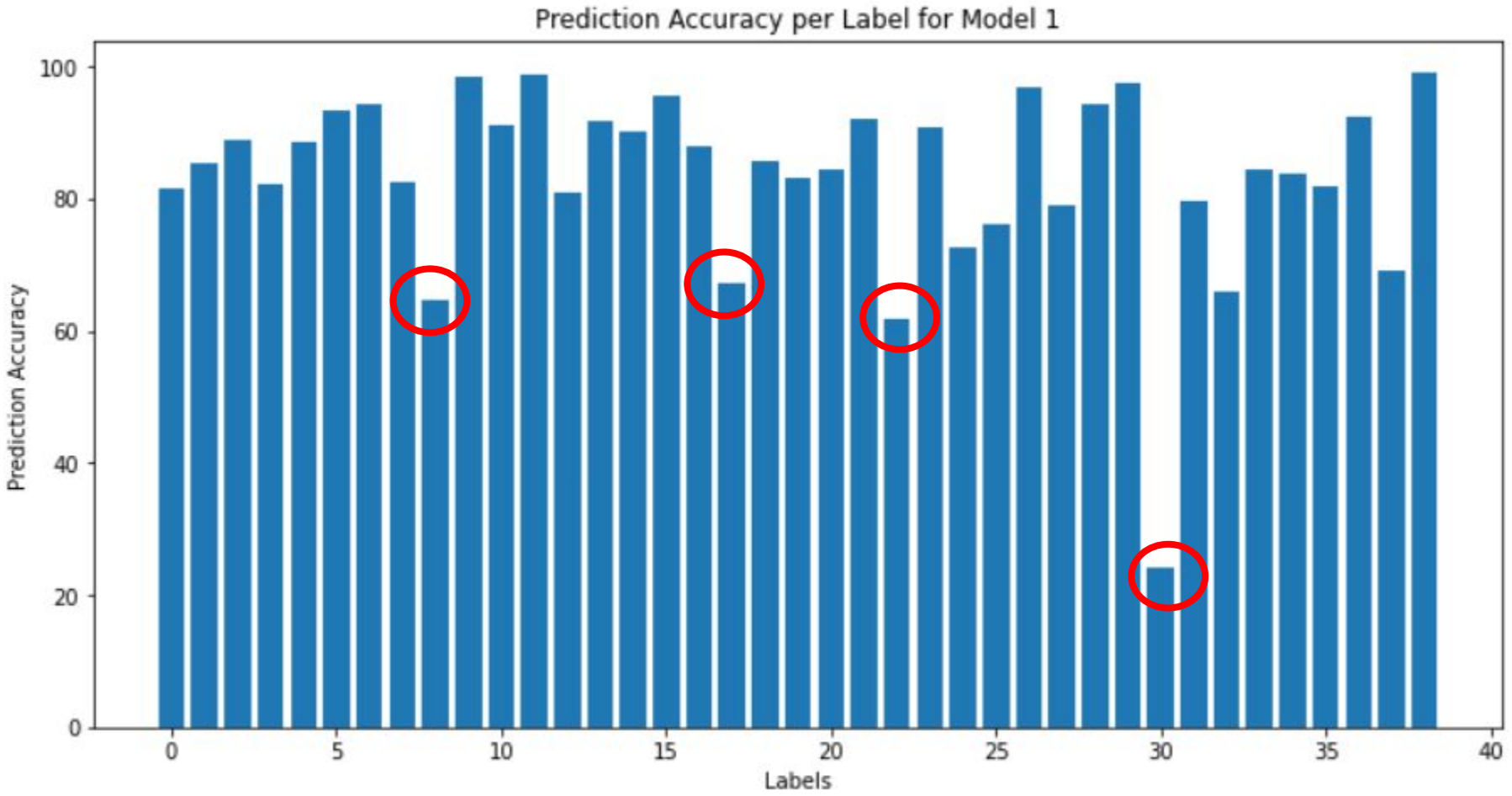


Showing few misclassified images

No of misclassified images per label for **Model 1**



Prediction accuracy per label for **Model 1**

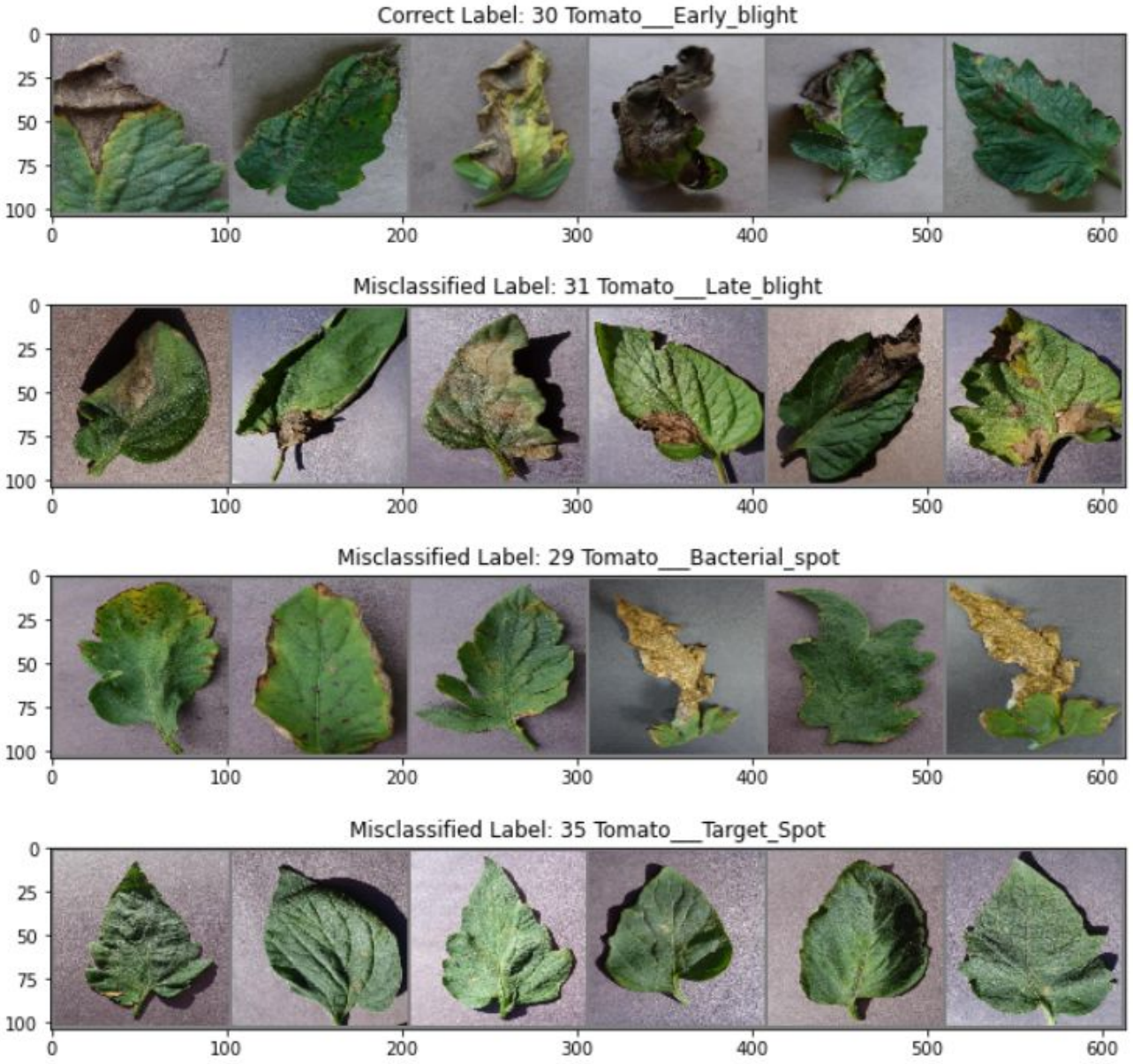


Note: Low accuracy prediction on labels 8, 17, 22, 30, and 37.

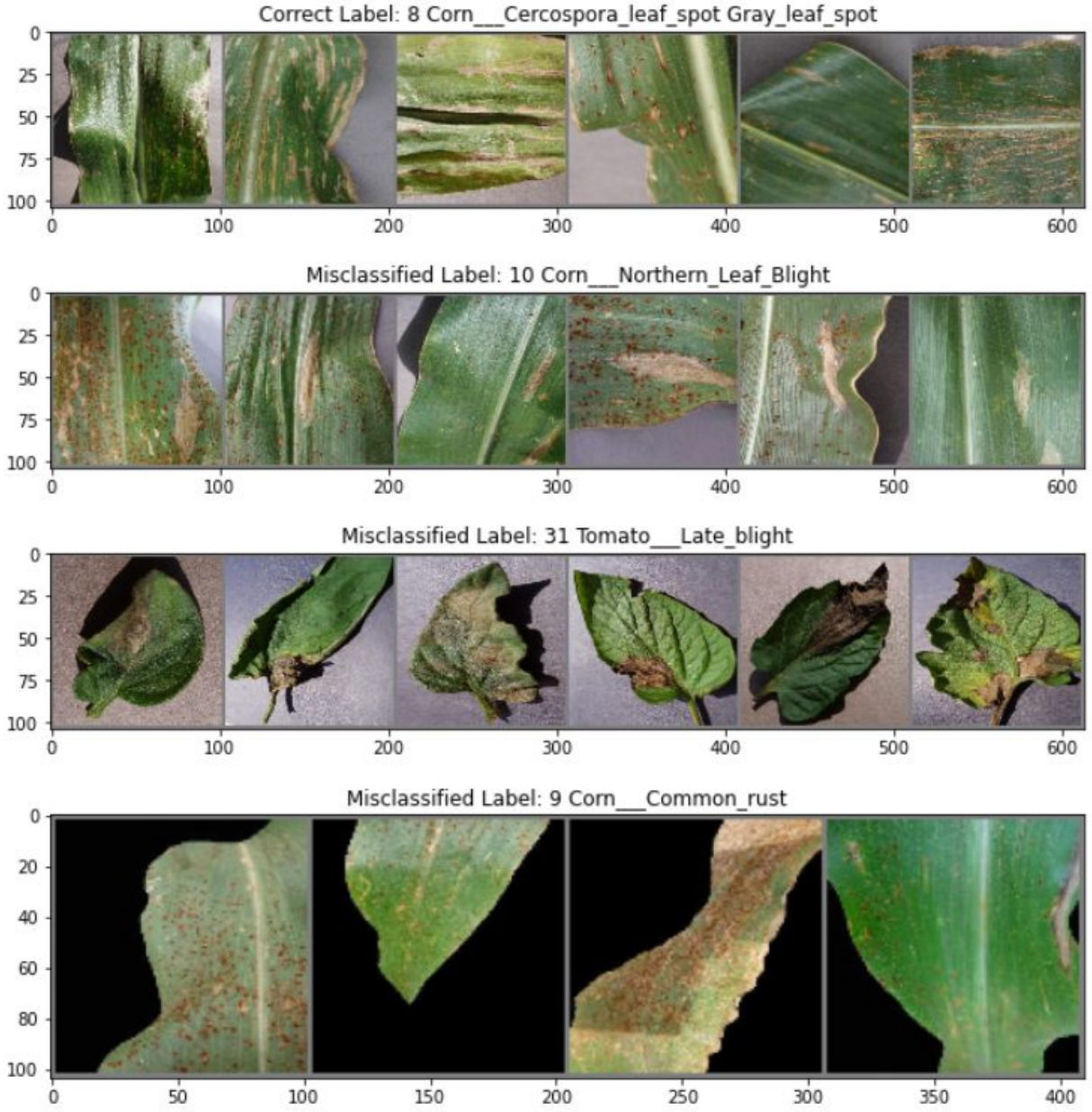
Top 3 mislabelled count: **Model 1**

```
For label 8, Corn___Cercospora_leaf_spot Gray_leaf_spot, top three misclassified labels are:
---> Mislabeled: 10 Corn___Northern_Leaf_Blight
Count: 47
---> Mislabeled: 31 Tomato___Late_blight
Count: 7
---> Mislabeled: 9 Corn___Common_rust
Count: 6

For label 17, Peach___Bacterial_spot, top three misclassified labels are:
---> Mislabeled: 29 Tomato___Bacterial_spot
Count: 31
---> Mislabeled: 31 Tomato___Late_blight
Count: 24
---> Mislabeled: 0 Apple___Apple_scab
Count: 23

For label 22, Potato___Late_blight, top three misclassified labels are:
---> Mislabeled: 33 Tomato___Septoria_leaf_spot
Count: 21
---> Mislabeled: 31 Tomato___Late_blight
Count: 15
---> Mislabeled: 19 Pepper,_bell___Bacterial_spot
Count: 8

For label 30, Tomato___Early_blight, top three misclassified labels are:
---> Mislabeled: 31 Tomato___Late_blight
Count: 42
---> Mislabeled: 29 Tomato___Bacterial_spot
Count: 28
---> Mislabeled: 35 Tomato___Target_Spot
Count: 20

For label 37, Tomato___Tomato_mosaic_virus, top three misclassified labels are:
---> Mislabeled: 33 Tomato___Septoria_leaf_spot
Count: 33
---> Mislabeled: 1 Apple___Black_rot
Count: 7
---> Mislabeled: 5 Blueberry___healthy
Count: 5
```

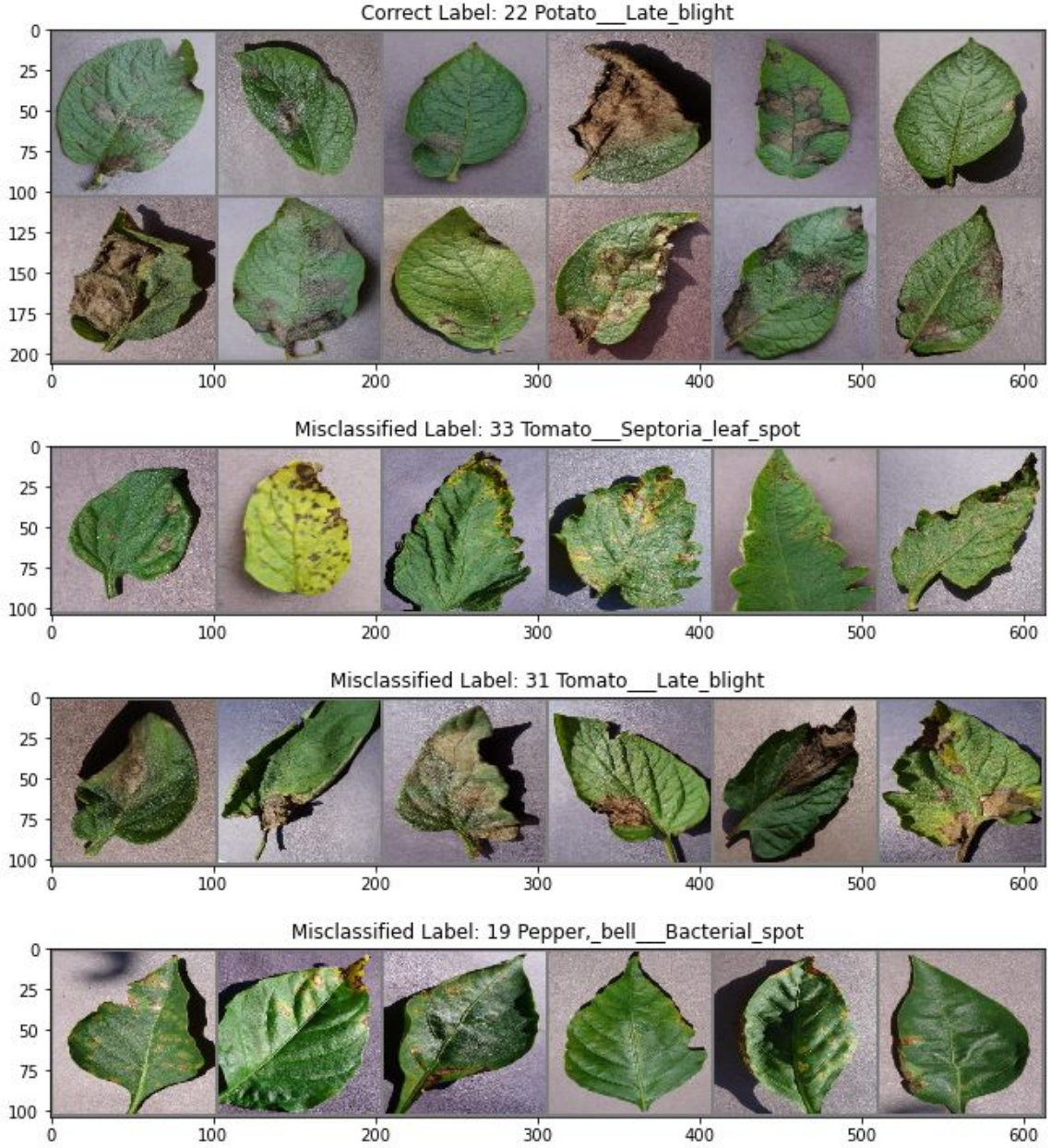
Showing some misclassified images for label 30



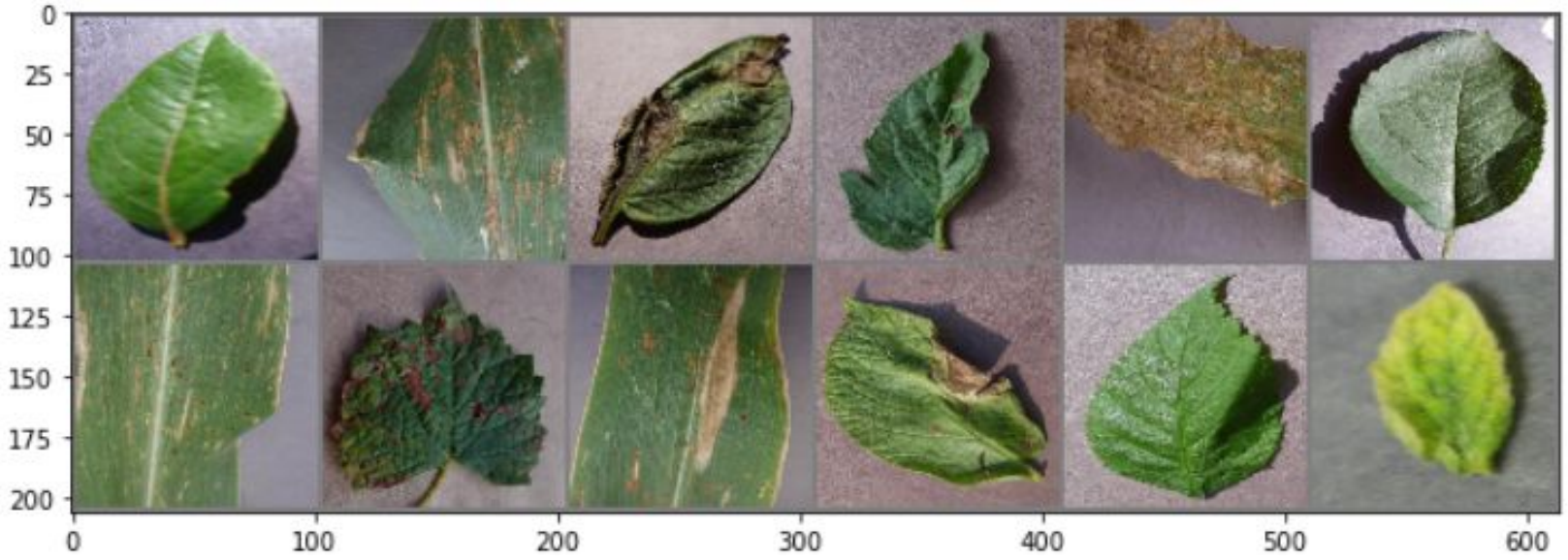
Showing some misclassified images for label 8



Showing some misclassified images for label 22

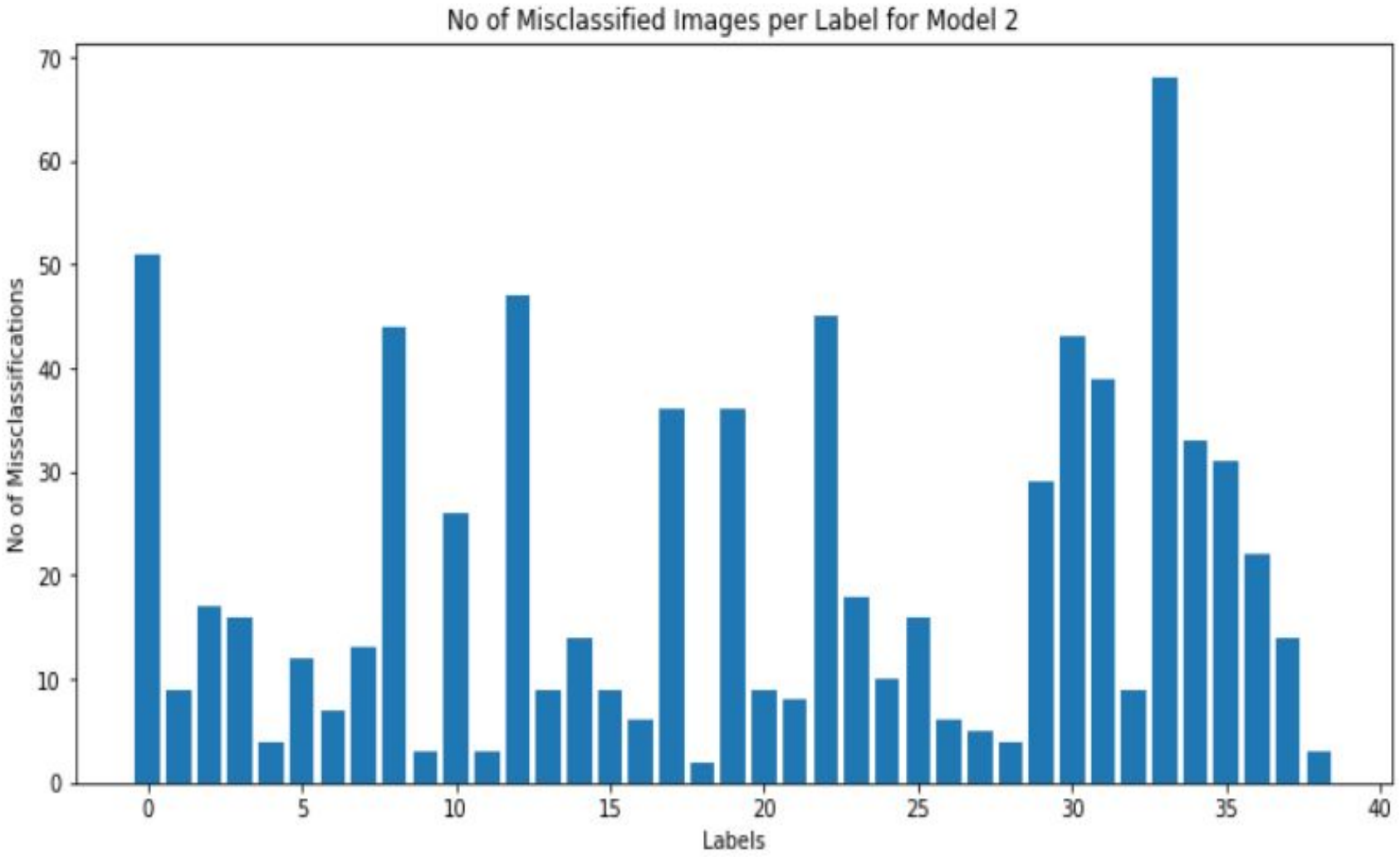


Examine misclassified images: **Model 2**

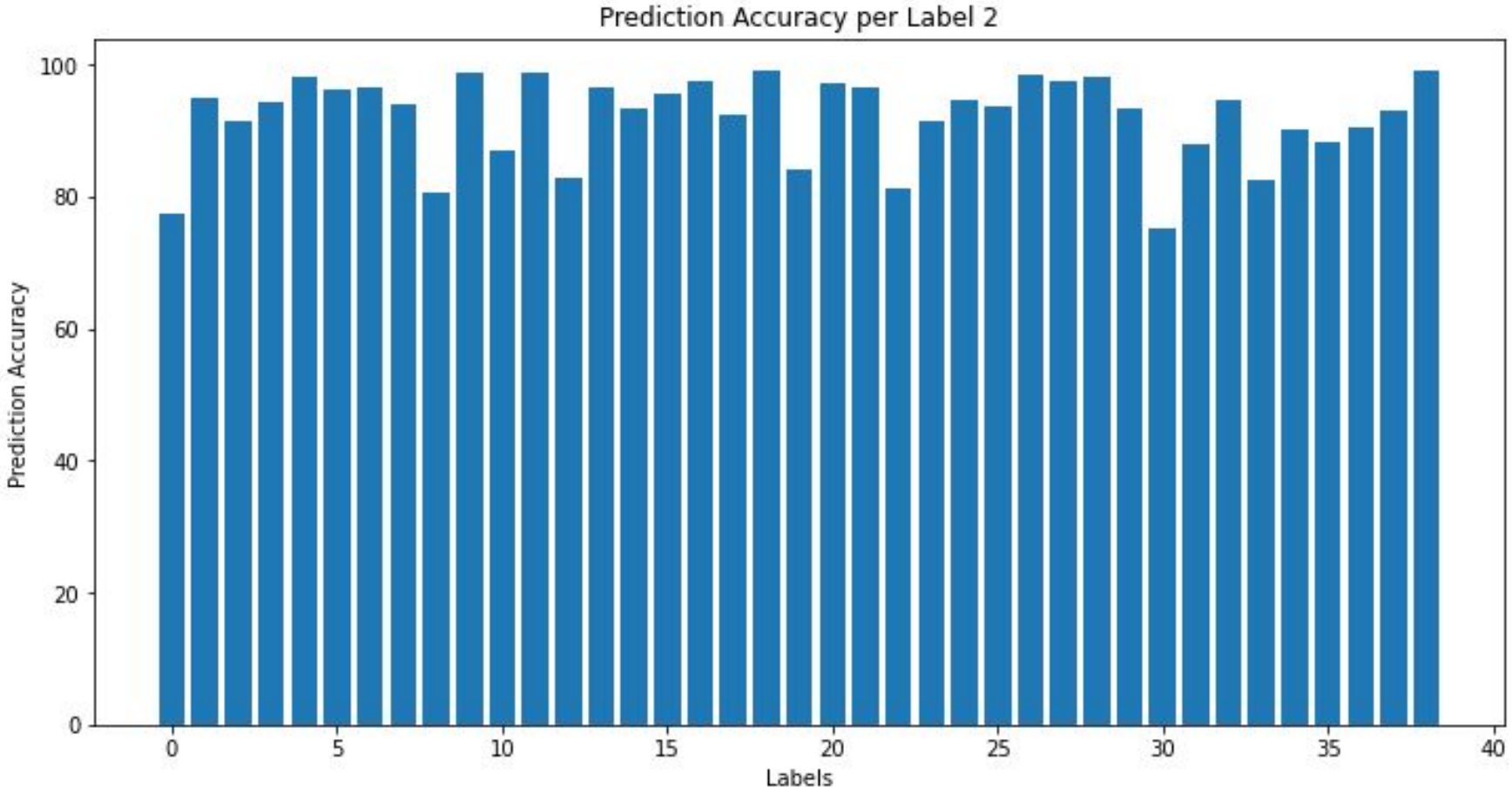


Showing few misclassified images

No of misclassified images per label for **Model 2**



Prediction accuracy per label for **Model 2**

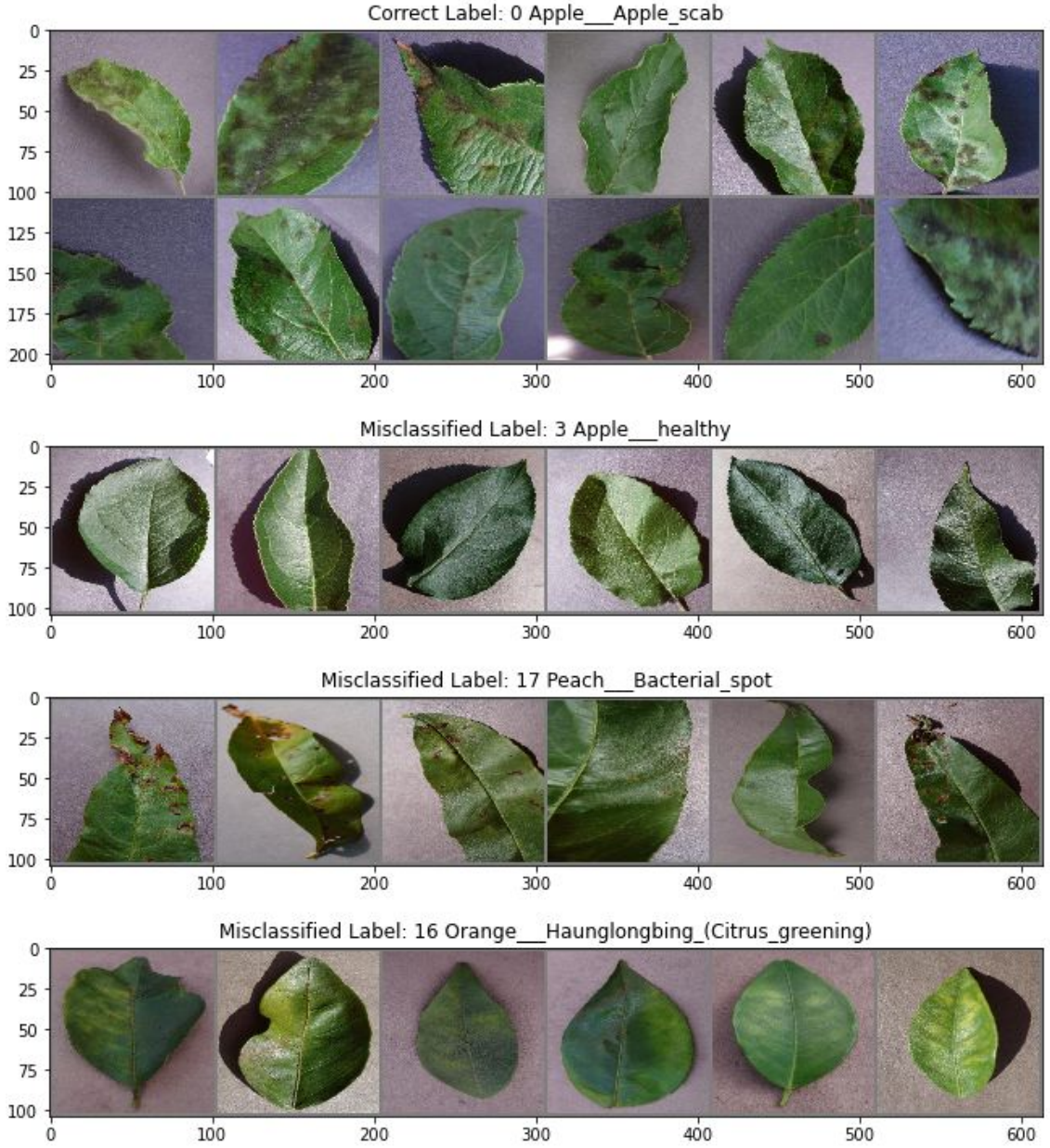


Note: Prediction accuracy on labels 0 and 30 lower compared to other labels

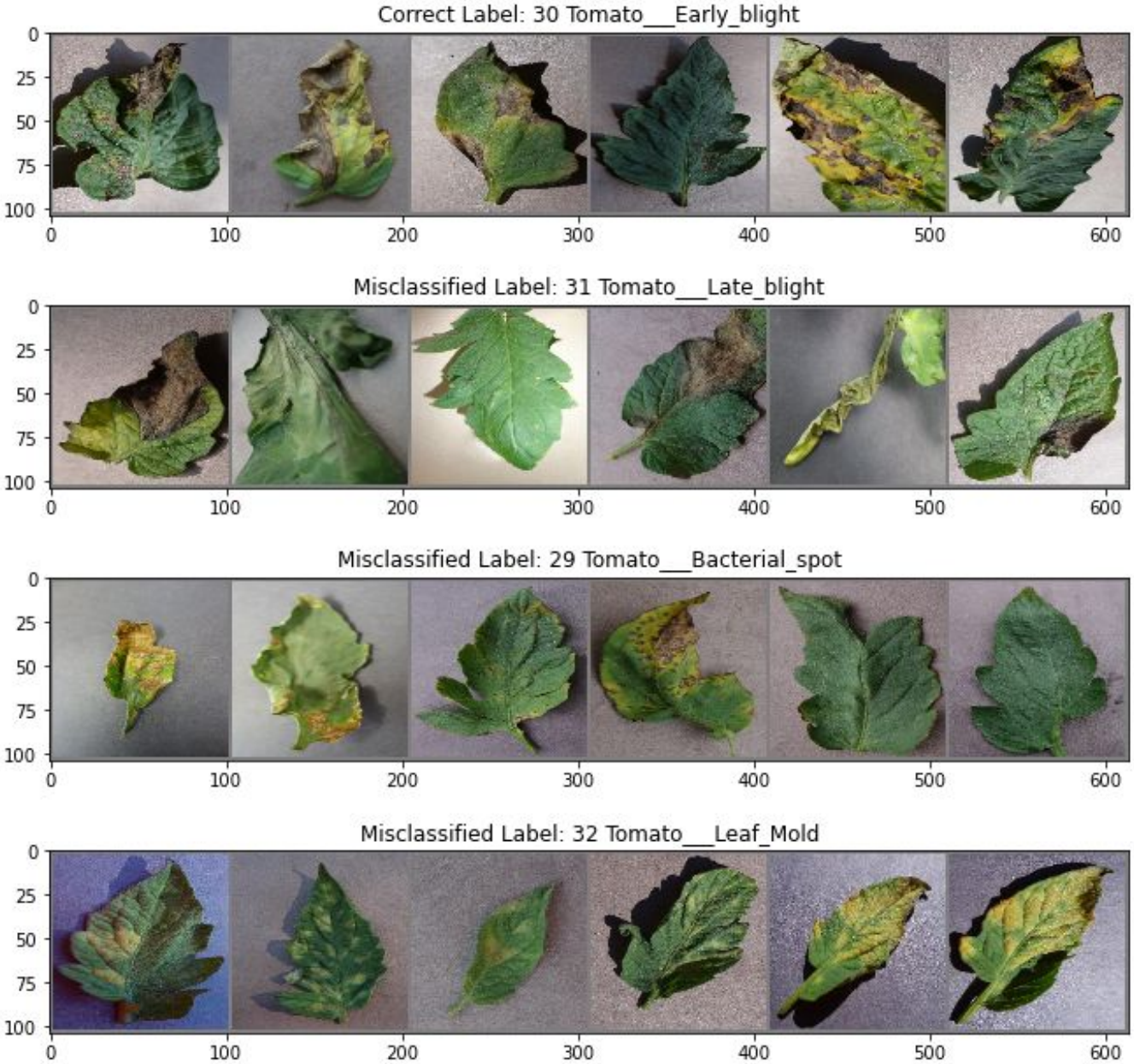
Top 3 mislabelled count: **Model 2**

```
For label 0, Apple___Apple_scab, top three misclassified labels are:
---> Mislabeled: 3 Apple___healthy
Count: 18
---> Mislabeled: 17 Peach___Bacterial_spot
Count: 13
---> Mislabeled: 16 Orange___Haunglongbing_(Citrus_greening)
Count: 4

For label 30, Tomato___Early_blight, top three misclassified labels are:
---> Mislabeled: 31 Tomato___Late_blight
Count: 29
---> Mislabeled: 29 Tomato___Bacterial_spot
Count: 4
---> Mislabeled: 32 Tomato___Leaf_Mold
Count: 3
```

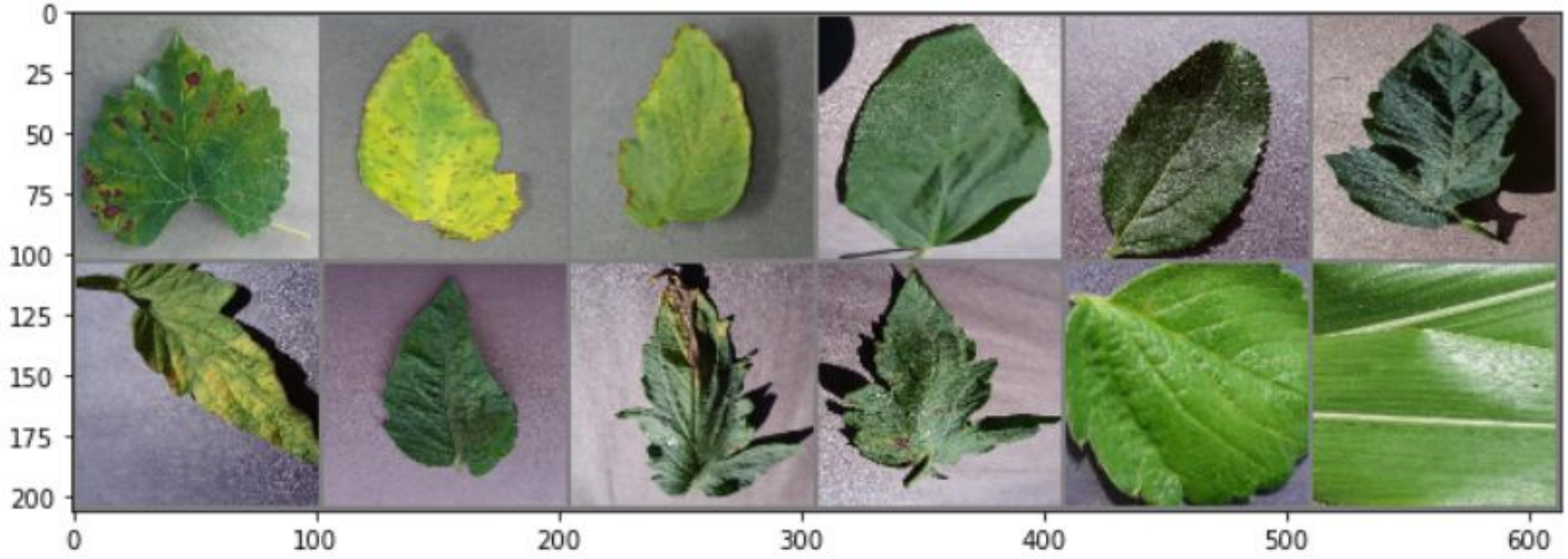
Showing some misclassified images for label 0



Showing some misclassified images for label 30

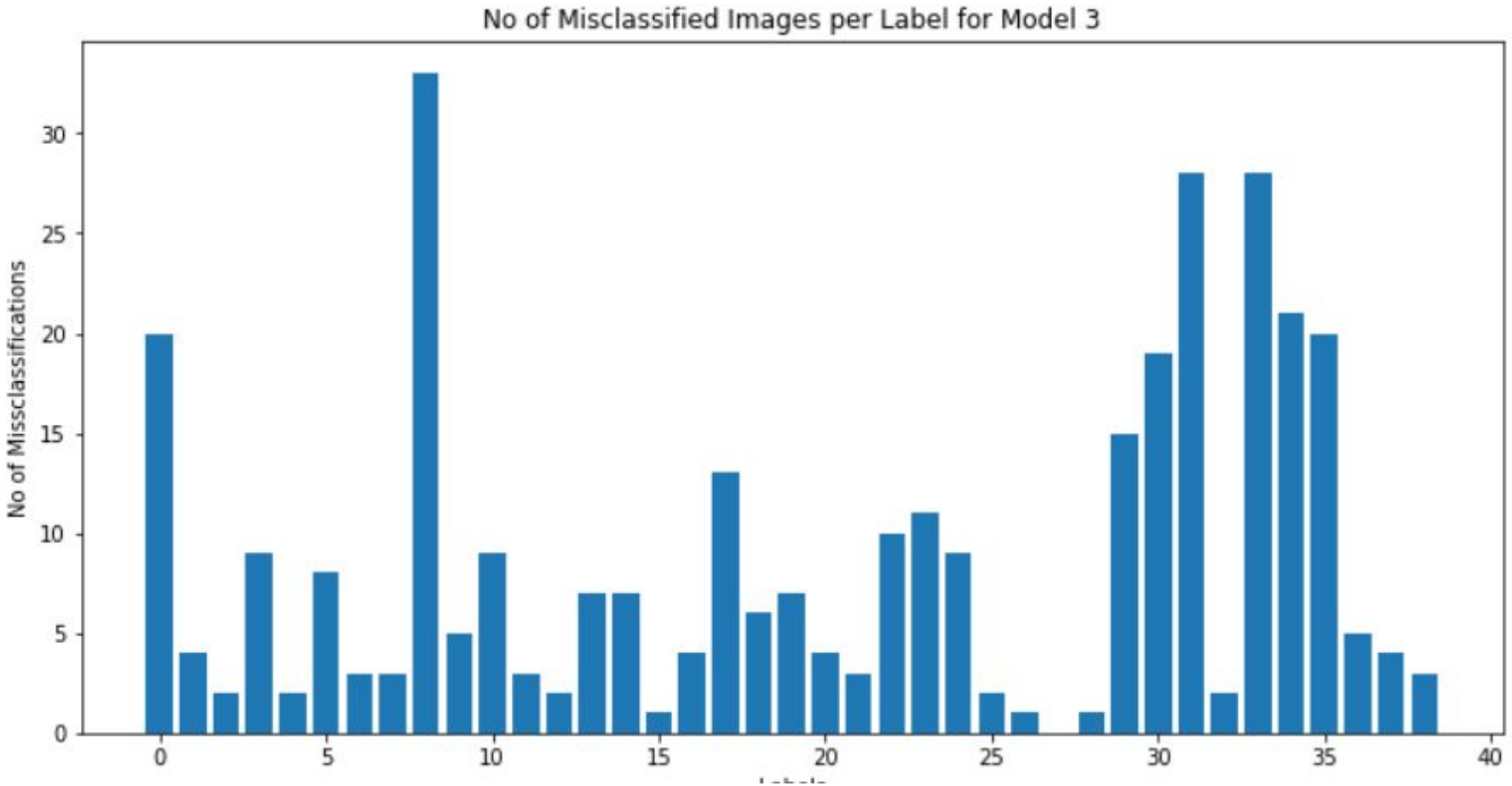


Examine misclassified images: **Model 3**

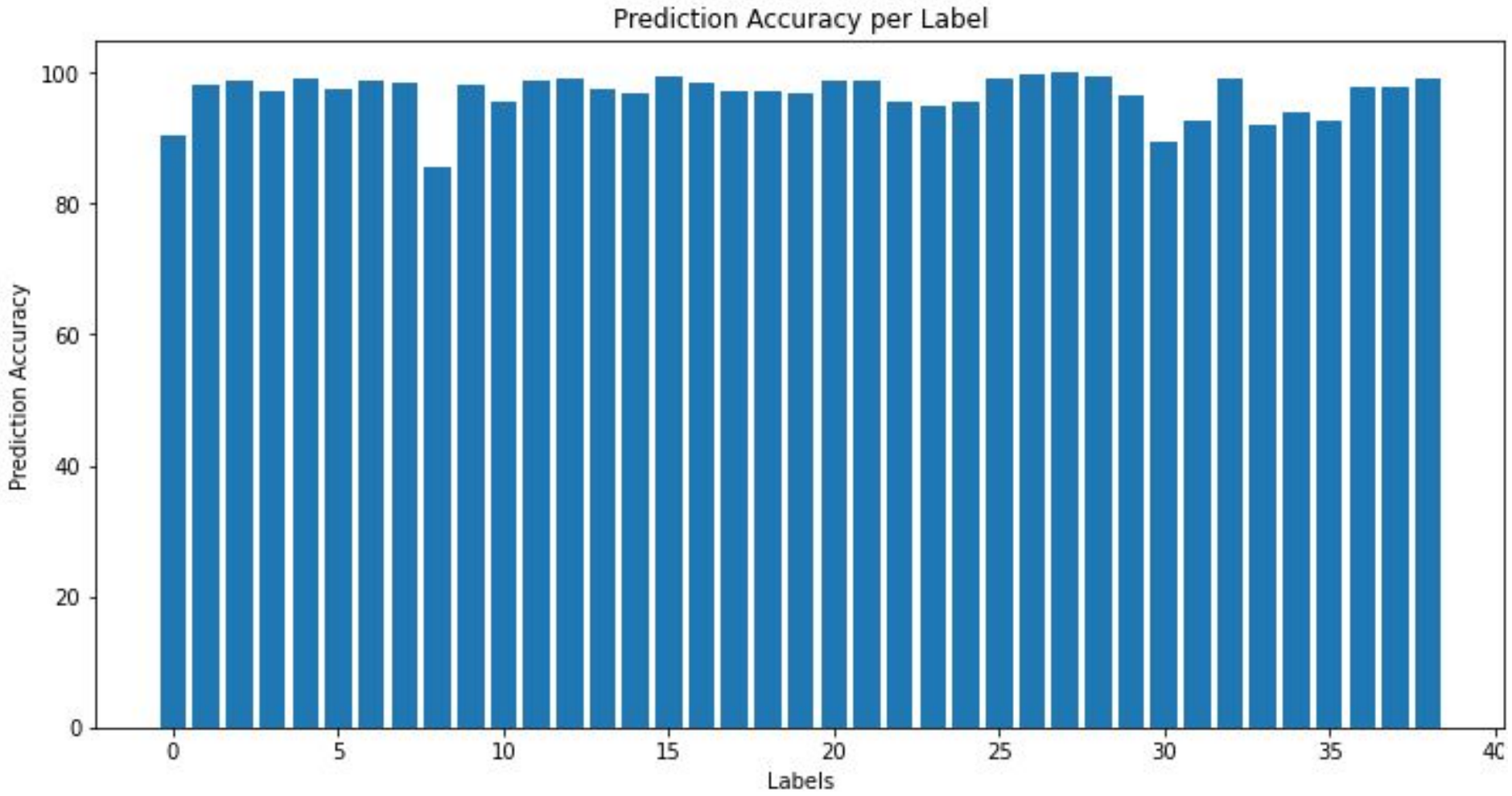


Showing few misclassified images

No of misclassified images per label for **Model 3**



Prediction accuracy per label for **Model 3**

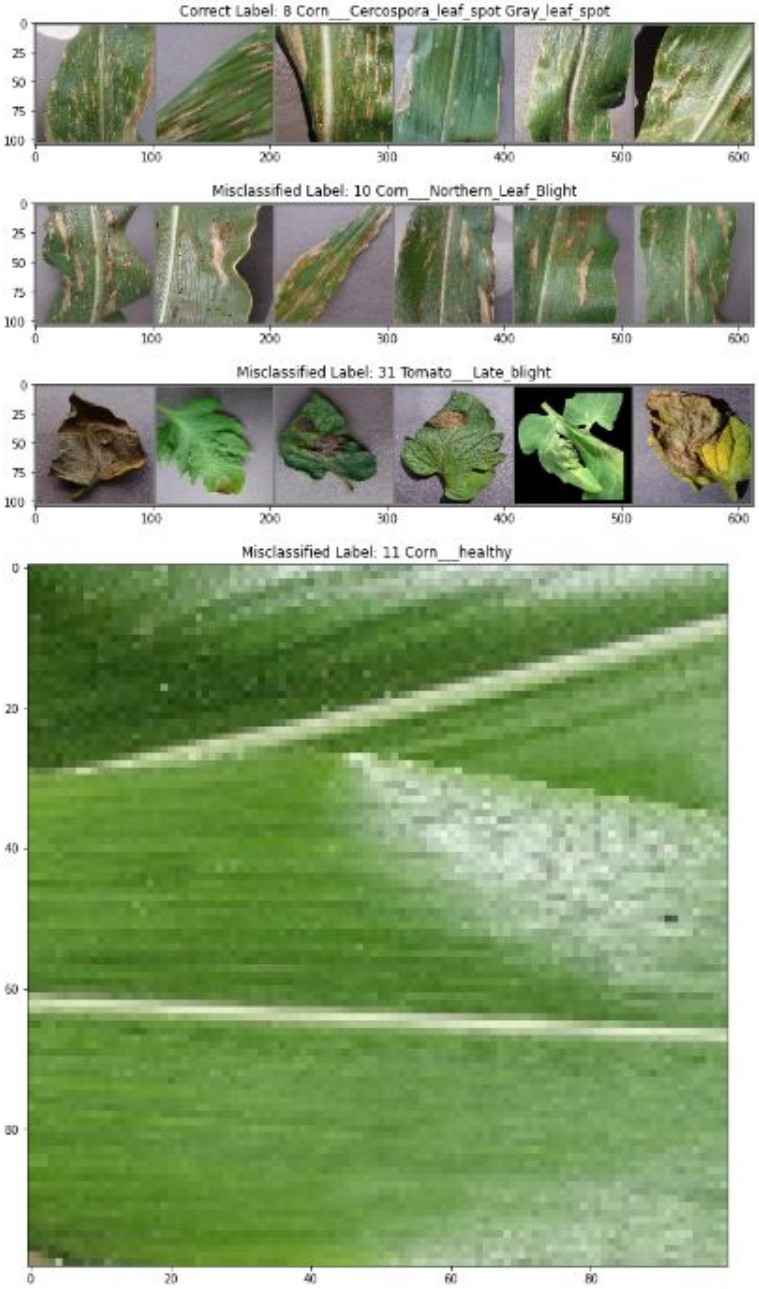


Note: Prediction accuracy on label 8 is lower compared to other labels

Top 3 mislabelled count: **Model 3**

```
For label 8, Corn___Cercospora_leaf_spot Gray_leaf_spot, top three misclassified labels are:
---> Mislabeled: 10 Corn___Northern_Leaf_Blight
Count: 28
---> Mislabeled: 31 Tomato___Late_blight
Count: 1
---> Mislabeled: 11 Corn___healthy
Count: 1
```

Showing some misclassified images for label 8



Result Summary

	Model 1	Model 2	Model 3
Number of Layers	2	5	7
Accuracy	84.14	92.24	96.66
Total Misclassifications	1586	776	334

7) Transfer Learning

Transfer Learning: **Alexnet**

```
alexnet = models.alexnet(weights=AlexNet_Weights.IMAGENET1K_V1).to(device)
```

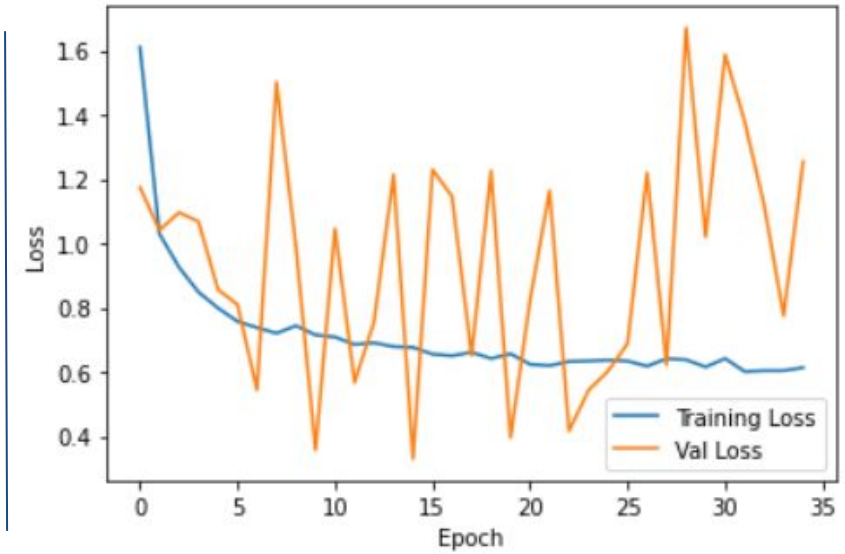
```
for param in alexnet.parameters():
    param.requires_grad=False

alexnet.classifier = nn.Sequential(
    nn.Linear(9216, 1024),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(1024, 700),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(700, 400),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(400, 39)
)

alexnet = alexnet.to(device)
```

```
J_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(alexnet.parameters(), lr=0.001)
```

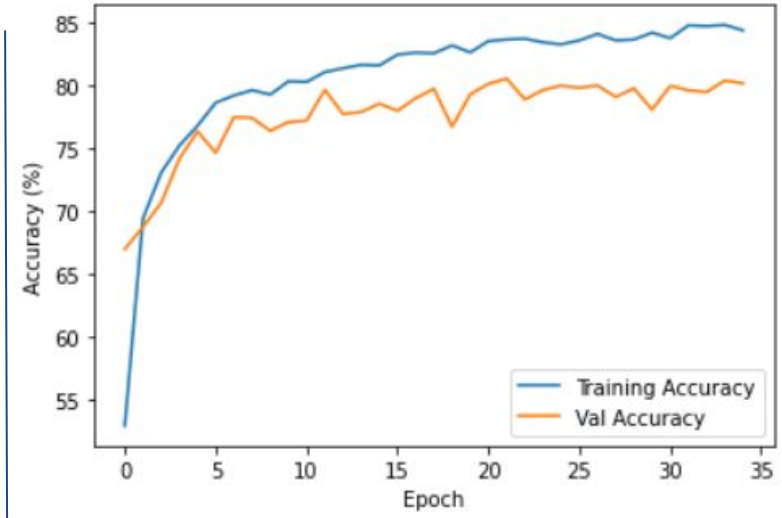
```
filepath4 = './ModelParameter/Transfer Learning/alexnetbestModelPara.pt'
```

Plotting loss, accuracy and training time: **Alexnet**

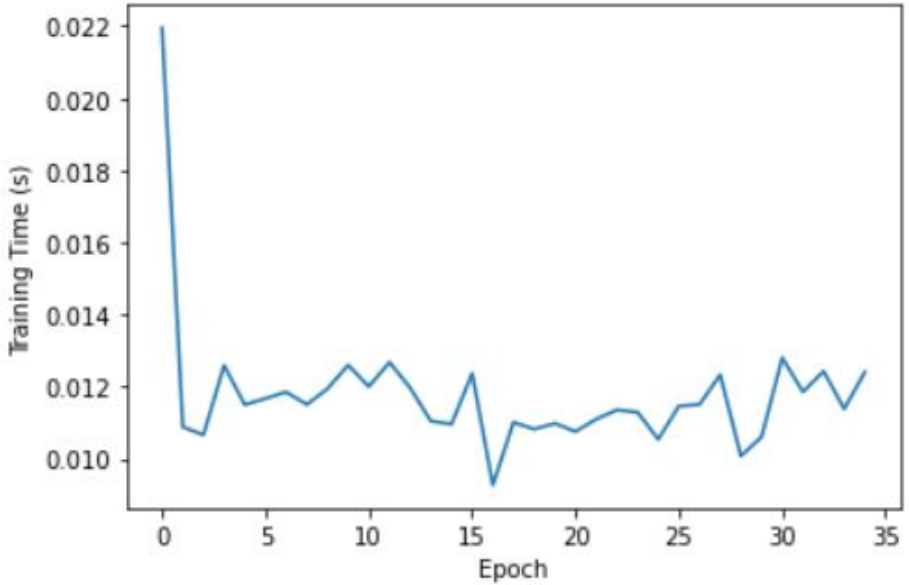
Loss Plot



Accuracy Plot



Training Time Plot



Transfer Learning: **Resnet**

```
resnet = models.resnet18(weights=ResNet18_Weights.IMAGENET1K_V1).to(device)
```

```
for param in resnet.parameters():
    param.requires_grad=False #this is essentially freezing all the weights

resnet.fc = nn.Linear(512, 39)

resnet = resnet.to(device)
```

```
random_image = torch.rand((50, 3, 100, 100)).to(device)
out = resnet(random_image)
print(out.shape)
```

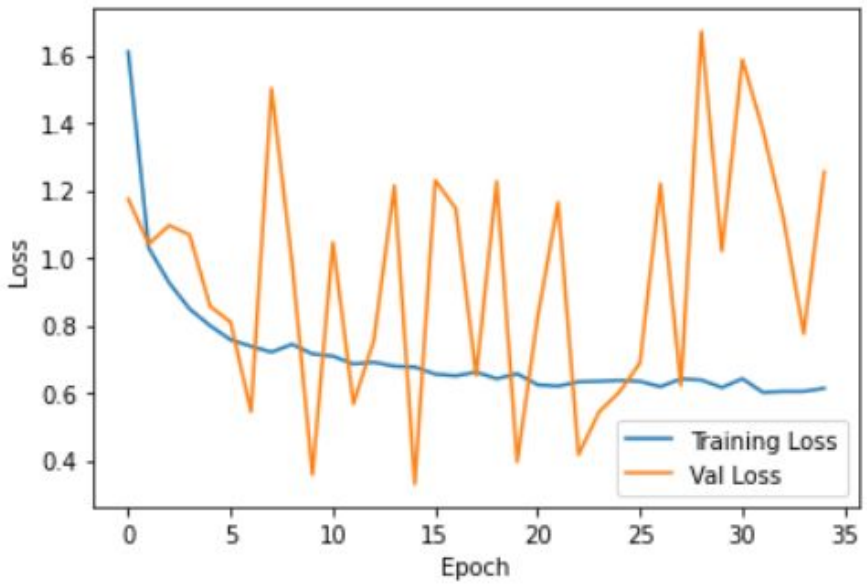
```
torch.Size([50, 39])
```

```
J_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(resnet.parameters(), lr=0.001)
```

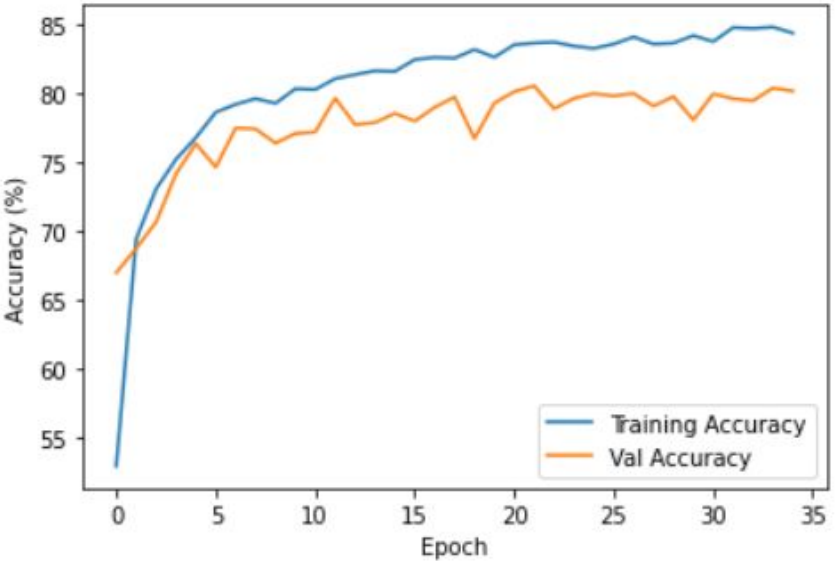
```
filepath5 = './ModelParameter/Transfer Learning/resnetbestModelPara.pt'
```

Plotting loss, accuracy and training time: **Resnet**

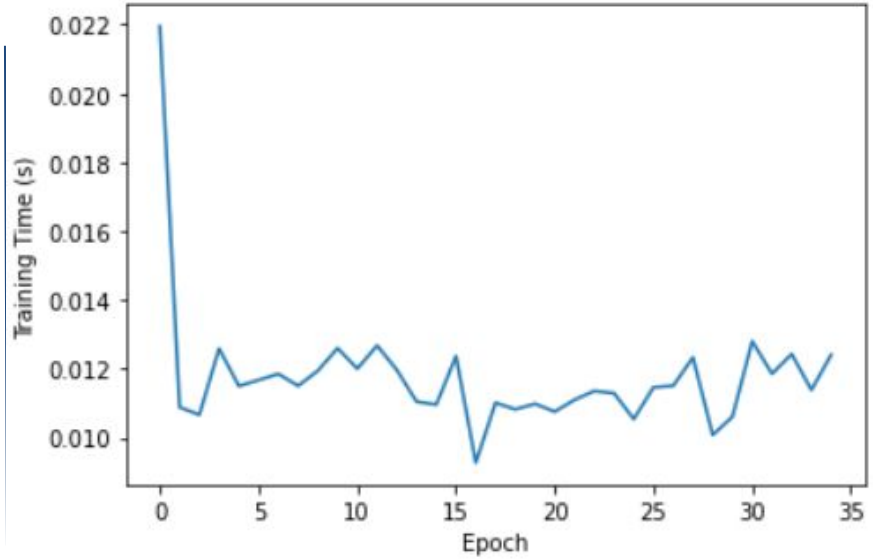
Loss Plot



Accuracy Plot



Training Time Plot



8) Testing/Inference

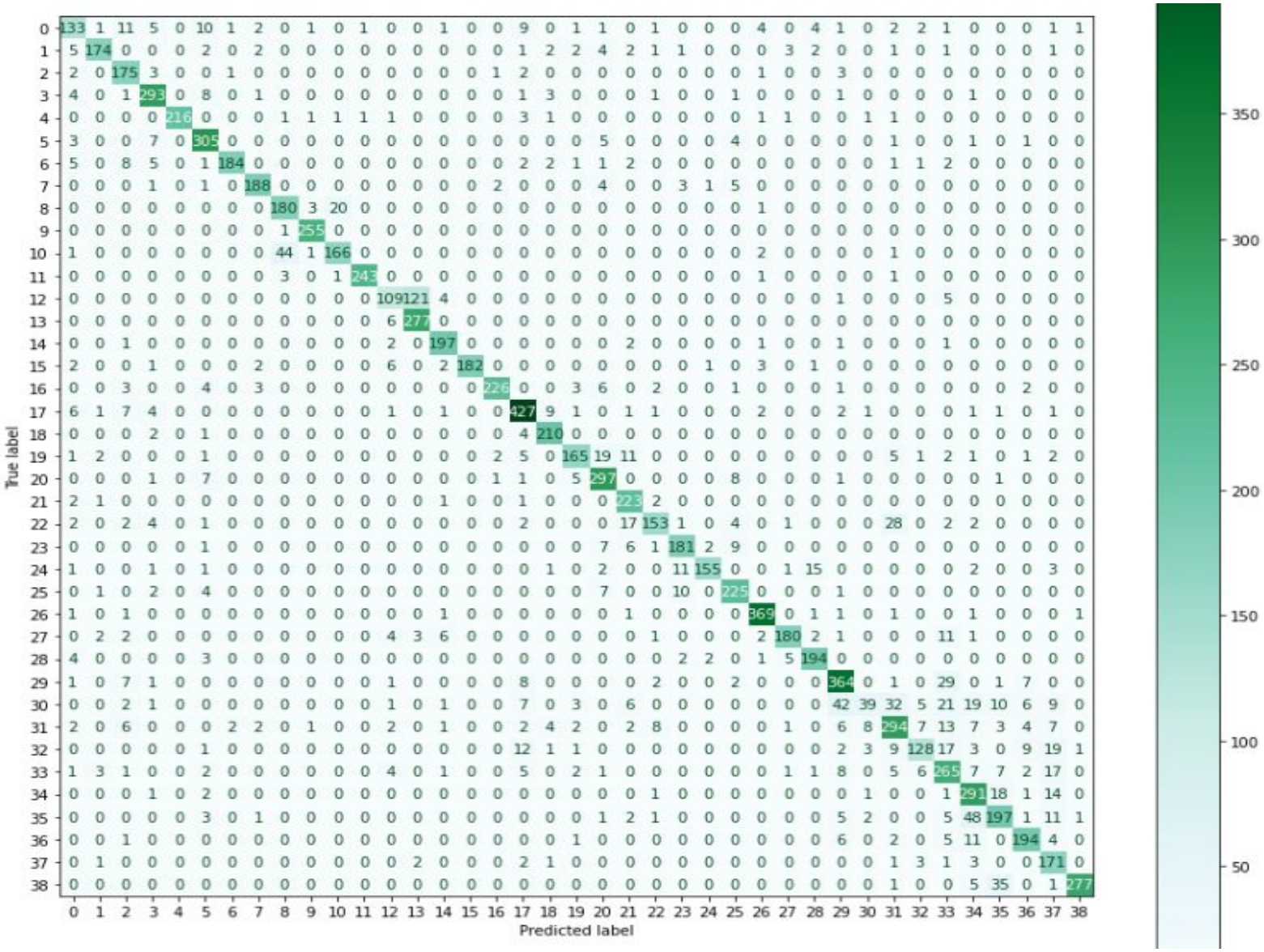
Accuracy on test data

Model	Accuracy
Alexnet	85.02
Resnet	88.35

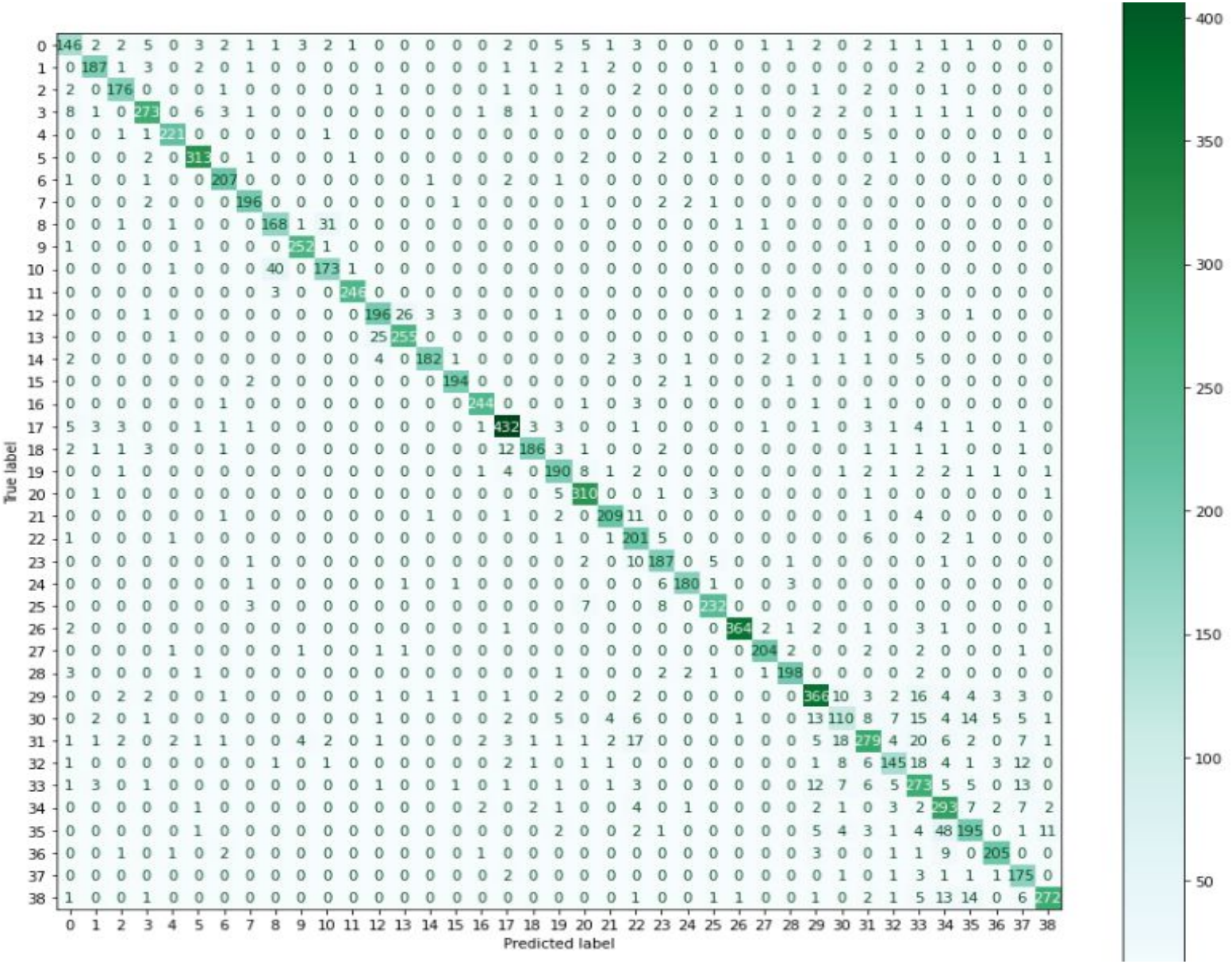
Total Number of Misclassification per Label

Model	Total Number of Misclassifications per Label
Alexnet	**1498**
Resnet	**1165**

Confusion Matrix: **Alexnet**



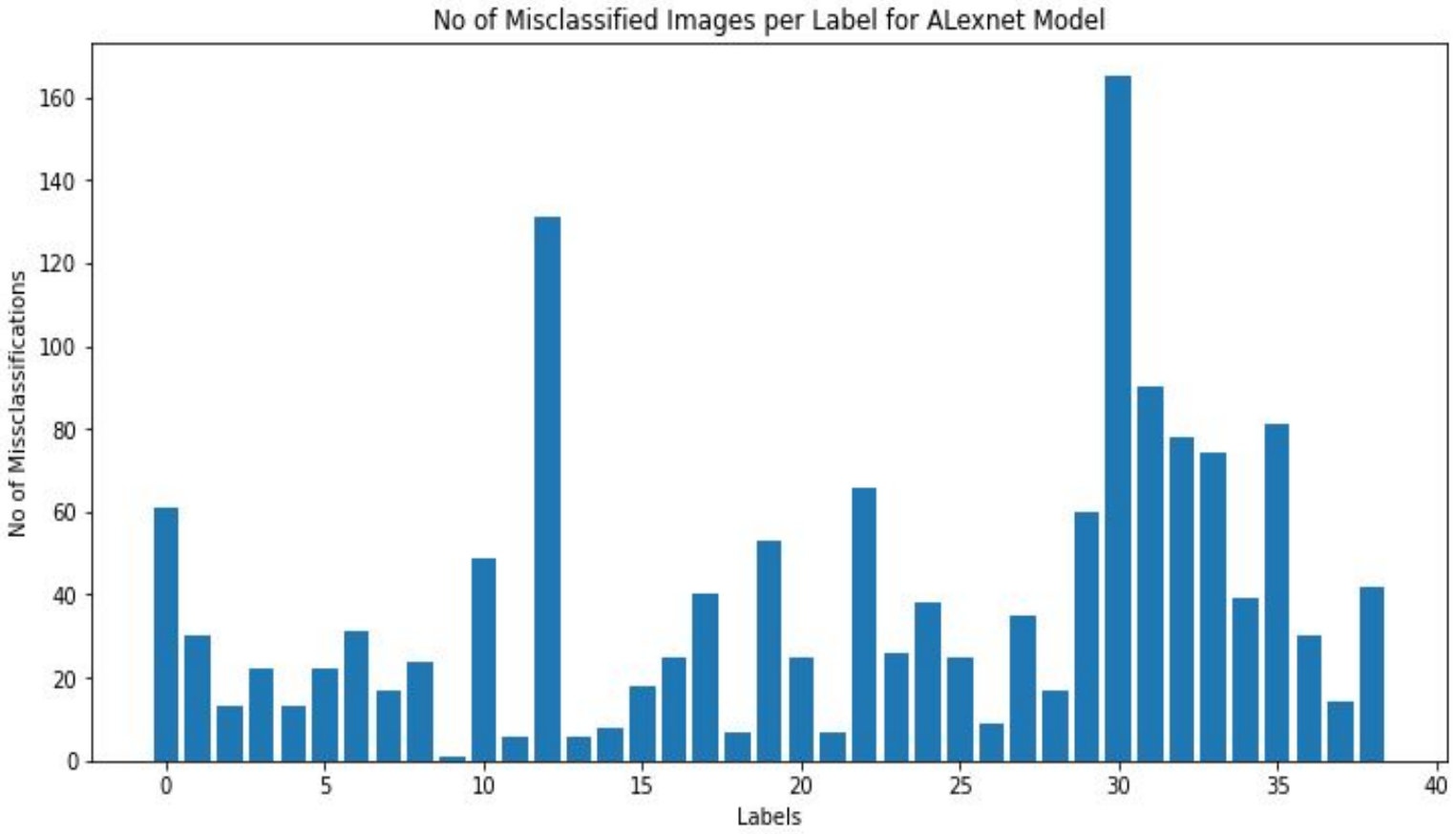
Confusion Matrix: **Resnet**



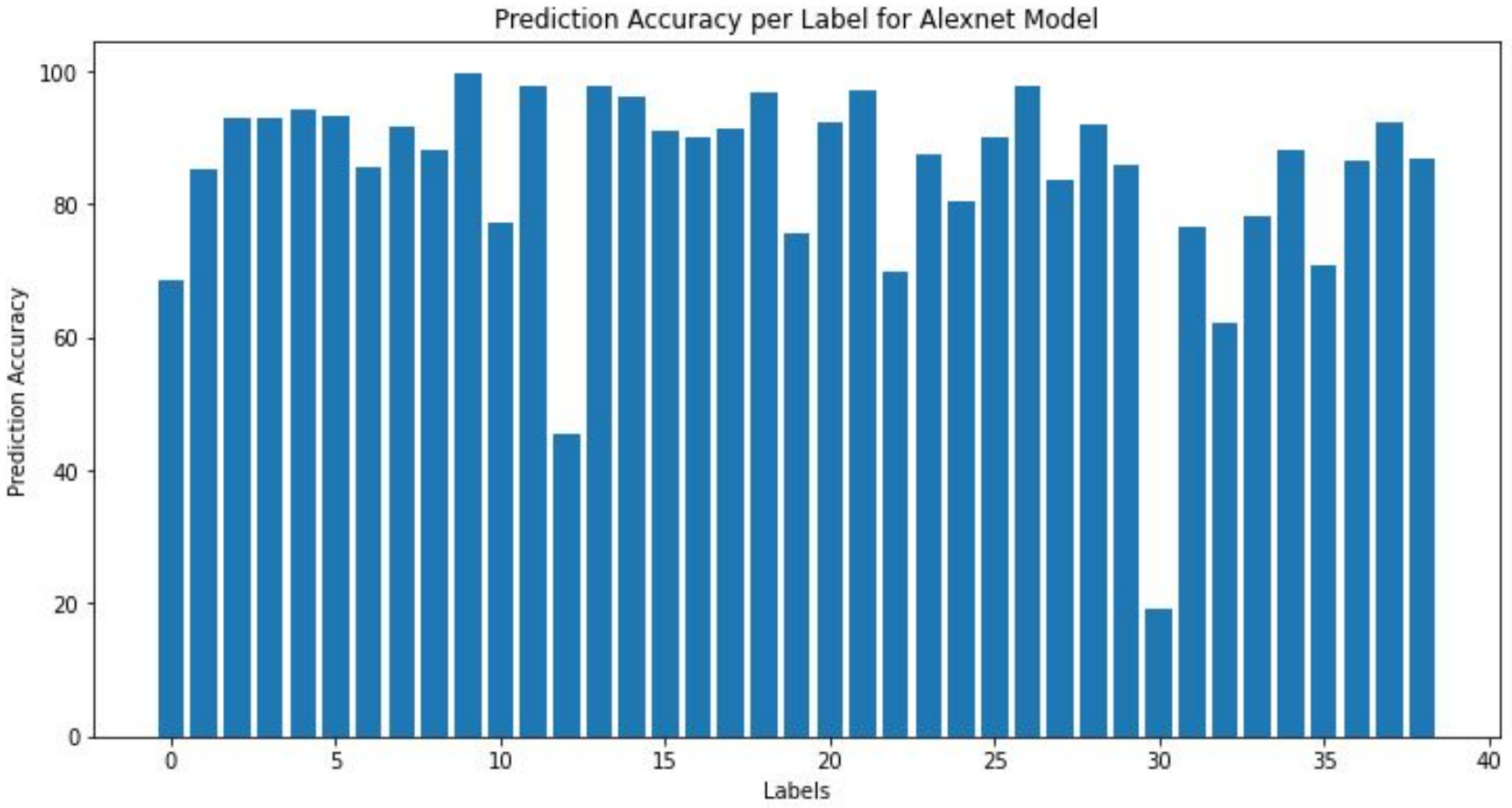
8.1) Examine misclassified images

No of misclassified images per label for **Alexnet**

Total misclassified images: 1498



Prediction accuracy per label for **Alexnet**

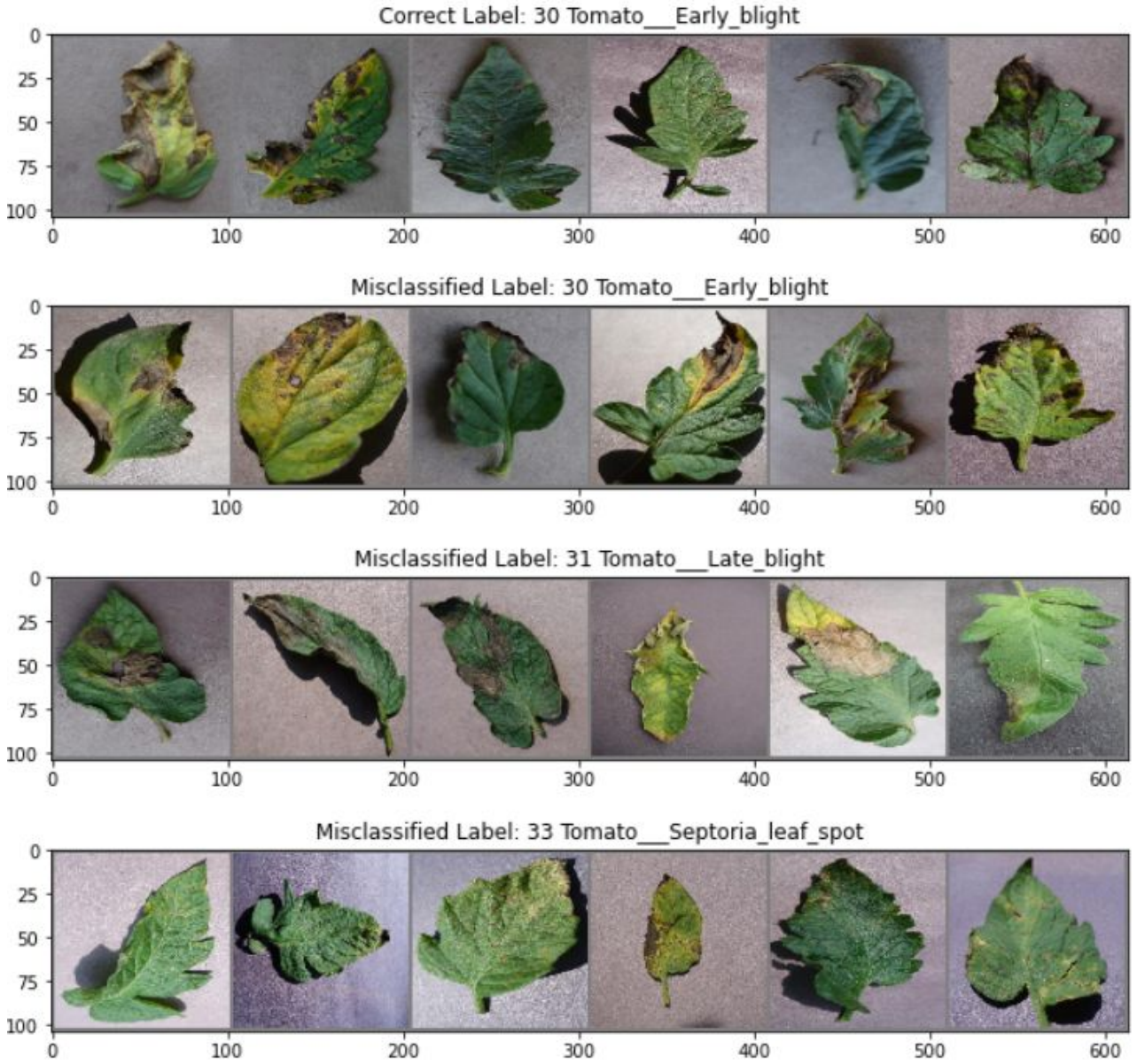


Note: Low accuracy prediction on labels 0, 10, 12, 22, 30, 32, and 35

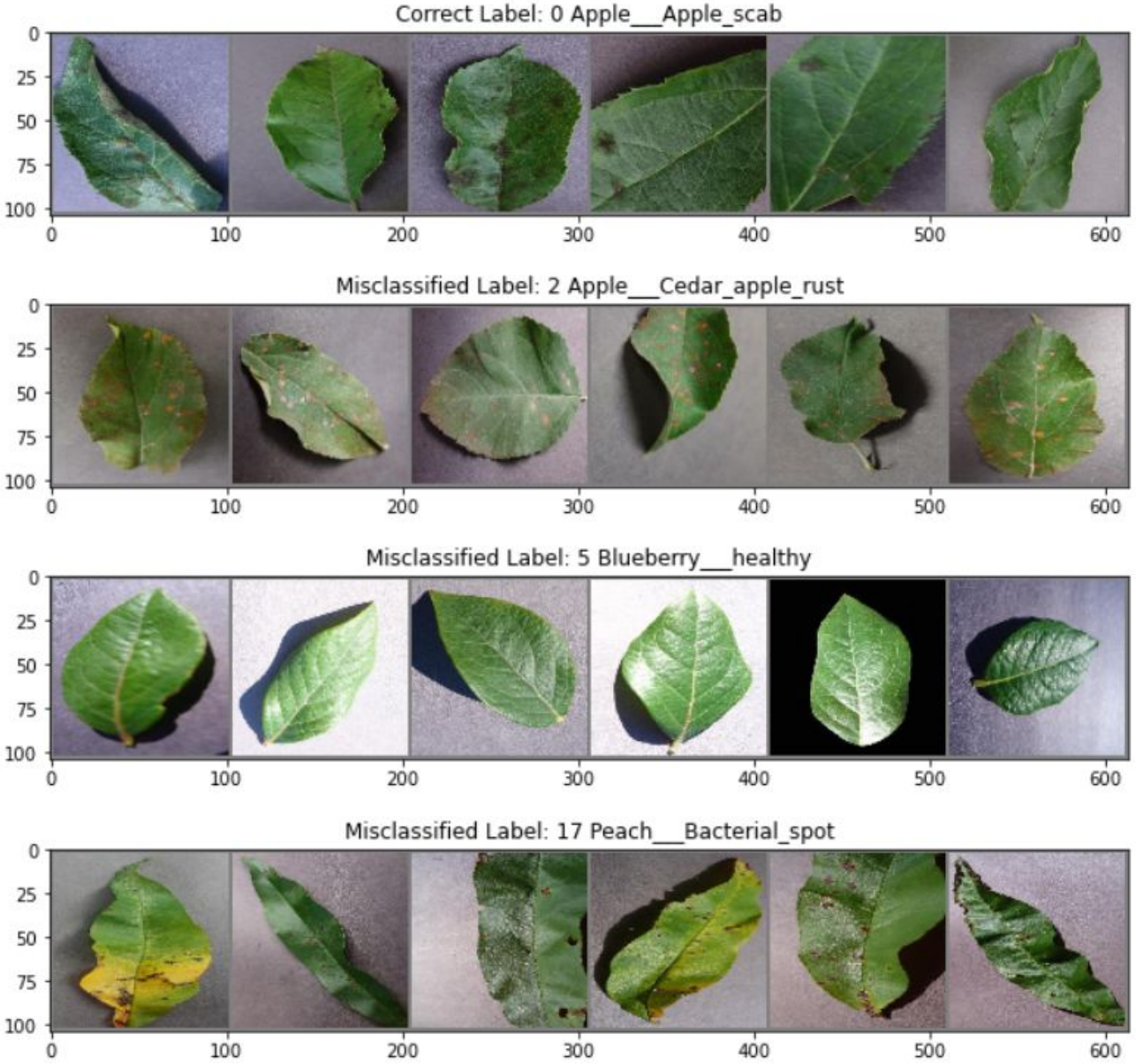
Top 3 mislabelled count: **Alexnet**

```
For label 0, Apple___Apple_scab, top three misclassified labels are:
---> Mislabeled: 2 Apple___Cedar_apple_rust
Count: 11
---> Mislabeled: 5 Blueberry___healthy
Count: 10
---> Mislabeled: 17 Peach___Bacterial_spot
Count: 9

For label 10, Corn___Northern_Leaf_Blight, top three misclassified labels are:
---> Mislabeled: 8 Corn___Cercospora_leaf_spot Gray_leaf_spot
Count: 44
---> Mislabeled: 26 Squash___Powdery_mildew
Count: 2
---> Mislabeled: 31 Tomato___Late_blight
Count: 1

For label 12, Grape___Black_rot, top three misclassified labels are:
---> Mislabeled: 12 Grape___Black_rot
Count: 109
---> Mislabeled: 33 Tomato___Septoria_leaf_spot
Count: 5
---> Mislabeled: 14 Grape___Leaf_blight_(Isariopsis_Leaf_Spot)
Count: 4

For label 22, Potato___Late_blight, top three misclassified labels are:
---> Mislabeled: 31 Tomato___Late_blight
Count: 28
---> Mislabeled: 21 Potato___Early_blight
Count: 17
---> Mislabeled: 3 Apple___healthy
Count: 4

For label 30, Tomato___Early_blight, top three misclassified labels are:
---> Mislabeled: 30 Tomato___Early_blight
Count: 39
---> Mislabeled: 31 Tomato___Late_blight
Count: 32
---> Mislabeled: 33 Tomato___Septoria_leaf_spot
Count: 21

For label 32, Tomato___Leaf_Mold, top three misclassified labels are:
---> Mislabeled: 37 Tomato___Tomato_mosaic_virus
Count: 19
---> Mislabeled: 33 Tomato___Septoria_leaf_spot
Count: 17
---> Mislabeled: 17 Peach___Bacterial_spot
Count: 12

For label 35, Tomato___Target_Spot, top three misclassified labels are:
---> Mislabeled: 34 Tomato___Spider_mites Two-spotted_spider_mite
Count: 48
---> Mislabeled: 37 Tomato___Tomato_mosaic_virus
Count: 11
---> Mislabeled: 33 Tomato___Septoria_leaf_spot
Count: 5
```

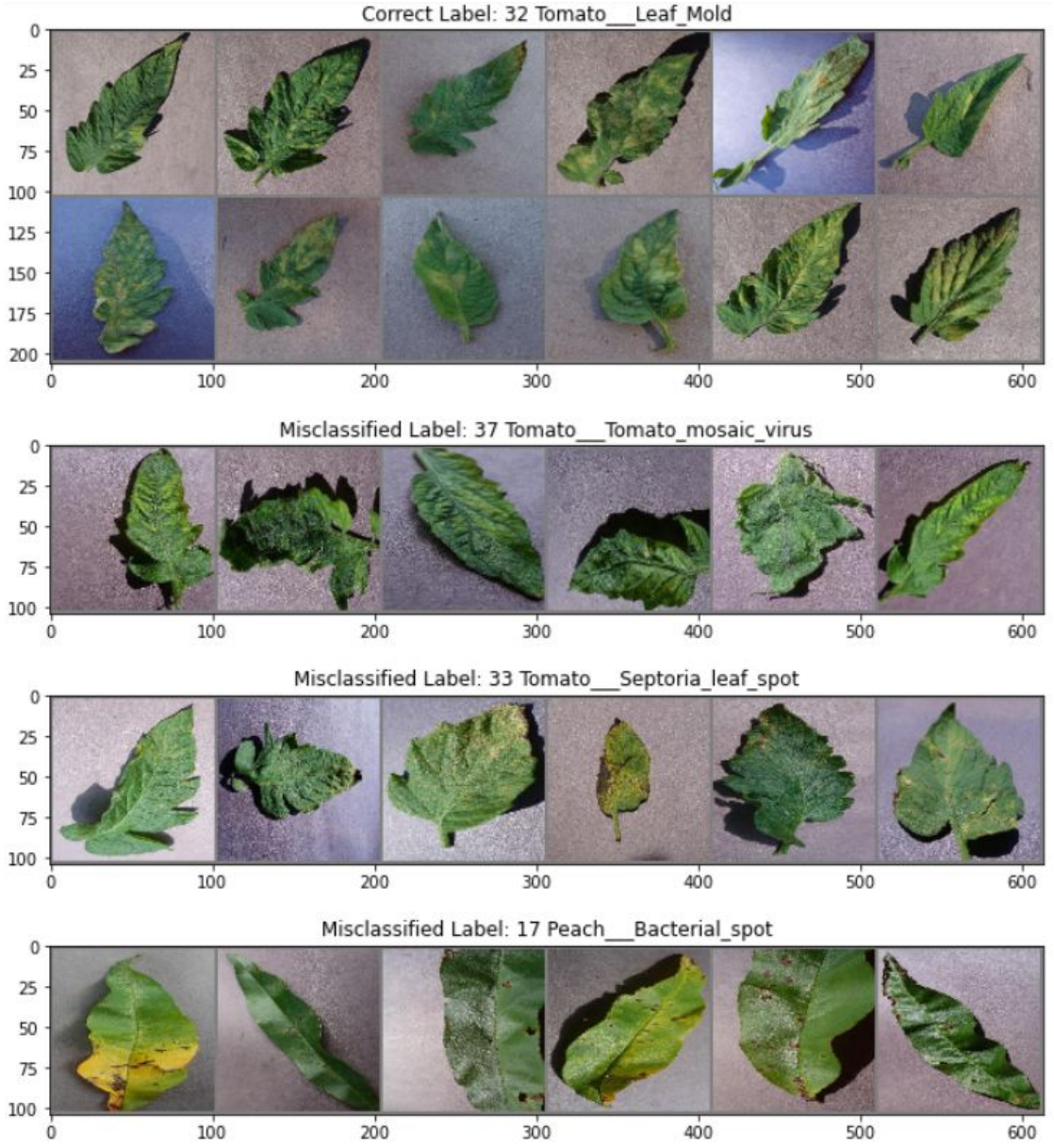
Showing some misclassified images for label 30



Showing some misclassified images for label 0

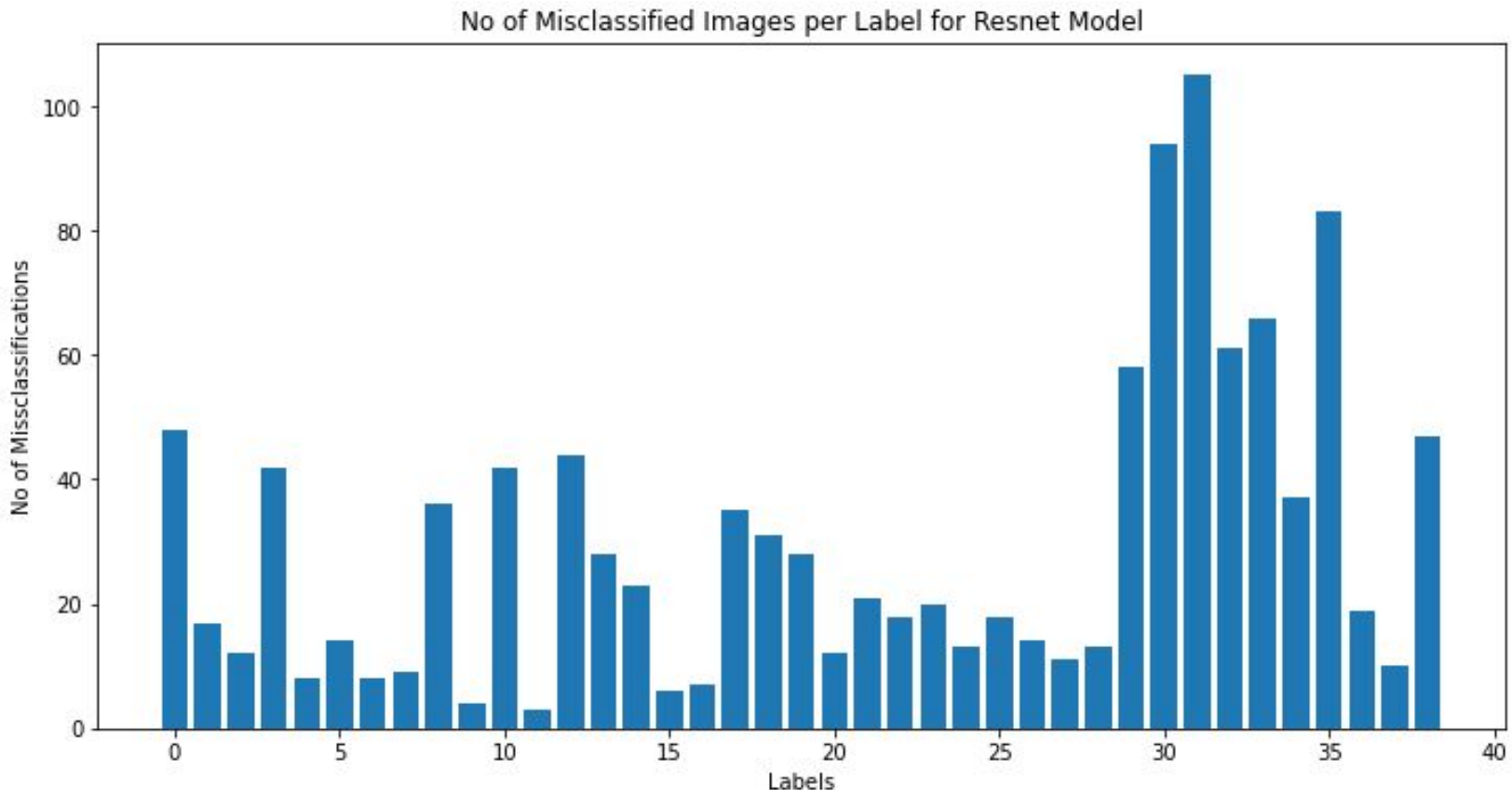


Showing some misclassified images for label 32

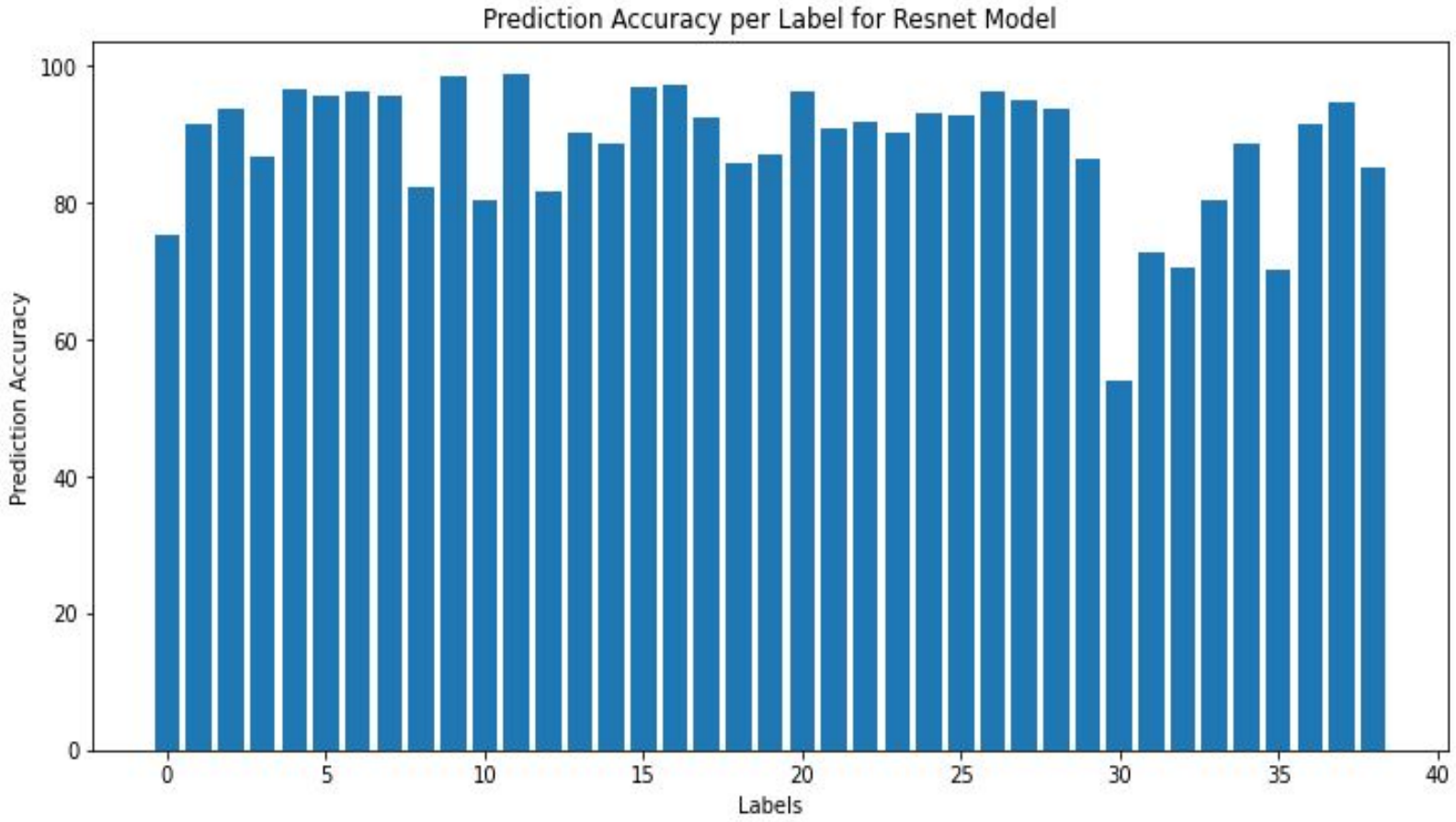


No of misclassified images per label for **Resnet**

Total misclassified images: 1165



Prediction accuracy per label for **Resnet**

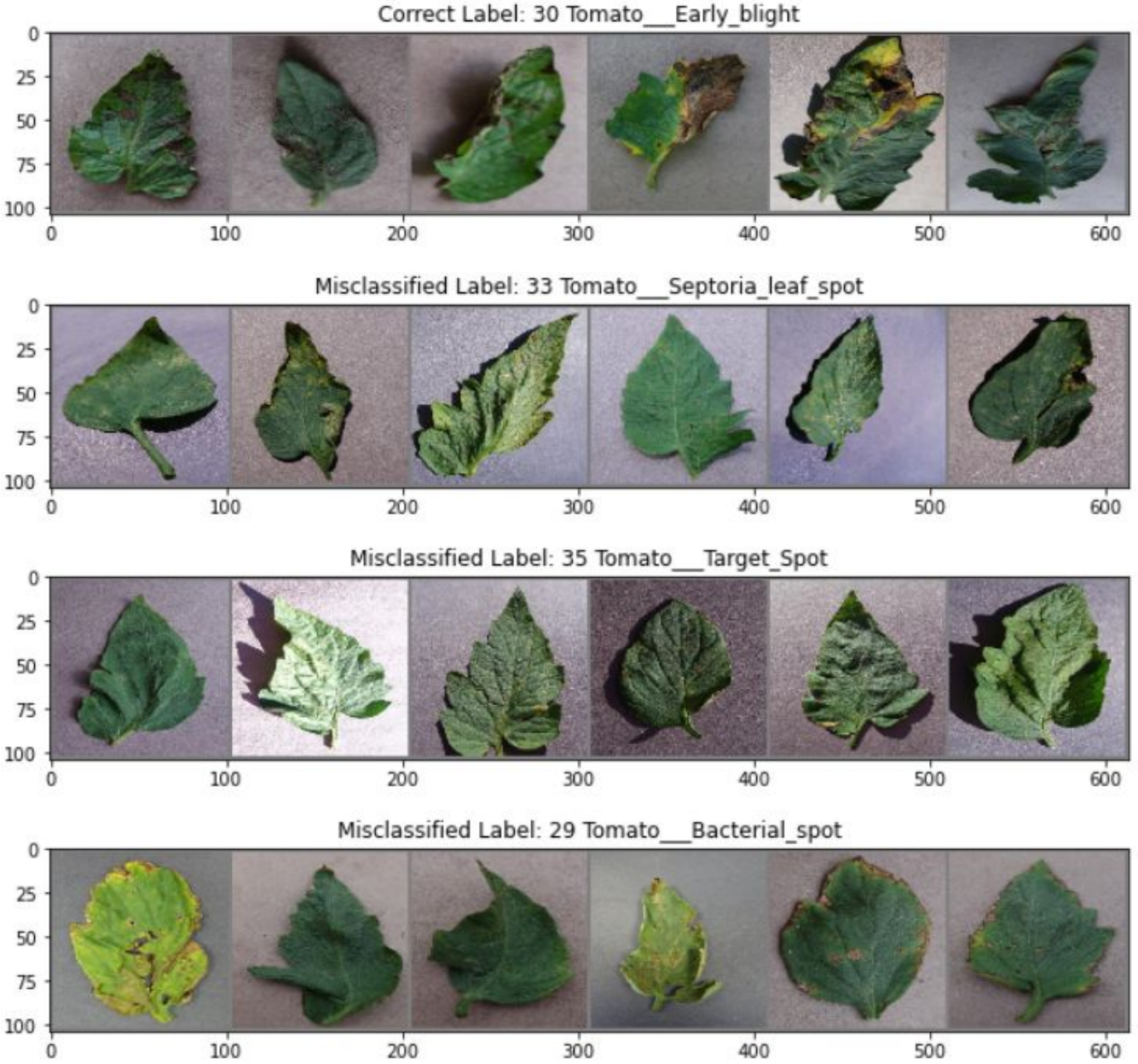


Note: Low accuracy prediction on labels 0, 30, 31, 32, and 35

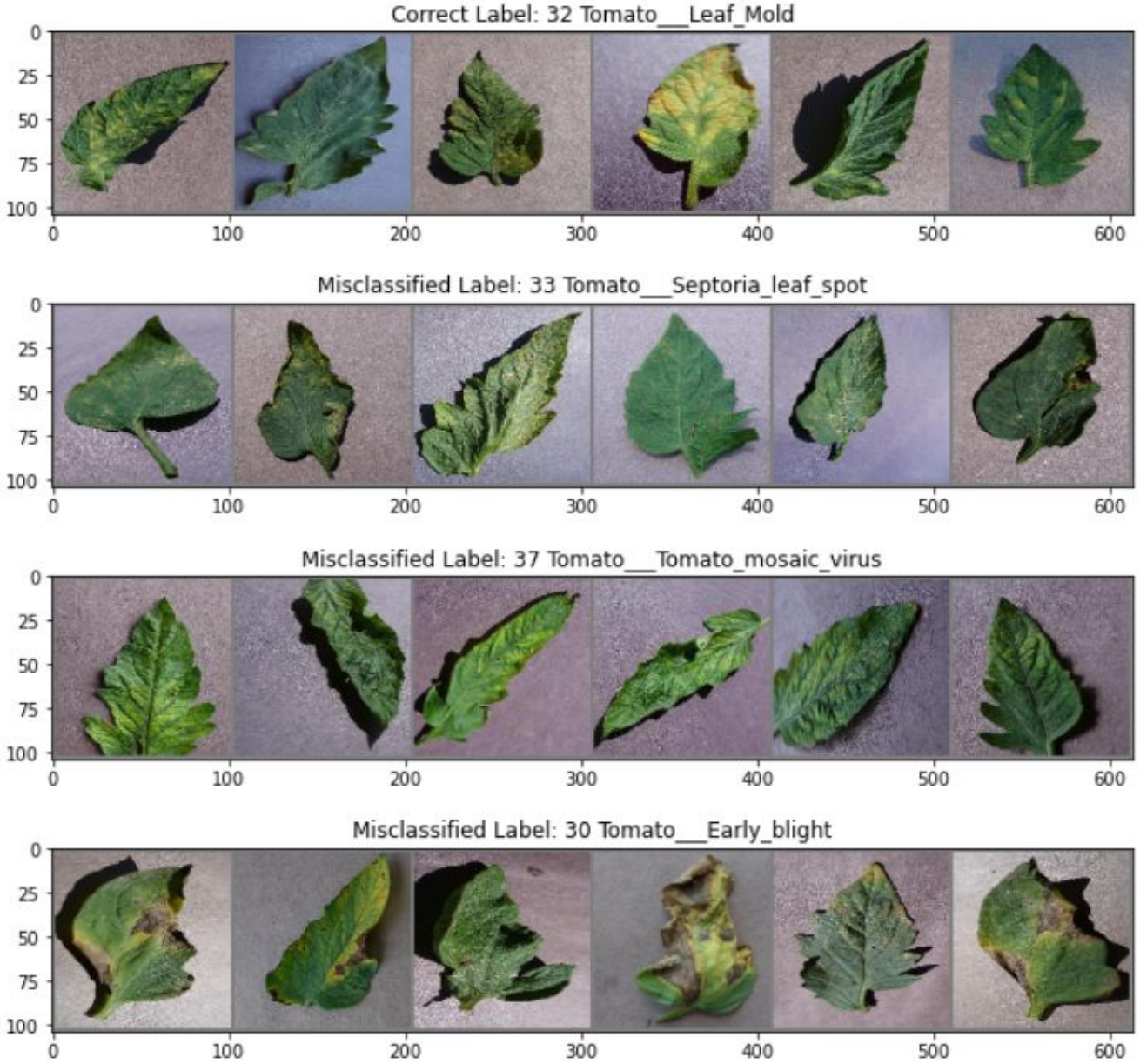
Top 3 mislabelled count: **Resnet**

```
For label 0, Apple___Apple_scab, top three misclassified labels are:
---> Mislabeled: 2 Apple___Cedar_apple_rust
Count: 11
---> Mislabeled: 5 Blueberry___healthy
Count: 10
---> Mislabeled: 17 Peach___Bacterial_spot
Count: 9

For label 10, Corn___Northern_Leaf_Blight, top three misclassified labels are:
---> Mislabeled: 8 Corn___Cercospora_leaf_spot Gray_leaf_spot
Count: 44
---> Mislabeled: 26 Squash___Powdery_mildew
Count: 2
---> Mislabeled: 31 Tomato___Late_blight
Count: 1

For label 12, Grape___Black_rot, top three misclassified labels are:
---> Mislabeled: 12 Grape___Black_rot
Count: 109
---> Mislabeled: 33 Tomato___Septoria_leaf_spot
Count: 5
---> Mislabeled: 14 Grape___Leaf_blight_(Isariopsis_Leaf_Spot)
Count: 4

For label 22, Potato___Late_blight, top three misclassified labels are:
---> Mislabeled: 31 Tomato___Late_blight
Count: 28
---> Mislabeled: 21 Potato___Early_blight
Count: 17
---> Mislabeled: 3 Apple___healthy
Count: 4

For label 30, Tomato___Early_blight, top three misclassified labels are:
---> Mislabeled: 30 Tomato___Early_blight
Count: 39
---> Mislabeled: 31 Tomato___Late_blight
Count: 32
---> Mislabeled: 33 Tomato___Septoria_leaf_spot
Count: 21

For label 32, Tomato___Leaf_Mold, top three misclassified labels are:
---> Mislabeled: 37 Tomato___Tomato_mosaic_virus
Count: 19
---> Mislabeled: 33 Tomato___Septoria_leaf_spot
Count: 17
---> Mislabeled: 17 Peach___Bacterial_spot
Count: 12

For label 35, Tomato___Target_Spot, top three misclassified labels are:
---> Mislabeled: 34 Tomato___Spider_mites Two-spotted_spider_mite
Count: 48
---> Mislabeled: 37 Tomato___Tomato_mosaic_virus
Count: 11
---> Mislabeled: 33 Tomato___Septoria_leaf_spot
Count: 5
```

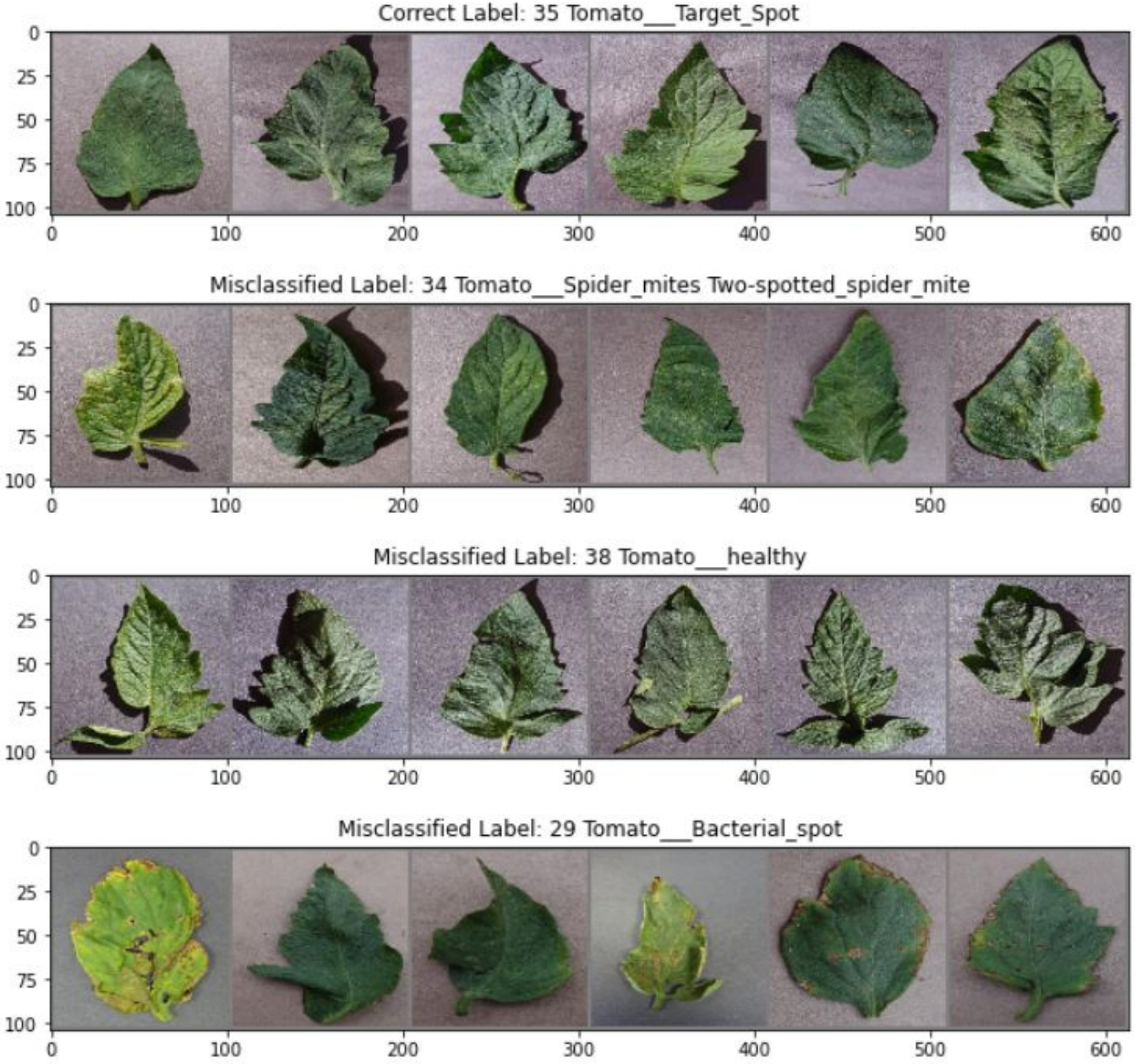
Showing some misclassified images for label 30



Showing some misclassified images for label 32



Showing some misclassified images for label 35



Overall Comparison in terms of Accuracy

Model	**Accuracy**
Model 1	84.14
Model 2	92.24
Model 3	**96.66**
Alexnet	85.02
Resnet	88.35

Overall Comparison in terms of number of image misclassification

Model	Total Number of Image Misclassifications
Model 1	1587
Model 2	776
Model 3	**334**
Alexnet	1498
Resnet	1165

Observation

- Increasing number of layers tend to improve model accuracy
  - However, it can also increase chances for model to overfit early
  - Can be mitigated with dropout to some extent
- Transfer learning models perform better than only model 1
  - original trained on 224X224 image size
  - Learned features were useful but need further training

Observation

- Good number of misclassifications seem to be related with tomato leaves
- E.g.
  - Model 1 had low accuracy on
    - Tomato___Early_blight
    - Tomato___Tomato_mosaic_virus
  - Model 2 had low accuracy
    - Tomato___Early_blight
  - Alex had low accuracy on
    - Tomato___Early_blight
    - Tomato___Leaf_Mold
    - Tomato___Target_Spot

Observation

- Resnet had low accuracy on
  - Tomato___Early_blight
  - Tomato___Late_blight
  - Tomato___Leaf_Mold
  - Tomato___Target_Spot
- Model 1 misclassified Peach__Bacrial_spot with
  - Tomato__Bacterial
- Model 2 misclassified Corn___Cercospora_leaf_spot Gray_leaf_spot with
  - Tomato___Late_blight

Observation

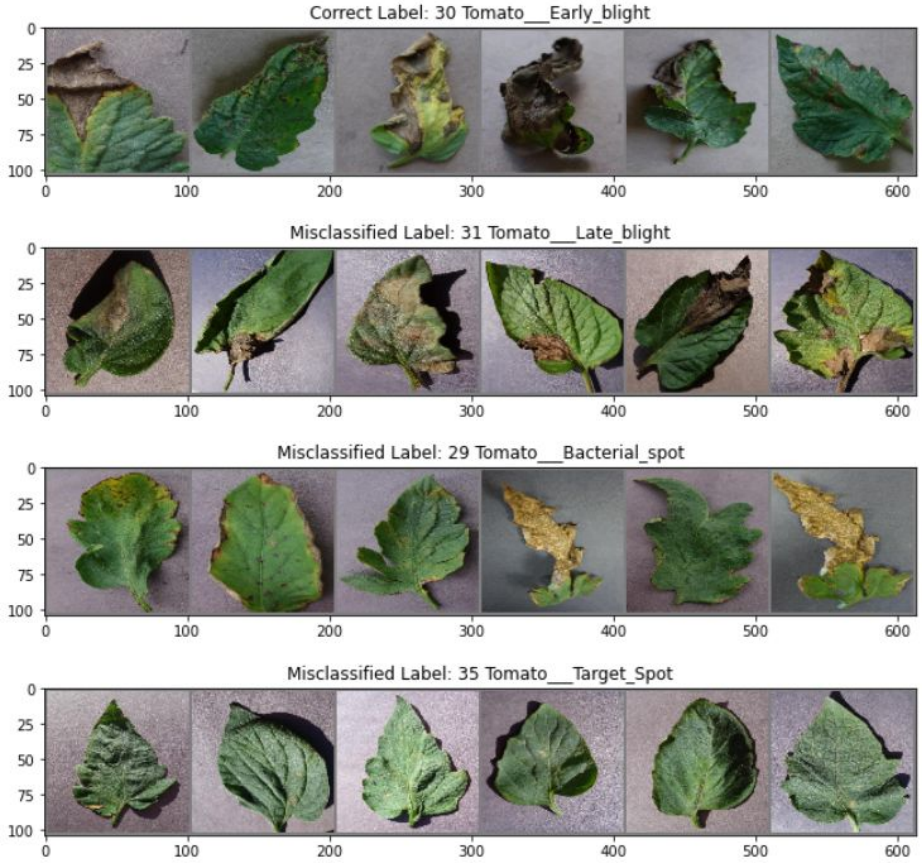
- Resnet had low accuracy on
  - Tomato___Early_blight
  - Tomato___Late_blight
  - Tomato___Leaf_Mold
  - Tomato___Target_Spot
- Model 1 misclassified Peach__Bacrial_spot with
  - Tomato__Bacterial
- Model 2 misclassified Corn___Cercospora_leaf_spot Gray_leaf_spot with
  - Tomato___Late_blight

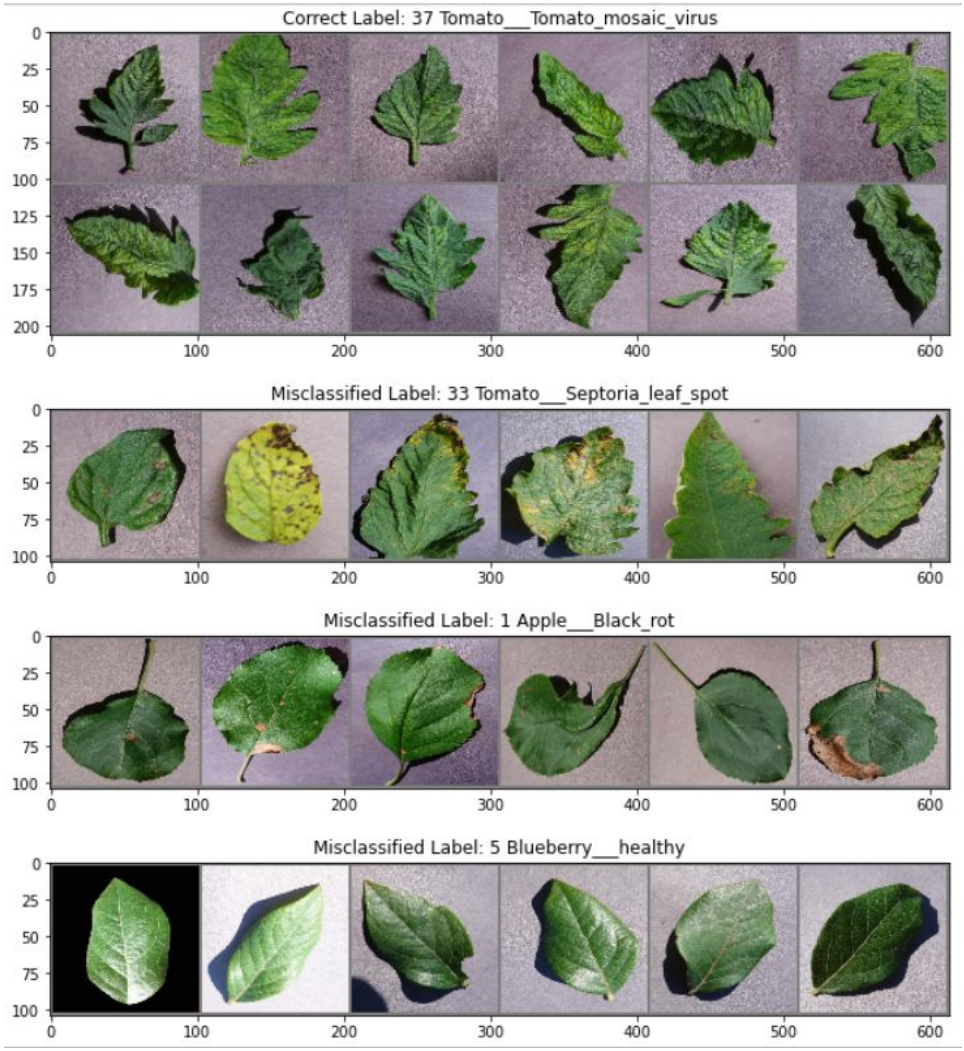
Observation

- Good number of misclassifications of and with also came from apple, corn, and peach but not as much as tomato leaves

Observation

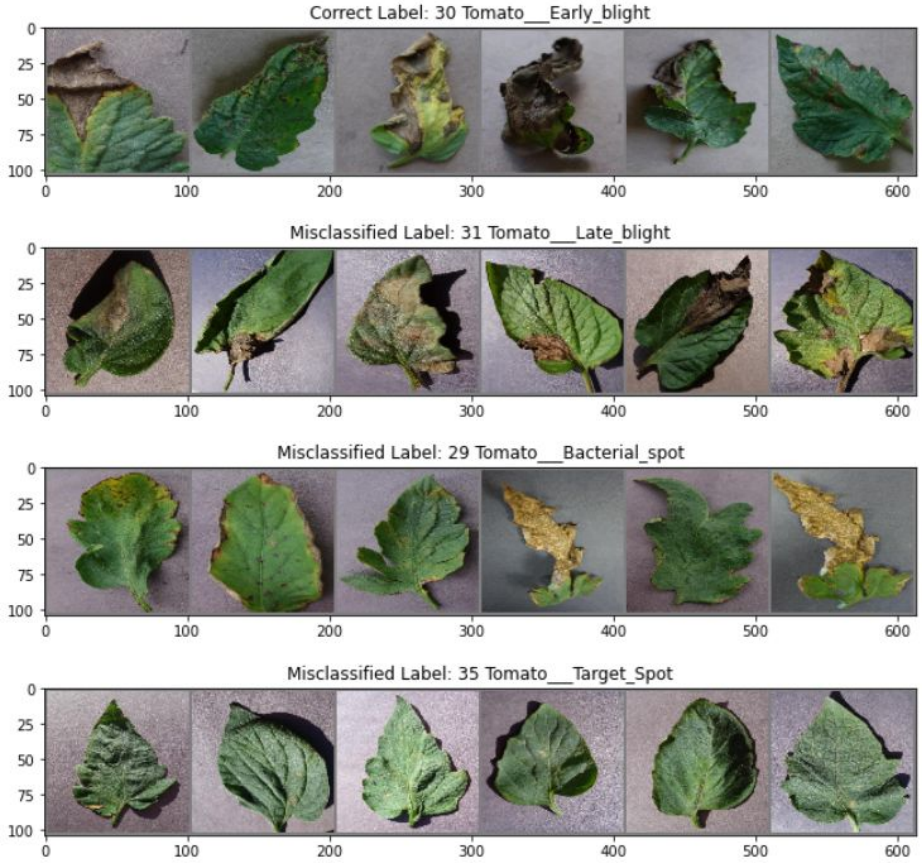
Model 1

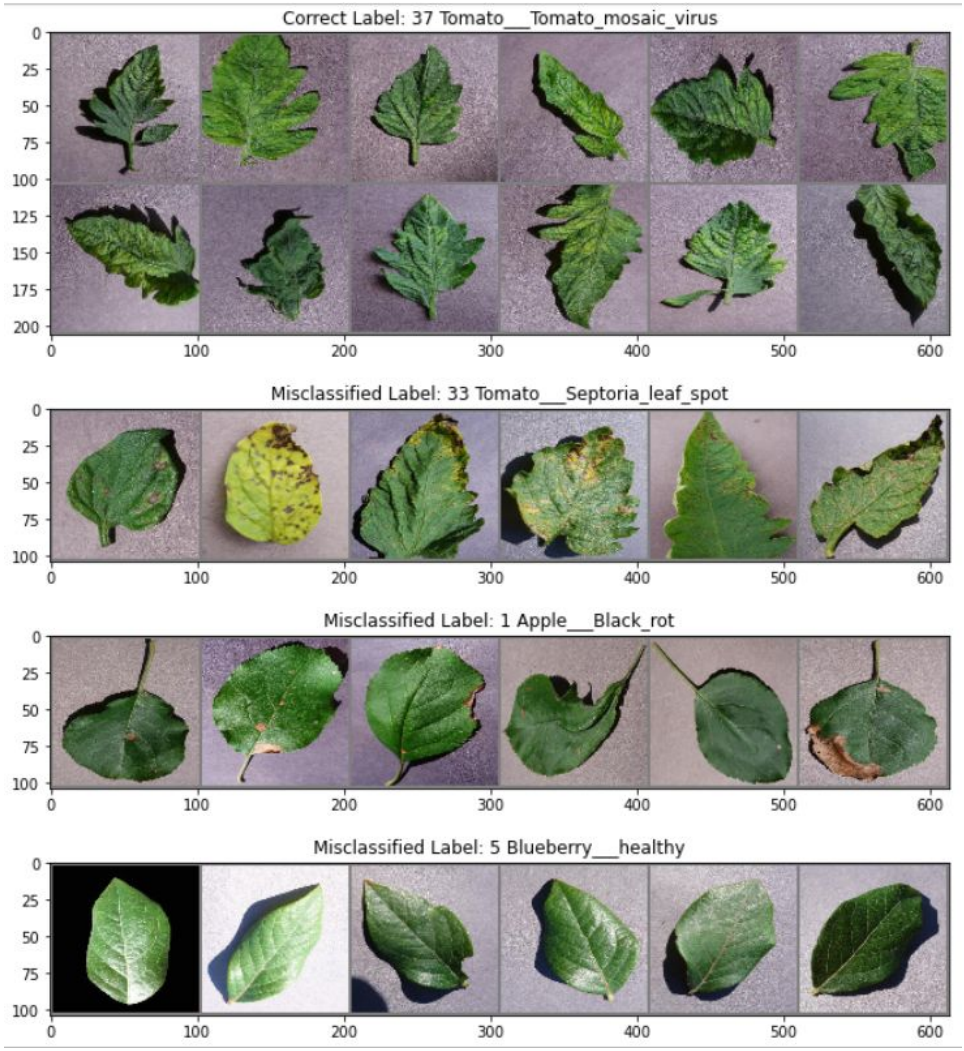




Observation

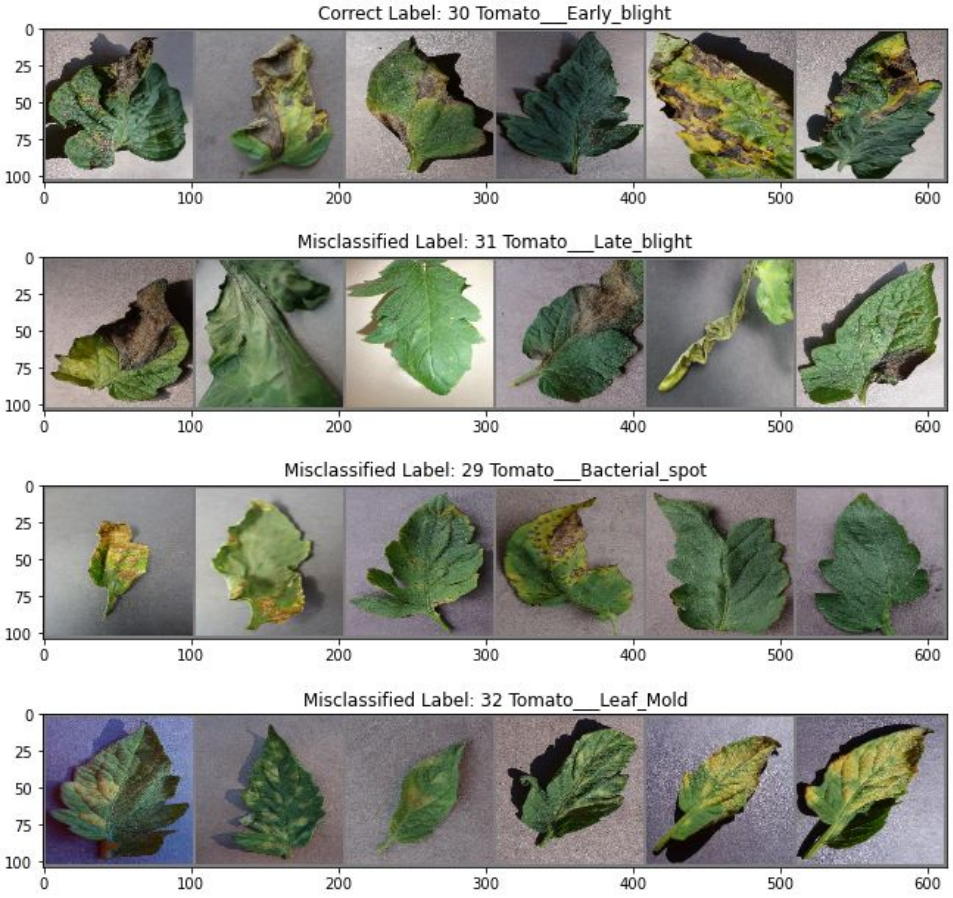
Model 1



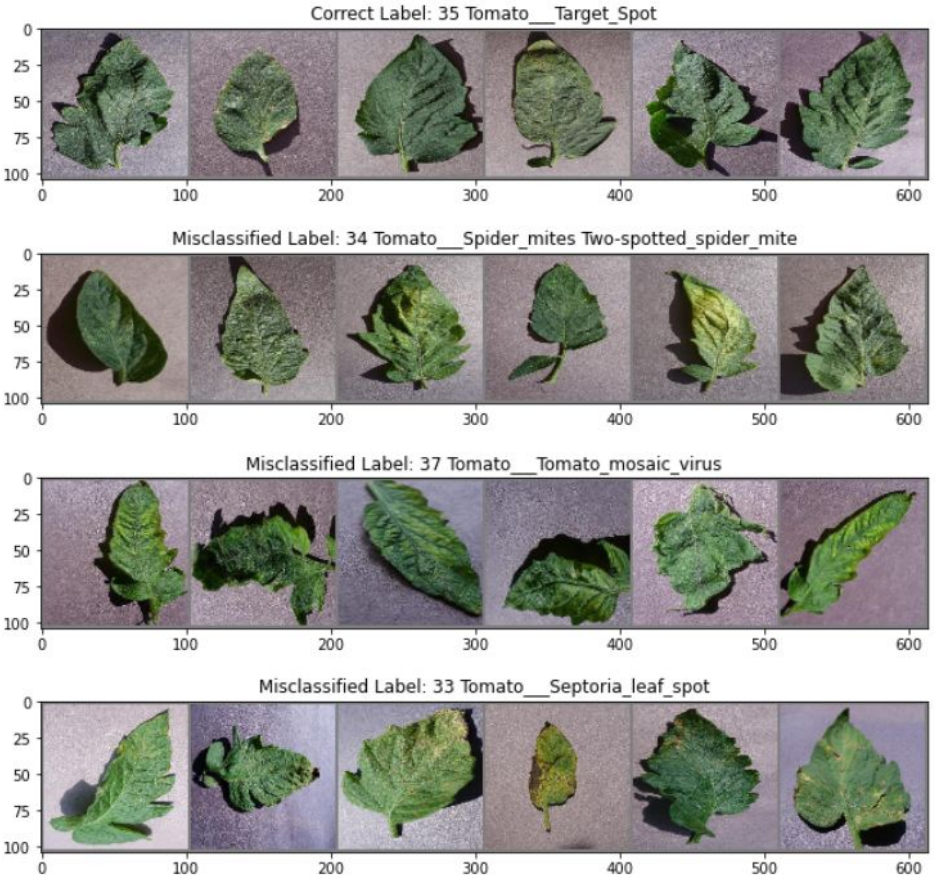


Observation

Model 2



Alexnet



Conclusion

- Having a deep CNN architecture can give a very good results
  - This is supported by pre-trained models where Resnet, having much more layers, performed better than Alexnet

Logistic Regression

Input layer → Logistic: Linear Function → SoftMax Function → Labels: Cross Entropy function

```
class LogisticRegressionModel(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LogisticRegressionModel, self).__init__()
        self.linear = nn.Linear(input_dim, output_dim)

    def forward(self, images):
        out = self.linear(images)
        return out
```

Parameter: 1,170,039

Layer : (Linear)1

Best Accuracy: 60%

Rerun for 3 times to get the accuracy

```
Epoch: 1. Iteration: 250. Loss: 1.3172693252563477. Accuracy: 59.310
Epoch: 2. Iteration: 500. Loss: 1.2904924154281616. Accuracy: 58.970
Epoch: 3. Iteration: 750. Loss: 1.1059702634811401. Accuracy: 59.380
Epoch: 4. Iteration: 1000. Loss: 1.3718109130859375. Accuracy: 59.41
Epoch: 5. Iteration: 1250. Loss: 1.497916579246521. Accuracy: 59.470
Epoch: 5. Iteration: 1500. Loss: 1.3298181295394897. Accuracy: 59.5
Epoch: 6. Iteration: 1750. Loss: 1.3828749656677246. Accuracy: 59.93
Epoch: 7. Iteration: 2000. Loss: 1.3359534740447998. Accuracy: 59.81
Epoch: 8. Iteration: 2250. Loss: 1.2590163946151733. Accuracy: 59.90
Epoch: 9. Iteration: 2500. Loss: 1.1226212978363037. Accuracy: 60.02
Epoch: 10. Iteration: 2750. Loss: 1.2472220659255981. Accuracy: 59.8
Epoch: 10. Iteration: 3000. Loss: 1.066394567489624. Accuracy: 59.95
```

Pros & Cons on Models

Pros

- CNN able to detects features without Human to supervised
- CNN can do no worse than Logistic because of number of layer. The more layer, the more learning capability.
- Logistic Regression easy to implement than CNN model

Cons

- CNN use a lot of data to train so image size is needed to be reduce due to Hardware's Capability
- Logistic Regression need to have a lot more of epoch to get the best accuracy

Future Work

It is yet to be seen how well these models generalize on the real data generated by the potential users. Therefore, it would be the next step to test our model in some real world data, may be produced by the ministry of agriculture of any country.

References

1. Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in plant science*, *7*, 1419.

2. Esgario, J. G., Krohling, R. A., & Ventura, J. A. (2020). Deep learning for classification and severity estimation of coffee leaf biotic stress. *Computers and Electronics in Agriculture*, *169*, 105162.

3. Antonio Bruno, Davide Moroni, & Riccardo Dainelli (2022) Improving plant disease classification by adaptive minimal ensembling. *Frontiers in Artificial Intelligence*. 10.3389/frai.2022.868926

Thank you!