Biologically inspired artificial intelligence

Report

Plant disease classification

Section members:

Maciej Czechowski

Kamil Kubina

1. Introduction

The project involves developing a neural network model for processing images of plant diseases. Convolutional neural networks will be used for the project to ensure efficient processing of the given images. The data will be divided into two sets - training and test in a ratio of 80/20. Technologies such as Python, PyTorch, JupiterNotebook will be used for implementation, and data processing will be performed using CUDA architecture.

2. Project analysis

a. Solution approaches

We decided to use a convolutional neural network (CNN) for our project. It operates on images which makes it ideally suited for creating our model. To solve the problem, we could use transfer learning with pre-trained models or create the structure of the artificial intelligence model from scratch. We decided to choose the second approach. This solution is more time-consuming and requires more resources than the alternative one using a pre-trained model. However, it makes the model learnable only for the data on which it will operate. It also provides full control over the structure of the model, making its behaviour easier to understand and making experiments on the model simpler to perform.

b. Datasets description

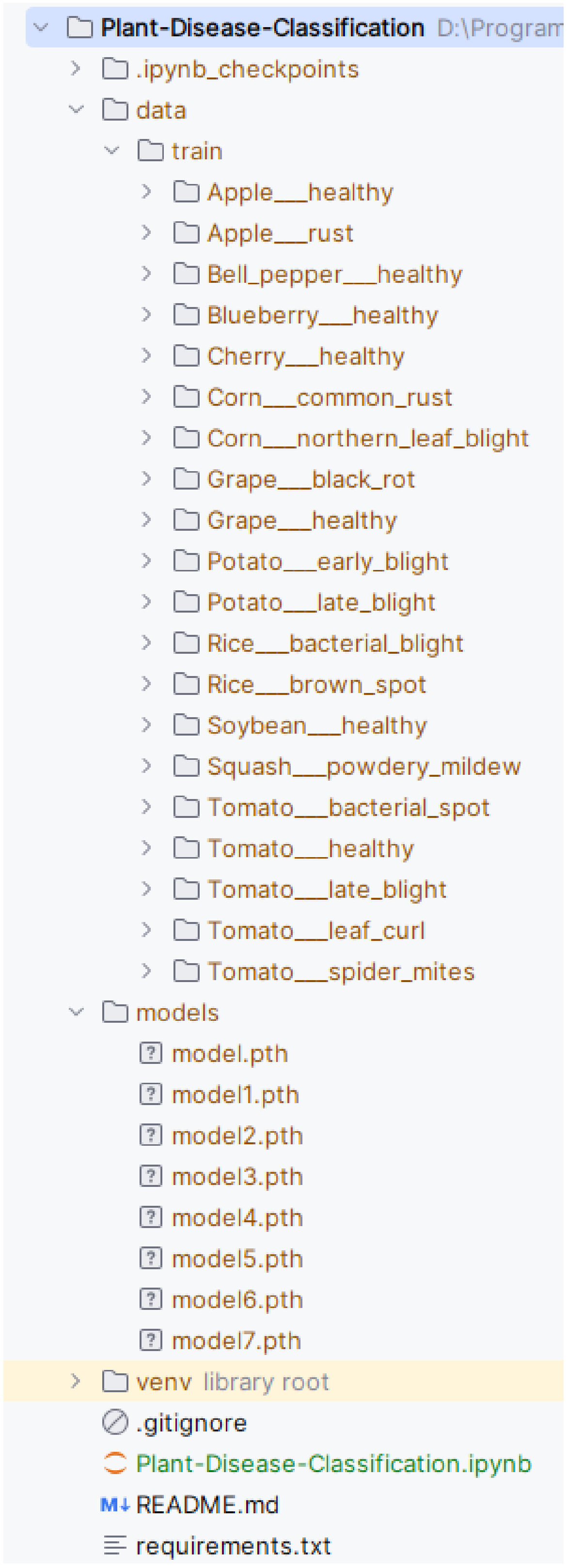
To find a suitable dataset we used the website kaggle.com, which contains a huge database of different types of datasets. The dataset we selected contains about 95,000 images divided into 60 classes of classified plants along with their diseases, and is a compound of other minor datasets.

c. Tools analysis

- Python, with libraries such as:
  - PyTorch – was selected over TensorFlow, because it is more accessible and easier to use
  - Matplotlib – visualisations
  - OpenCV2 – image processing
  - Numpy and pandas – arithmetical operations
- Jupiter Notebook – better code presentation and output visualization
- CUDA – architecture that enables GPU usage in data processing

3. Specification

a. Project structure



```
Plant-Disease-Classification  D:\Program
  .ipynb_checkpoints
  data
    train
      Apple___healthy
      Apple___rust
      Bell_pepper___healthy
      Blueberry___healthy
      Cherry___healthy
      Corn___common_rust
      Corn___northern_leaf_blight
      Grape___black_rot
      Grape___healthy
      Potato___early_blight
      Potato___late_blight
      Rice___bacterial_blight
      Rice___brown_spot
      Soybean___healthy
      Squash___powdery_mildew
      Tomato___bacterial_spot
      Tomato___healthy
      Tomato___late_blight
      Tomato___leaf_curl
      Tomato___spider_mites
  models
    model.pth
    model1.pth
    model2.pth
    model3.pth
    model4.pth
    model5.pth
    model6.pth
    model7.pth
  venv  library root
  .gitignore
  Plant-Disease-Classification.ipynb
  README.md
  requirements.txt
```

Screenshot 1. Project structure

b. Code structure

1. Data preparation

a. Creating image path lists

Creating lists of paths where data is saved. Lists are divided - 80% in training set and 20% in test set. Paths are shuffled before assigning.

```python
train_data_path = 'data\\train'

classes, train_image_paths  = [], []

for data_path in glob(train_data_path + '\\*'):
    classes.append(" ".join(" - ".join(data_path.split('\\')[-1].split('___')).split('_')))
    train_image_paths.append(glob(data_path + '\\*'))

train_image_paths = list(pandas.core.common.flatten(train_image_paths))

split_index = int(len(train_image_paths) * 0.8)
random.shuffle(train_image_paths)
train_image_paths, test_image_paths = train_image_paths[:split_index], train_image_paths[split_index:]

print('class example: ', classes[0])
print('Path example: ', train_image_paths[0], end='\n\n')

print(f'Classes size: {len(classes)}', end='\n\n')

print(f'Train size: {len(train_image_paths)}')
print(f'Test size: {len(test_image_paths)}')
```

Output:

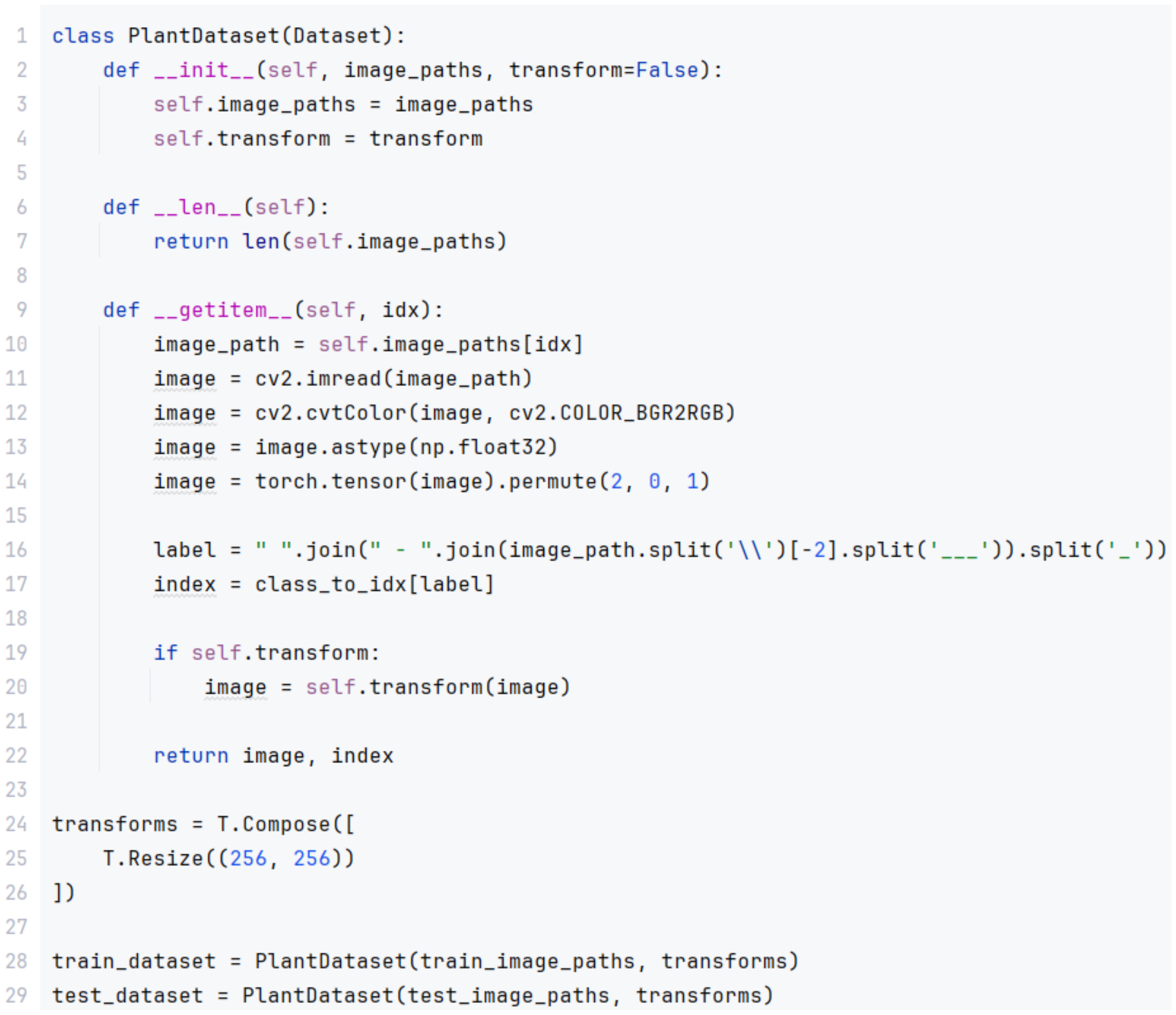
```
class example:  Apple - healthy
Path example:  data\train\Tomato___Late_blight\94f53287-b156-4ece-94a2-3c12783e4fbb___RS_Late.B 5409.JPG

Classes size: 20

Train size: 22114
Test size: 5529
```

b. Mapping classes with indexes

```python
idx_to_class = {i: j for i, j in enumerate(classes)}
class_to_idx = {value: key for key, value in idx_to_class.items()}
```

c. Creating Datasets

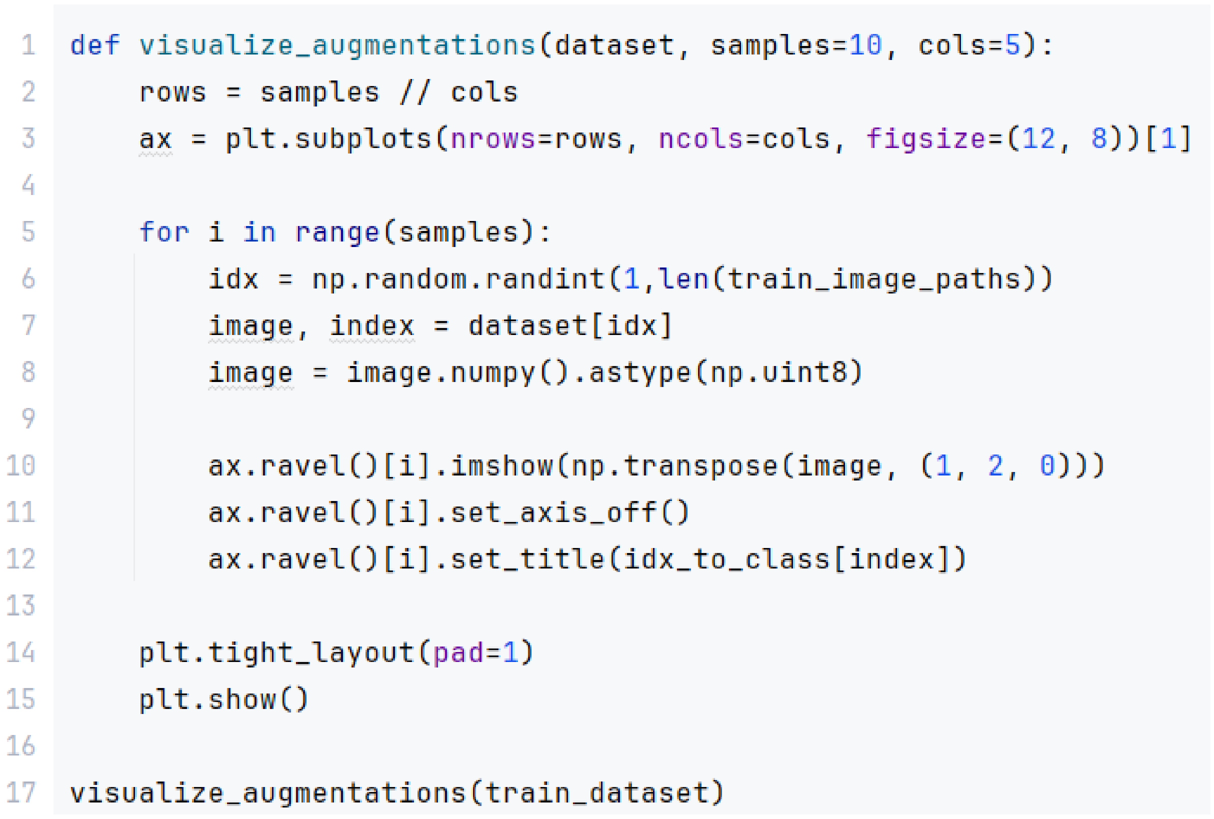
Datasets contain images loaded from provided paths. Each image gets label that inform about type and disease of following plant and is scaled to 256x256 pixels.



```python
class PlantDataset(Dataset):
    def __init__(self, image_paths, transform=False):
        self.image_paths = image_paths
        self.transform = transform

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image_path = self.image_paths[idx]
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = image.astype(np.float32)
        image = torch.tensor(image).permute(2, 0, 1)

        label = " ".join(" - ".join(image_path.split('\\')[-2].split('___')).split('_'))
        index = class_to_idx[label]

        if self.transform:
            image = self.transform(image)

        return image, index

transforms = T.Compose([
    T.Resize((256, 256))
])

train_dataset = PlantDataset(train_image_paths, transforms)
test_dataset = PlantDataset(test_image_paths, transforms)
```

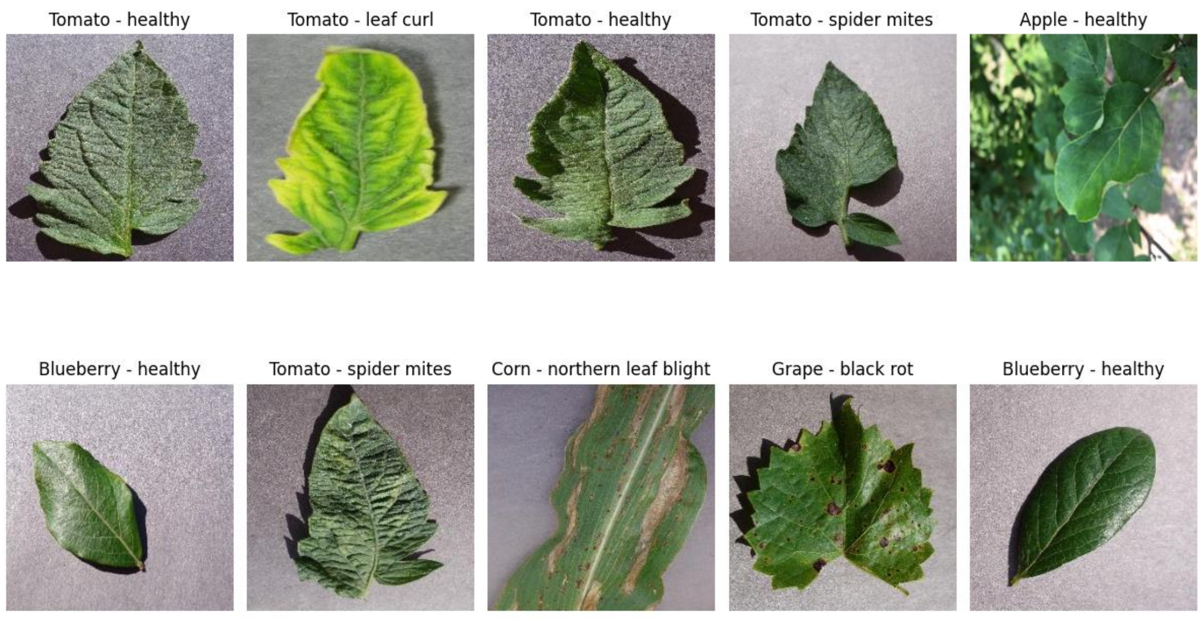
d. Data visualization

Visualization of 10 random images from training set.



```python
def visualize_augmentations(dataset, samples=10, cols=5):
    rows = samples // cols
    ax = plt.subplots(nrows=rows, ncols=cols, figsize=(12, 8))[1]

    for i in range(samples):
        idx = np.random.randint(1,len(train_image_paths))
        image, index = dataset[idx]
        image = image.numpy().astype(np.uint8)

        ax.ravel()[i].imshow(np.transpose(image, (1, 2, 0)))
        ax.ravel()[i].set_axis_off()
        ax.ravel()[i].set_title(idx_to_class[index])

    plt.tight_layout(pad=1)
    plt.show()

visualize_augmentations(train_dataset)
```

Output:



e. Defining dataloaders

```python
train_loader = DataLoader(
    train_dataset, batch_size=32, shuffle=True
)

test_loader = DataLoader(
    test_dataset, batch_size=32, shuffle=False
)
```

2. AI model

a. Choosing data processing device.

```python
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
```

b. Defining AI model structure

```python
def convBlock(in_channels, out_channels, pool=False):
    layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
             nn.ReLU(inplace=True)]
    if pool:
        layers.append(nn.MaxPool2d(4))
    return nn.Sequential(*layers)
```

```python
class NeuralNetwork(nn.Module):

    def __init__(self):
        super().__init__()

        self.conv1 = convBlock(3, 64)
        self.conv2 = convBlock(64, 128, pool=True)
        self.res1 = nn.Sequential(convBlock(128, 128), convBlock(128, 128))

        self.conv3 = convBlock(128, 256, pool=True)
        self.res2 = nn.Sequential(convBlock(256, 256), convBlock(256, 256))

        self.classifier = nn.Sequential(nn.AdaptiveAvgPool2d(1),
                                         nn.Flatten(),
                                         nn.Linear(256, len(classes)))

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.res1(out)
        out = F.interpolate(out, scale_factor=2, mode='nearest')
        out = self.conv3(out)
        out = self.res2(out)
        out = F.interpolate(out, scale_factor=2, mode='nearest')
        out = self.classifier(out)
        return out
```

c. Model creation

```
net = NeuralNetwork().to(device)
print(net)
```

Output:

```
NeuralNetwork(
  (conv1): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
  )
  (conv2): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (res1): Sequential(
    (0): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
    )
  )
  (conv3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=4, stride=4, padding=0, dilation=1, ceil_mode=False)
  )
  (res2): Sequential(
    (0): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
    )
    (1): Sequential(
      (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): ReLU(inplace=True)
    )
  )
  (classifier): Sequential(
    (0): AdaptiveAvgPool2d(output_size=1)
    (1): Flatten(start_dim=1, end_dim=-1)
    (2): Linear(in_features=256, out_features=20, bias=True)
  )
)
```

d. Optimizer and criterion creation

```python
criterion = nn.CrossEntropyLoss()
optimizer = Adam(net.parameters(), lr=1e-3)
```

e. Model training and validation

Training model and recording losses and accuracy either training and test DataLoader at the end of each epoch.

```python
start = torch.cuda.Event(enable_timing=True)
end = torch.cuda.Event(enable_timing=True)
train_losses, val_losses, train_accs, val_accs = [], [], [], []

start.record()

for epoch in range(10):

    train_loss = 0.0
    train_correct = 0
    train_total = 0
    net.train()
    for i, train_data in enumerate(train_loader, 0):

        train_inputs, train_labels = train_data
        train_inputs, train_labels = train_inputs.to(device), train_labels.to(device)

        optimizer.zero_grad()
        outputs = net(train_inputs)
        loss = criterion(outputs, train_labels)
        loss.backward()
        optimizer.step()

        train_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        train_correct += (predicted == train_labels).sum().item()
    train_loss_avg = train_loss / len(train_loader)
    train_acc = train_correct / len(train_loader.dataset)
    print(f'[Epoch {epoch + 1}] Train loss: {train_loss_avg:.3f}')
    print(f'[Epoch {epoch + 1}] Train acc: {train_acc:.3f}')

    valid_loss = 0.0
    valid_correct = 0
    valid_total = 0
    net.eval()
    with torch.no_grad():
        for i, valid_data in enumerate(test_loader, 0):
            valid_inputs, valid_labels = valid_data
            valid_inputs, valid_labels = valid_inputs.to(device), valid_labels.to(device)

            valid_outputs = net(valid_inputs)
            loss = criterion(valid_outputs, valid_labels)
            valid_loss += loss.item()

            _, predicted = torch.max(valid_outputs.data, 1)
            valid_correct += (predicted == valid_labels).sum().item()

    valid_loss_avg = valid_loss / len(test_loader)
    valid_acc = valid_correct / len(test_loader.dataset)
    print(f'[Epoch {epoch + 1}] Valid loss: {valid_loss_avg:.3f}')
    print(f'[Epoch {epoch + 1}] Valid acc: {valid_acc:.2%}')

    train_losses.append(train_loss_avg)
    train_accs.append(train_acc)
    val_losses.append(valid_loss_avg)
    val_accs.append(valid_acc)
```

```python
end.record()
torch.cuda.synchronize()

print('Finished Training')
print(start.elapsed_time(end))
```

Output:

```
[1, 692] Train loss: 2.742
[1, 692] Train acc: 0.138
[Epoch 1] Valid loss: 5.239
[Epoch 1] Valid acc: 17.04%
[2, 692] Train loss: 1.305
[2, 692] Train acc: 0.586
[Epoch 2] Valid loss: 0.813
[Epoch 2] Valid acc: 73.65%
[3, 692] Train loss: 0.573
[3, 692] Train acc: 0.821
[Epoch 3] Valid loss: 0.581
[Epoch 3] Valid acc: 82.46%
```

f. Save model to file

```python
torch.save(net.state_dict(), "./models/model8.pth")
```

3. Testing

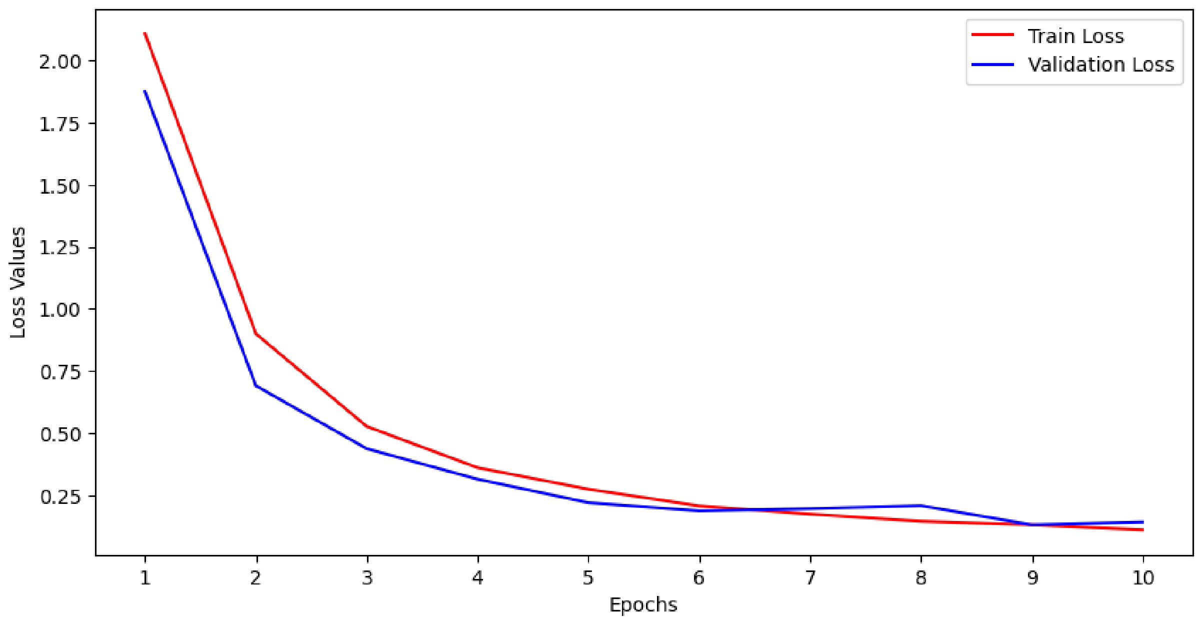
a. Load model from file

```python
net.load_state_dict(torch.load("./models/model8.pth"))
net.eval()
net.to(device)
```

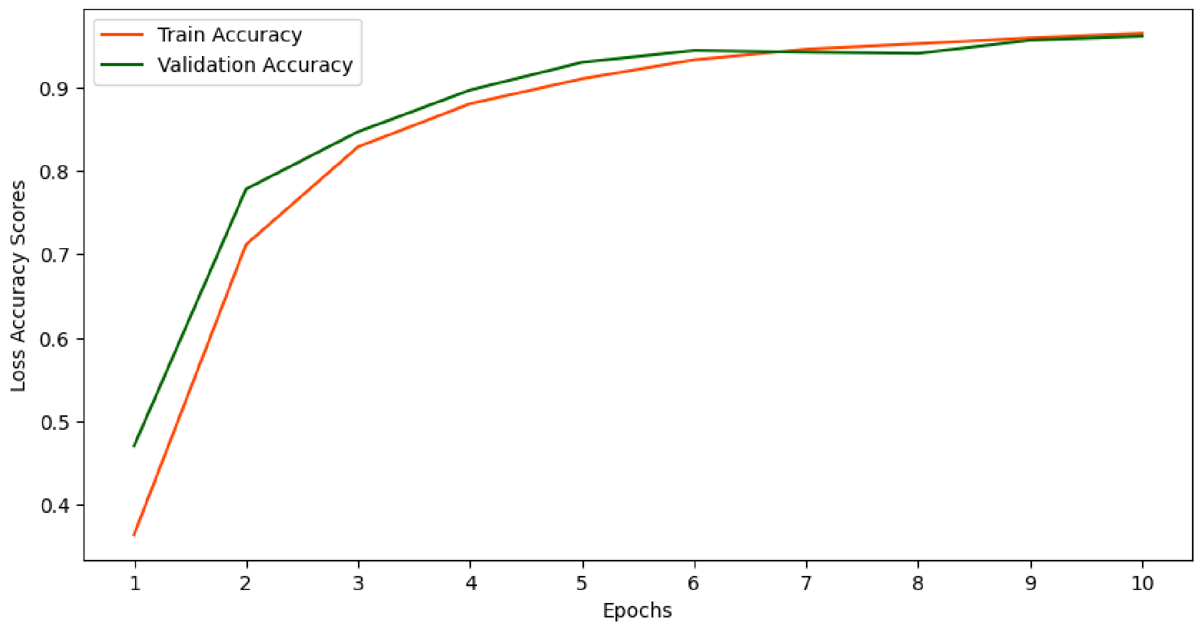
b. Drawing loss and accuracy plots

```python
plt.figure(figsize = (10, 5))
plt.plot(train_losses, label = "Train Loss", c = "red")
plt.plot(val_losses, label = "Validation Loss", c = "blue")
plt.xlabel("Epochs")
plt.ylabel("Loss Values")
plt.xticks(ticks = np.arange(len(train_losses)), labels = [i for i in range(1, len(train_losses) + 1)])
plt.legend()
plt.show()

plt.figure(figsize = (10, 5))
plt.plot(train_accs, label = "Train Accuracy", c = "orangered")
plt.plot(val_accs, label = "Validation Accuracy", c = "darkgreen")
plt.xlabel("Epochs")
plt.ylabel("Loss Accuracy Scores")
plt.xticks(ticks = np.arange(len(train_accs)), labels = [i for i in range(1, len(train_accs) + 1)])
plt.legend()
plt.show()
```

Output:



Train Loss
Validation Loss
Loss Values
Epochs
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
1
2
3
4
5
6
7
8
9
10



Train Accuracy
Validation Accuracy
Loss Accuracy Scores
Epochs
0.9
0.8
0.7
0.6
0.5
0.4
1
2
3
4
5
6
7
8
9
10

c. Function for getting predicted and actual labels for provided model and DataLoader

```python
def get_preds_and_actual(model, loader):
    predictions = torch.tensor([]).to(device)
    actuals = torch.tensor([]).to(device)
    for batch in loader:
        images, actual = batch
        images, actual = images.to(device), actual.to(device)

        pred = model(images)
        predictions = torch.cat((predictions, pred),dim=0)
        actuals = torch.cat((actuals, actual),dim=0)

    return predictions.argmax(dim=1).int(), actuals.int()
```

d. Function for getting correctness of the model

```python
def get_correct(predictions, actuals):
    result = 0
    for i, _ in enumerate(predictions):
        if predictions[i] == actuals[i]:
            result += 1
    return result
```

e. Getting predictions and actuals

```python
with torch.no_grad():
    predictions, actuals = get_preds_and_actual(net, test_loader)
```

f. Getting model's correctness

```python
correct = get_correct(predictions, actuals)
print(f'Accuracy of the network on the valid images: {(correct / len(actuals)):.2%}')
```

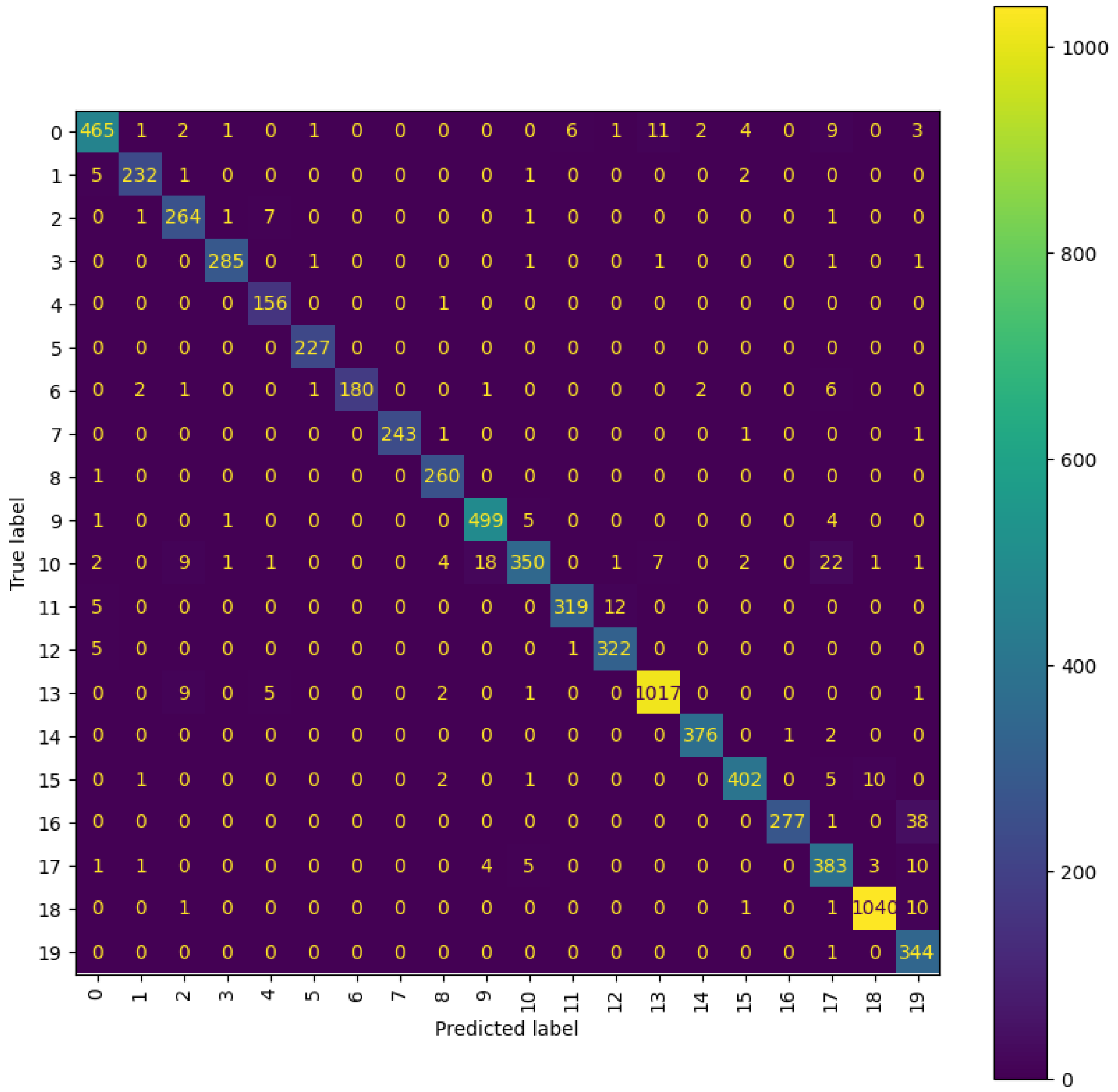
Output:

```
Accuracy of the network on the valid images: 94.16%
```

g. Draw confusion matrix

```python
cm = confusion_matrix(actuals.cpu(), predictions.cpu())

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

fig, ax = plt.subplots(figsize=(10, 10))
disp.plot(ax=ax, cmap='viridis', xticks_rotation='vertical')
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.grid(False)

plt.show()
```

Output:

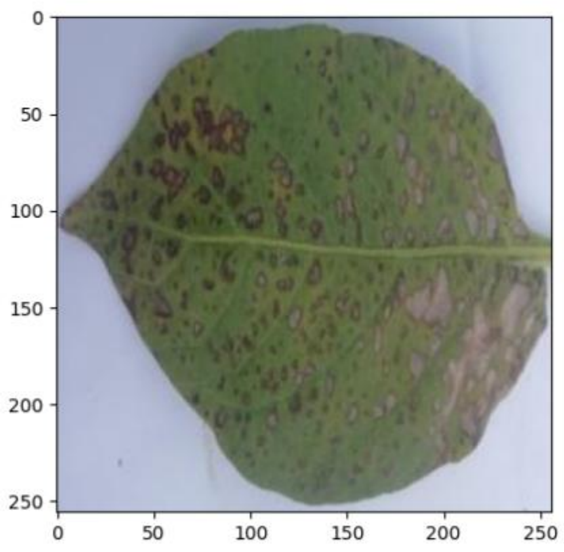


True label \ Predicted label	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	465	1	2	1	0	1	0	0	0	0	0	6	1	11	2	4	0	9	0	3
1	5	232	1	0	0	0	0	0	0	0	1	0	0	0	0	2	0	0	0	0
2	0	1	264	1	7	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
3	0	0	0	285	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	1
4	0	0	0	0	156	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	227	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	2	1	0	0	1	180	0	0	1	0	0	0	0	2	0	0	6	0	0
7	0	0	0	0	0	0	0	243	1	0	0	0	0	0	0	1	0	0	0	1
8	1	0	0	0	0	0	0	0	260	0	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0	499	5	0	0	0	0	0	0	4	0	0
10	2	0	9	1	1	0	0	0	4	18	350	0	1	7	0	2	0	22	1	1
11	5	0	0	0	0	0	0	0	0	0	0	319	12	0	0	0	0	0	0	0
12	5	0	0	0	0	0	0	0	0	0	0	1	322	0	0	0	0	0	0	0
13	0	0	9	0	5	0	0	0	2	0	1	0	0	1017	0	0	0	0	0	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	376	0	1	2	0	0
15	0	1	0	0	0	0	0	0	2	0	1	0	0	0	0	402	0	5	10	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	277	1	0	38
17	1	1	0	0	0	0	0	0	0	4	5	0	0	0	0	0	0	383	3	10
18	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1040	10
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	344

h. Showing random image and model prediction about its class

```python
image, index = next(iter(train_loader))

image = image.to(device)
pred = net(image)

pred_label = pred.argmax(dim=1).int()
pred_value = torch.nn.functional.softmax(pred[0], dim=0)
percentage = (pred_value.max() * 100).cpu().item()

image = image[0].cpu().numpy().astype(np.uint8)
plt.imshow(np.transpose(image, (1, 2, 0)))
plt.show()
print(f"Prediction: {percentage:.2f}% | Prediction: {idx_to_class[pred_label[0].item()]} | Actual: {idx_to_class[index[0].item()]}")
```

Output:



```
Prediction: 84.26% | Prediction: Potato - early blight | Actual: Potato - early blight
```

c. Interface

To visualize the results and comment blocks of code, we used Markdown and the outputs in Jupyter Notebook.

4. Experiments

The first models we created had a simple structure and relatively short training times (30-40 minutes), but their results were poor. After developing a model with better accuracy, we began further experiments, gradually changing various parameters to improve the model's accuracy. Initially, we trained our models on a complete dataset containing 60 classes. Training on the entire dataset was time-consuming, and the model we created was too simple to handle such a large number of different classes. Thus, after consulting with our supervisor, we decided to reduce the number of classes. To do this, we created a confusion matrix and used it to eliminate the worst-performing classes.

All our models utilized the convBlock structure we designed. This structure features a two-dimensional convolution layer that processes the image data. By incorporating additional layers, we increased the number of input and output channels. The convBlock also includes a ReLU activation layer and an optional MaxPool2d layer, which reduces the number of data points by pooling adjacent pixel values into a single value. This pooling reduces the data size without compromising accuracy. Once the data has passed through all the convolution layers, the data passes through an adaptive averaging layer that reduces the dimensions to 1. Then, the data is flattened into one-dimensional vectors. Finally, the processed vectors pass through a linear layer that generates classification results for each class.

- ## Model 1

Model Structure

```
self.conv1 = convBlock(3, 64)
self.conv2 = convBlock(64, 128, pool=True)
self.res1 = nn.Sequential(convBlock(128, 128), convBlock(128, 128))

self.conv3 = convBlock(128, 256, pool=True)
self.conv4 = convBlock(256, 512, pool=True)
self.res2 = nn.Sequential(convBlock(512, 512), convBlock(512, 512))

self.classifier = nn.Sequential(nn.AdaptiveAvgPool2d(1),
                                nn.Flatten(),
                                nn.Linear(512, len(classes)))
```

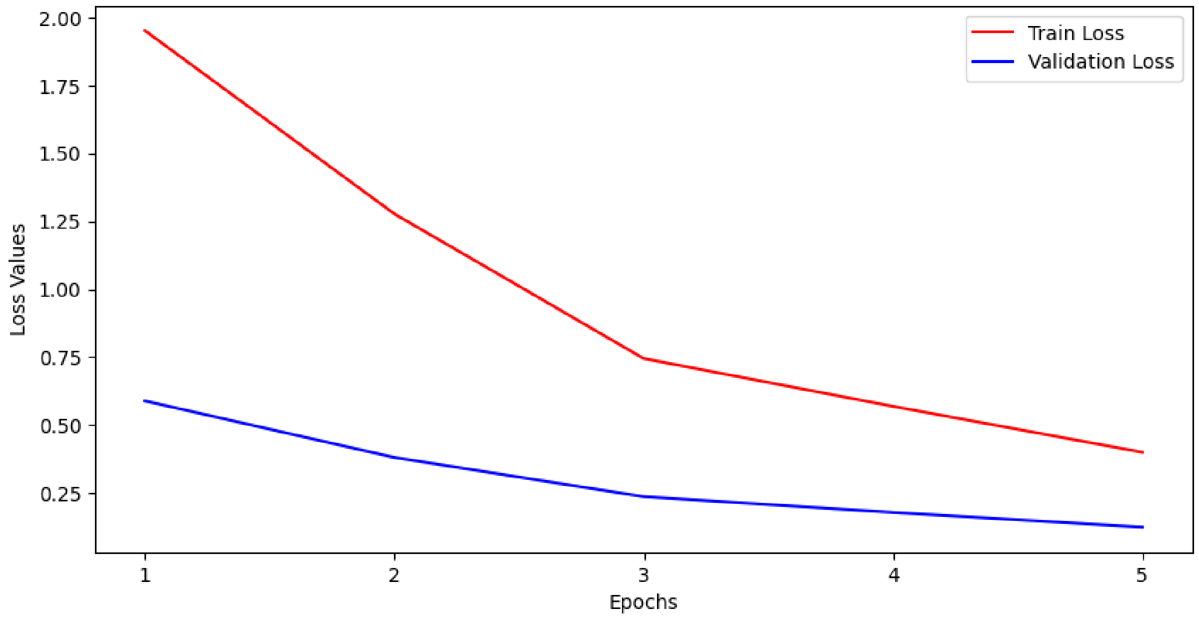
Characteristics

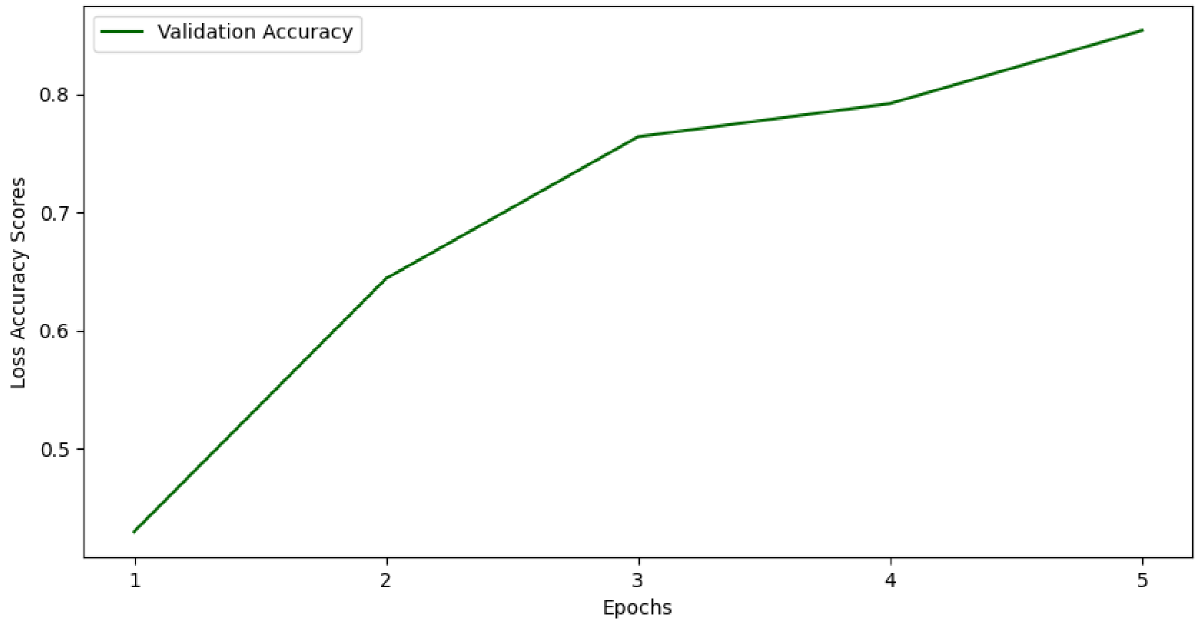
Classes: 30

Training time: 13h15m23s

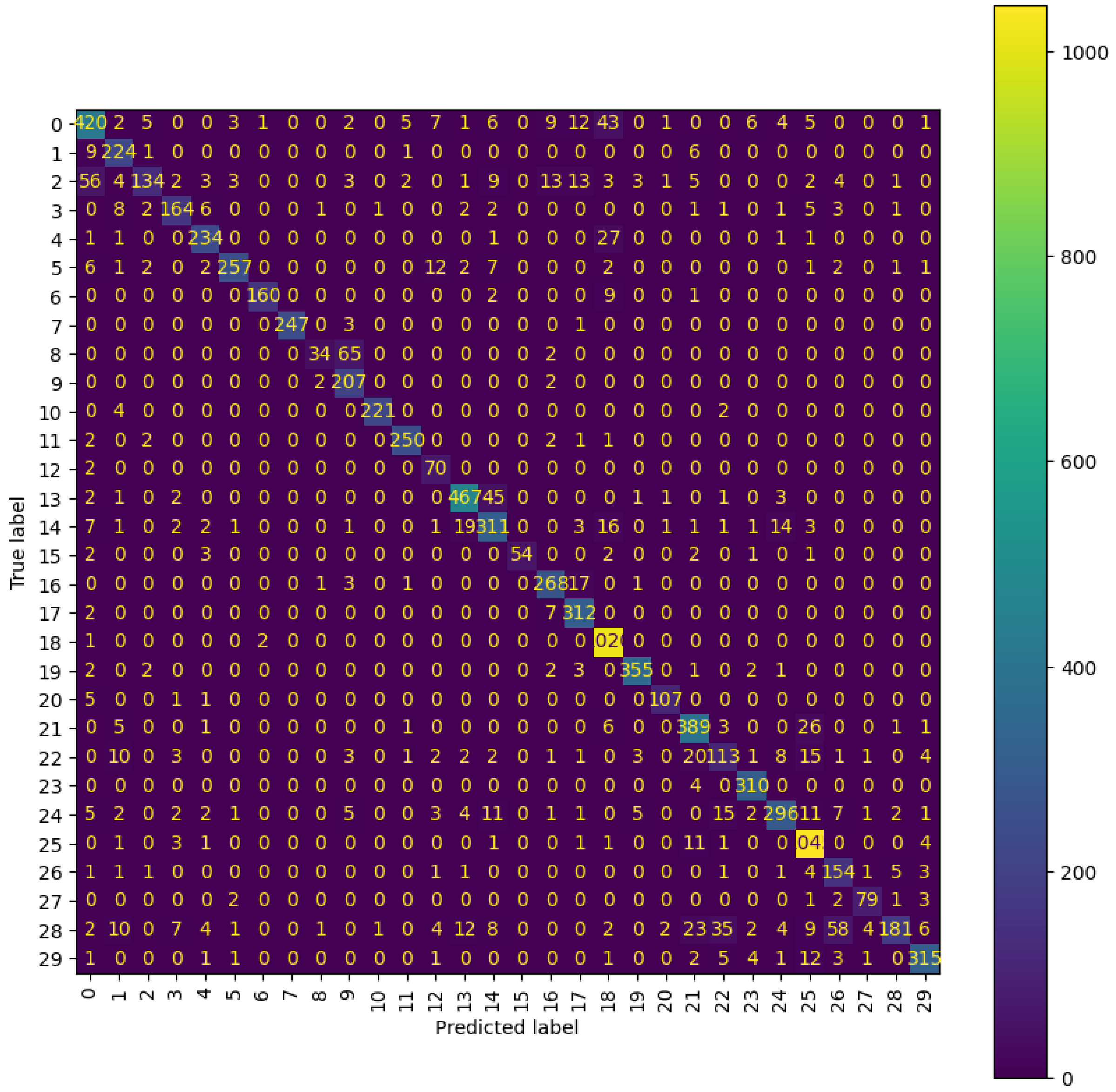
Accuracy: 85.39%

Plots


Train Loss
Validation Loss
Loss Values
Epochs
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
1
2
3
4
5


Validation Accuracy
Loss Accuracy Scores
Epochs
0.8
0.7
0.6
0.5
1
2
3
4
5

Confusion matrix

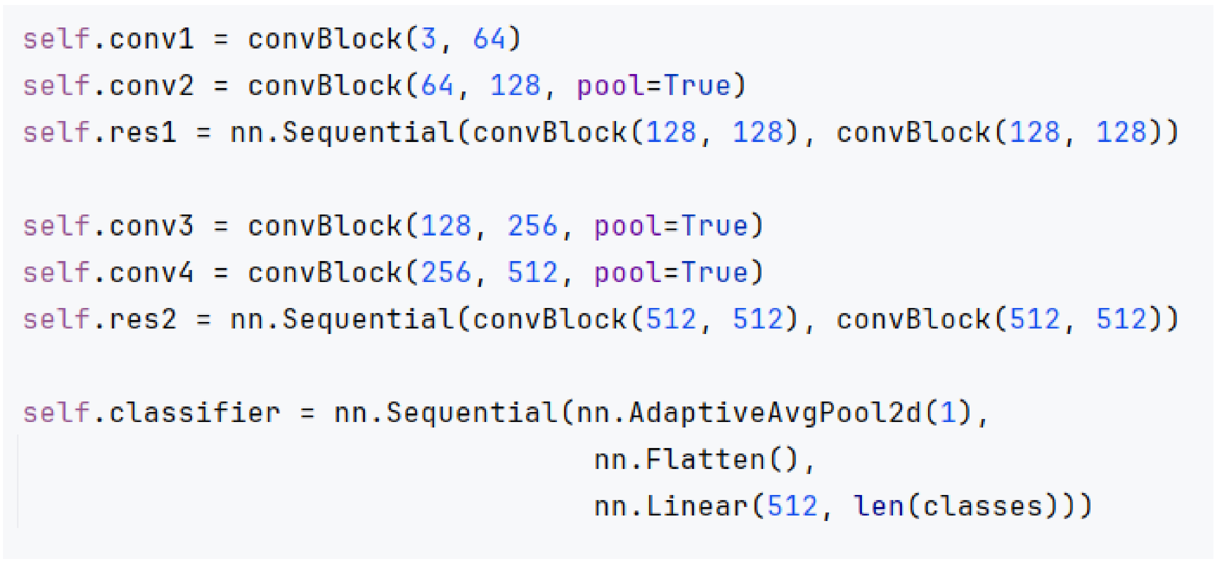


Summary

Analysing the loss graphs on the training set and the validation set, we note that the model generalizes well and does not under-fit the training data (overfitting effect).

- ## Model 2

Model Structure



```
self.conv1 = convBlock(3, 64)
self.conv2 = convBlock(64, 128, pool=True)
self.res1 = nn.Sequential(convBlock(128, 128), convBlock(128, 128))

self.conv3 = convBlock(128, 256, pool=True)
self.conv4 = convBlock(256, 512, pool=True)
self.res2 = nn.Sequential(convBlock(512, 512), convBlock(512, 512))

self.classifier = nn.Sequential(nn.AdaptiveAvgPool2d(1),
                                nn.Flatten(),
                                nn.Linear(512, len(classes)))
```

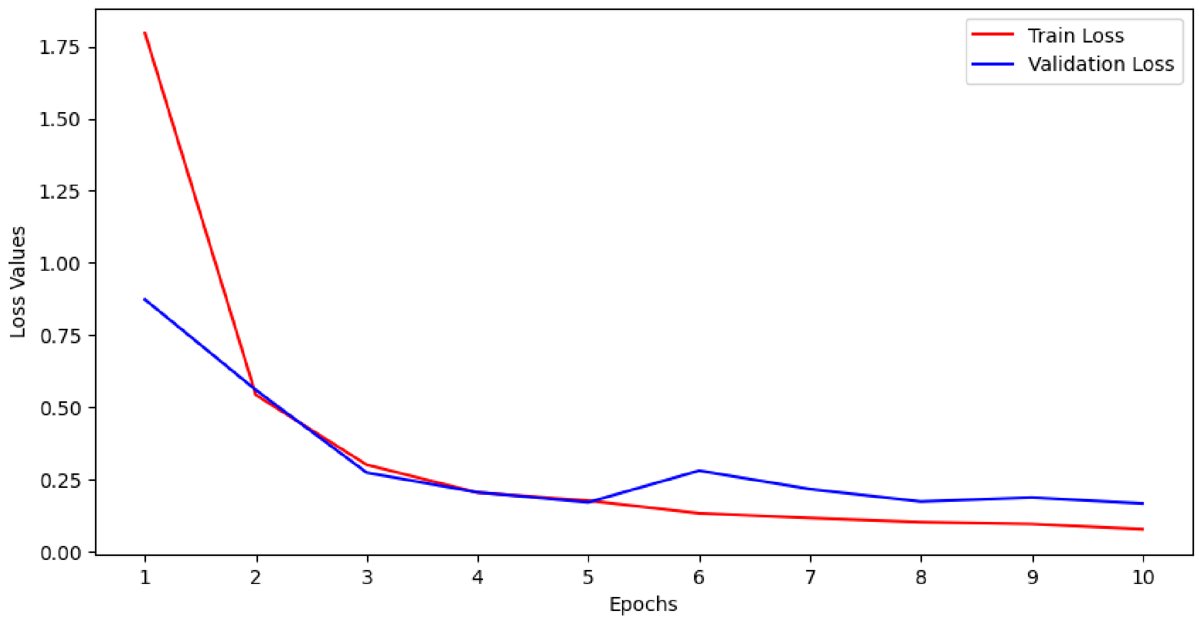
Characteristics

Classes: 20

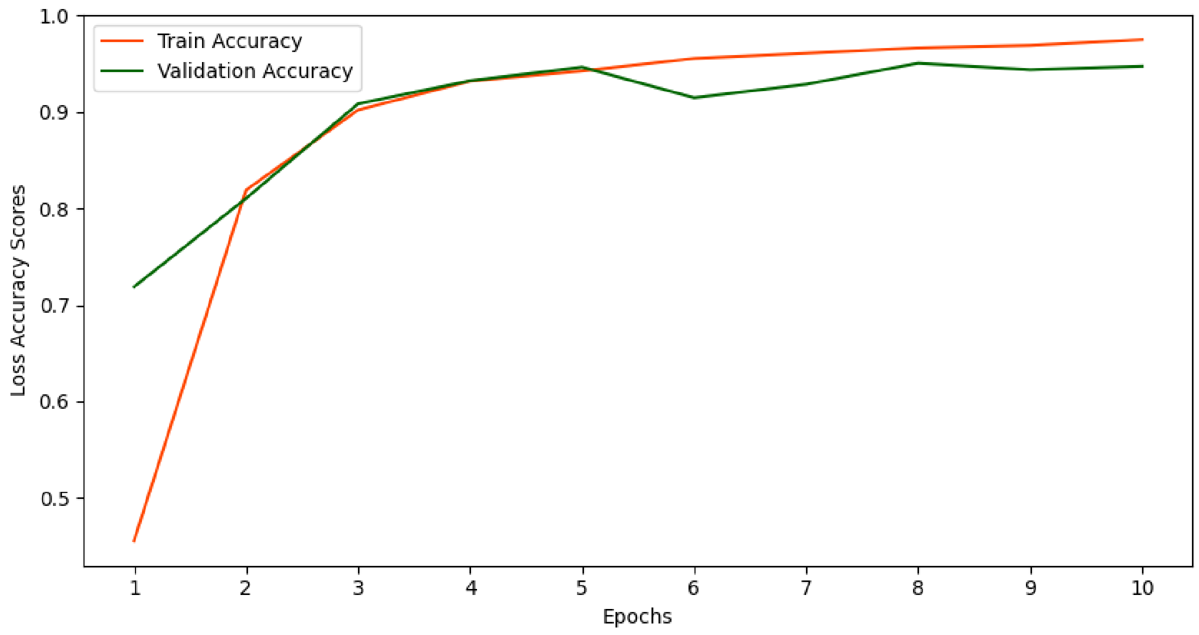
Training time: 17h56m03s

Accuracy: 94.75%

Plots

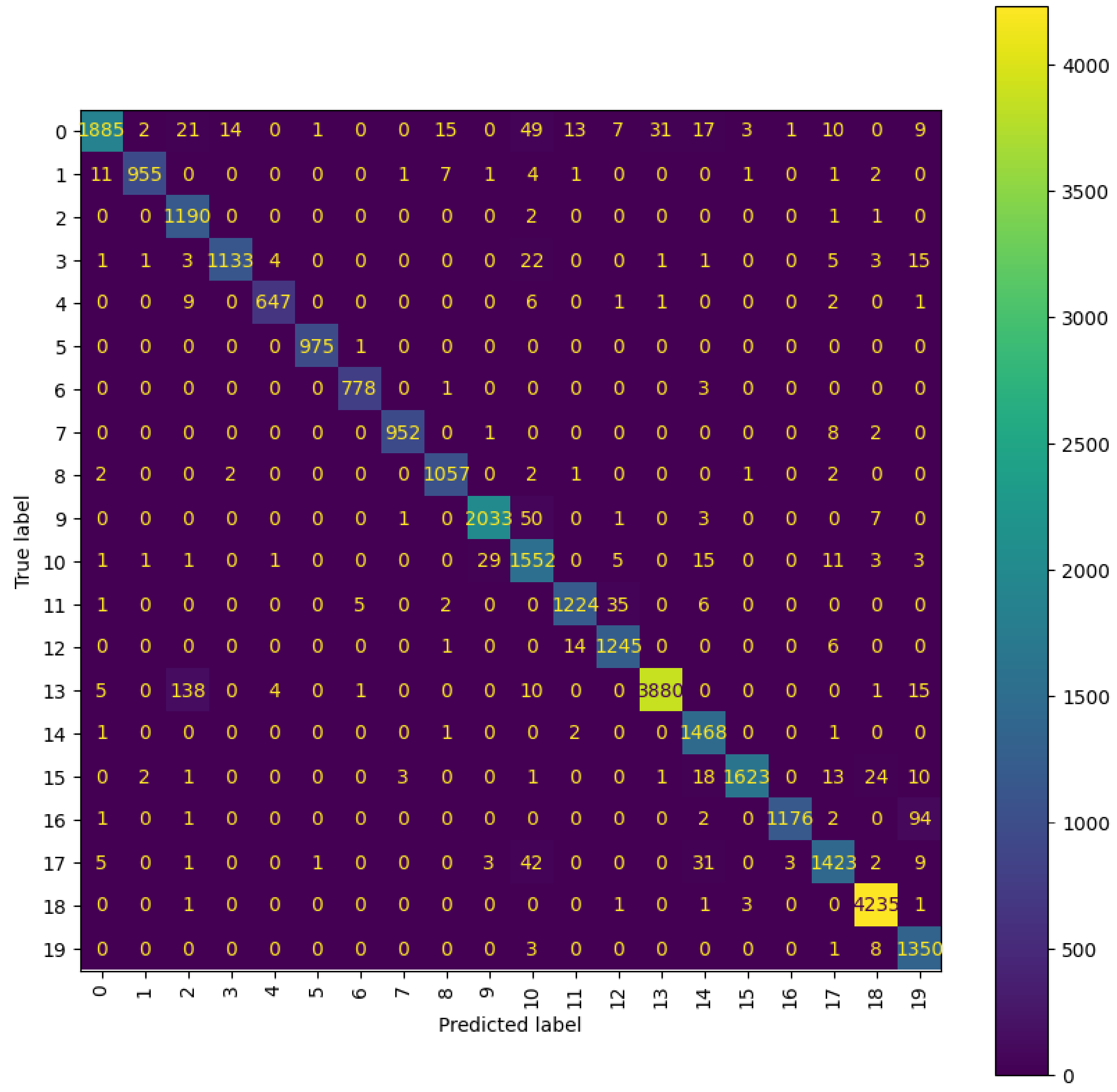


Train Loss
Validation Loss
Loss Values
Epochs
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
1
2
3
4
5
6
7
8
9
10



Train Accuracy
Validation Accuracy
Loss Accuracy Scores
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
1
2
3
4
5
6
7
8
9
10

Confusion matrix



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1885	2	21	14	0	1	0	0	15	0	49	13	7	31	17	3	1	10	0	9
1	11	955	0	0	0	0	0	1	7	1	4	1	0	0	0	1	0	1	2	0
2	0	0	1190	0	0	0	0	0	0	0	2	0	0	0	0	0	0	1	1	0
3	1	1	3	1133	4	0	0	0	0	0	22	0	0	1	1	0	0	5	3	15
4	0	0	9	0	647	0	0	0	0	0	6	0	1	1	0	0	0	2	0	1
5	0	0	0	0	0	975	1	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	778	0	1	0	0	0	0	0	3	0	0	0	0	0
7	0	0	0	0	0	0	0	952	0	1	0	0	0	0	0	0	0	8	2	0
8	2	0	0	2	0	0	0	0	1057	0	2	1	0	0	0	1	0	2	0	0
9	0	0	0	0	0	0	0	1	0	2033	50	0	1	0	3	0	0	0	7	0
10	1	1	1	0	1	0	0	0	0	29	1552	0	5	0	15	0	0	11	3	3
11	1	0	0	0	0	0	5	0	2	0	0	1224	35	0	6	0	0	0	0	0
12	0	0	0	0	0	0	0	0	1	0	0	14	1245	0	0	0	0	6	0	0
13	5	0	138	0	4	0	1	0	0	0	10	0	0	3880	0	0	0	0	1	15
14	1	0	0	0	0	0	0	0	1	0	0	2	0	0	1468	0	0	1	0	0
15	0	2	1	0	0	0	0	3	0	0	1	0	0	1	18	1623	0	13	24	10
16	1	0	1	0	0	0	0	0	0	0	0	0	0	0	2	0	1176	2	0	94
17	5	0	1	0	0	1	0	0	0	3	42	0	0	0	31	0	3	1423	2	9
18	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	3	0	0	4235	1
19	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	1	8	1350

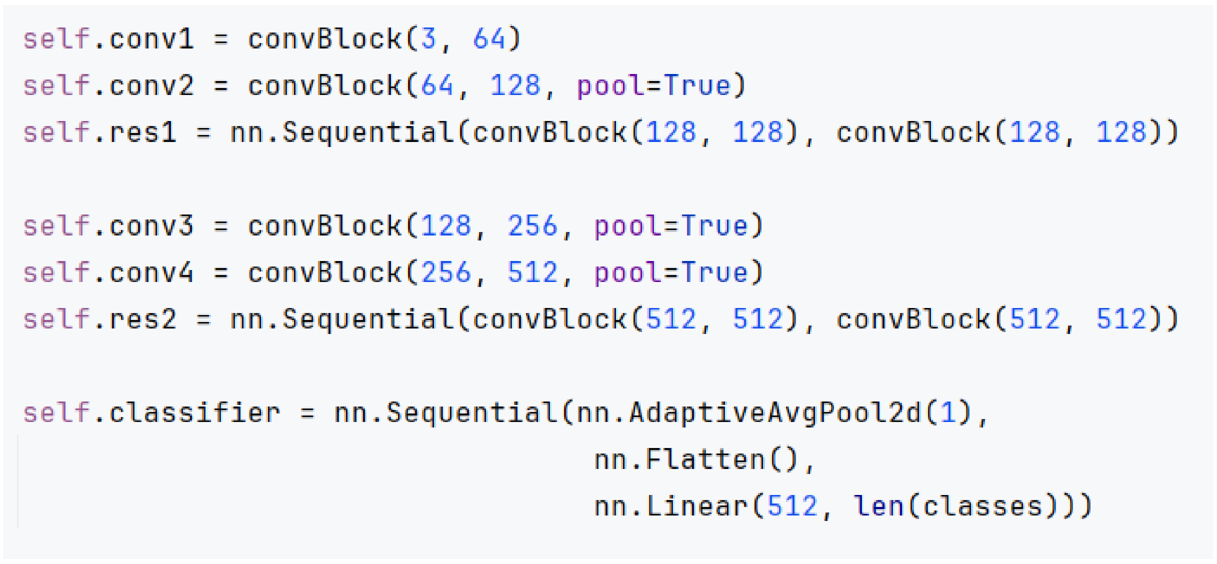
Summary

In model no. 2, we decided not to change the structure of the model. We increased the number of epochs, in order to improve model accuracy and generalization. We limited the number of classes to 20 in order to reduce the complexity of the model and to reduce training time.

We can see an improvement in the precision of the model over the previous one by 9.36%. After the 5th epoch, we can see a slight increase in losses on the validation set, which means that the model fits the training data too closely, affecting the final precision of the model.

- ## Model 3

Model Structure



```
self.conv1 = convBlock(3, 64)
self.conv2 = convBlock(64, 128, pool=True)
self.res1 = nn.Sequential(convBlock(128, 128), convBlock(128, 128))

self.conv3 = convBlock(128, 256, pool=True)
self.conv4 = convBlock(256, 512, pool=True)
self.res2 = nn.Sequential(convBlock(512, 512), convBlock(512, 512))

self.classifier = nn.Sequential(nn.AdaptiveAvgPool2d(1),
                                nn.Flatten(),
                                nn.Linear(512, len(classes)))
```

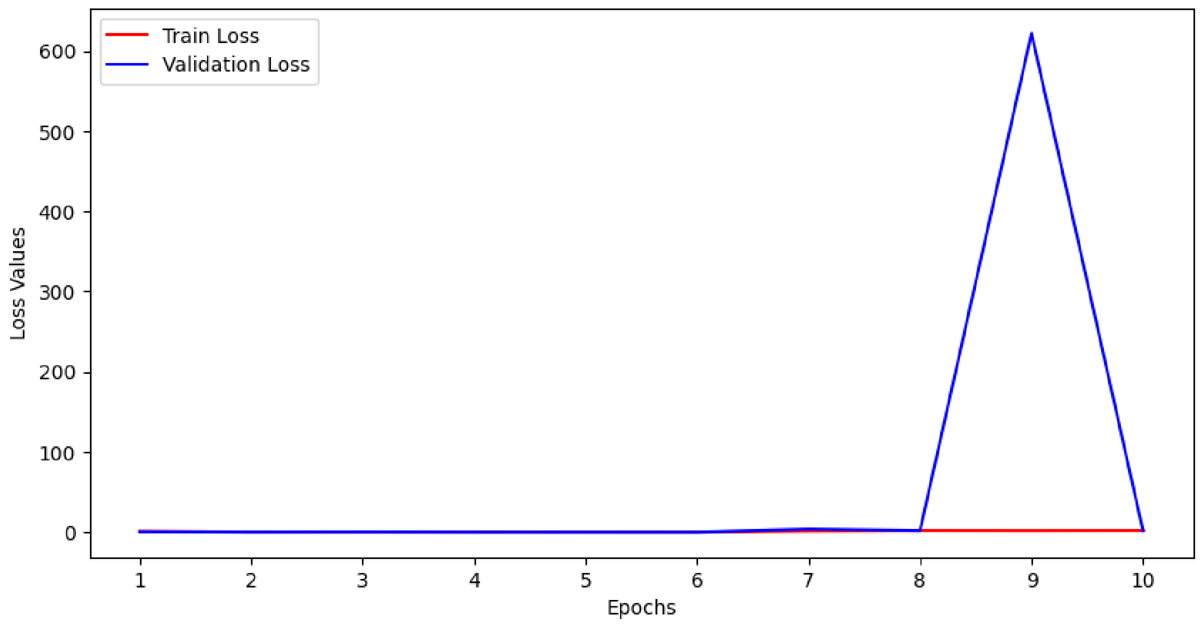
Characteristics

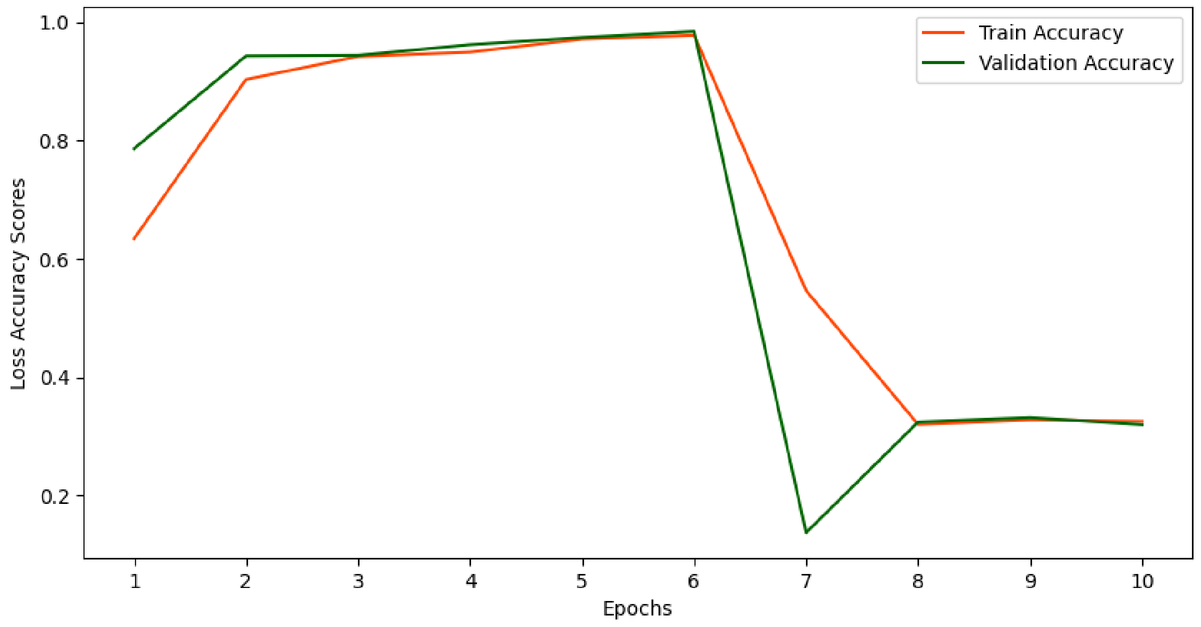
Classes: 10

Training time: 10h03m25s

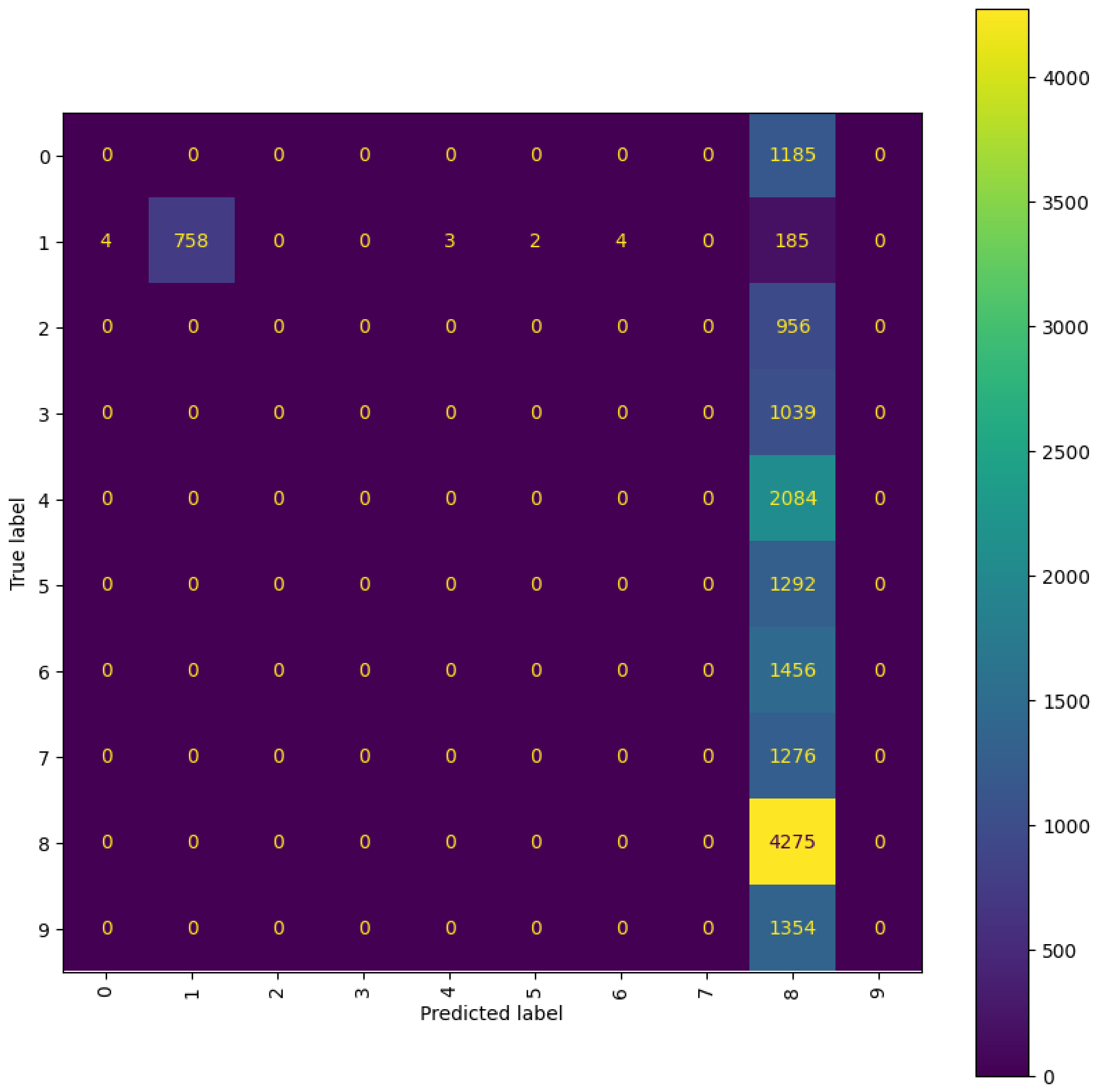
Accuracy: 31.95%

Plots


Train Loss
Validation Loss
Loss Values
600
500
400
300
200
100
0
1
2
3
4
5
6
7
8
9
10
Epochs


Train Accuracy
Validation Accuracy
Loss Accuracy Scores
1.0
0.8
0.6
0.4
0.2
1
2
3
4
5
6
7
8
9
10
Epochs

Confusion matrix



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	1185	0
1	4	758	0	0	3	2	4	0	185	0
2	0	0	0	0	0	0	0	0	956	0
3	0	0	0	0	0	0	0	0	1039	0
4	0	0	0	0	0	0	0	0	2084	0
5	0	0	0	0	0	0	0	0	1292	0
6	0	0	0	0	0	0	0	0	1456	0
7	0	0	0	0	0	0	0	0	1276	0
8	0	0	0	0	0	0	0	0	4275	0
9	0	0	0	0	0	0	0	0	1354	0

Summary

In model no. 3, we decided not to change the structure of the model or number of epochs. We limited the number of classes to 10.

By epoch 6, the precision of the validation set could be seen at 98%. In the next epochs, a drastic drop in precision appeared on both sets and stabilized at 32%. From the graphs and the confusion matrix, we can conclude that due to insufficient or unbalanced data, the model erroneously learned a single case instead of actual patterns. This resulted in the recognition of all images as class no. 8 images.

- Model 4

Model Structure

```
self.conv1 = convBlock(3, 64)
self.conv2 = convBlock(64, 128, pool=True)
self.res1 = nn.Sequential(convBlock(128, 128), convBlock(128, 128))

self.conv3 = convBlock(128, 256, pool=True)
self.conv4 = convBlock(256, 512, pool=True)
self.res2 = nn.Sequential(convBlock(512, 512), convBlock(512, 512))

self.classifier = nn.Sequential(nn.AdaptiveAvgPool2d(1),
                                nn.Flatten(),
                                nn.Linear(512, len(classes)))
```

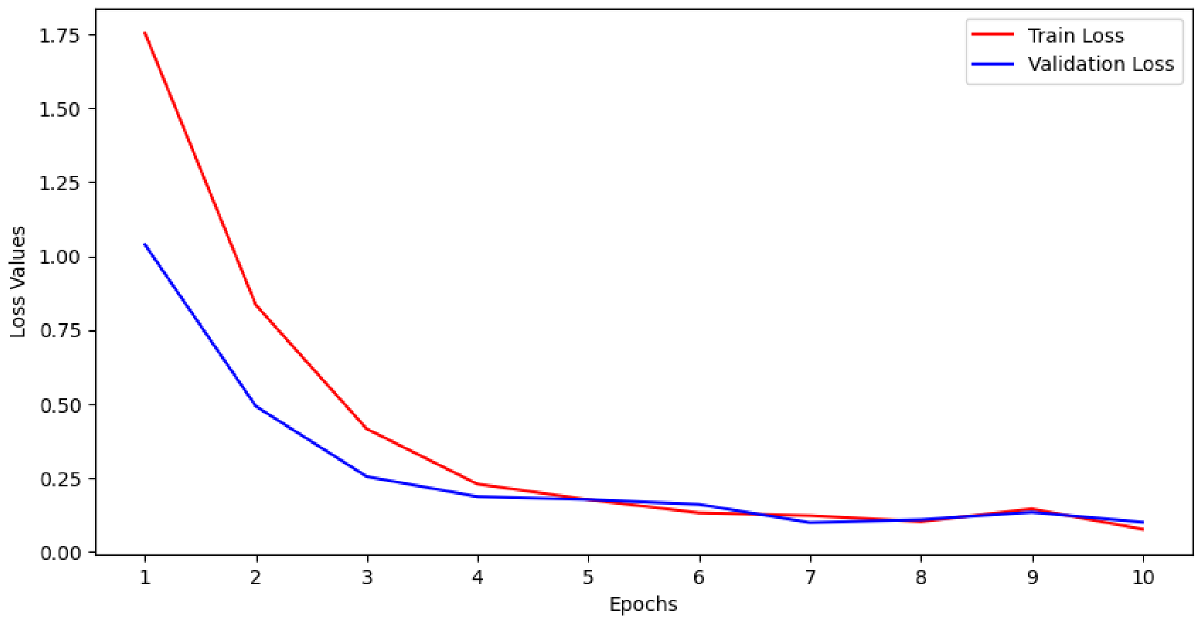
Characteristics

Classes: 10

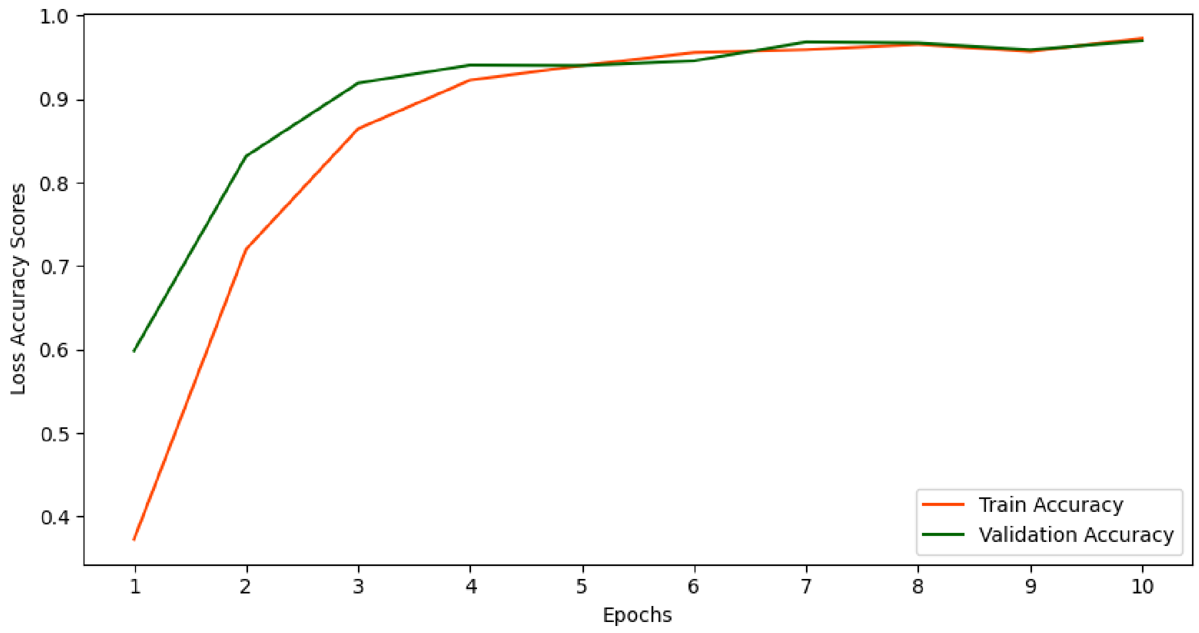
Training time: 6h25m45s

Accuracy: 96.99%

Plots

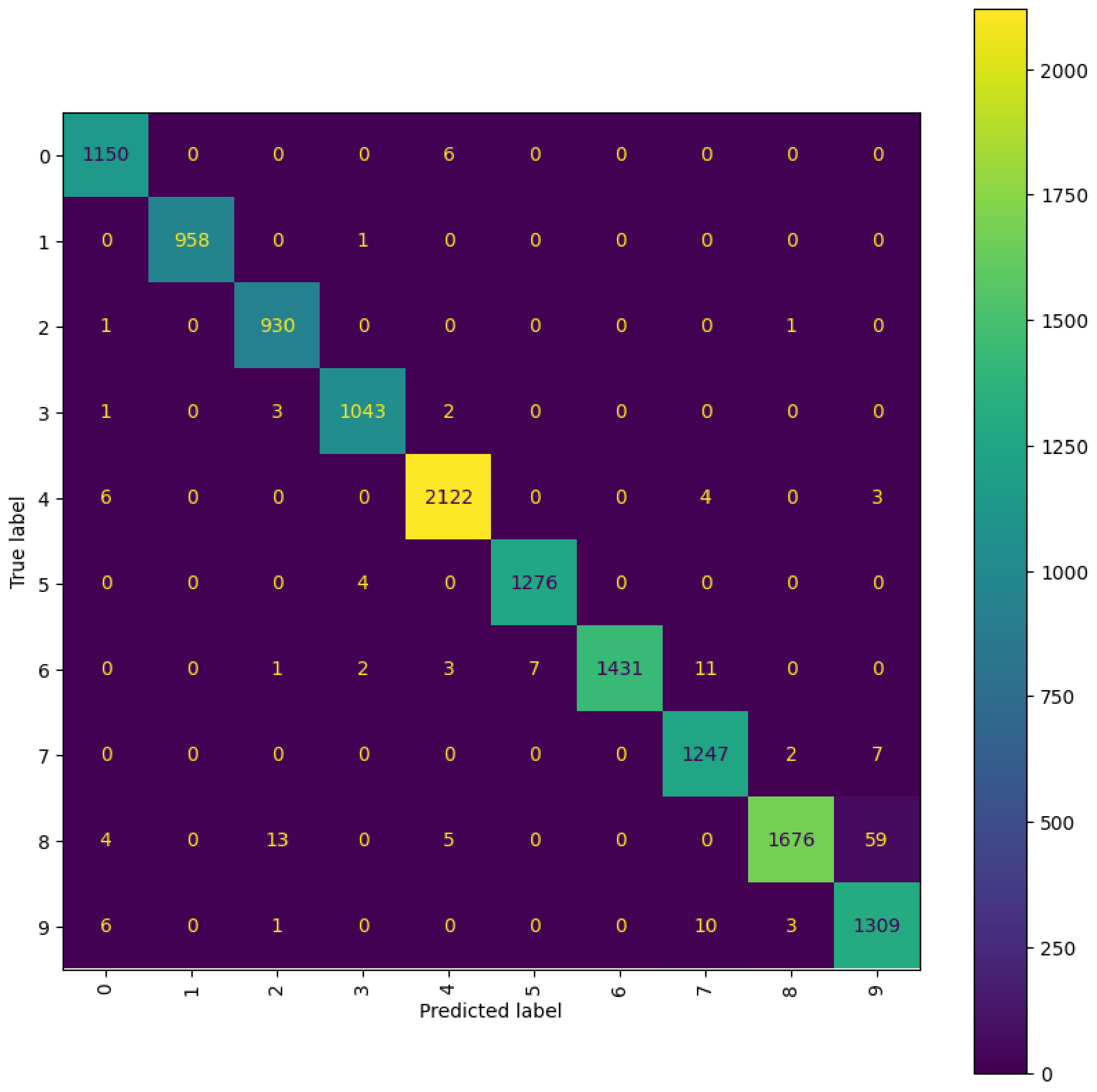


Train Loss
Validation Loss
Loss Values
Epochs
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
1
2
3
4
5
6
7
8
9
10



Train Accuracy
Validation Accuracy
Loss Accuracy Scores
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
0.4
1
2
3
4
5
6
7
8
9
10

Confusion matrix



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	1150	0	0	0	6	0	0	0	0	0
1	0	958	0	1	0	0	0	0	0	0
2	1	0	930	0	0	0	0	0	1	0
3	1	0	3	1043	2	0	0	0	0	0
4	6	0	0	0	2122	0	0	4	0	3
5	0	0	0	4	0	1276	0	0	0	0
6	0	0	1	2	3	7	1431	11	0	0
7	0	0	0	0	0	0	0	1247	2	7
8	4	0	13	0	5	0	0	0	1676	59
9	6	0	1	0	0	0	0	10	3	1309

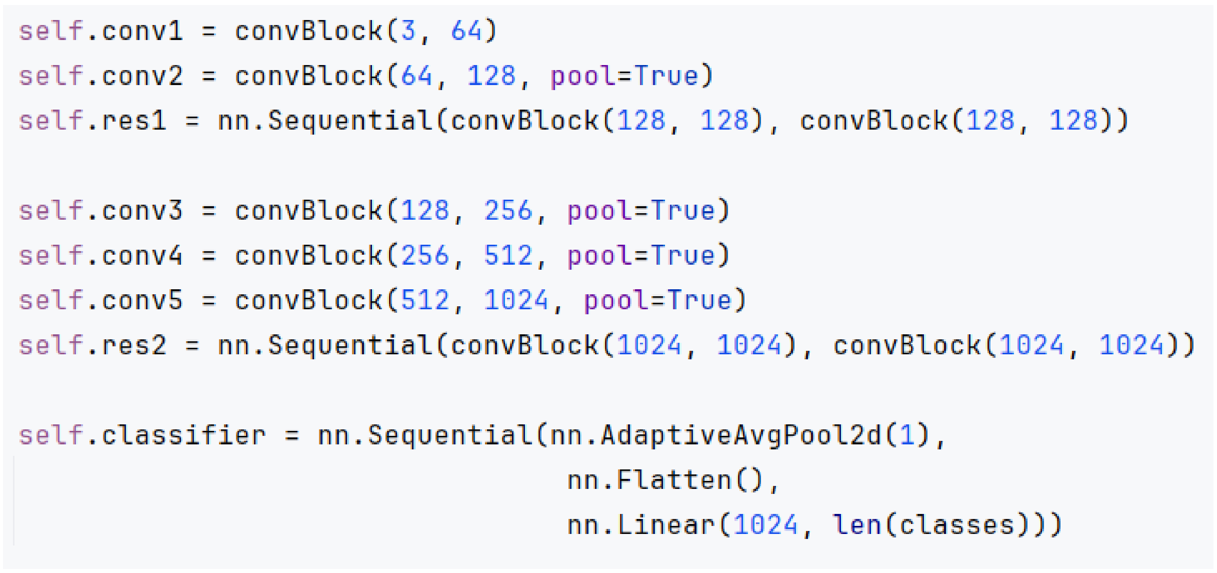
Summary

In the 4th model, we reduced images in class no. 8 so that the model would not be biased by huge amount of data form a single class.

We can see an improvement in the precision of the model over the previous one by 65.04%. Achieving the best accuracy so far.

- ## Model 5

Model Structure



```python
self.conv1 = convBlock(3, 64)
self.conv2 = convBlock(64, 128, pool=True)
self.res1 = nn.Sequential(convBlock(128, 128), convBlock(128, 128))

self.conv3 = convBlock(128, 256, pool=True)
self.conv4 = convBlock(256, 512, pool=True)
self.conv5 = convBlock(512, 1024, pool=True)
self.res2 = nn.Sequential(convBlock(1024, 1024), convBlock(1024, 1024))

self.classifier = nn.Sequential(nn.AdaptiveAvgPool2d(1),
                                nn.Flatten(),
                                nn.Linear(1024, len(classes)))
```

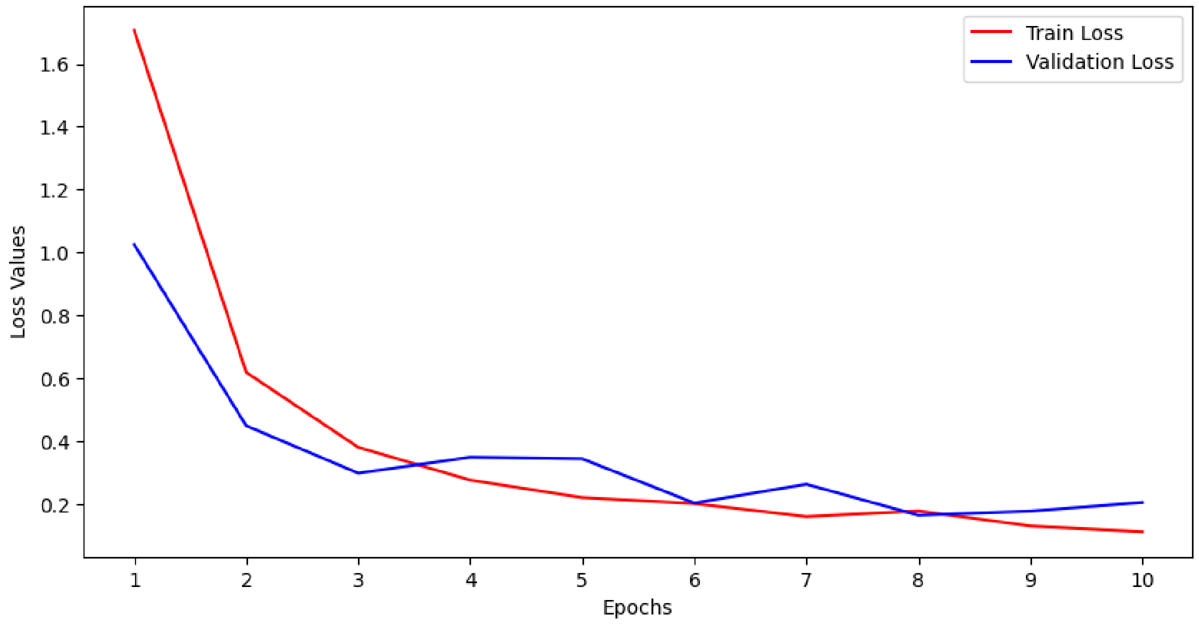
Characteristics

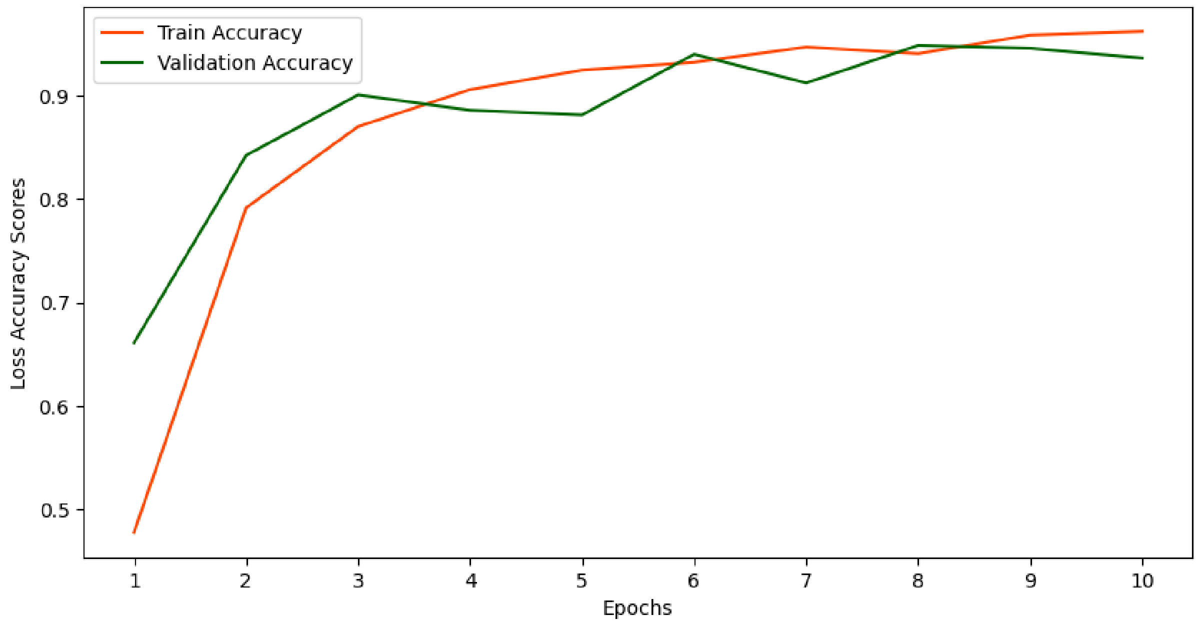
Classes: 20

Training time: 30h25m04s

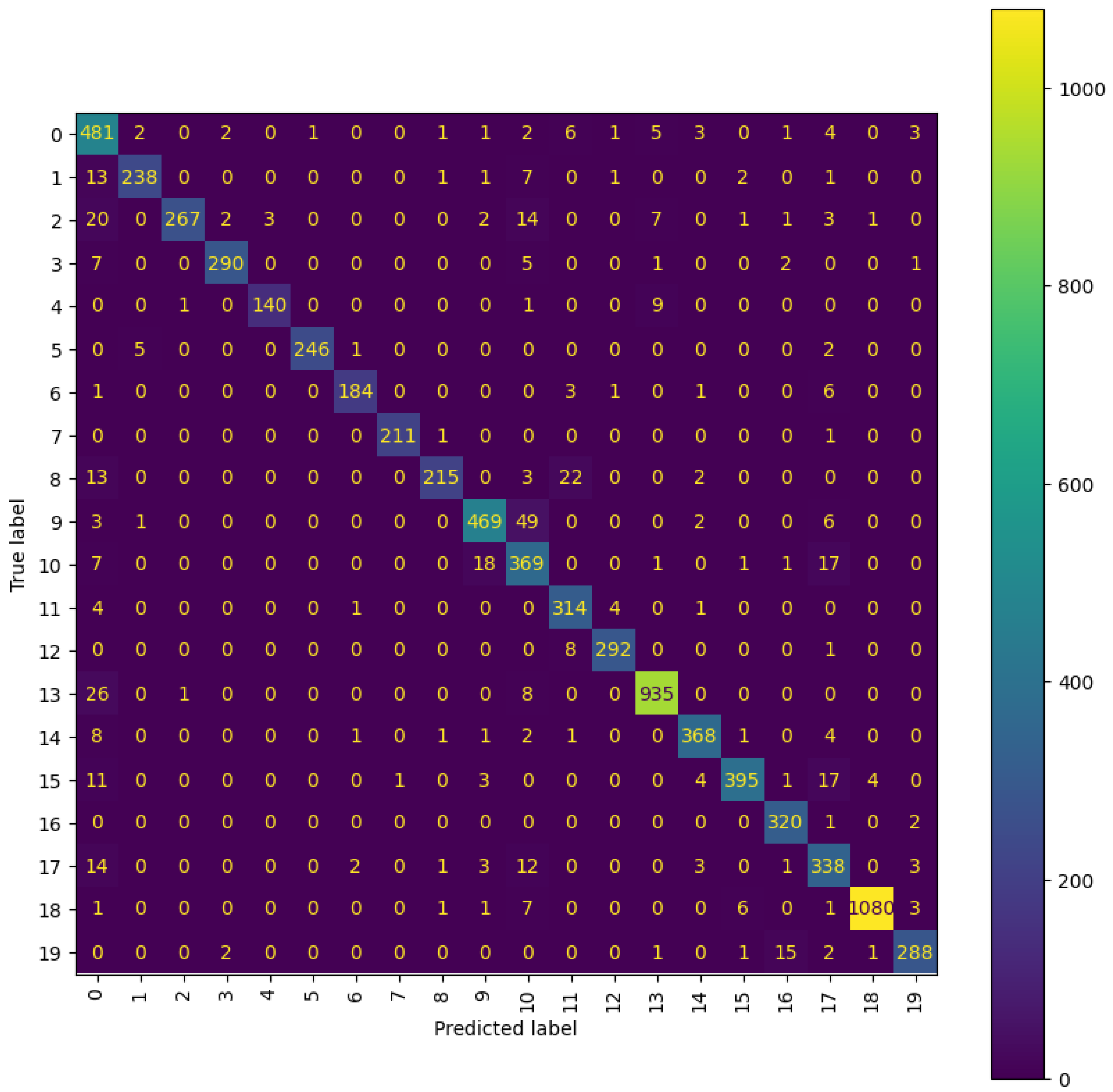
Accuracy: 93.64%

Plots


Train Loss
Validation Loss
Loss Values
Epochs
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
1
2
3
4
5
6
7
8
9
10


Train Accuracy
Validation Accuracy
Loss Accuracy Scores
Epochs
0.9
0.8
0.7
0.6
0.5
1
2
3
4
5
6
7
8
9
10

Confusion matrix



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	481	2	0	2	0	1	0	0	1	1	2	6	1	5	3	0	1	4	0	3
1	13	238	0	0	0	0	0	0	1	1	7	0	1	0	0	2	0	1	0	0
2	20	0	267	2	3	0	0	0	0	2	14	0	0	7	0	1	1	3	1	0
3	7	0	0	290	0	0	0	0	0	0	5	0	0	1	0	0	2	0	0	1
4	0	0	1	0	140	0	0	0	0	0	1	0	0	9	0	0	0	0	0	0
5	0	5	0	0	0	246	1	0	0	0	0	0	0	0	0	0	0	2	0	0
6	1	0	0	0	0	0	184	0	0	0	0	3	1	0	1	0	0	6	0	0
7	0	0	0	0	0	0	0	211	1	0	0	0	0	0	0	0	0	1	0	0
8	13	0	0	0	0	0	0	0	215	0	3	22	0	0	2	0	0	0	0	0
9	3	1	0	0	0	0	0	0	0	469	49	0	0	0	2	0	0	6	0	0
10	7	0	0	0	0	0	0	0	0	18	369	0	0	1	0	1	1	17	0	0
11	4	0	0	0	0	0	1	0	0	0	0	314	4	0	1	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	8	292	0	0	0	0	1	0	0
13	26	0	1	0	0	0	0	0	0	0	8	0	0	935	0	0	0	0	0	0
14	8	0	0	0	0	0	1	0	1	1	2	1	0	0	368	1	0	4	0	0
15	11	0	0	0	0	0	0	1	0	3	0	0	0	0	4	395	1	17	4	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	320	1	0	2
17	14	0	0	0	0	0	2	0	1	3	12	0	0	0	3	0	1	338	0	3
18	1	0	0	0	0	0	0	0	1	1	7	0	0	0	0	6	0	1	1080	3
19	0	0	0	2	0	0	0	0	0	0	0	0	0	1	0	1	15	2	1	288

Summary

In model no. 5, we applied an additional convolution layer, ReLU activation function and max pooling 2D layer increasing the output channels to 1024. We have also restored the number of classes to 20.

This resulted in a significant increase in training time and model complexity. Despite training time and an increase in the number of classes, the model's precision remained above 90%.

- ## Model 6

Model Structure

```
self.conv1 = convBlock(3, 64)
self.conv2 = convBlock(64, 128, pool=True)
self.res1 = nn.Sequential(convBlock(128, 128), convBlock(128, 128))

self.conv3 = convBlock(128, 256, pool=True)
self.res2 = nn.Sequential(convBlock(256, 256), convBlock(256, 256))

self.classifier = nn.Sequential(nn.AdaptiveAvgPool2d(1),
                                 nn.Flatten(),
                                 nn.Linear(256, len(classes)))
```

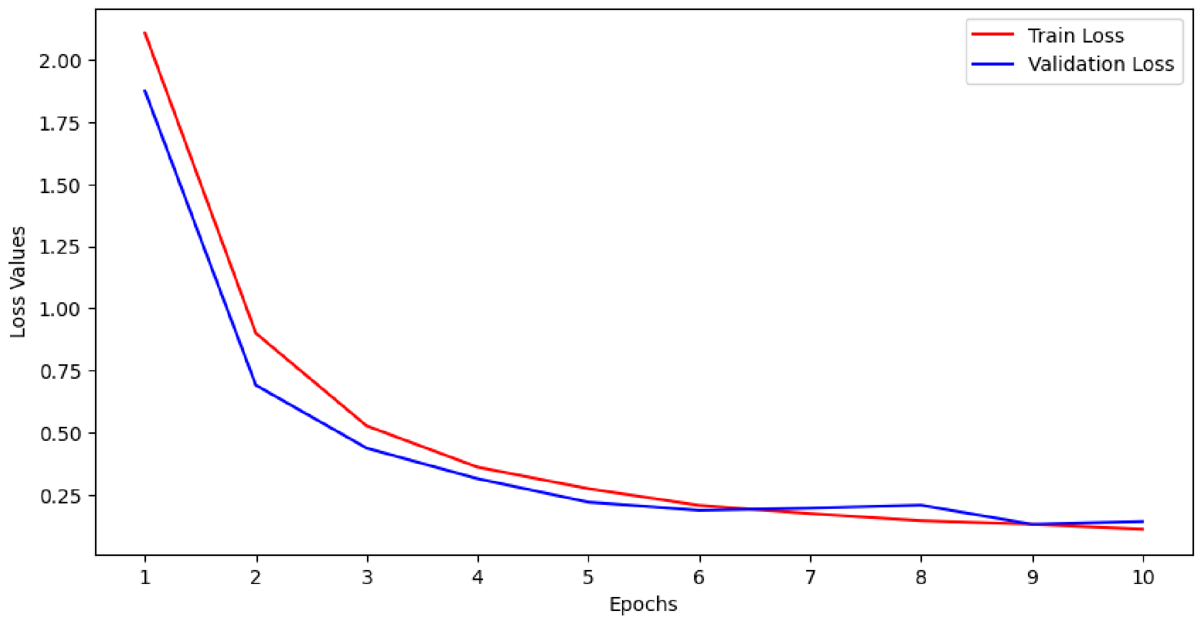
Characteristics

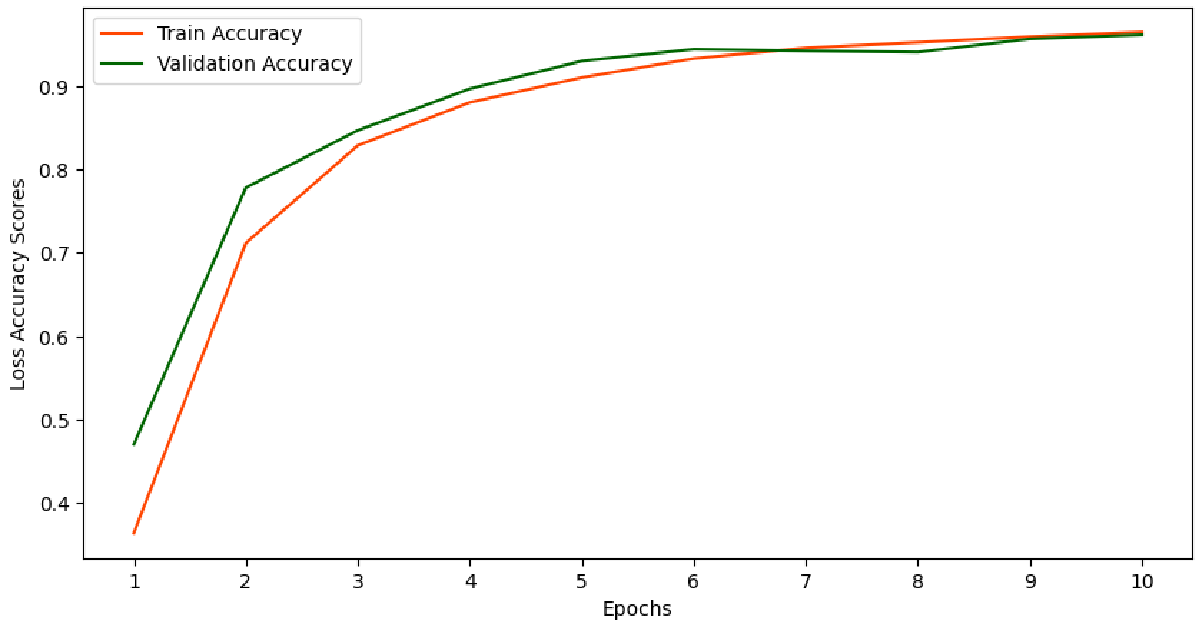
Classes: 20

Training time: 18h37m40s

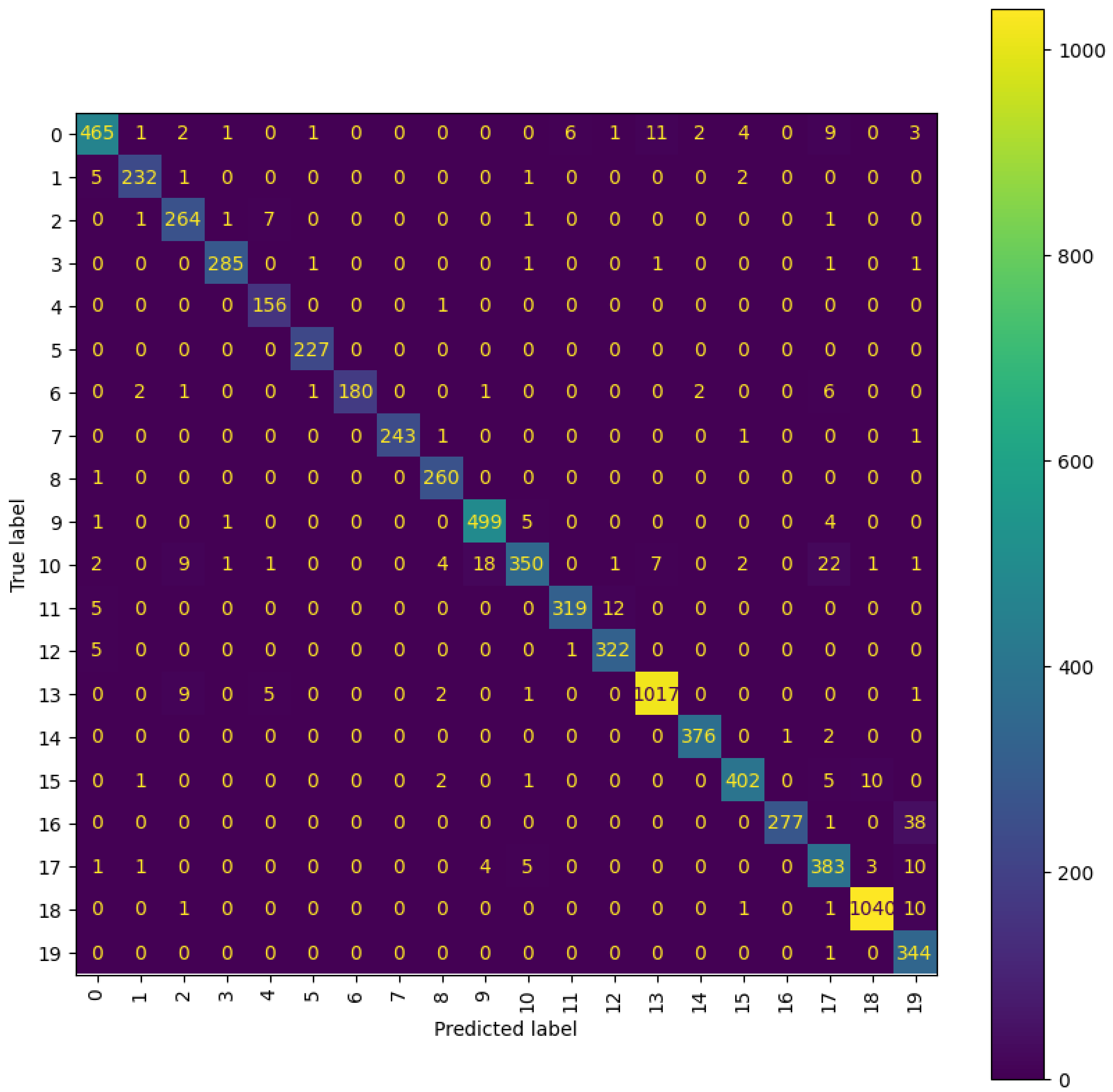
Accuracy: 96.17%

Plots


Train Loss
Validation Loss
Loss Values
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
1
2
3
4
5
6
7
8
9
10
Epochs


Train Accuracy
Validation Accuracy
Loss Accuracy Scores
0.9
0.8
0.7
0.6
0.5
0.4
1
2
3
4
5
6
7
8
9
10
Epochs

Confusion matrix



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	465	1	2	1	0	1	0	0	0	0	0	6	1	11	2	4	0	9	0	3
1	5	232	1	0	0	0	0	0	0	0	1	0	0	0	0	2	0	0	0	0
2	0	1	264	1	7	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
3	0	0	0	285	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	1
4	0	0	0	0	156	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	227	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	2	1	0	0	1	180	0	0	1	0	0	0	0	2	0	0	6	0	0
7	0	0	0	0	0	0	0	243	1	0	0	0	0	0	0	1	0	0	0	1
8	1	0	0	0	0	0	0	0	260	0	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	0	499	5	0	0	0	0	0	0	4	0	0
10	2	0	9	1	1	0	0	0	4	18	350	0	1	7	0	2	0	22	1	1
11	5	0	0	0	0	0	0	0	0	0	0	319	12	0	0	0	0	0	0	0
12	5	0	0	0	0	0	0	0	0	0	0	1	322	0	0	0	0	0	0	0
13	0	0	9	0	5	0	0	0	2	0	1	0	0	1017	0	0	0	0	0	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	376	0	1	2	0	0
15	0	1	0	0	0	0	0	0	2	0	1	0	0	0	0	402	0	5	10	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	277	1	0	38
17	1	1	0	0	0	0	0	0	0	4	5	0	0	0	0	0	0	383	3	10
18	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1040	10
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	344

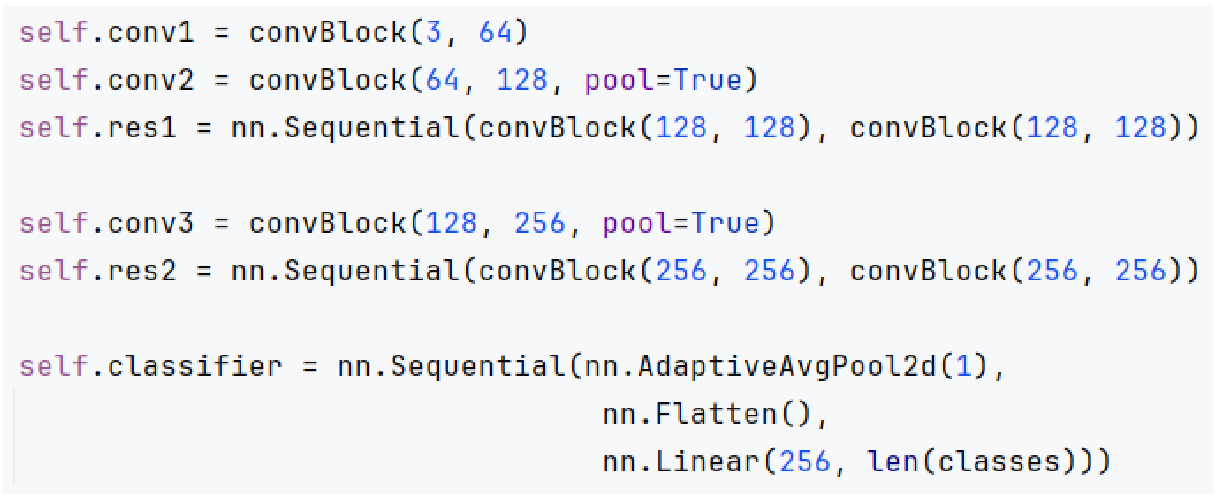
Summary

In model no. 6, we removed 2 ‘convBlock’ functions decreasing the output channels to 256 in order to reduce the complexity of the model.

We can see an in the precision of the model over the 4th one by 0.82%. This means that the previous model was too complicated for our dataset and learned the wrong patterns. Reducing the layers had a positive effect on the final precision, achieving the best accuracy so far.

- ## Model 7

Model Structure



```
self.conv1 = convBlock(3, 64)
self.conv2 = convBlock(64, 128, pool=True)
self.res1 = nn.Sequential(convBlock(128, 128), convBlock(128, 128))

self.conv3 = convBlock(128, 256, pool=True)
self.res2 = nn.Sequential(convBlock(256, 256), convBlock(256, 256))

self.classifier = nn.Sequential(nn.AdaptiveAvgPool2d(1),
                                nn.Flatten(),
                                nn.Linear(256, len(classes)))
```

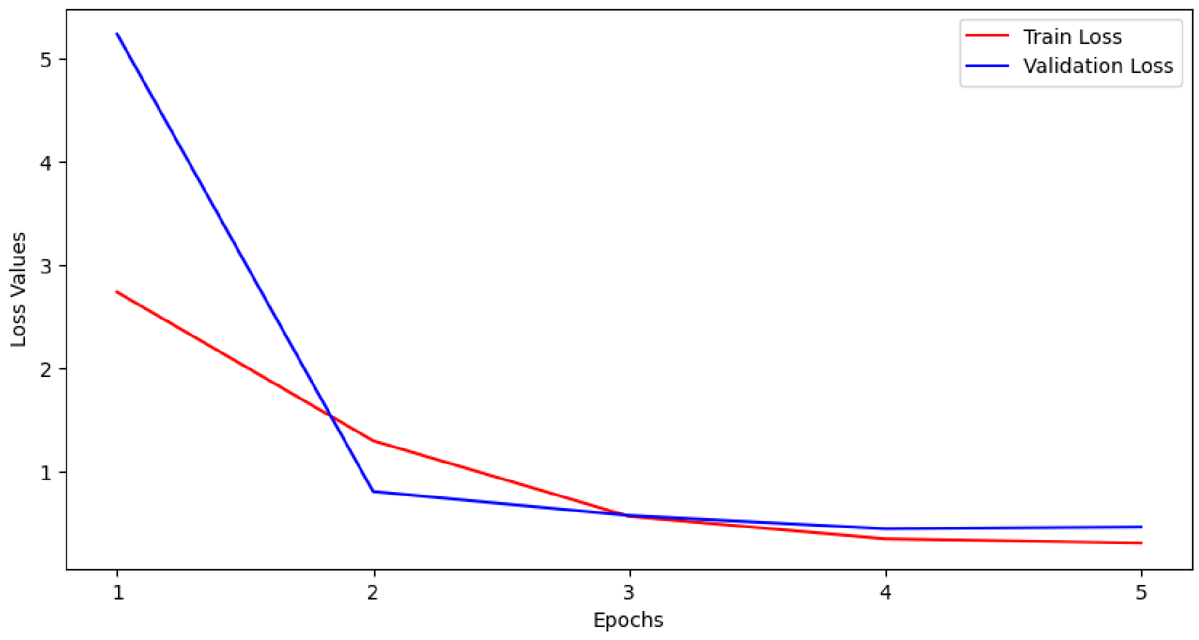
Characteristics

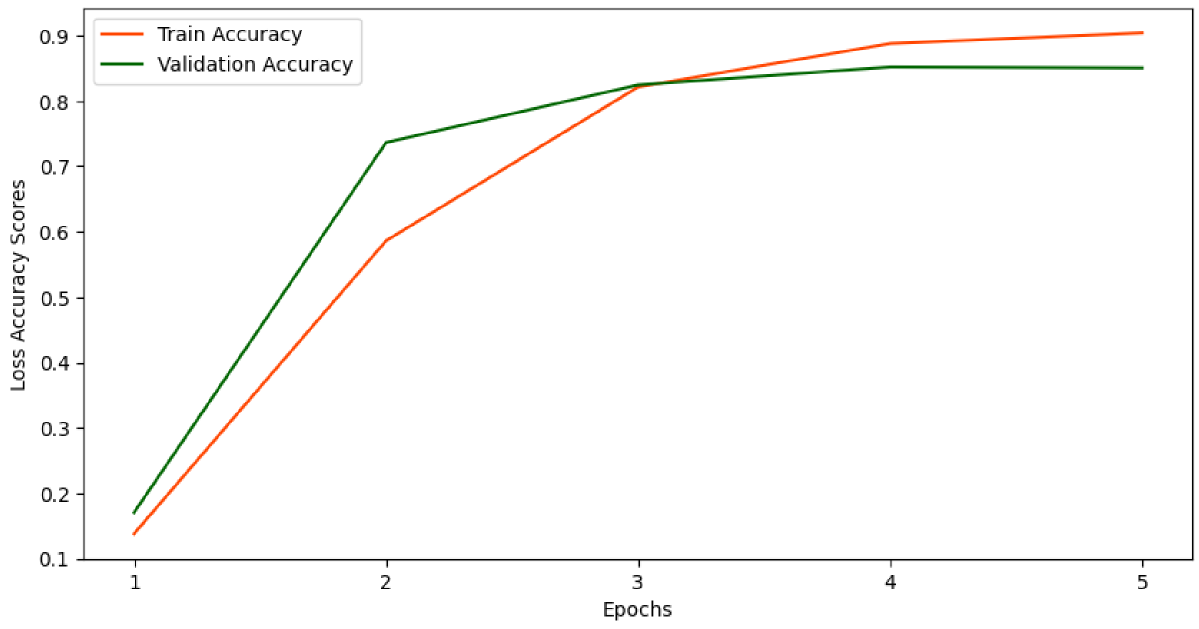
Classes: 20

Training time: 11h01m26s

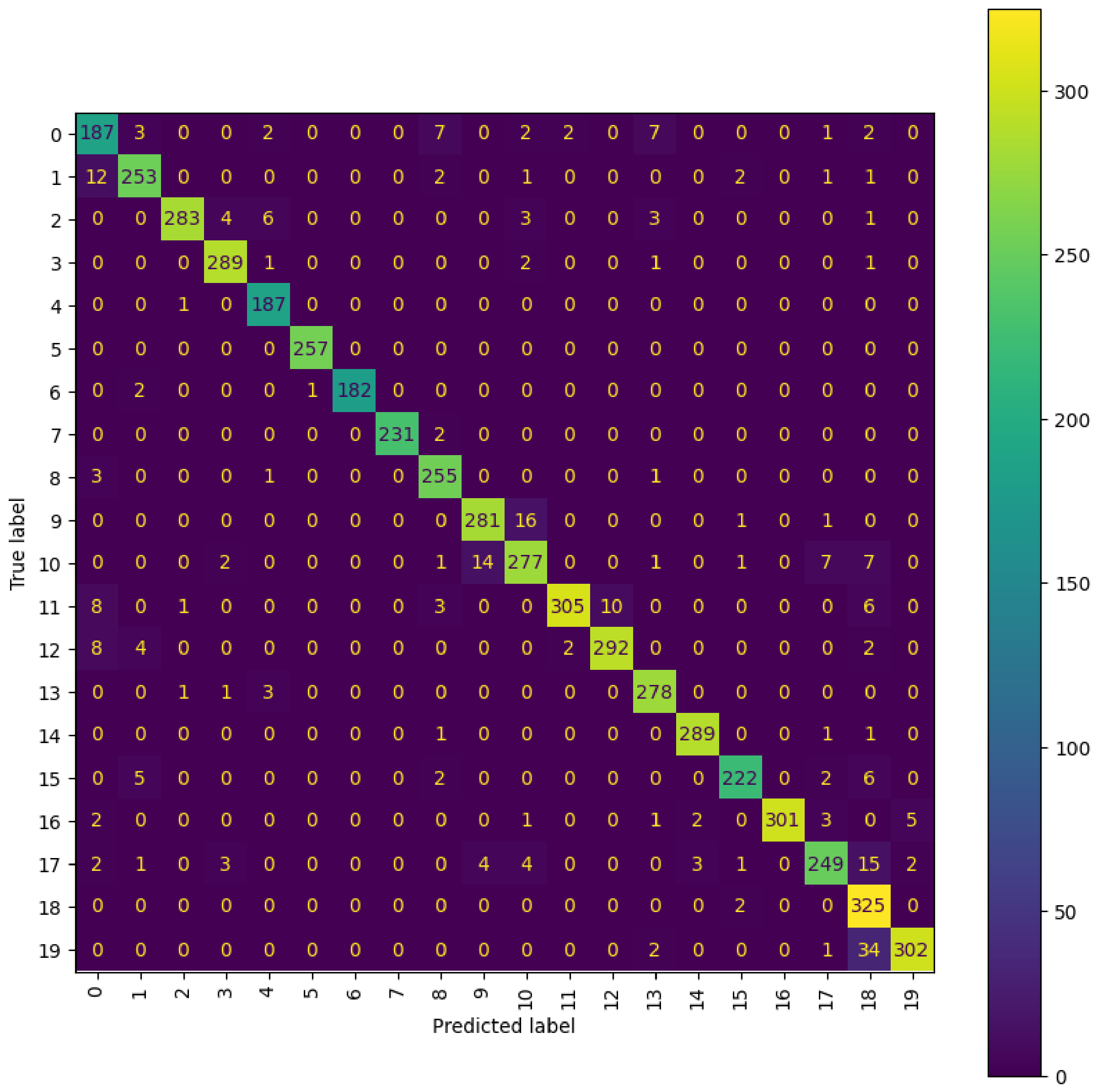
Accuracy: 85.06%

Plots


Train Loss
Validation Loss
Loss Values
Epochs
1
2
3
4
5
1
2
3
4
5


Train Accuracy
Validation Accuracy
Loss Accuracy Scores
Epochs
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
2
3
4
5

Confusion matrix



True \ Predicted	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	187	3	0	0	2	0	0	0	7	0	2	2	0	7	0	0	0	1	2	0
1	12	253	0	0	0	0	0	0	2	0	1	0	0	0	0	2	0	1	1	0
2	0	0	283	4	6	0	0	0	0	0	3	0	0	3	0	0	0	0	1	0
3	0	0	0	289	1	0	0	0	0	0	2	0	0	1	0	0	0	0	1	0
4	0	0	1	0	187	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	257	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	2	0	0	0	1	182	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	231	2	0	0	0	0	0	0	0	0	0	0	0
8	3	0	0	0	1	0	0	0	255	0	0	0	0	1	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	281	16	0	0	0	0	1	0	1	0	0
10	0	0	0	2	0	0	0	0	1	14	277	0	0	1	0	1	0	7	7	0
11	8	0	1	0	0	0	0	0	3	0	0	305	10	0	0	0	0	0	6	0
12	8	4	0	0	0	0	0	0	0	0	0	2	292	0	0	0	0	0	2	0
13	0	0	1	1	3	0	0	0	0	0	0	0	0	278	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	1	0	0	0	0	0	289	0	0	1	1	0
15	0	5	0	0	0	0	0	0	2	0	0	0	0	0	0	222	0	2	6	0
16	2	0	0	0	0	0	0	0	0	0	1	0	0	1	2	0	301	3	0	5
17	2	1	0	3	0	0	0	0	0	4	4	0	0	0	3	1	0	249	15	2
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	325	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	34	302

Summary

In model no. 7, we balanced data trying to improve accuracy on lesser classes. During training accuracy for each epoch was lower than in previous model.

5. Summary

The implementation of convolutional neural network using the PyTorch library allowed us to learn about the operation of different layers such as convolution layer, normalization layer, activation layer, etc., and their usage in models.

The CUDA architecture had a big impact on the tests performed, which allowed us to significantly reduce the training time, which allowed us to launch more tests.

The tests conducted allowed us to observe various phenomena occurring when training the model. One of the most important is the relationship between the complexity of the model and the amount of data on which it was tested. Models with high complexity require large data sets to avoid overfitting. For small data sets, reducing model complexity or using data augmentation techniques such as data balancing positively affects model precision.

In our research, we did not use layers such as dropout layer that could have had a positive impact on the model's precision. We did not test different activation layer functions.

6. References

Plant Diseases Training Dataset - https://www.kaggle.com/

Plant Disease Test Dataset - https://www.kaggle.com/

Building an Image Classification Model From Scratch Using PyTorch | by Benedict Neo | bitgrit Data Science Publication | Medium - https://medium.com/

Learn the Basics — PyTorch Tutorials 2.3.0+cu121 documentation - https://pytorch.org/

7. Project link

Link: mc090/Plant-Disease-Classification - https://github.com/