Report

Biologically inspired artificial intelligence
Heart

Fruit and vegetables classification

Supervisor: Dr inż. Grzegoż Baron

Date: 03.07.2023

Jakub Gil
Michał Lenort

Introduction to the project topic:

The accurate classification of fruits and vegetables is a crucial task in various domains, including agriculture, food processing, and retail. Efficient identification and categorization of produce items are essential for quality control, inventory management, and meeting consumer demands. With recent advancements in machine learning and artificial intelligence, neural networks have emerged as powerful tools for automating classification processes.

In this report, we present a comprehensive study on the development and evaluation of a unidirectional neural network-based system for the classification of fruits and vegetables. Our primary objective is to design a robust model capable of accurately identifying and categorizing various types of produce. By harnessing the capabilities of deep learning and unidirectional neural networks, we aim to enhance the efficiency and reliability of the classification process.

Problem statement:

The classification of fruits and vegetables presents unique challenges due to the wide variety of shapes, sizes, colors, and textures exhibited by different produce items. Traditional classification methods often rely on manual inspection, which is time-consuming, subjective, and prone to human error. Automation of this process is necessary to meet the increasing demand for efficiency and accuracy in various industries.

To address these challenges, we propose the utilization of a unidirectional neural network, specifically a feedforward neural network, for fruit and vegetable classification. Feedforward neural networks have demonstrated remarkable performance in image recognition tasks, making them well-suited for this application. By training a neural network on a large dataset of labeled images, we aim to develop a model that can generalize well and accurately classify unseen fruits and vegetables.

Dataset properties:

- The total number of images: 90483.
- Training set size: 67692 images

- Test set size: 22688 images
- The number of classes: 131
- Image size: 100x100 pixels

Methodology:

Our approach involves several key steps. Firstly, we gather a comprehensive dataset comprising high-resolution images of different fruits and vegetables. This dataset is meticulously labeled with the corresponding class labels to facilitate supervised learning. Subsequently, we preprocess the images by applying techniques such as resizing, normalization, and augmentation to enhance the model's robustness and generalization ability.

Next, we design and train a unidirectional neural network architecture specifically tailored for fruit and vegetable classification. The network consists of multiple layers, including input, hidden, and output layers. We utilize activation functions such as ReLU (Rectified Linear Unit) to introduce non-linearity and enable the network to learn complex patterns and features from the images.

To optimize the network's parameters, we employ a suitable optimization algorithm, such as stochastic gradient descent (SGD), and utilize a loss function, such as categorical cross-entropy. During the training process, we monitor the model's performance on a separate validation set to fine-tune the hyperparameters and prevent overfitting.

Internal and external specification of the software solution:

The dataset is extracted to the directory data/fruits-360. It contains 2 folders (train and test), containing the training set (67,692 images) and test set (22,688 images) respectively. Each of them contains 131 folders, one for each class of images :

```python
dataset_path = './input/fruits/fruits-360/Training'
test_path = './input/fruits/fruits-360/Test'
print(os.listdir(dataset_path))
print(os.listdir(test_path))
print(torch.cuda.is_available())
```

```
['Apple Braeburn', 'Apple Crimson Snow', 'Apple Golden 1', 'Apple Golden 2', 'Apple Golden 3',
['Apple Braeburn', 'Apple Crimson Snow', 'Apple Golden 1', 'Apple Golden 2', 'Apple Golden 3',
True
```

```python
image_size = 75

fruit_data = ImageFolder(dataset_path, transform = transforms.Compose([
    Resize((image_size, image_size)),
    ToTensor()
]))

fruit_test = ImageFolder(test_path, transform = transforms.Compose([
    Resize((image_size, image_size)),
    ToTensor()
]))
```

The dataset was splited into 3 parts :

Training set : used to train the model i.e. compute the loss and adjust the weights of the model using gradient descent .

Validation set : used to evaluate the model while training, adjust hyperparameters (learning rate etc.) and pick the best version of the model.

Test set : used to compare different models, or different types of modeling approaches, and report the final accuracy of the model.

```python
torch.manual_seed(20)
validation_length = len(fruit_data) // 10
training_length = len(fruit_data) - validation_length
```

```python
training_dataset, validation_dataset = random_split(fruit_data, [training_length,
validation_length])
print(len(training_dataset))
print(len(validation_dataset))
```

```
60923
6769
```

```python
batch_length = 64
training_loader = DataLoader(training_dataset, batch_length, shuffle = True, num_workers
= 4, pin_memory = True)
validation_loader = DataLoader(validation_dataset, batch_length * 2, num_workers = 4,
pin_memory = True)
test_loader = DataLoader(fruit_test, batch_length * 2, num_workers = 4, pin_memory =
True)
```

The neural network was trained using the CUDA architecture, which required a graphics card. This made it possible to speed up the training of the model.

```python
device = None
if torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')

print(device)
```

```
cuda
```

```python
def move_to_device(data, device):
    if (isinstance(data, (list, tuple))):
        return [move_to_device(d, device) for d in data]

    return data.to(device, non_blocking=True)
```

```python
class DeviceLoader():
    def __init__(self, dataloader, device):
        self.dataloader = dataloader
        self.device = device

    def __iter__(self):
        for d in self.dataloader:
            yield move_to_device(d, self.device)

    def __len__(self):
        return len(self.dataloader)
```

```python
training_loader = DeviceLoader(training_loader, device)
validation_loader = DeviceLoader(validation_loader, device)
test_loader = DeviceLoader(test_loader, device)
```

A model consisting of the following layers was then created:

1. Input layer (self.in_layer): It takes an input of size input_length and linearly transforms it into a space of size 8384.
2. First hidden layer (self.hidden1): It takes an input of size 8384 and linearly transforms it into a space of size 4192. Then, a ReLU activation function is applied.
3. Second hidden layer (self.hidden2): It takes an input of size 4192 and linearly transforms it into a space of size 2096. Then, a ReLU activation function is applied.
4. Third hidden layer (self.hidden3): It takes an input of size 2096 and linearly transforms it into a space of size 1048. Then, a ReLU activation function is applied.

5. Output layer (self.out_layer): It takes an input of size 1048 and linearly transforms it into a space of size output_length. Then, a ReLU activation function is applied.

```python
def calc_accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim = 1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))
```

Python

```python
class Classification(nn.Module):
    def train_batch(self, batch):
        images, labels = batch
        outputs = self(images)
        loss = F.cross_entropy(outputs, labels)
        return loss
    
    def validate_batch(self, batch):
        images, labels = batch
        outputs = self(images)
        loss = F.cross_entropy(outputs, labels)
        accuracy = calc_accuracy(outputs, labels)
        return {'validation_loss': loss.detach(), 'validation_accuracy': accuracy}
    
    def calc_validation_epoch(self, outputs):
        batch_losses = [o['validation_loss'] for o
        epoch_loss = torch.stack(batch_losses).mean
        batch_accs = [o['validation_accuracy'] for o in outputs]
        epoch_accuracy = torch.stack(batch_accs).mean()
        return {'validation_loss': epoch_loss.item(), 'validation_accuracy': epoch_accuracy.item()}
    
    def print_epoch_result(self, epoch, result):
        print("Epoch [{}], training_loss: {:.4f}, validation_loss: {:.4f}, validation_accuracy: {:.4f}".format(
            epoch, result['training_loss'], result['validation_loss'], result['validation_accuracy']))
```

(variable) o: Any

Python

```python
class Model(Classification):
    def __init__(self, input_length, output_length):
        super().__init__()
        self.in_layer = nn.Linear(input_length, 8384)
        self.hidden1 = nn.Linear(8384, 4192)
        self.hidden2 = nn.Linear(4192, 2096)
        self.hidden3 = nn.Linear(2096, 1048)
        self.out_layer = nn.Linear(1048, output_length)

    def forward(self, xb):
        # Flatten images into vectors
        output = xb.view(xb.size(0), -1)
        # Apply layers & activation functions
        # Input layer
        output = self.in_layer(output)
        # Hidden layers w/ ReLU
        output = self.hidden1(F.relu(output))
        output = self.hidden2(F.relu(output))
        output = self.hidden3(F.relu(output))
        # Class output layer
        output = self.out_layer(F.relu(output))
        return output
```

Python

```python
def learn_model(epochs, learing_rate, model, training_loader, validation_loader,
opt_func=torch.optim.SGD):
    tm = []
    optimizer = opt_func(model.parameters(), learing_rate)
    for epoch in range(epochs):
        # Training Phase
        model.train()
        training_losses = []
        for batch in tqdm(training_loader):
            loss = model.train_batch(batch)
            train (function) backward: Any
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
        # Validation phase
        result = verify(model, validation_loader)
        result['training_loss'] = torch.stack(training_losses).mean().item()
        model.print_epoch_result(epoch, result)
        tm.append(result)
    return tm
```

Experiments

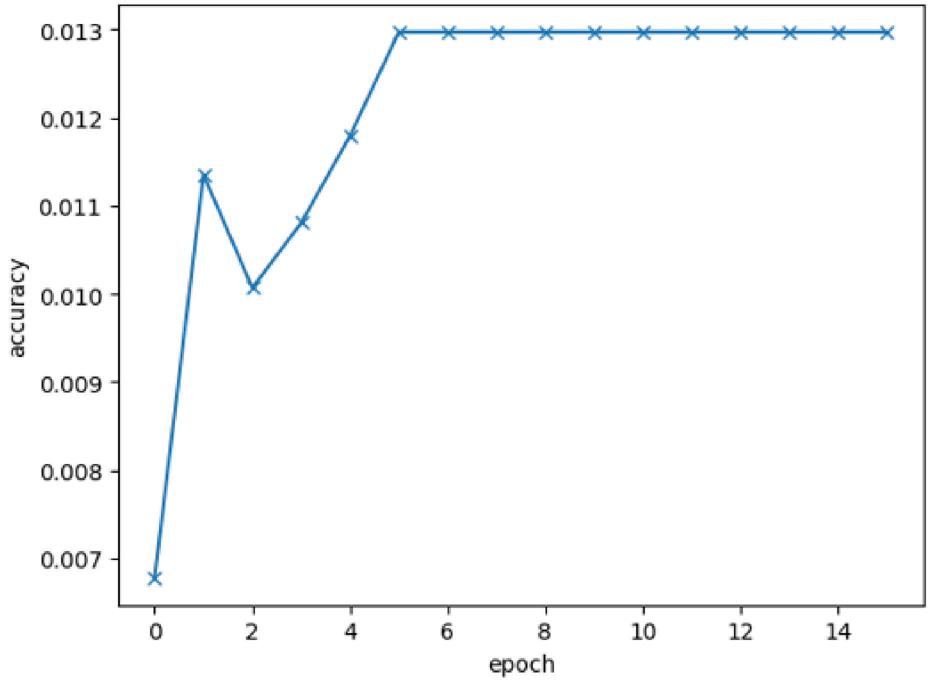
Testing impact of optimizers on accuracy

In the project we used five different optimizers:

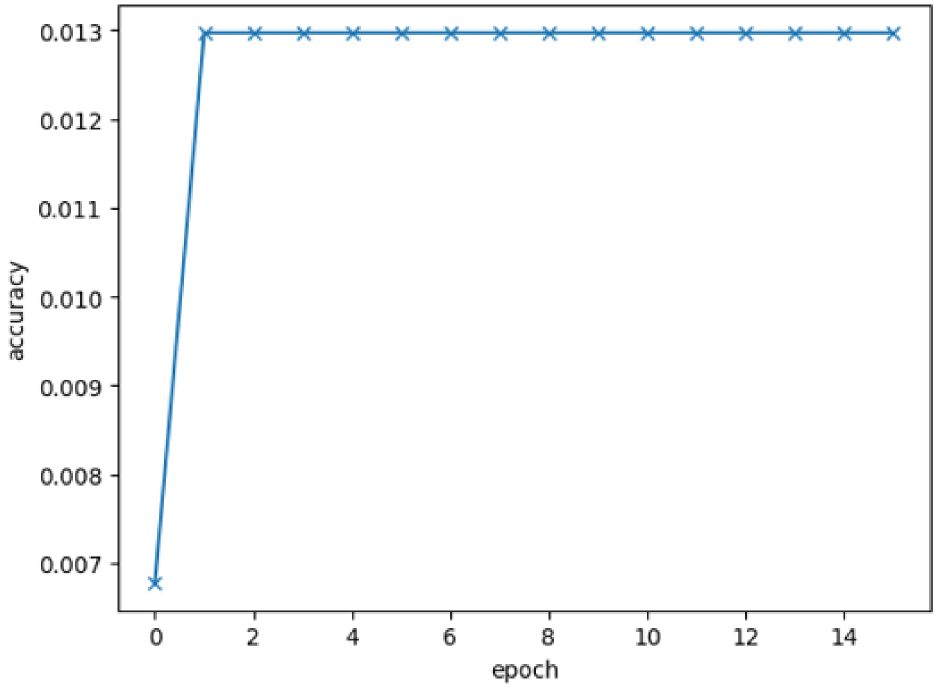
- **Stochastic Gradient Descent (SGD)** - This is a basic optimizer that updates network weights based on the gradient of the loss function for single samples. Despite its simplicity, SGD can be effective in many cases, especially with properly selected batch size and learning rate.
- **Adam** - Adam (Adaptive Moment Estimation) is a popular optimizer that combines momentum and RMSprop methods. It uses gradient moment estimation and second gradient moment estimation to adaptively adjust the learning rate for each parameter. Adam tends to work well on a wide variety of problems and does not require too many hyperparameters.
- **Adagrad** - Adagrad (Adaptive Gradient) is an optimizer that adjusts the learning rate for each parameter based on the sum of squares of previous gradients. It is effective in cases where some parameters are more important than others.
- **Adagrad** - Adagrad (Adaptive Gradient) is an optimizer that adjusts the learning rate for each parameter based on the sum of squares of previous gradients. It is effective in cases where some parameters are more important than others.

- **Adadelta** - Adadelta is an optimizer that uses an estimate of the mean squares of gradient updates to adjust the learning rate. This optimizer does not require setting a global learning rate.

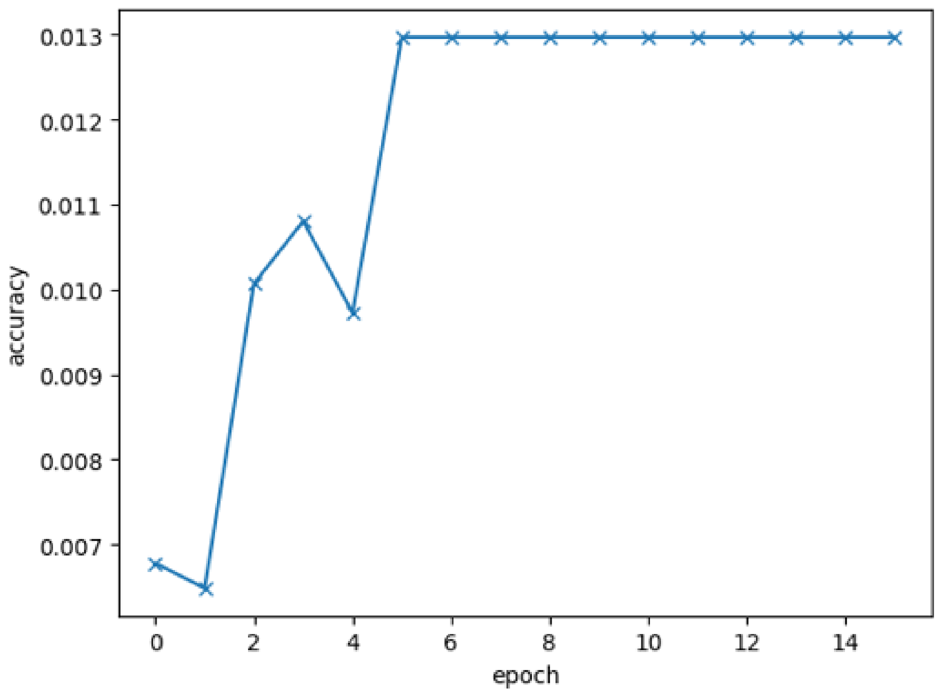
Results:



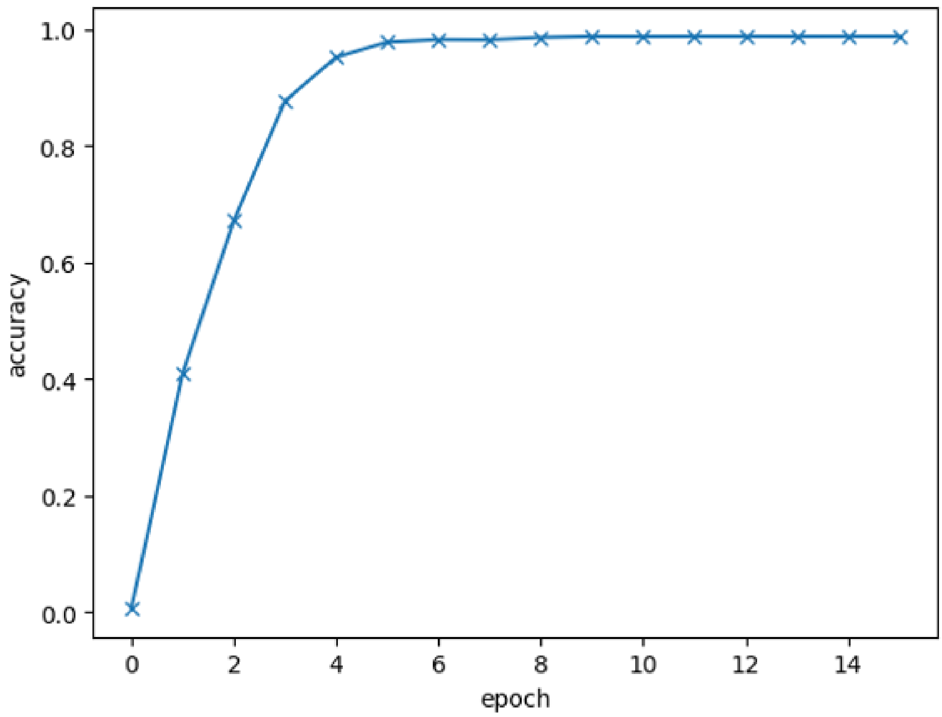
Adam – 1.44% accuracy



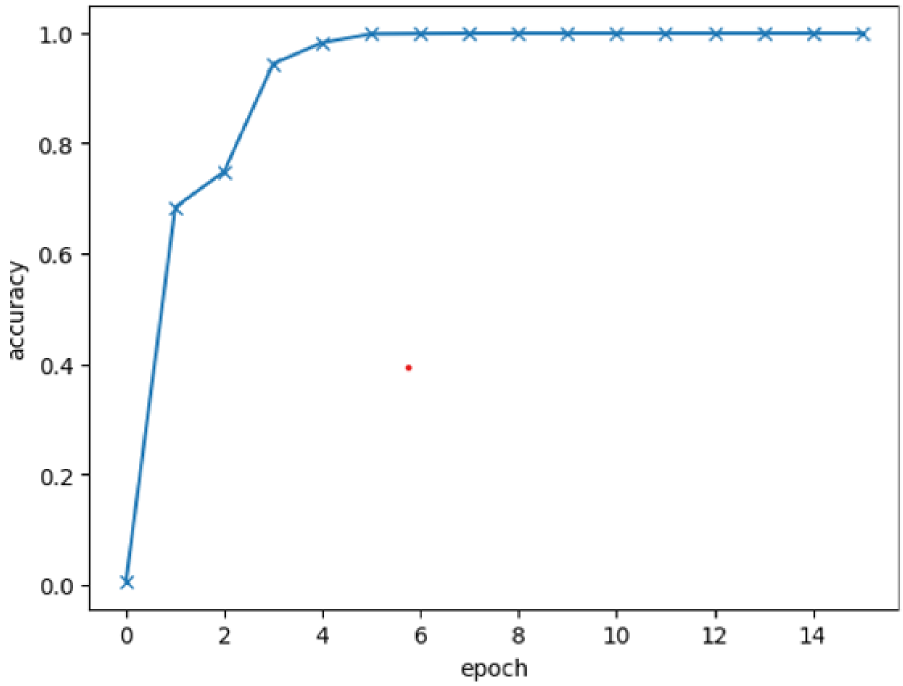
Adagrad – 1.44% accuracy



RMSprop – 1.44% accuracy



Adadelta – 88.91% accuracy

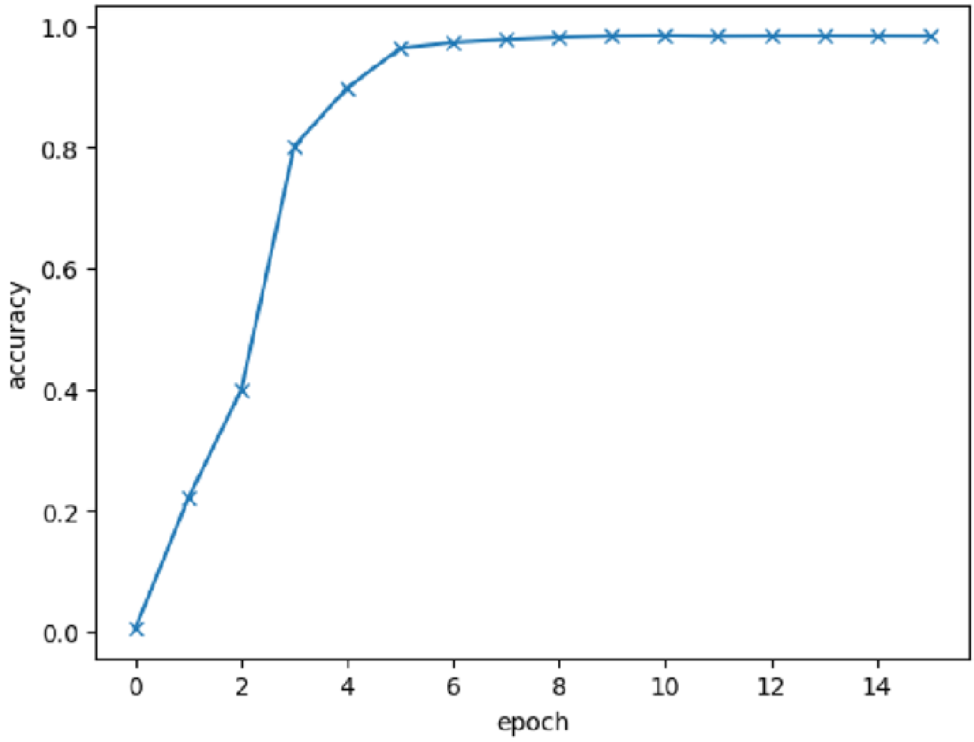


SGD – 93.44% accuracy

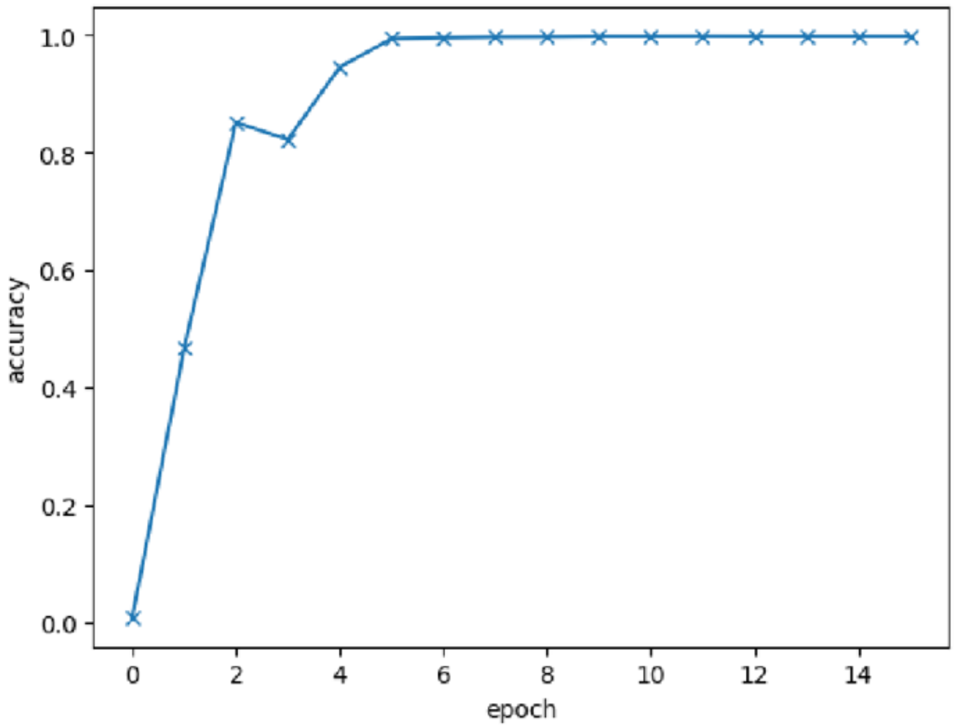
Testing impact of image size on accuracy

Tested image sizes:

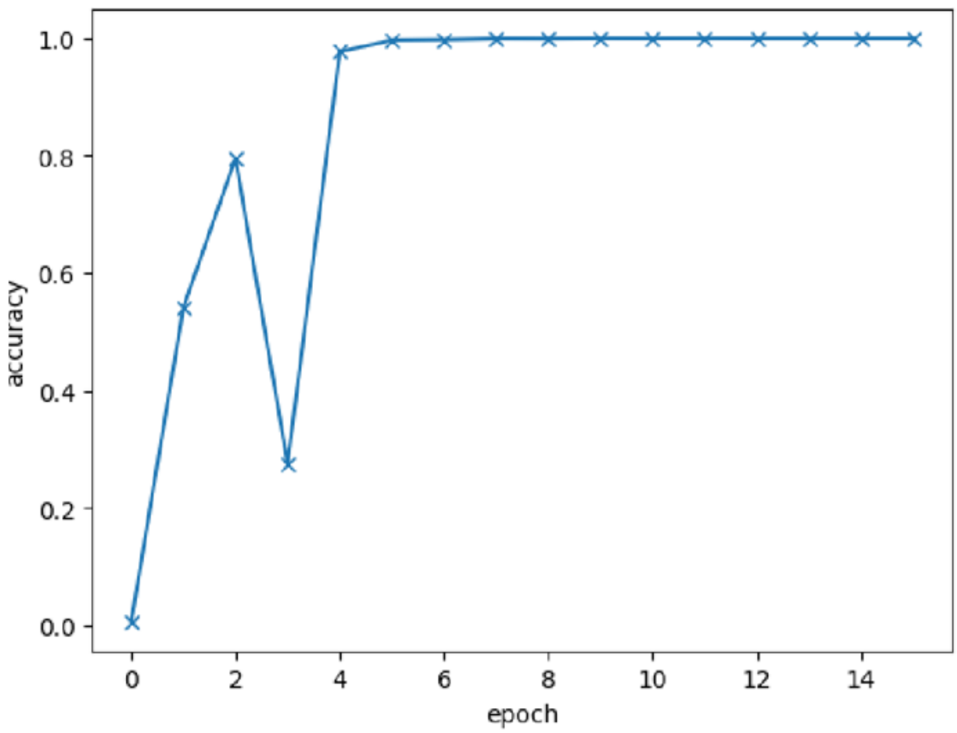
- 50x50 – 91.63% accuracy



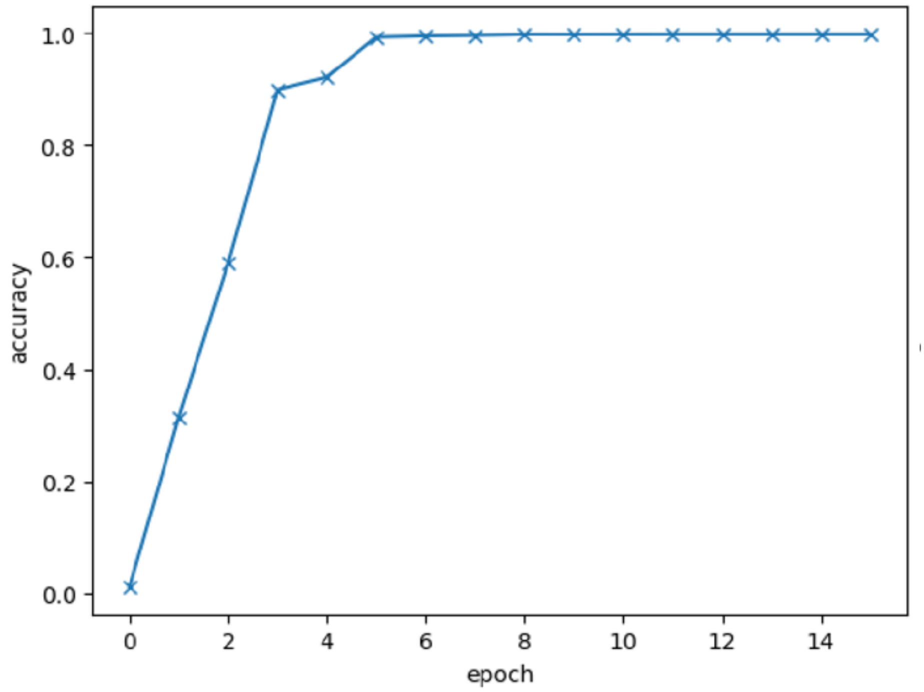
- 75x75 – 90.99% accuracy



- 100x100 – 91.6% accuracy



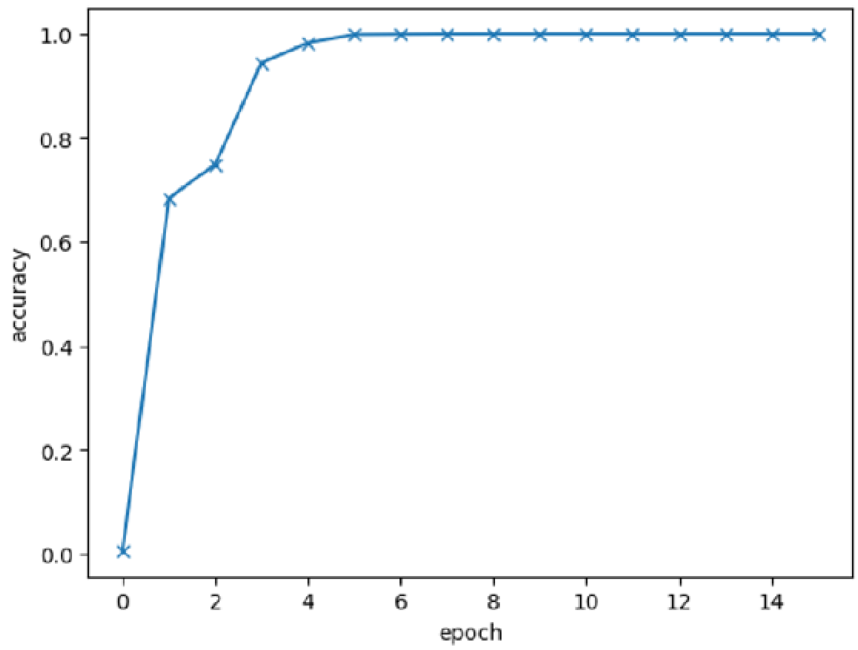
- 125x125 – 91.12% accuracy



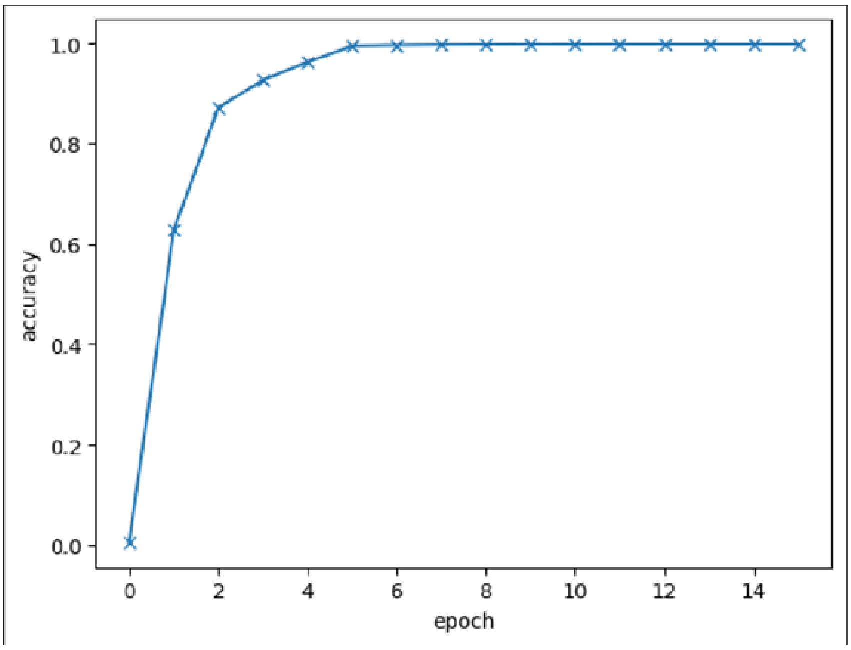
Testing impact of batch size on accuracy

Tested batch sizes:

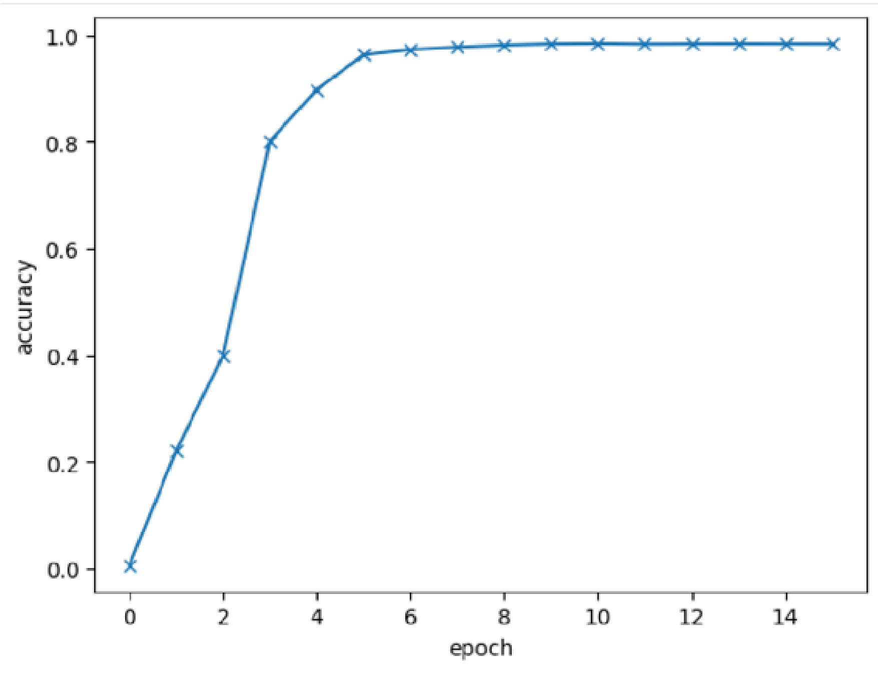
- 32 – 93.44%



- 64 – 91.63%



- 128 – 87.52%



Conclusion

In conclusion, this report presents a detailed investigation into the development of a unidirectional neural network-based system for the classification of fruits and vegetables. By leveraging the power of deep learning and feedforward neural networks, we aim to create an accurate and efficient classification model that can enhance productivity and quality control in industries related to produce.

Through our comprehensive methodology and evaluation process, we strive to provide valuable insights and practical recommendations for the implementation of such systems. The successful deployment of a robust fruit and vegetable classification system can have significant implications, ranging from reducing manual labor to enabling automated inventory management and improving customer satisfaction.

This project allowed us to learn the basics of Machine Learning technology. We learned how to easily and effectively create a neural network to classify a large data set. This project also allowed us to get acquainted with the basics of the Python language, and the Pytorch library, which greatly facilitates the creation of Neural Networks by providing ready-made tools.

Based on the experiments, it can be concluded that the selection of paramtrs has a significant role in the performance of the neural network. The selection of the optimizer has the greatest impact. In the case of FNN, SGD and Adadelta perform best. Batch size was also of great importance. The best results were achieved when its value was 32.

Refrences

- www.kaggle.com
- https://towardsdatascience.com/machine-learning-basics-part-1-a36d38c7916
- en.wikipedia.org/wiki/Feedforward_neural_network
- https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network

Link to the shared files

Github:
https://github.com/Michallenort/Fruits_Classification

Dataset:
https://www.kaggle.com/datasets/moltean/fruits