

ITRI 626 Mini-Project Submission

A. du Plessis

Lecturer: Prof. Abasalom E. Ezuguw

Student number: 34202676

Contents

1	Introduction	1
2	Architecture of Model	2
2.1	Input Layer	3
2.2	Convolution Layer	3
2.3	Batch Normalisation Layer	3
2.4	Activation Function (Nonlinearity Layer)	4
2.5	Pooling Layer	4
2.6	Dropout Layer	4
2.7	Fully Connected Layer	4
2.8	Output Layer	5
3	Performance Evaluation	6
3.1	Loss	6
3.2	Accuracy	6
3.3	F1-Score	6
3.4	ROC AUC	6
4	Conclusion	7

List of Figures

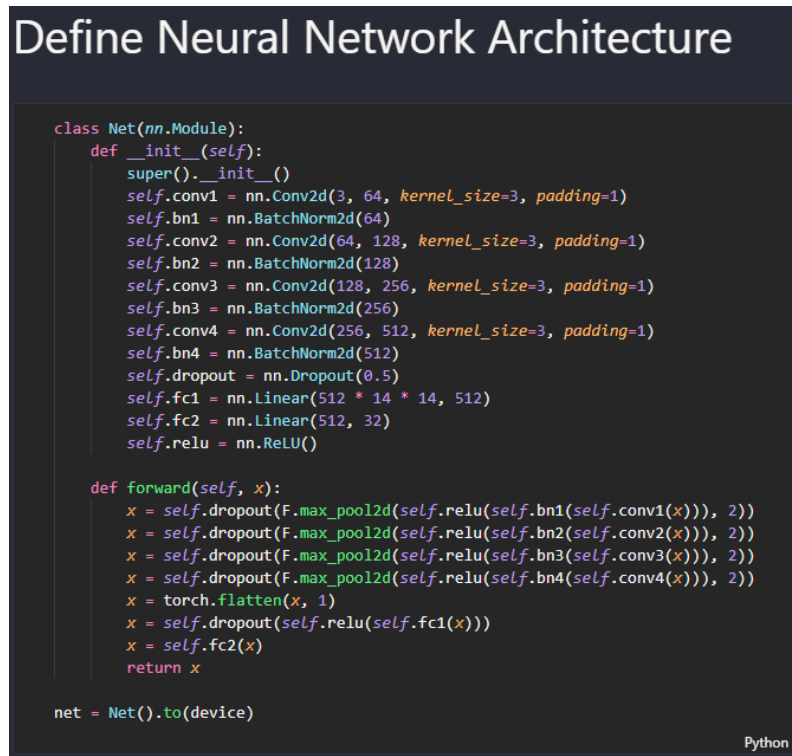
2.1	Neural Network Architecture	2
2.2	Representation of Input Batch	3
2.3	Representation of the Layers	5

Introduction

Architecture of Model

The model that was developed is a convolutional neural network (CNN). CNNs are based on neurons that are organised in layers, this enables CNNs to hierarchical representations (Kattenborn et al., 2021). The architecture of any CNN, this includes the model that was developed to write up this report, consists out of the following layers as stated by Bhatt et al. (2021):

- Input layer
- Convolution layer
- Batch normalisation layer
- Activation function (Nonlinearity layer)
- Pooling layer
- Dropout layer
- Fully connected layer
- Output layer



```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)
        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(256)
        self.conv4 = nn.Conv2d(256, 512, kernel_size=3, padding=1)
        self.bn4 = nn.BatchNorm2d(512)
        self.dropout = nn.Dropout(0.5)
        self.fc1 = nn.Linear(512 * 14 * 14, 512)
        self.fc2 = nn.Linear(512, 32)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.dropout(F.max_pool2d(self.relu(self.bn1(self.conv1(x))), 2))
        x = self.dropout(F.max_pool2d(self.relu(self.bn2(self.conv2(x))), 2))
        x = self.dropout(F.max_pool2d(self.relu(self.bn3(self.conv3(x))), 2))
        x = self.dropout(F.max_pool2d(self.relu(self.bn4(self.conv4(x))), 2))
        x = torch.flatten(x, 1)
        x = self.dropout(self.relu(self.fc1(x)))
        x = self.fc2(x)
        return x

net = Net().to(device)
```

Python

Figure 2.1: Neural Network Architecture

2.1 Input Layer

The images of the fruits that the model is trained gets inputted into the model in batches of 32. The images are then resized to 224x224 pixels with the 3 colour channels, RGB (Red Green Blue). Each of the colour channels are normalised by using a mean of [0.485, 0.456, 0.406] and a standard deviation of [0.229, 0.224, 0.225].

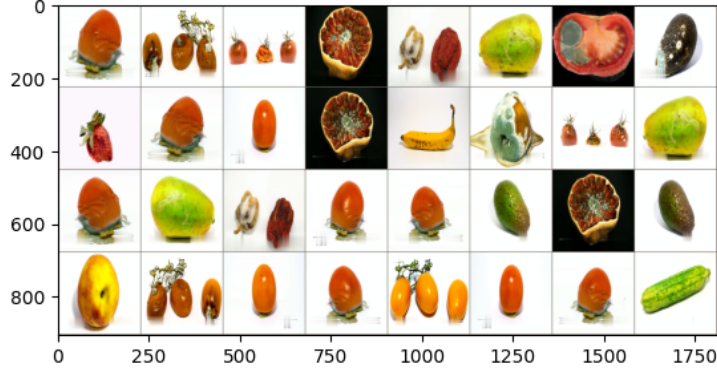


Figure 2.2: Representation of Input Batch

2.2 Convolution Layer

This model consists out of 4 convolutional layers. Each of the convolutional layers are followed by batch normalisation, activation, pooling, and dropout.

The parameters for each convolutional layer is structured as follows. The input colour channels, the colour channels as the images are entered into the model. The output channels, the new colour channels after the images pass through a convolutional layer this becomes for the following convolutional layer. The kernel size, the kernel size refers to the dimensions of the sliding window that moves across the input image, the kernel performs element-wise multiplication with the input data it covers, followed by a summation to produce a single output value, this helps in feature extraction, such as detecting edges, textures, or more complex patterns in deeper layers (Ding et al., 2022). The padding, the adding of extra pixels around the border of an image after it has passed through the convolutional layer.

2.3 Batch Normalisation Layer

The batch normalisation helps to standardise and accelerate the training of the model, by normalising the inputs of each of the batches, the batch normalisation is located before the activation function (Kumar and Shankar Hati, 2021).

2.4 Activation Function (Nonlinearity Layer)

The activation function that was used in this model is the ReLU activation function. This activation function introduces non-linearity into the model, by enabling the model to learn complex patterns. ReLU demonstrated that a activation function in the hidden layers can improve the training speed of the model (Ide and Kurita, 2017).

2.5 Pooling Layer

The purpose of this layer is to down size the convolved feature's spatial size, this helps to reduce the computing power that is needed to process the data (Bhatt et al., 2021). For this model maximum pooling was used in the pooling layer. The input tensor is processed by the pooling layer, where a 2x2 kernel moves across the matrix, selecting the maximum value at each position to populate the output matrix (Bhatt et al., 2021).

2.6 Dropout Layer

The dropout layer is a regularisation technique that helps to prevent overfitting by randomly setting a fraction of input units to zero during training (Khan et al., 2019). As stated by Khan et al. (2019) a dropout rate of 0.5 is a standard dropout rate.

2.7 Fully Connected Layer

The fully connected layer, also known as the dense layer, is located at the end of all the hidden layer, and allowse the model to preform classification. The fully connected layer takes input from the final pooling layer, which is flattened before being passed to it (Bhatt et al., 2021). Flattening transforms the 3D output from the previous layer into a single vector (Bhatt et al., 2021). The FC layer then learns nonlinear combinations of high-level features from the convolutional layer, allowing it to model complex functions in that space (Bhatt et al., 2021).

2.8 Output Layer

The output layer is the last layer for a CNN model. The output that the model provides can be between any of the 32 different classes that the dataset exist out of. The activation function is also applied here to determine the probability of the model accurately predicting the output. Based on this predicting the model does backpropagation and change values to help improve the training process. Since the model is being saved, this makes it possible to apply transfer learning, by using the saved model to train on a new dataset.

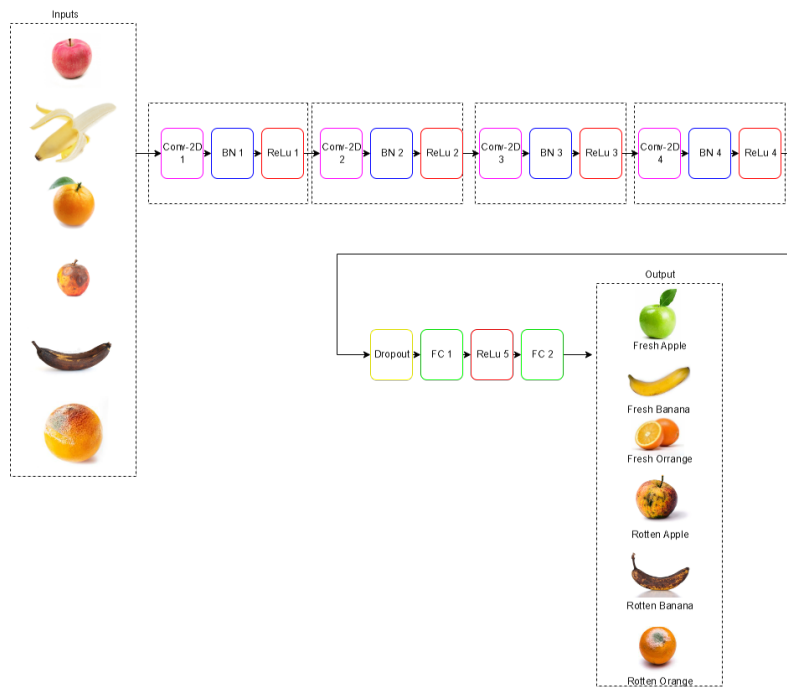


Figure 2.3: Representation of the Layers

Performance Evaluation

The model that was developed could train for a maximum epoch of 150. A patience of 10 epochs was added to prevent overfitting the model, by stopping if no improvements to the model has been made after 10 epochs. During the training process of the model metrics such as loss per epoch, accuracy per epoch, F1-score per epoch, and the ROC and AUC were collected and saved in their own respective text files. This section will discuss each of these metrics of the model.

3.1 Loss

3.2 Accuracy

3.3 F1-Score

3.4 ROC AUC

Conclusion

Bibliography

- Bhatt, D., Patel, C., Talsania, H., Patel, J., Vaghela, R., Pandya, S., Modi, K., and Ghayvat, H. (2021). Cnn variants for computer vision: History, architecture, application, challenges and future scope. *Electronics*, 10(20):2470.
- Ding, X., Zhang, X., Han, J., and Ding, G. (2022). Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11963–11975.
- Ide, H. and Kurita, T. (2017). Improvement of learning for cnn with relu activation by sparse regularization. In *2017 international joint conference on neural networks (IJCNN)*, pages 2684–2691. IEEE.
- Kattenborn, T., Leitloff, J., Schiefer, F., and Hinz, S. (2021). Review on convolutional neural networks (cnn) in vegetation remote sensing. *ISPRS journal of photogrammetry and remote sensing*, 173:24–49.
- Khan, S. H., Hayat, M., and Porikli, F. (2019). Regularization of deep neural networks with spectral dropout. *Neural Networks*, 110:82–90.
- Kumar, P. and Shankar Hati, A. (2021). Convolutional neural network with batch normalisation for fault detection in squirrel cage induction motor. *IET electric power applications*, 15(1):39–50.