Advanced topics in Deep Learning - Mid Semester Project

Stav Elizur Tommy Afek

Submitted as mid-semester project report for Advanced topics in Deep Learning course, Colman, 2023

1 Introduction

This project is a conclusion of all the skills we have acquired during our classes mainly relating to building a convolutional neural network by solving a computing vision multi class classification problem. For this project the problem is a classification of 4 types of Coffee Beans. This project is built with Tensorflow Keras platform, which is a neural network python library that is mainly used for building a Neural Network in much easier and simpler way.

1.1 Data

The data we have for the project is a data set divided to 2 folders of test and train. Each folder has inside images of 4 different Coffee beans types: Dark, Light, Green and Medium. The data set is compressed to a zip file that had 4 folders for each type. Total of 1600 examples. We also have a CSV that has data of each image and its classification which we didn't use because the Keras library does the classification by the images folder path. Basically if the image is inside a "Dark" category folder, The Keras method we used to load the data set automatically classifies it as as a "Dark" image.

1.2 Problem

The problem of our project is a multi class classification of 4 different types of Coffee Beans images.

2 Solution

2.1 General approach

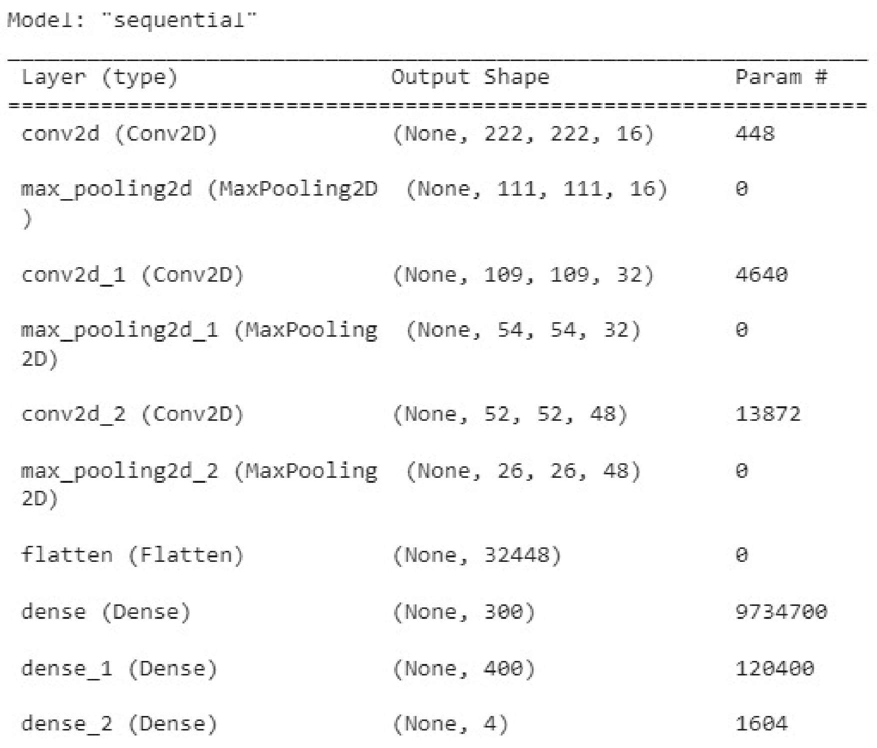
In order to solve our multi class images classification problem we built a convolutional neural network. The primary feature that were different between each Coffee Bean type was their color and also their shapes. A convolutional neural network is a Deep Learning algorithm that can take in an input image, assign importance of features in the image and learn them and to be able to differentiate one from the other and classify them by their features it extracted.

2.2 Design

Our project code divides into 5 parts: Data pre-processing, Building the CNN, Training the CNN, Making a single prediction and experiences with Performance evaluation. In the Data pre-processing part we explore the data, we learn how much classes we have, what are those classes and we view random images that belong to each class with functions we wrote in order to visualize to us what kind of features do we have. Also, we learn from the exploration how difficult is the data set, cause we need to be able to give a better approximation to how our CNN architecture is supposed to look like. Then, we process our Training and Data set by using the Keras Image Data Generator method which generates batches of tensor image data with real-time data augmentation (rescale=1./255) to adjust our data to be able to use it in our convolutional neural network. In the next part, we always load the best model in order to save run time and we skip the CNN building and training process because we get from those processes the same model we loaded. When we build the CNN, We use Keras's tools for the building and training processes, which makes each stage of our building and training processes easier. We initialize our CNN network, add max pooling and filtering layers for the feature extraction layers and the flatten layer. The filtering layers use Relu activation function that takes the maximum value after computing. After that, we add the fully connected layers with Sigmoid activation function for the label classification itself and the final output layer with the Softmax activation function for the prediction. Softmax activation function is used to give a probability for each class prediction. Then we compile our CNN with the Adam optimizer, which is the best variation of the Gradient Descent algorithm because it has the smallest slope between the rest of the algorithms so it gives the best results for loss per epochs. We use the loss categorical cross entropy function which does severity punishment for our model depending on the distance between the predicted value and the actual value. Also we defined that the accuracy metrics between each epochs will be calculated and saved to make sure our model is not over fitted. Then, that we train our CNN with the training set, test set and number of epochs and after that we save it. The training process took 58 seconds. In the last part, after our CNN is built, we evaluate our performances using the accuracy metrics our model has to calculate our the validation and test loss and accuracy in each epoch and we create the confusion matrix for our experiments to define which experiment has better results. The technical challenges were understanding how many layers we should use for building the CNN, both for the feature extraction and classification layers which we improve by experimenting values.

2.3 Base Model

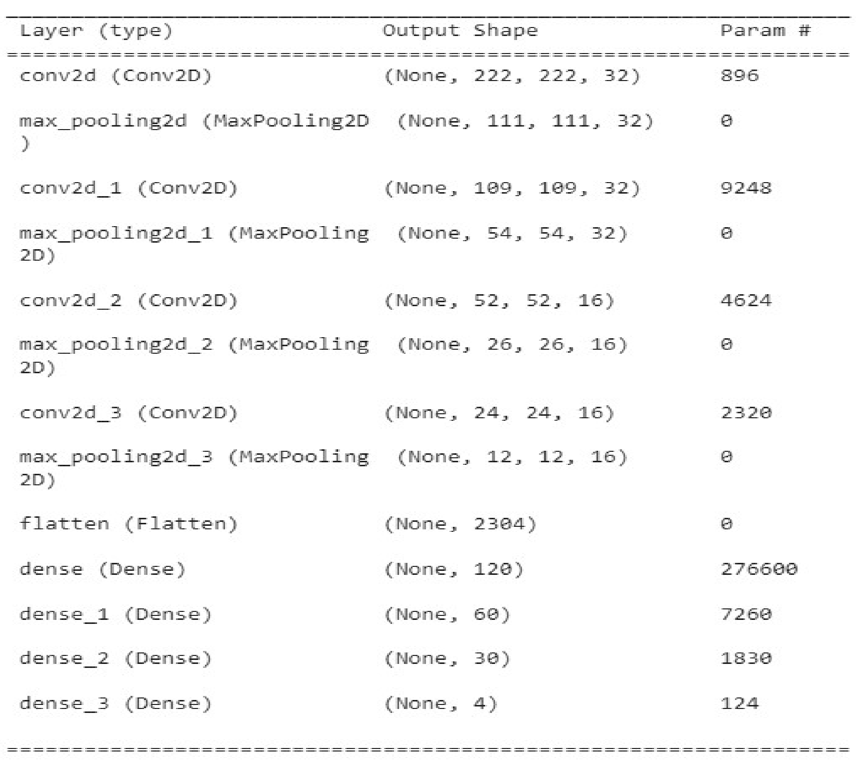
Our base model is a convolutional neuron network. A convolutional neural network has the ability to extract this feature and use it for our classification. This ability is called feature extraction which is a method that uses a filtering pooling layers. The filtering layer is basically a matrix with the same deepness of our image target that checks each section in the image with the convolutional method, which is a multiplication of the cells of both matrices: the filter matrix in the image and the section matrix that the filtering is currently on, then we do a summary of all the cells. Also, we can do this with many filters as we like. The layers of pooling takes the maximum cell value of the section the pooling matrix is currently computing and it can have striding to skip sections. After the feature extraction there is the flatten method which creates a vector of our matrix output, which is basically a label representing our image. Then, there are fully connected layers that are used for the classification itself. Our base model has layers in the following order: a convolutional layer with 16 filters that each filter is 3x3x3 and a 224x224x3 input shape, a 2x2 max pooling layer with 2 strides, a convolutional layer with 32 filters that each filter is 3x3x16, a 2x2 max pooling layer with 2 strides, a 48 filters that each filter is 3x3x32, a 2x2 max pooling layer with 2 strides. Then, we have the flatten layer from the last layer which is 26x26x48 to a vector in length of 32448, a fully connected layer of 300 neurons, a fully connected layer of 400 layers, an output layer of neuron per class, in summary, 4 neurons.


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 16)      448

 max_pooling2d (MaxPooling2D  (None, 111, 111, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 48)        13872

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 48)       0
 2D)

 flatten (Flatten)           (None, 32448)             0

 dense (Dense)               (None, 300)               9734700

 dense_1 (Dense)             (None, 400)               120400

 dense_2 (Dense)             (None, 4)                 1604
```

CNN architecture of the base model

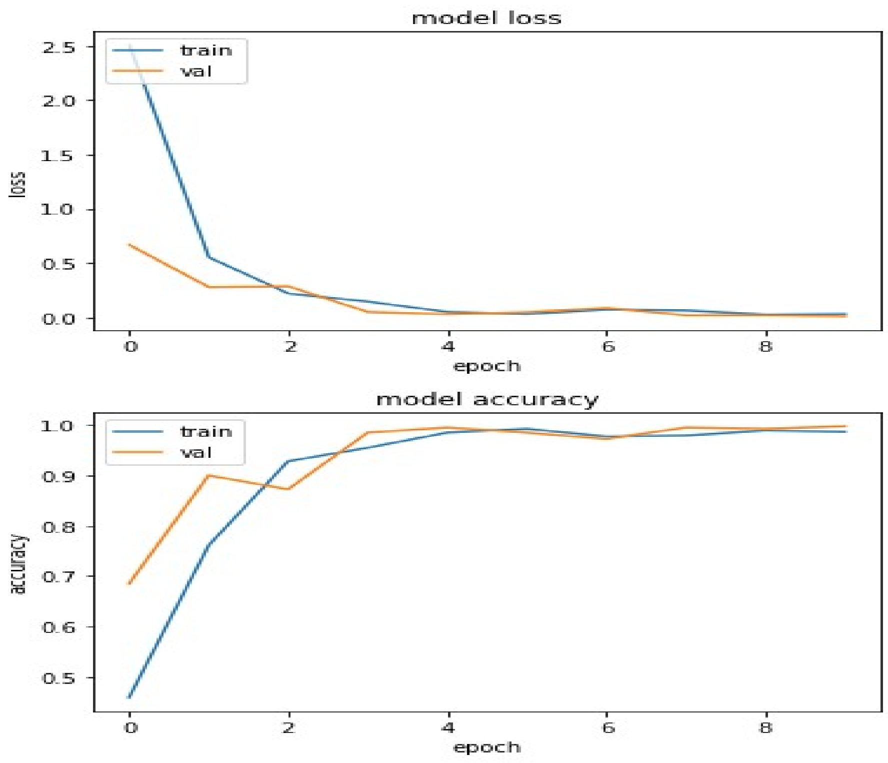
2.4 First Experiment

In our first experiment, we paid more attention to our CNN architecture. we had the same feature extraction layers using the same filter matrix for the convolutional layers and filter matrix and strides for the pooling layers, it only differs from the base model with the number of filters: a convolutional layer with 32 filters, a max pooling layer, a convolutional layer with 32 filters, a max pooling layer, a convolutional layer with 16 filters, a max pooling layer, a convolutional layer with 16 filters, a max pooling layer and a flatten layer. Then, in the classification layers we had 1 more fully connected layer than our base model and also a different number of neurons per fully connected layer than our base model. The order of the layers is: a fully connected layer with 120 neurons, a fully connected layer with 60 neurons, a fully connected layer with 30 neurons and an output layer with 4 neurons. The number of epochs we used to train our neural network was 10, the each batch size of our data was 40 and Learning Rate of 0.001, same as the base model hyper parameters.

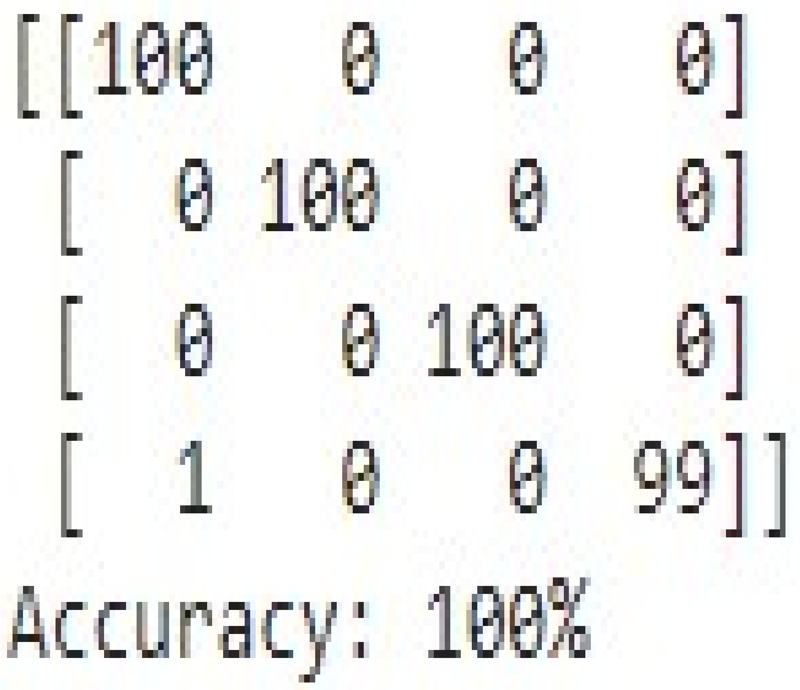


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2D  (None, 111, 111, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 32)      9248

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 16)        4624

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 16)       0
 2D)

 conv2d_3 (Conv2D)           (None, 24, 24, 16)        2320

 max_pooling2d_3 (MaxPooling  (None, 12, 12, 16)       0
 2D)

 flatten (Flatten)           (None, 2304)              0

 dense (Dense)               (None, 120)               276600

 dense_1 (Dense)             (None, 60)                7260

 dense_2 (Dense)             (None, 30)                1830

 dense_3 (Dense)             (None, 4)                 124

=================================================================
```

CNN architecture of the first experiment



Graphs of loss and accuracy by epochs of the first experiment

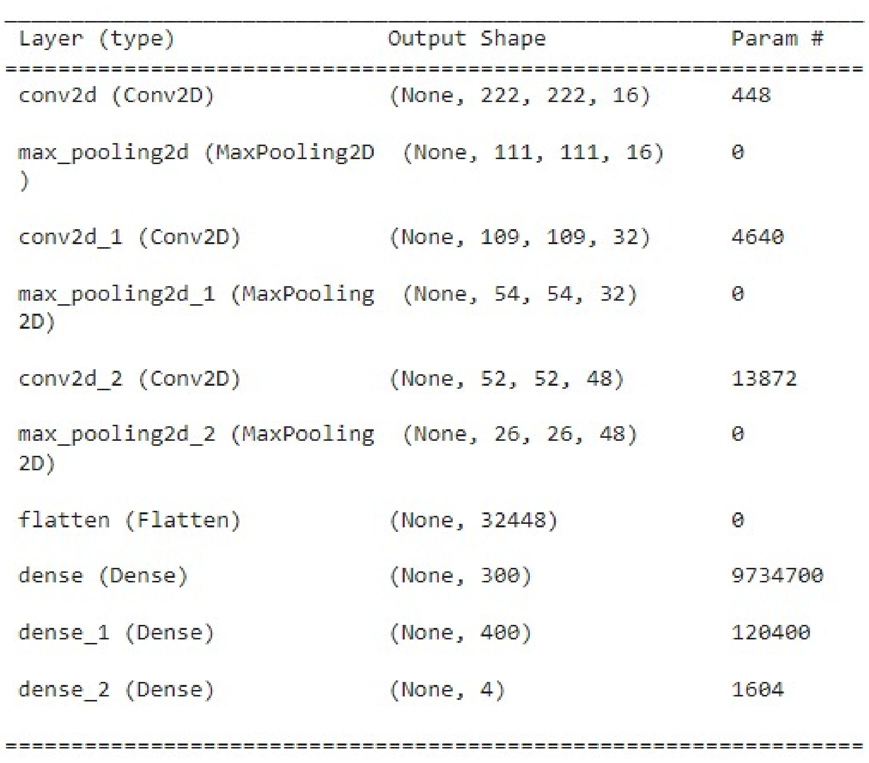


```
[[100   0   0   0]
 [  0 100   0   0]
 [  0   0 100   0]
 [  1   0   0  99]]
Accuracy: 100%
```

Graph of the confusion matrix of the first experiment

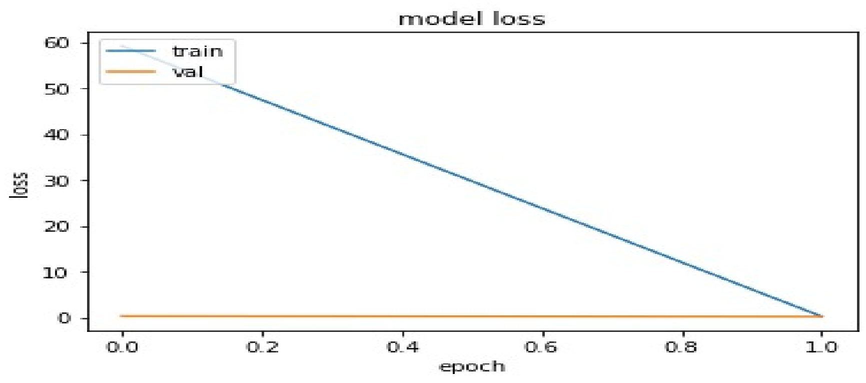
2.5 Second Experiment

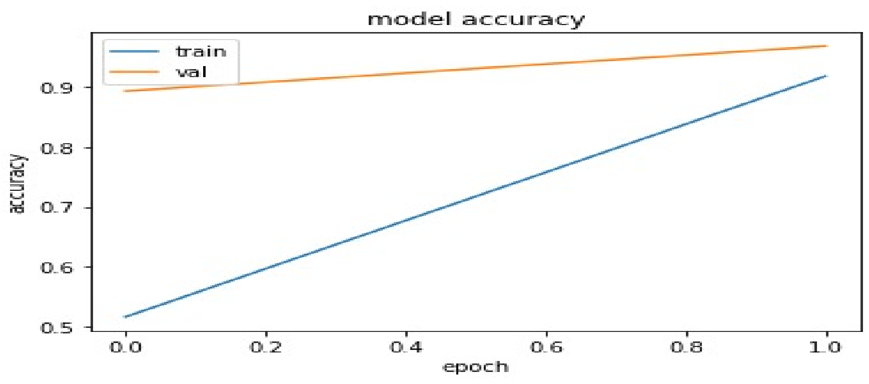
In our second experiment, we focused more in our hyper-parameters and used different numbers than the base model. We had the exact same architecture as the base model CNN, using different hyper-parameters which are the batch size of 20, learning rate of 0.001 and 2 number of Epochs.



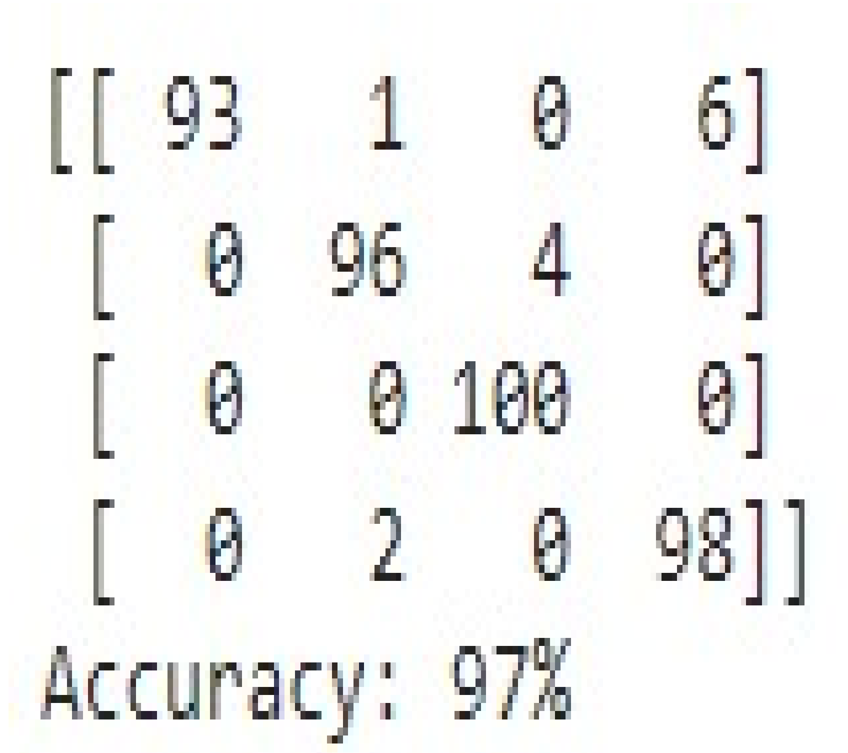
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 16)      448

 max_pooling2d (MaxPooling2D  (None, 111, 111, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 109, 109, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 48)        13872

 max_pooling2d_2 (MaxPooling  (None, 26, 26, 48)       0
 2D)

 flatten (Flatten)           (None, 32448)             0

 dense (Dense)               (None, 300)               9734700

 dense_1 (Dense)             (None, 400)               120400

 dense_2 (Dense)             (None, 4)                 1604

=================================================================
```

CNN architecture of the two experiment





Graphs of loss and accuracy by epochs of the second experiment

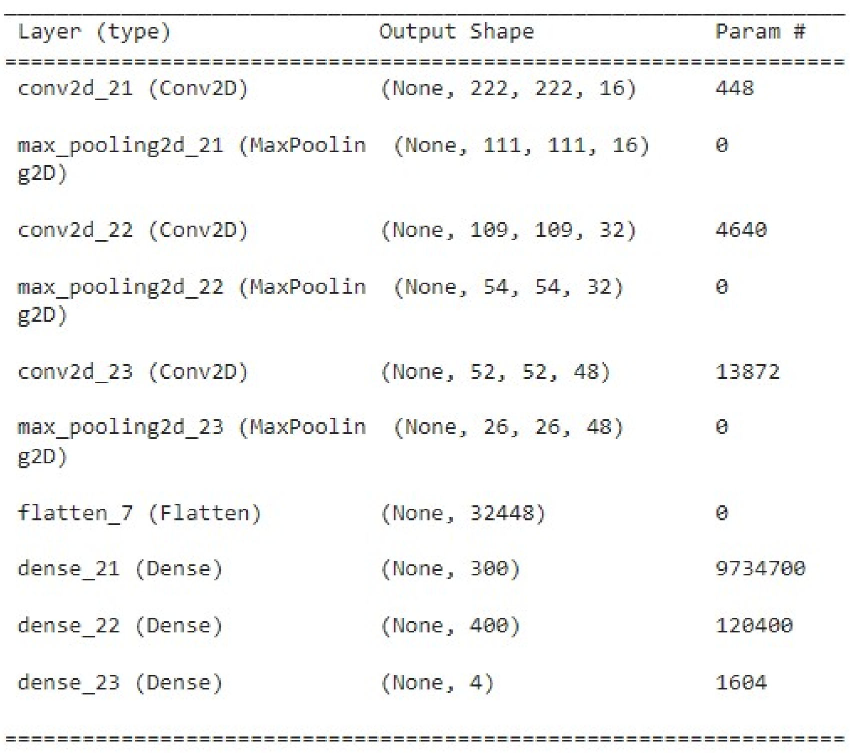


```
[[ 93   1   0   6]
 [  0  96   4   0]
 [  0   0 100   0]
 [  0   2   0  98]]
Accuracy: 97%
```

Graph of the confusion matrix of the second experiment

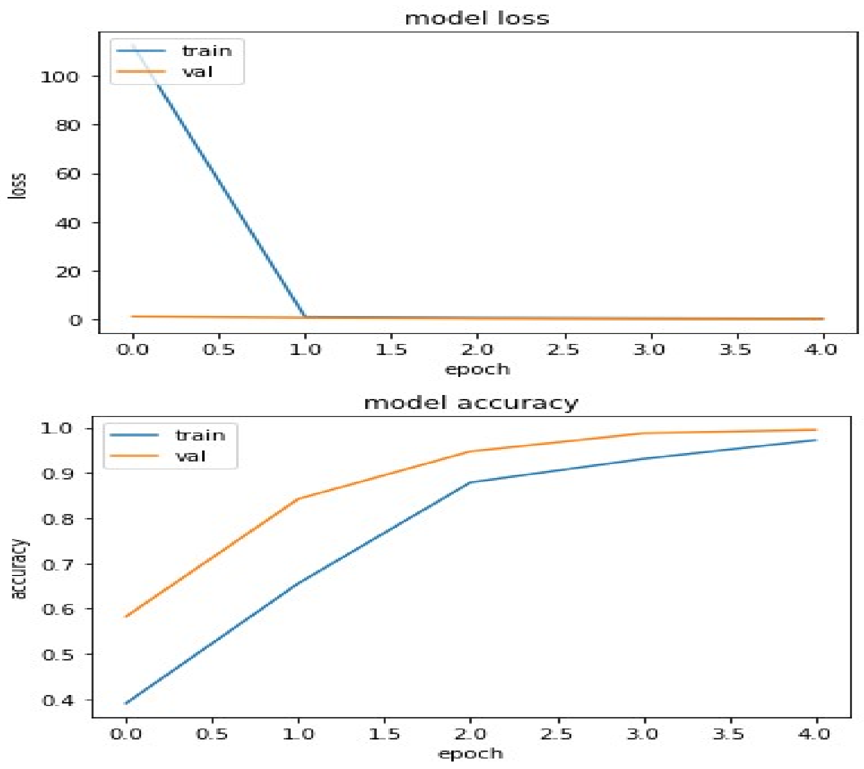
2.6 Best model Results and Metrics

After we ran the 2 experiments, we created a combination of them with our base model. We saw the our base model was enough and adding more convolutional layers was unnecessary so We used the architecture of our base model with different hyper parameters: Learning rate- 0.001, Epochs- 5, Batch size- 40. The result was the best model we trained.

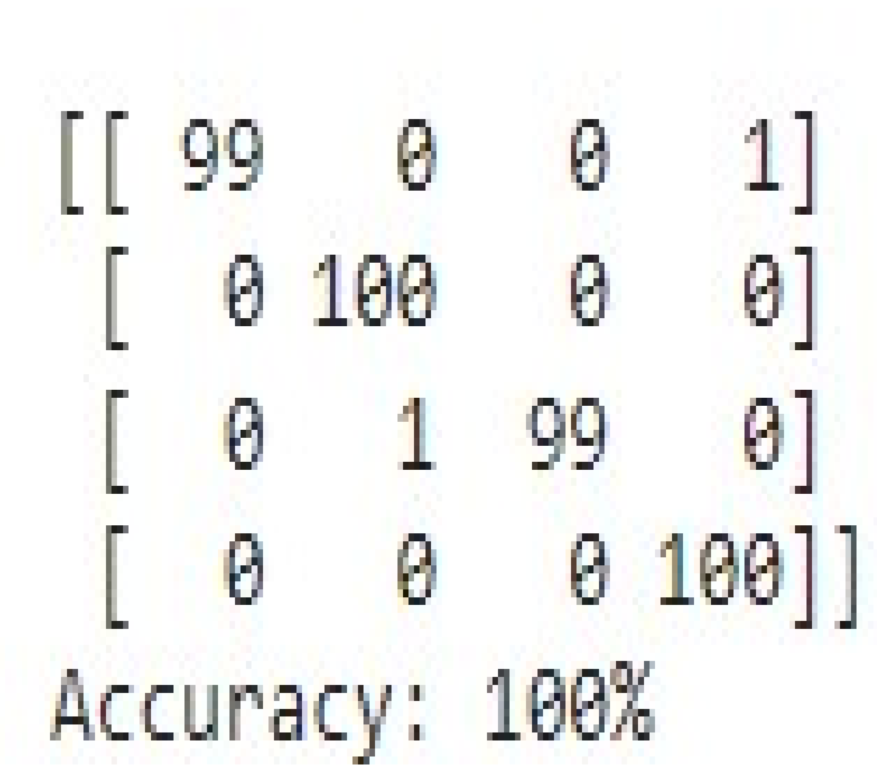


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_21 (Conv2D)          (None, 222, 222, 16)      448

 max_pooling2d_21 (MaxPoolin  (None, 111, 111, 16)     0
 g2D)

 conv2d_22 (Conv2D)          (None, 109, 109, 32)      4640

 max_pooling2d_22 (MaxPoolin  (None, 54, 54, 32)       0
 g2D)

 conv2d_23 (Conv2D)          (None, 52, 52, 48)        13872

 max_pooling2d_23 (MaxPoolin  (None, 26, 26, 48)       0
 g2D)

 flatten_7 (Flatten)         (None, 32448)             0

 dense_21 (Dense)            (None, 300)               9734700

 dense_22 (Dense)            (None, 400)               120400

 dense_23 (Dense)            (None, 4)                 1604

=================================================================
```

CNN architecture of the best model



Graphs of loss and accuracy by epochs of the best model



```
[[ 99   0   0   1]
 [  0 100   0   0]
 [  0   1  99   0]
 [  0   0   0 100]]
Accuracy: 100%
```

Graph of the confusion matrix of the best model

3 Discussion

In summary, after we ran the experiments we understood from the first experiment which was more focused in the CNN architecture itself that not always having more convolutional layers can give us better performance. We learned from the lectures a term about receptive fields which means that the inside layers can see bigger pictures than the previous convolutional layers. Also it can affect badly on our algorithm because the more layers we have, we take a look on unnecessary pixels such as background pixels. After we ran the second experiment which was more focused on the hyper parameters, we learned that having more epochs should cause over fitting, and the learning rate is dependable on the epochs because when we have a small learning rate, getting to the minimal loss will take a lot of epochs and we needed to find the balance between them. We also learned that the batch size controls the accuracy of the estimate of the error gradient when training neural networks.

4 Code

Colab project