Final report - Tools classification Project

Project Guides:

Prof. GL Samuel

Prof Somashekhar S Hiremath

By:

Shreeniwas Jagdale(ME17B115)

Utkarsh Kumar(ME17B123)

Index

Abstract:

Segregation between various objects is the core idea behind this project. The idea of segregation can be extended to many industries, where automation is required as this reduces the human effort and time. This helps in optimizing a process making the assembly line or the work area more efficient, incurring profits.

Introduction:

The problem was to segregate mixed up tools in a manufacturing unit. There is a possibility that different types of spanners, screwdrivers can get mixed up and it would be time-consuming to sort these and arrange. Not only will it consume time but reduces the productivity of the workers which could have been applied to other places. This can also mean that we can sort between various sizes of nuts, bolts, and washers. Segregation when completely automated, saves time and increases profits.

Applications:

The above code has been applied to segregate various tools namely spanner, hammer, and screwdriver. This can be extended to segregate various objects, tools, and their varieties. A few applications of the same are mentioned as follows

- Nuts and bolts of different sizes can be separated with the help of an embedded mechanical system. This will help out big manufacturing plants save time.
- Sometimes flaws in products are too small to be noticed by the naked eye. Very sensitive cameras can be used to detect the smallest defects. This can reduce human effort, time, and money.
- An example of the above is in the automotive industry where engine cylinders are manufactured. The cylinders can be checked using this technology as it involves a high degree of precision.

Approach

Basically it’s a classification problem with 3 classes namely hammer, screwdriver, and spanners. We are using a convolutional as well as a dense layer model for this problem. Let’s discuss every step taken by us to approach the final model

Data Set creation

As this problem is uniquely defined there was no available dataset for this problem on the internet or any other online depository such as Kaggle. So we had to create our dataset manually. Creating a dataset manually was not an easy task, we had to download 512 images from google and had to take photographs from our cameras to form the required dataset. Finally, we were able to get around 917 images totally. Various images were taken by our phone cameras in the central workshop and CFI.

Here is the **link to our dataset:**
https://drive.google.com/open?id=1XerDv803_-uLUW-ANTuoMGMOORYYPRne

Pre-processing on the obtained dataset

- **Resizing of images to a particular size was done in order to get uniform output**

    Resizing of all the images to a particular size was done in order to get a standardized data set, and as we know standardized data set helps in removal of non-uniformity from the output. It also helps the algorithm to converge to the required output in an optimum way.
- **Images were divided and stored in 3 categories namely Hammer, screwdrivers, and Spanner.**

    This was done so that classifier will automatically recognize there are 3 classes and gives an output accordingly. Three categories were formed for this problem namely hammer, screwdrivers, and spanners.
- **Each Image was given named sequentially in every group**
- **All the datasets were uploaded and saved on a drive**

    As we were using google collab, we uploaded all the data on our google drives so that while training our model we can just mount the google drive and no need to upload data every time. This eased our job and improved our efficiency.

Dataset division

As you know for classifications problems, the data set has to be divided into train data as well as test data. The formed dataset was divided manually into the test dataset and the training dataset(1:15 ratio was taken for division). The training dataset was used to train the model while the test dataset was used to test various images and crosscheck the output. The maximum of the dataset was used for training our model.

Model creation for training

- **The model was created using convolutional layers with an active function of 'relu'**

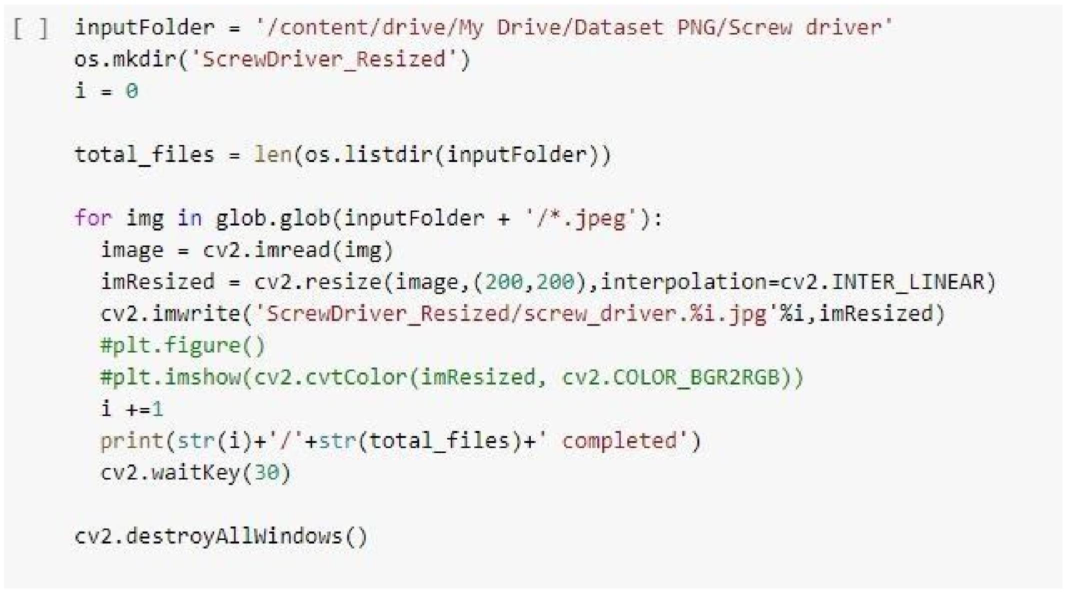
  When programming a **CNN,** the input is a tensor with shape (number of images) x (image width) x (image height) x (image depth). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map width) x (feature map height) x (feature map channels).
  Relu function is the modification of normally used sigmoid function and known for its a good performance with CNNs
- **The max-pooling method was used for downsampling**

  Convolutional networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.
- **Then output was flattened and normal dense layers were used to train with relu function.**

  As max-pooling operation reduces the dimensions of the data, we can now introduce dense layer neural networks to make our model further smart and efficient. Here again, relu function was used during activation. A densely connected layer provides learning features from all the combinations of the features of the previous layer, whereas a convolutional layer relies on consistent features with a small repetitive field.

The coding part of our project

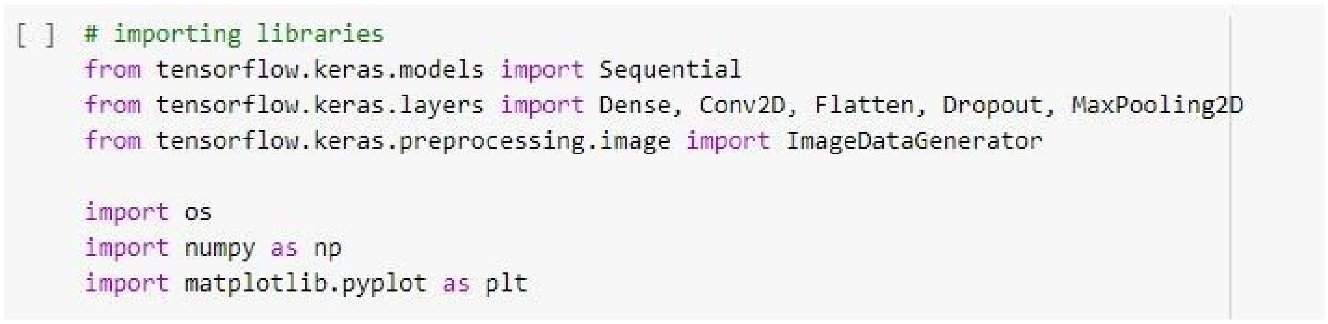
Preprocessing of data:



```
[ ] inputFolder = '/content/drive/My Drive/Dataset PNG/Screw driver'
    os.mkdir('ScrewDriver_Resized')
    i = 0

    total_files = len(os.listdir(inputFolder))

    for img in glob.glob(inputFolder + '/*.jpeg'):
      image = cv2.imread(img)
      imResized = cv2.resize(image,(200,200),interpolation=cv2.INTER_LINEAR)
      cv2.imwrite('ScrewDriver_Resized/screw_driver.%i.jpg'%i,imResized)
      #plt.figure()
      #plt.imshow(cv2.cvtColor(imResized, cv2.COLOR_BGR2RGB))
      i +=1
      print(str(i)+'/'+str(total_files)+' completed')
      cv2.waitKey(30)

    cv2.destroyAllWindows()
```

As you can see above in code, data was resized and saved in different folders with names like 1,2,3 and so on for each folder. This reduced the complexity of training and helped us to code effectively. As told previously resizing helped us to remove non-uniformity from the model as well as it helped to converge to an output in an optimum way.

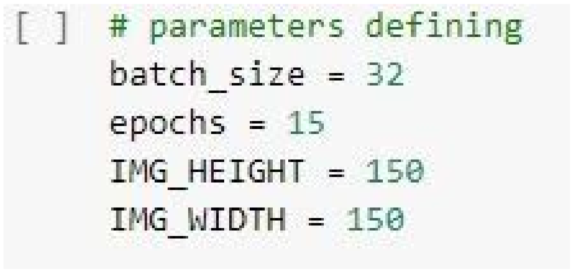
Importing libraries:



```
[ ] # importing libraries
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
    from tensorflow.keras.preprocessing.image import ImageDataGenerator

    import os
    import numpy as np
    import matplotlib.pyplot as plt
```

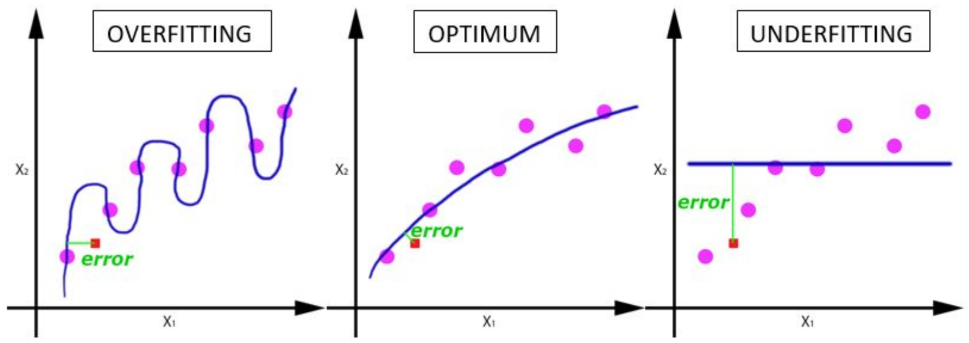
Various libraries were imported for the model, which you can see in the image attached above.

Hyper-parameters tuning:

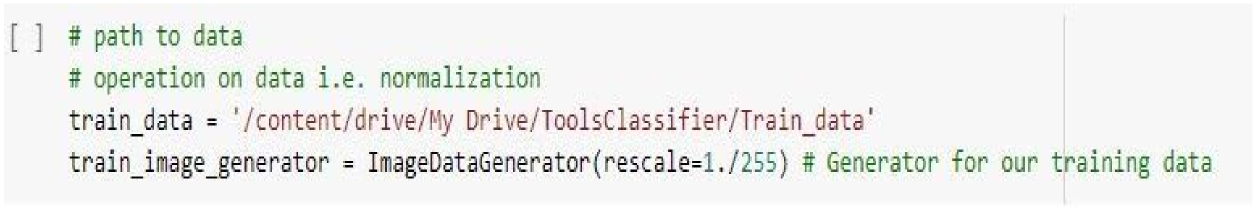


```
[ ] # parameters defining
    batch_size = 32
    epochs = 15
    IMG_HEIGHT = 150
    IMG_WIDTH = 150
```

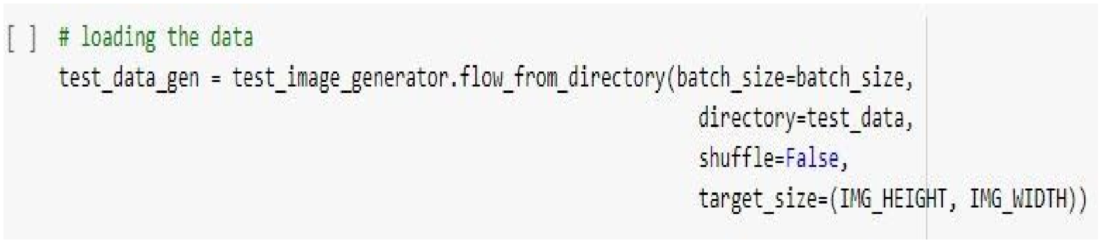
Various hyper-parameters such as batch size, epochs, and image properties were tuned according to the observations that we found during various experiments. As we know parameters such as batch size and epoch can significantly change the behavior of curve fitting during gradient descent. As epochs increase, they tend to change from underfitting curves to overfitting curves. Hence choosing optimum values for no. of epochs and batch size is essential during the training of CNN.



Data loading:



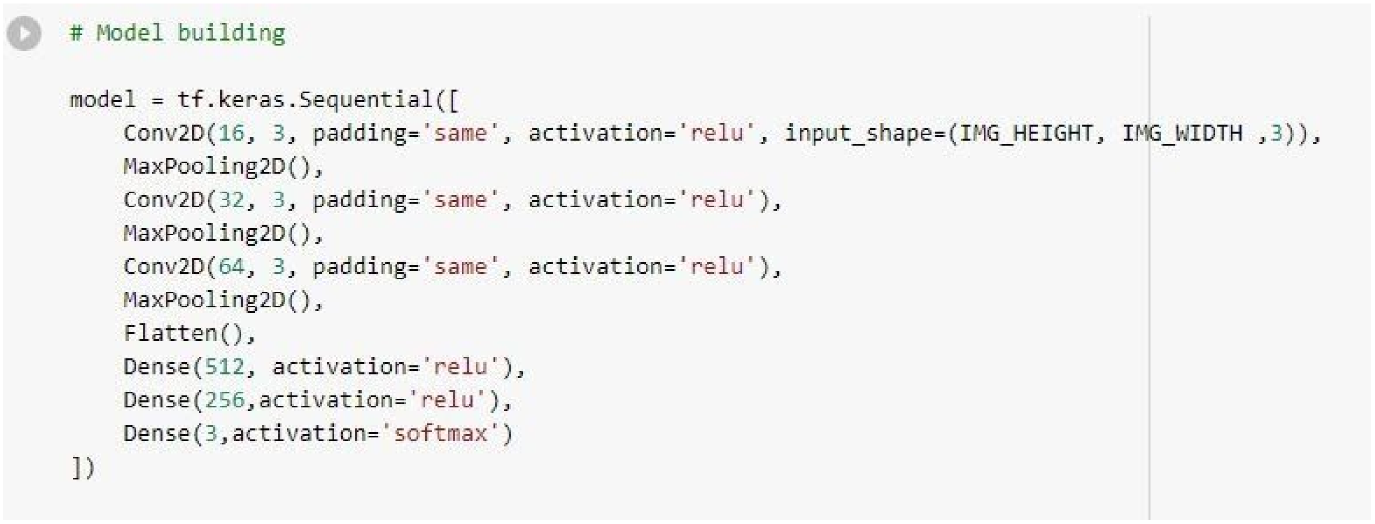
```
[ ] # path to data
    # operation on data i.e. normalization
    train_data = '/content/drive/My Drive/ToolsClassifier/Train_data'
    train_image_generator = ImageDataGenerator(rescale=1./255) # Generator for our training data
```



```
[ ] # loading the data
    test_data_gen = test_image_generator.flow_from_directory(batch_size=batch_size,
                                                             directory=test_data,
                                                             shuffle=False,
                                                             target_size=(IMG_HEIGHT, IMG_WIDTH))
```

Data was uploaded on a drive and then it was loaded into code after mounting the drive into a collab. As you can see above, testing data and training data was loaded into the code using various commands.

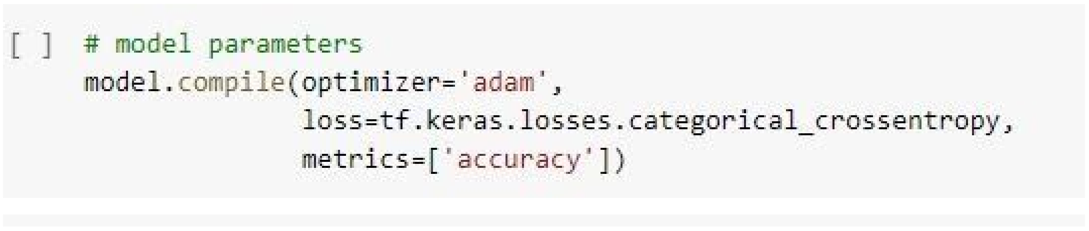
Model building:



```
# Model building

model = tf.keras.Sequential([
    Conv2D(16, 3, padding='same', activation='relu', input_shape=(IMG_HEIGHT, IMG_WIDTH ,3)),
    MaxPooling2D(),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(256,activation='relu'),
    Dense(3,activation='softmax')
])
```

As told previously first convolutional layers were used followed by max-pooling and in the end dense layers were used. No. of layers was decided according to previous experience and observations during experiments

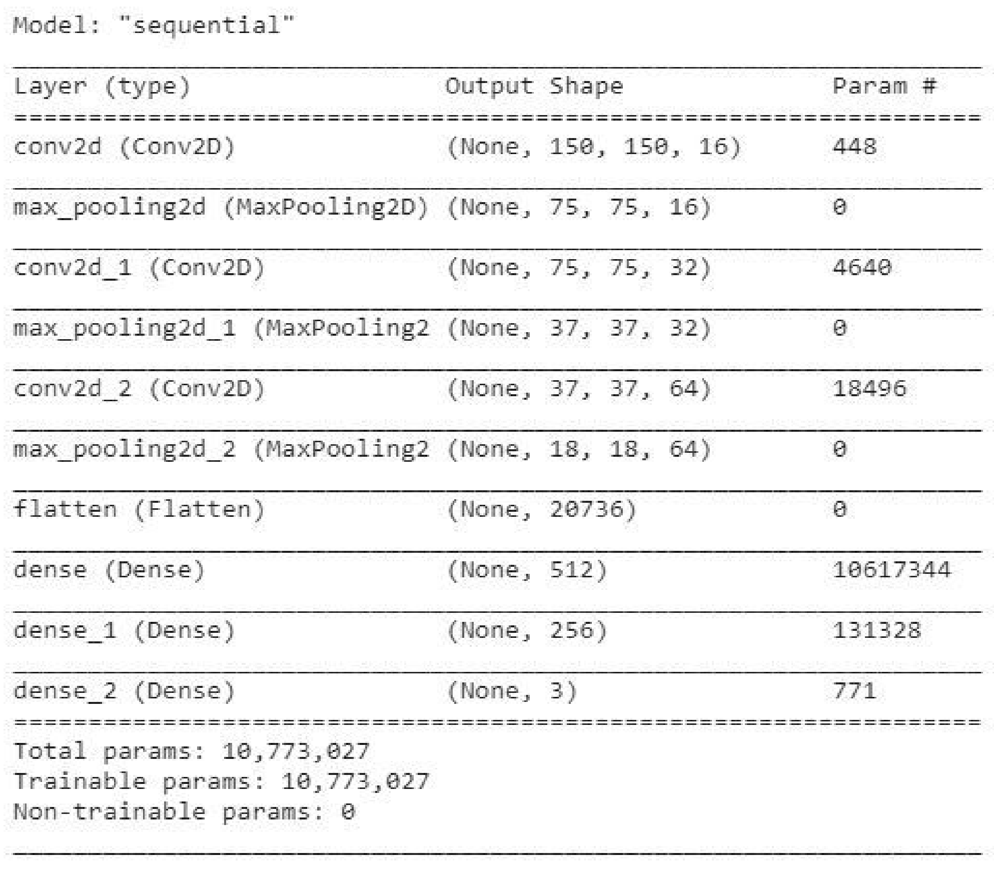
Model parameters:



```
[ ]  # model parameters
     model.compile(optimizer='adam',
                   loss=tf.keras.losses.categorical_crossentropy,
                   metrics=['accuracy'])
```

- Adam is an adaptive learning rate optimization algorithm that’s been designed specifically for training deep neural networks.
- Cross-entropy loss function was used to calculate loss fraction.

Model summary:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 150, 150, 16)      448       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 16)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 32)        4640      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 64)        18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 64)        0         
_________________________________________________________________
flatten (Flatten)            (None, 20736)             0         
_________________________________________________________________
dense (Dense)                (None, 512)               10617344  
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328    
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 771       
=================================================================
Total params: 10,773,027
Trainable params: 10,773,027
Non-trainable params: 0
_________________________________________________________________
```

Tools used:

Dataset for the problem set was created manually and tools used for the project were google collab and it’s libraries & functions.

- Advantages of **google collab**:
  - Each notebook can use high-end google GPU
  - Free version
  - Inbuilt version controlling system
  - We can write customized code

Conclusion:

- **As the no. of layers are increasing, accuracy is increasing**
- **As the batch size is increasing, accuracy is decreasing**

  There is a trade-off between batch size and no. of epochs, we have to hyper tune these parameters, in order to get high accuracy.
- **As dataset size increases, accuracy increases**

  We were limited to the dataset as there was no online resource for that, we can further add images to increase the span of the dataset i.e to train our model more effectively.

- **After looking at the above parameters, we have hyper tuned our model parameters for better accuracy.**
- **After hyperparameter tuning, we got an accuracy of around 0.87**
- **Various improvements can be done in a model to increase the accuracy**

  We will talk about those improvements in future modifications

Future modifications:

- **Datasets can be enlarged to improve accuracy and to reduce loss function.**

  As told before, we were limited to dataset because of unavailability of online sources, our accuracy can be further increased by enlarging our dataset, as it will allow more training sets for the model to train
- **The batch size can be reduced to improve performance.**
- **Data(images) can be passed with different orientations so that model will be trained better and span of the dataset will be increased**

Thank you note:

We would like to thank our professors Dr. Somashekhar Hiremath and Dr. Samuel for constantly motivating us throughout the project, and providing us with valuable feedback and comments upon which we have worked and successfully completed this project, and also for helping us with our doubts after the classes. We would also like to thank our TA for the course, Mr. Rajesh Babu for helping us in various ways throughout the course.

Full code:

https://colab.research.google.com/drive/1yrxyNrK0h7K1kb3XWouJ3t4cchOVZYUF