INSTITUTE FOR ADVANCED
COMPUTING AND
SOFTWARE
DEVELOPMENT
AKURDI, PUNE

Documentation On
“Mechanical Tool Images Using CNN and VGG16”
EDBDA 2021

Submitted By:

Group No: 27
Foram Suratwala 1549
Prajakta Umbarkar 1552

Mr. Prashant Karhale
Centre Coordinator

Mr. Akshay Tilekar (Project Guide)
Mr. Rahul Paunde (Internal Guide)
Mr. Manish (Internal Guide)

Contents

List of Figure

Abstract

Convolutional neural network models were developed to perform tools detection, through deep learning methodologies. Training of the models performed with the use of an open database of 6261 images, containing different types of mechanical tools in a set of 8 distinct labels. Transfer learning VGG16 model architectures were trained, with the best performance reaching 86% accuracy success rate in Mechanical tools such as (Gasoline can, Hammer, Pebble, Pilers, Screw Driver, Tool box, Wrench). Also use feature selection of different techniques perform of image.

Chapter 1

1.1 Introduction:

All machine tools have some means of constraining the workpiece and provide a guided movement of the partMs of the machine. Machine tools employ some sort of tool that does the cutting or shaping. All machine tools have some means of constraining the work piece and provide a guided movement of the parts of the machine.

Mechanical engineers design power-producing machines, such as electric generators, internal combustion engines, and steam and gas turbines, as well as power-using machines, such as refrigeration and air-conditioning systems. Convolution Neural Network and VGG16 perform different Accuracy of model and also used feature selection of different technique.

1.2 Purpose

Hammer, pliers, gasoline can, Wrench, tool box, rope, pliers, Screw drive where different class apply the CNN and VGG16 transfer learning models are check accuracy. Feature selection are used different images performed how to show images effect.

Chapter 2

2.1 Software Life Cycle Model

In order to make this Project we are going to use Classic LIFE CYCLE MODEL. Classic life cycle model is also known as WATER FALL MODEL. The life cycle model demands a Systematic sequential approach to software development that begins at the system level and progress through analysis design coding, testing and maintenance.

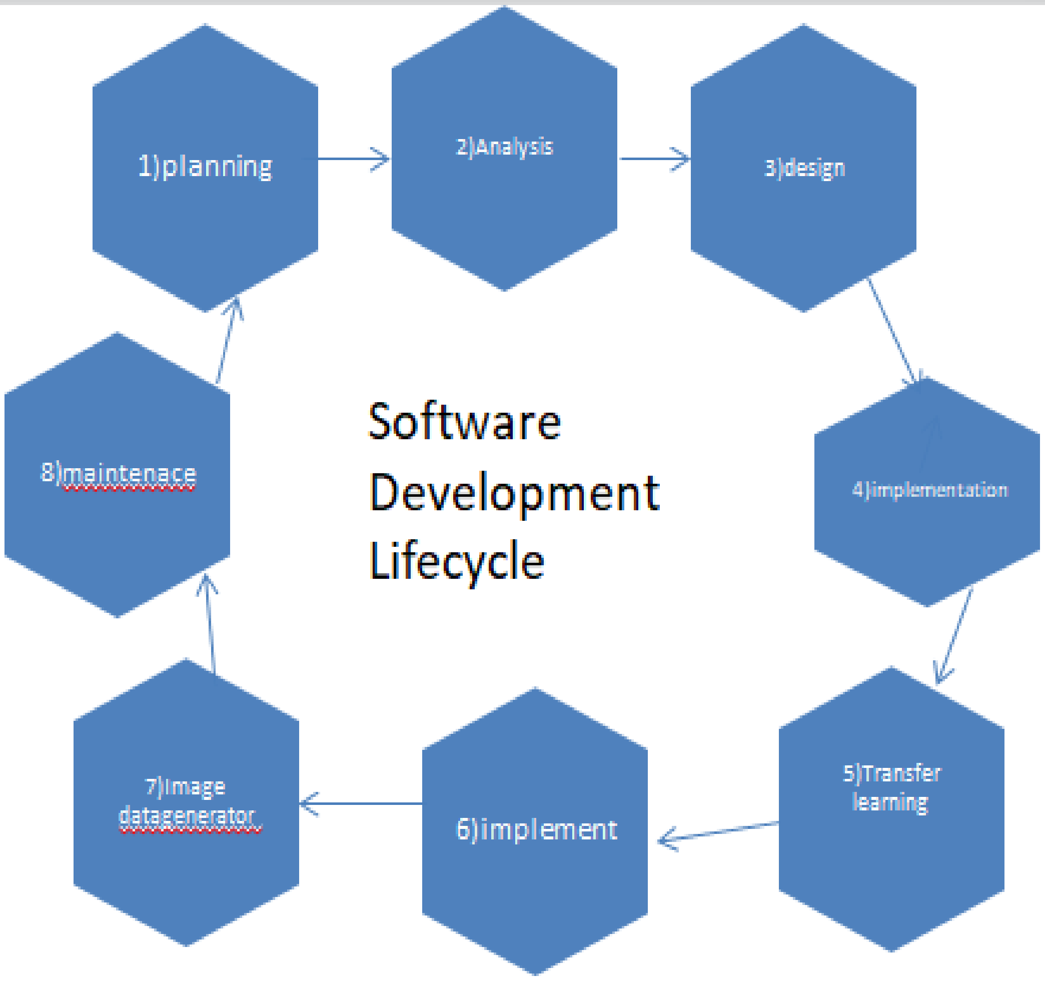


Fig 2.1: - Software development lifecycle

2.2 Overall Description

The waterfall model is sequential software development process, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of conception initiation, Analysis, Design (splitting of dataset using train image and train label images), Implementation, maintained (accuracy test CNN), transfer learning (VGG16), maintained (accuracy test VGG16), feature selection (rotate, crop, Thresholding, gaussian blur, salt and pepper noise, speckle noise, gaussian noise, canny edge detection.

2.2.1 Dataset:

Train directory name	count
Gasoline Can	0-210
Hammer	211-1837
Pebble	1838-2435
Pliers	2436-2807
Rope	2808-3177
Screw driver	3178-4503
Toolbox	4504-4930
Wrench	4931-6261

2.2.2 Imports library:

➢ Matplotlib:

Matplotlib is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure. We have used it to show visualizations of analysis.

➢ NumPy:

NumPy is used to for mathematical operations. This package provides easy use of mathematical function.

- TensorFlow:

TensorFlow is open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.

- Keras:

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Up until version 2.3 Keras supported multiple backends, including TensorFlow, Microsoft Cognitive Toolkit, R, sequential, optimizer, flatten, dense, maxpooling, dropout, adam, rmsproop, model.fit, imagedatagenerate, convolution2d.

- Glob:

The glob module is used to retrieve files/pathnames matching a specified pattern. The pattern rules of glob follow standard Unix path expansion rules.

2.3 Requirement Specification

2.3.1 Initial nonfunctional requirement will be:

- Getting the large datasets which can provide developer enough data to train the model.
- Maintain the minimum variance and bias so the model is successfully work.
- Avoid the underfitting and overfitting.

2.3.2 Initial functional requirement will be:

- Selecting the appropriate algorithms.
- Determining the appropriate input format to algorithm.
- Train the model.
- Test the model.

2.3.3 Hardware Requirement:

- Processor: Intel core i5 with 7th generate and above
- RAM: 8gb
- OS: Windows 10 and above, Linux

2.3.4 Software Requirement:

- Anaconda Navigator
- Jupyter NoteBook
- Spyder 3.8.5
- Anaconda Prompt, collab

Chapter 3

3.1 Mechanical Tool Execution

3.1.1 Feasible study phase

Study of different distributed algorithms. Different class mechanical tool images are studied and corresponding. If data are unstructured then rename name and create continuous type data then apply apply CNN method but model is not goo then apply transfer learning model that time split folder library are import and create new folder create and see train, test data inside same class is given.

3.1.2 Requirement analysis

The database is pre-processed such as image, reshaping image for row, column (150,150) and np.zero are apply train data image and np.one matrix are apply train label image and then create train data then data after reshape divide 255 and after split x and y train and test data. Train data for 80% and test data for 20% after apply random state because data do not mismatch.

All split x and y split then deep neural method are apply and keras application for transfer learning vgg16 method apply and feature selection apply different images.

3.1.3 Design phase

- Designing algorithm and graphs for getting more accuracy and satisfaction.
- Adjusting the Images in categories so that our model identifies the it.

3.1.4 Implementation phase

- CNN has different layers that are Dense, Dropout, Activation, Flatten, Convolution2D and MaxPooling2D these all will implement.

- VGG16 has the different layers are Flatten, imagedatagenerator, dense.
- Feature selection method are use salt and pepper noise, gaussian noise, speckle noise, thresholding, gaussian blur, canny edge detection, rotate, crop, gray scale image, horizontal and vertical flip image.

3.1.5 Testing phase

- After the model is trained successfully the software can identify the different types of mechanical tools images in the database
- After successful training and pre-processing, comparison of the test image and trained model takes place to predict.

3.2 System Design

3.2.1 Flowchart of the System:

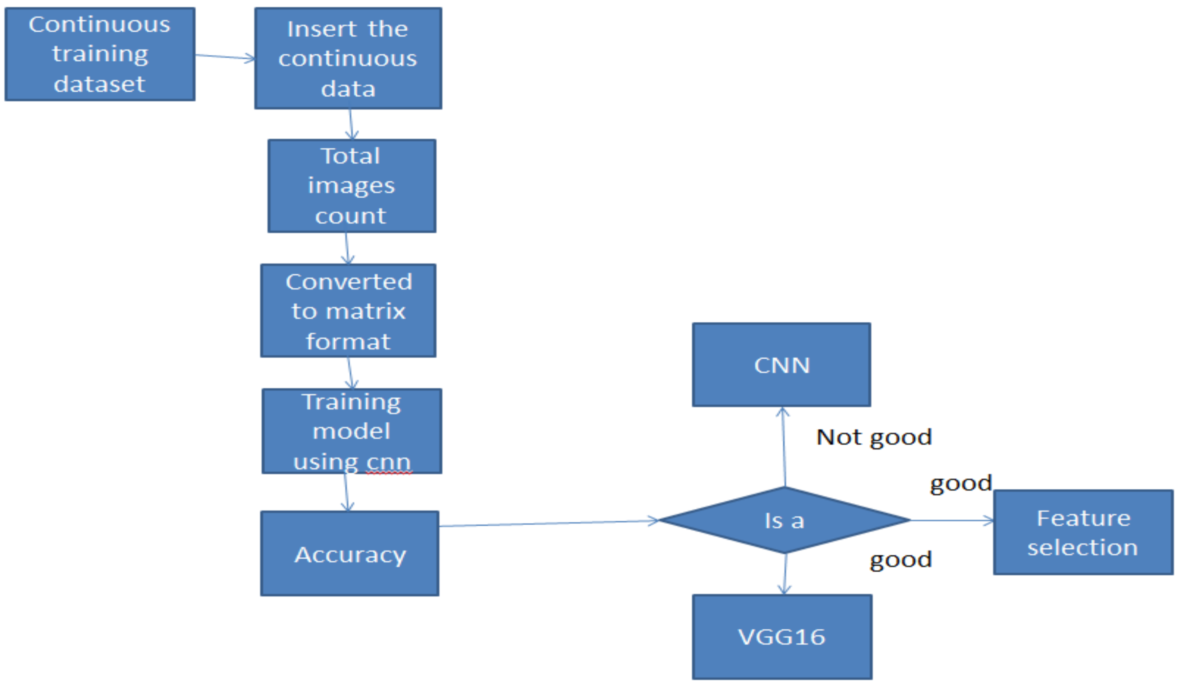


Fig 3.1 flow-diagram of system design

The above flowchart describes the working flow of the project. First matrix creates, the data split and selected the model check accuracy is good or not. If accuracy is not good then model skip and apply another model whichever accuracy is good when the accuracy is good then apply feature selection method and check how to effect different images.

Chapter 4

4.1 Model Building

4.1.1 Algorithm Research and Selection:

For this classification problem, following deep learning model was used:

Convolution Neural Network

It is well known for its widely used in applications of image and video recognition and also in recommender systems and Natural Language Processing (NLP). However, convolutional is more efficient because it reduces the number of parameters whichmakes different from other deep learning models. In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. The input layer which is a grayscale image. The Output layer which is a binary or multi-class labels. Hidden layers consisting of convolution layers, ReLU (rectified linear unit) layers, the pooling layers, and a fully connected Neural Network. CNNs are regularized versions of multilayer perceptron.

Multilayer perceptron usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks makes them prone to overfitting data.

Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters.

Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field as shown in below image.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns to optimize the filters (or kernels) through automated learning, whereas in traditional algorithms these filters are hand-engineered. This independence from prior knowledge and human intervention in feature extraction is a major advantage.

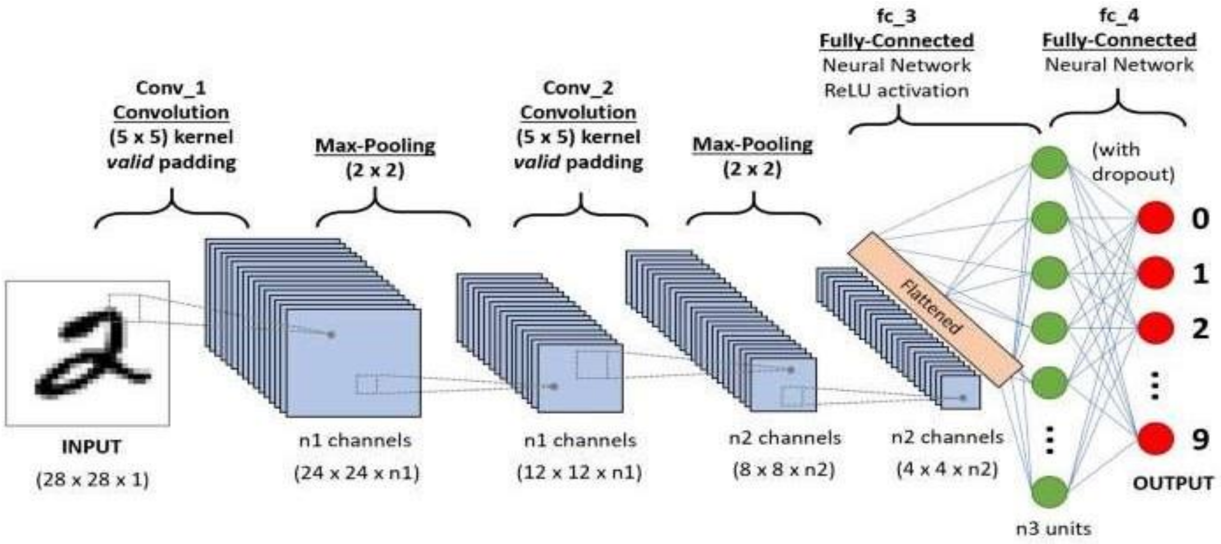


Fig 4.1: - CNN Architecture

4.1.2 Main Steps to build a CNN (or) Conv. net:

- Convolution Operation
- ReLU Layer (Rectified Linear Unit)
- Pooling Layer (Max Pooling)

- Flattening
- Fully Connected Layer

- **Convolution Operation**:

Convolution is the first layer to extract features from the input image and it learns therelationship between features using kernel or filters with input images.

- **ReLU Layer:**

ReLU stands for the Rectified Linear Unit for a non-linear operation. The output is *f(x) = max (0, x).* we use this because to introduce the non-linearity to CNN.

- **Pooling Layer:**

It is used to reduce the number of parameters by down sampling and retainonly the valuable information to process further. There are types of Pooling:

  - Max Pooling.
  - Average and Sum pooling.

- **Flattening:**

We flatten our entire matrix into a vector like a vertical one. So, that it will bepassed to the input layer.

- **Fully Connected Layer:**

We pass our flatten vector into input layer. We combined these features to create a model. Finally, we have an activation function such as SoftMax or sigmoidto classify the outputs.

Short information about Activation Functions:

These functions are needed to introduce a non-linearity into the network. Activation functionis applied and that output is passed to the next layer.

➢ **Sigmoid:**

The Sigmoid function used for **binary classification** in logistic regressionmodel. While creating artificial neurons sigmoid function used as the **activation function**. In statistics, the **sigmoid function graphs** are common as a cumulative distribution function.

➢ **Hyperbolic Tangent:**

In neural networks, as an alternative to sigmoid function, hyperbolic tangent function could be used as activation function. derivative of activation function would be involved in calculation for erroreffects on weights.

➢ **ReLU:**

ReLU is the max function(x,0) with input x e.g., matrix from a convolved image. ReLU then sets all negative values in the matrix x to zero and all other values are kept constant.

➢ **SoftMax:**

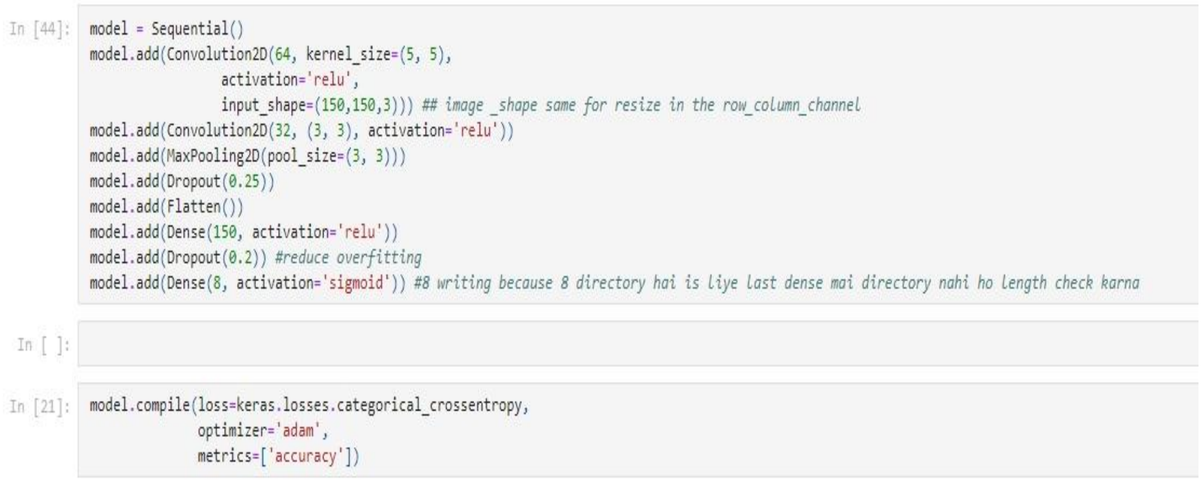
Used in multiple classification logistic regression model. In building neuralnetworks SoftMax functions used in different layer level.

4.2 Algorithm Implementation

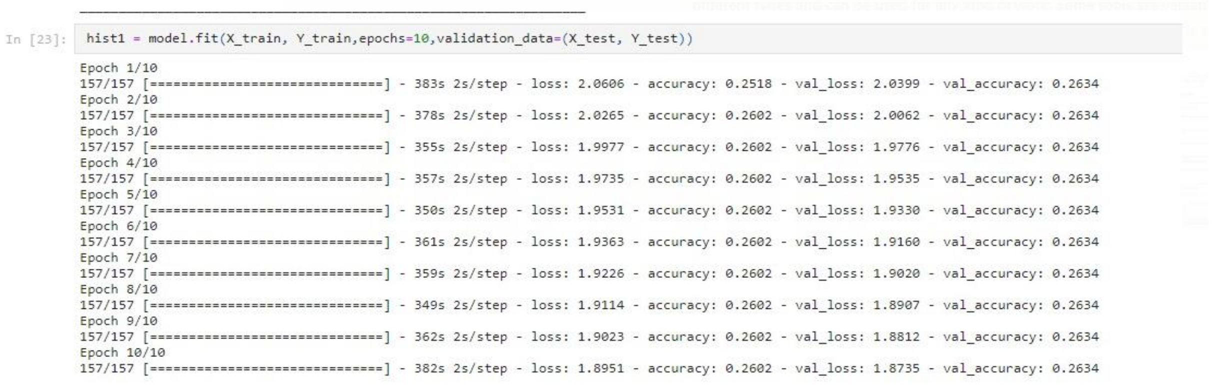
To implement the idea of disease detection and to train the machine accordinglyrequires lot of steps which are mentioned below: -

a. Label Data for input like Training Data, Training label data and in differentfolders.
b. Import all libraries like NumPy, Pandas, Matplotlib, Tensorflow, Convolution2D, MaxPooling2D, Flatten, Dense, Sequential Model, image, etc.
c. Add Convolutional Layer with 64 Filters each of filter size 5*5 and apply Reluactivation function.

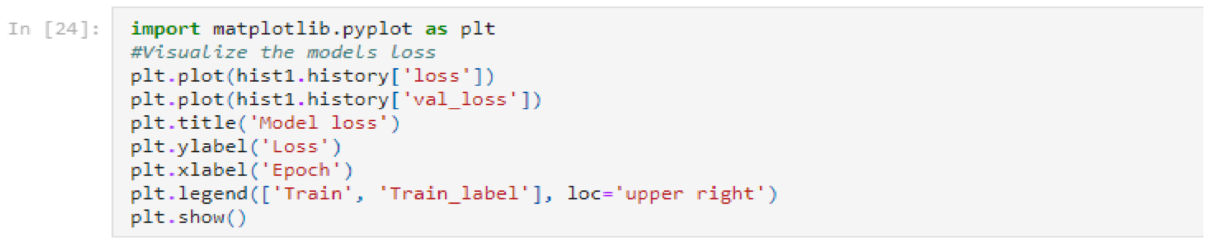
d. Add Maxpooling layer for extracting the features from convolutional layer.
e. Repeat Step3 and Step4.
f. Add Flatten Layer to convert 3D Array to 1D Array.
g. Add Hidden Layers or Dense Layers with 150 neurons and activation function is relu.
h. Add Hidden layer or Dropout layer with value 0.25
i. Add Hidden Layers or Dense Layers with 128 neurons and activation function is relu. 10.Add Output Layer with 8 neurons and Activation function SoftMax.
j. Apply Compile function to compile all the layers with loss function categorical_crossentropy and optimizer='adam'.
k. Model Fit Function is used to fit all variables like training set, X-train, y-trainepochs=5, validation data= x-train, y-train set etc.
l. Start Training, minimum time taken to train data in this hyper tunning dataset will be 8 hours.


```
In [44]: model = Sequential()
         model.add(Convolution2D(64, kernel_size=(5, 5),
                         activation='relu',
                         input_shape=(150,150,3))) ## image _shape same for resize in the row_column_channel
         model.add(Convolution2D(32, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(3, 3)))
         model.add(Dropout(0.25))
         model.add(Flatten())
         model.add(Dense(150, activation='relu'))
         model.add(Dropout(0.2)) #reduce overfitting
         model.add(Dense(8, activation='sigmoid')) #8 writing because 8 directory hai is liye last dense mai directory nahi ho length check karna

In [ ]:

In [21]: model.compile(loss=keras.losses.categorical_crossentropy,
                       optimizer='adam',
                       metrics=['accuracy'])
```

Accuracy is very low CNN Model is bad accuracy is 25.64% and epochs for 10 iteration is used so CNN model is underfitting.



```
In [23]: hist1 = model.fit(X_train, Y_train,epochs=10,validation_data=(X_test, Y_test))

Epoch 1/10
157/157 [==============================] - 383s 2s/step - loss: 2.0606 - accuracy: 0.2518 - val_loss: 2.0399 - val_accuracy: 0.2634
Epoch 2/10
157/157 [==============================] - 378s 2s/step - loss: 2.0265 - accuracy: 0.2602 - val_loss: 2.0062 - val_accuracy: 0.2634
Epoch 3/10
157/157 [==============================] - 355s 2s/step - loss: 1.9977 - accuracy: 0.2602 - val_loss: 1.9776 - val_accuracy: 0.2634
Epoch 4/10
157/157 [==============================] - 357s 2s/step - loss: 1.9735 - accuracy: 0.2602 - val_loss: 1.9535 - val_accuracy: 0.2634
Epoch 5/10
157/157 [==============================] - 350s 2s/step - loss: 1.9531 - accuracy: 0.2602 - val_loss: 1.9330 - val_accuracy: 0.2634
Epoch 6/10
157/157 [==============================] - 361s 2s/step - loss: 1.9363 - accuracy: 0.2602 - val_loss: 1.9160 - val_accuracy: 0.2634
Epoch 7/10
157/157 [==============================] - 359s 2s/step - loss: 1.9226 - accuracy: 0.2602 - val_loss: 1.9020 - val_accuracy: 0.2634
Epoch 8/10
157/157 [==============================] - 349s 2s/step - loss: 1.9114 - accuracy: 0.2602 - val_loss: 1.8907 - val_accuracy: 0.2634
Epoch 9/10
157/157 [==============================] - 362s 2s/step - loss: 1.9023 - accuracy: 0.2602 - val_loss: 1.8812 - val_accuracy: 0.2634
Epoch 10/10
157/157 [==============================] - 382s 2s/step - loss: 1.8951 - accuracy: 0.2602 - val_loss: 1.8735 - val_accuracy: 0.2634
```



```
In [24]: import matplotlib.pyplot as plt
         #Visualize the models loss
         plt.plot(hist1.history['loss'])
         plt.plot(hist1.history['val_loss'])
         plt.title('Model loss')
         plt.ylabel('Loss')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Train_label'], loc='upper right')
         plt.show()
```

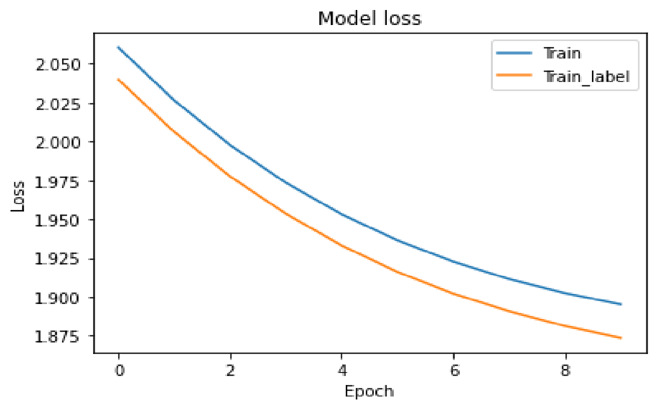
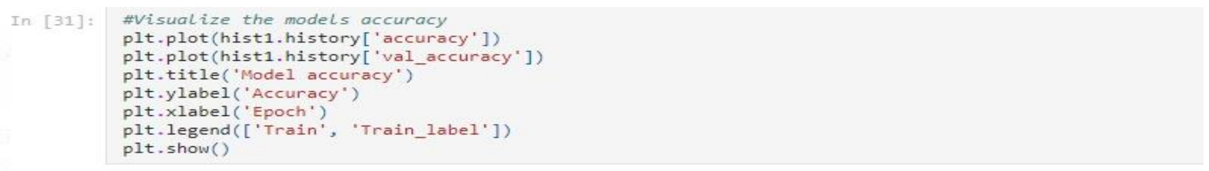


Fig 4.2 CNN Model loss



```
In [31]: #Visualize the models accuracy
         plt.plot(hist1.history['accuracy'])
         plt.plot(hist1.history['val_accuracy'])
         plt.title('Model accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Train_label'])
         plt.show()
```

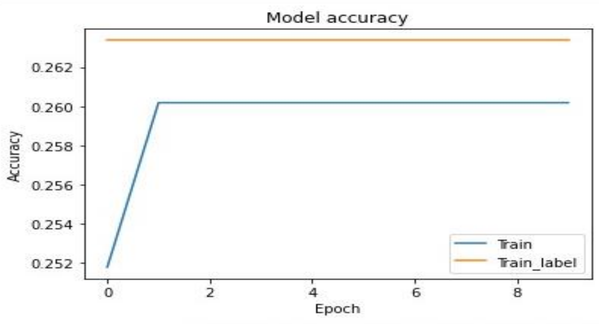


Fig 4.3 CNN Model Accuracy

4.3Model Flowchart

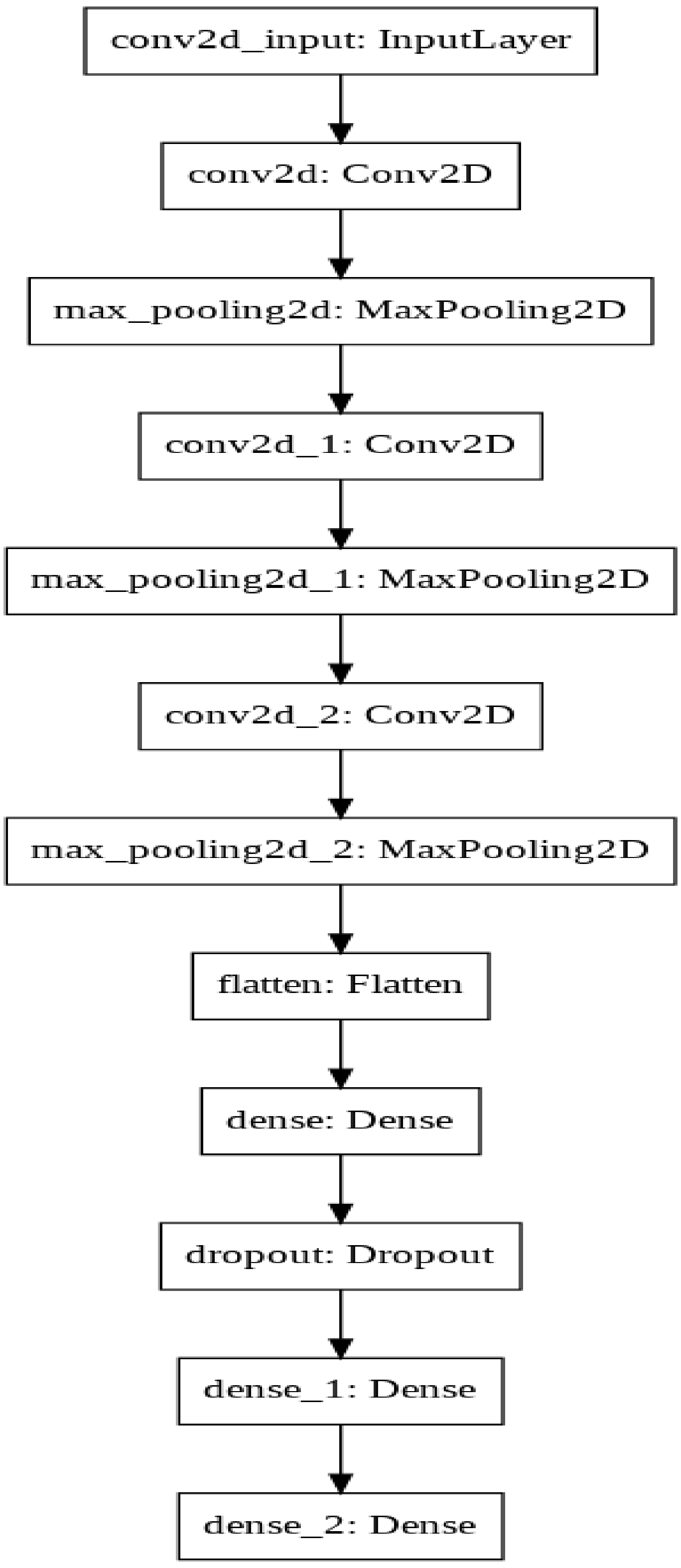


Fig 4.4: - Model Flowchart

Chapter 5

5.1 Transfer learning

Transfer learning generally refers to a process where a model trained on one problem is used in some way on a second related problem. In deep learning, transfer learning is a technique whereby a neural network model is first trained on a problem similar to the problem that is being solved. One or more layers from the trained model are then used in a new model trained on the problem of interest. Transfer learning has the benefit of decreasing the training time for a neural network model and can result in lower generalization error. The weights in re-used layers may be used as the starting point for the training process and adapted in response to the new problem.

This usage treats transfer learning as a type of weight initialization scheme. For these types of problems, it is common to use a deep learning model pre-trained for a large and challenging image classification task such as the ImageNet 1000-class photograph classification competition.

The research organizations that develop models for this competition and do well often release their final model under a permissive license for reuse. These models can take days or weeks to train on modern hardware. These models can be downloaded and incorporated directly into new models that expect image data as input.

Weight Initialization: The pre-trained model, or some portion of the model, is integrated into a new model, and the layers of the pre-trained model are trained in concert with the new model.

Transfer learning keras application are following types: -

- VGG (e.g., VGG16 or VGG19).
- Google Net (e.g., InceptionV3).
- Residual Network (e.g., ResNet50).

5.2 VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition".

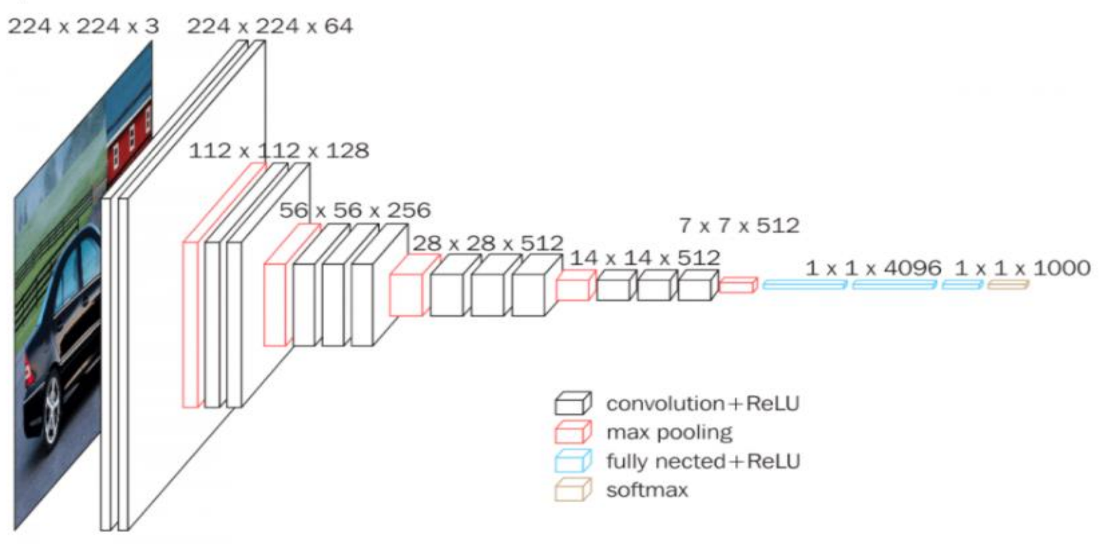
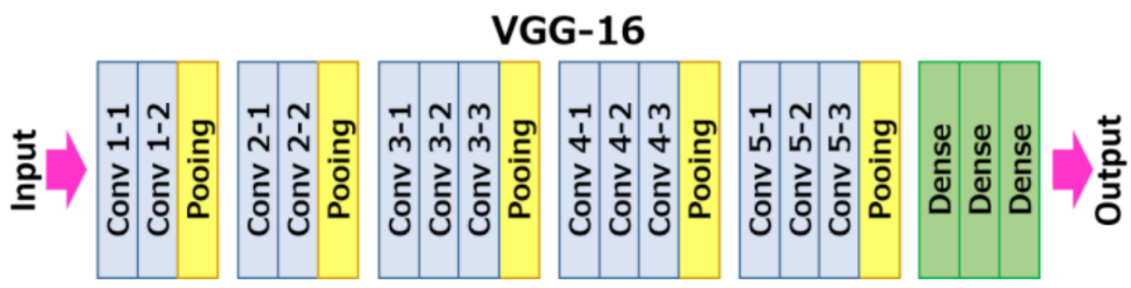


Fig 5.1: - Vgg16 Architecture

VGG16 is a convolution neural net (CNN) architecture which was used to win ILSVR(ImageNet) competition in 2014. It is considered to be one of the excellent vision model architecture till date. Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with a stride 1 and always used same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end it has 2 FC (fully connected layers) followed by a SoftMax for output. The 16 in VGG16 refers to it has 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx.) parameters. Vgg16 structure perform as shown in blow diagram.



VGG16

Fig 5.2: - VGG16 diagram

Include top = False means first and last layer is not including

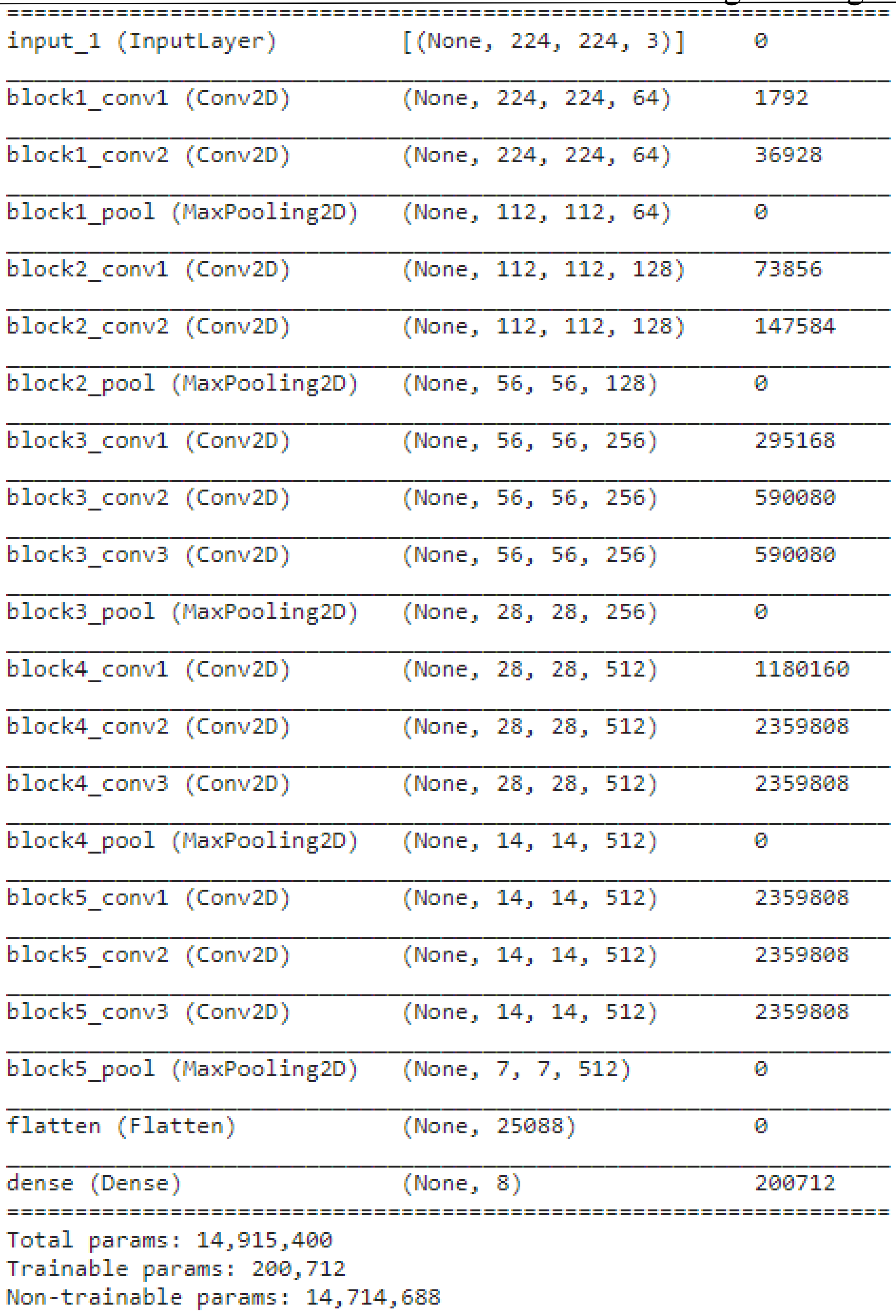
```
In [3]: vgg = VGG16(input_shape=image_resize + [3], weights='imagenet', include_top=False)
```

So, add flatten layer added vgg output and prediction is combine dense layer inside directory length and flatten layer combine and then apply model are combine input for vgg input and output for prediction.

```
In [6]: s = Flatten()(vgg.output)
```

```
In [7]: prediction = Dense(len(gloab_value_change), activation='softmax')(s)

        # create a model object
        model = Model(inputs=vgg.input, outputs=prediction)
```

Model.summary()

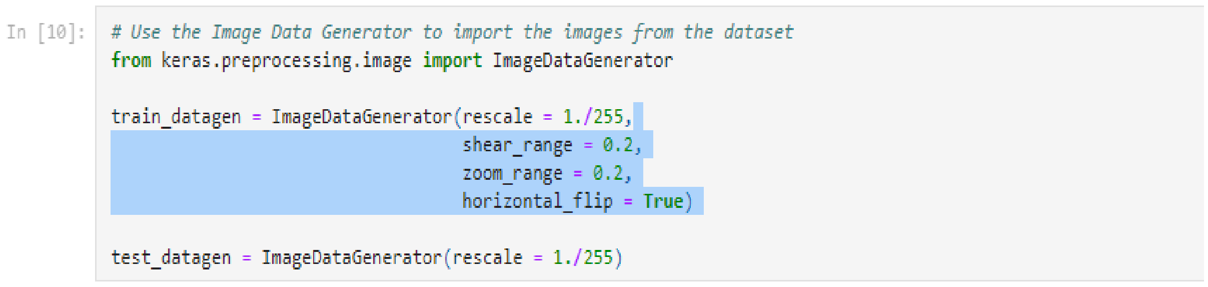


```
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 8)                 200712
=================================================================
Total params: 14,915,400
Trainable params: 200,712
Non-trainable params: 14,714,688
```

5.3 Data Pre-processing

- The dataset is divided into 80% for training and 20% for validation.
- First, augmentation settings are applied to the training data.
- set height and width of input image.
- The settings applied include flipping (horizontal), shearing of range (0.2) and zoom (0.2). Rescaling image values between (0 –1) called normalization. All these parameters are stored in the variable

"train_datagen" and "test_datagen".



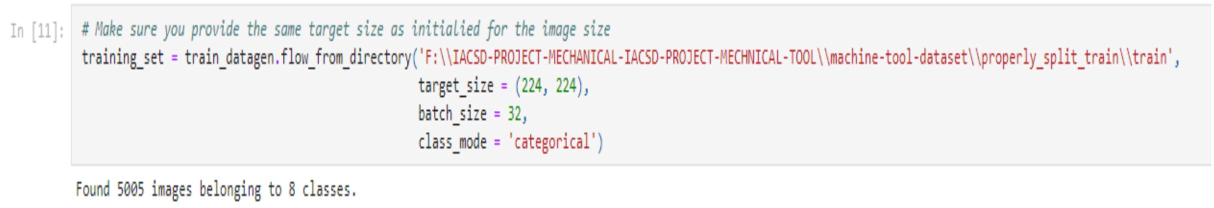
```
In [10]: # Use the Image Data Generator to import the images from the dataset
         from keras.preprocessing.image import ImageDataGenerator

         train_datagen = ImageDataGenerator(rescale = 1./255,
                                            shear_range = 0.2,
                                            zoom_range = 0.2,
                                            horizontal_flip = True)

         test_datagen = ImageDataGenerator(rescale = 1./255)
```

5005 images training image 80% and categorical means 8 directory is present so use categorial data, 1256 images test image 20% and categorical means 8 directory is present so use class mode as categorical, flow_from_directory means all directory images are covered, the **batch size** is a number of samples processed before the model is updated. The number of epochs is the number of complete passes through the training dataset.

The **size** of a **batch** must be more than or equal to one and less than or equal to the number of samples in the training dataset, target size means resize the row and directory as same as input size, batch size is given 32 output layers. ImageDataGenerator from keras.preprocessing. The objective of ImageDataGenerator is to import data with labels easily into the model.

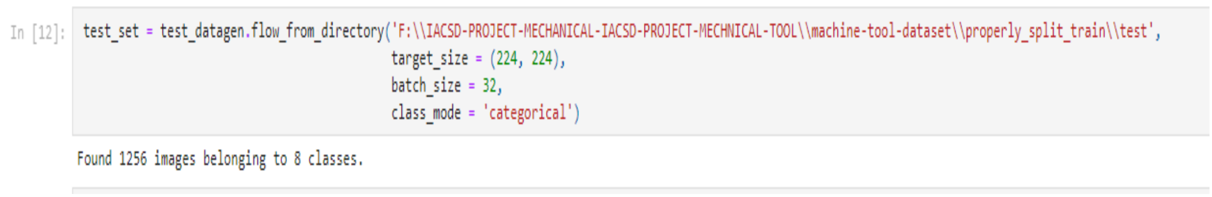
It is a very useful class as it has many functions to rescale, rotate, zoom, flip etc. The most useful thing about this class is that it doesn't affect the data stored on the disk. This class alters the data on the go while passing it to the model.

ImageDataGenerator for both training and testing data and passing the folder which has train data to the object train_datagen and similarly passing the folder which has test data to the object test_datagen. The folder structure of the data will be as follows -



```
In [11]: # Make sure you provide the same target size as initialied for the image size
         training_set = train_datagen.flow_from_directory('F:\\IACSD-PROJECT-MECHANICAL-IACSD-PROJECT-MECHNICAL-TOOL\\machine-tool-dataset\\properly_split_train\\train',
                                                          target_size = (224, 224),
                                                          batch_size = 32,
                                                          class_mode = 'categorical')

         Found 5005 images belonging to 8 classes.
```

Training set

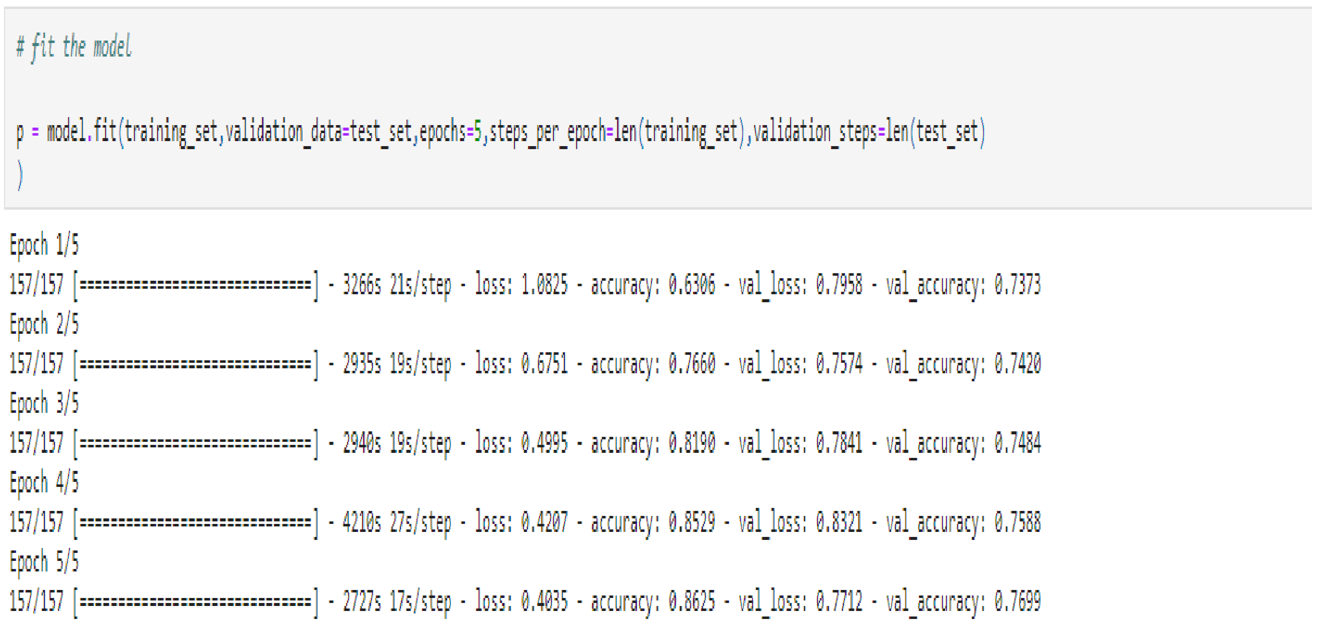


```
In [12]: test_set = test_datagen.flow_from_directory('F:\\IACSD-PROJECT-MECHANICAL-IACSD-PROJECT-MECHNICAL-TOOL\\machine-tool-dataset\\properly_split_train\\test',
                                                     target_size = (224, 224),
                                                     batch_size = 32,
                                                     class_mode = 'categorical')

         Found 1256 images belonging to 8 classes.
```

Test set

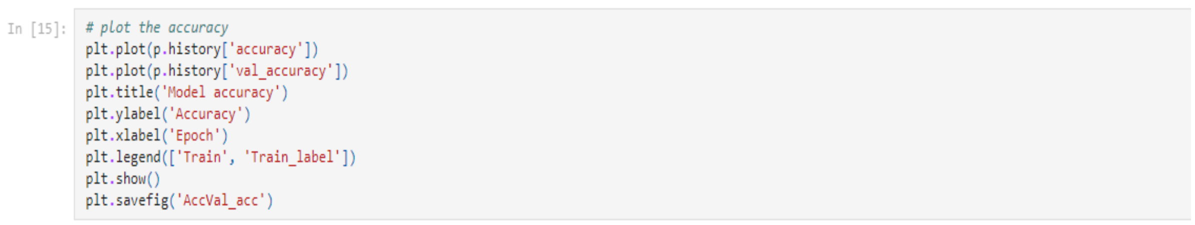
5.4 Accuracy

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, as above, an epoch that has one batch is called the batch gradient descent learning algorithm. Epochs are line how many iterations you want to give to you optimize something is called as callback.



```
# fit the model

p = model.fit(training_set,validation_data=test_set,epochs=5,steps_per_epoch=len(training_set),validation_steps=len(test_set)
)
```

```
Epoch 1/5
157/157 [==============================] - 3266s 21s/step - loss: 1.0825 - accuracy: 0.6306 - val_loss: 0.7958 - val_accuracy: 0.7373
Epoch 2/5
157/157 [==============================] - 2935s 19s/step - loss: 0.6751 - accuracy: 0.7660 - val_loss: 0.7574 - val_accuracy: 0.7420
Epoch 3/5
157/157 [==============================] - 2940s 19s/step - loss: 0.4995 - accuracy: 0.8190 - val_loss: 0.7841 - val_accuracy: 0.7484
Epoch 4/5
157/157 [==============================] - 4210s 27s/step - loss: 0.4207 - accuracy: 0.8529 - val_loss: 0.8321 - val_accuracy: 0.7588
Epoch 5/5
157/157 [==============================] - 2727s 17s/step - loss: 0.4035 - accuracy: 0.8625 - val_loss: 0.7712 - val_accuracy: 0.7699
```



```
In [15]: # plot the accuracy
         plt.plot(p.history['accuracy'])
         plt.plot(p.history['val_accuracy'])
         plt.title('Model accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epoch')
         plt.legend(['Train', 'Train_label'])
         plt.show()
         plt.savefig('AccVal_acc')
```

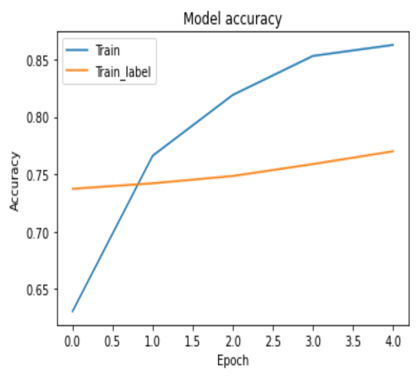


Fig 5.3: - Accuracy model

Accuracy is continuously increase so VGG16 model is good accuracy so this model is not overfit and underfit model and accuracy is training data is 86% and testing data is 76%.

Chapter 6

6.1 Feature Selection

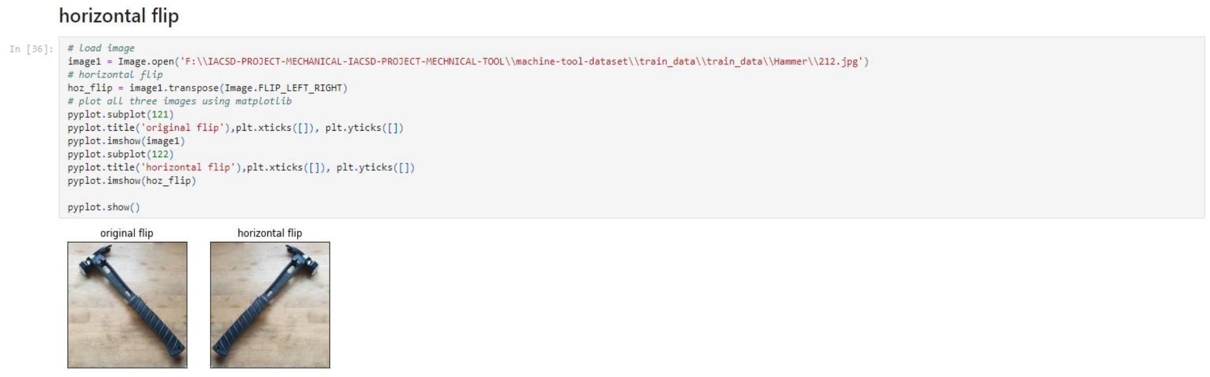
Feature selection, also known as **variable selection**, **attribute selection** or **variable subset selection**, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction.

The central premise when using a feature selection technique is that the data contains some features that are either redundant or irrelevant, and can thus be removed without incurring much loss of information. Redundant and irrelevant are two distinct notions, since one relevant feature may be redundant in the presence of another relevant feature with which it is strongly correlated.

Feature selection techniques should be distinguished from feature extraction. Feature selection method use different technics uses as following.

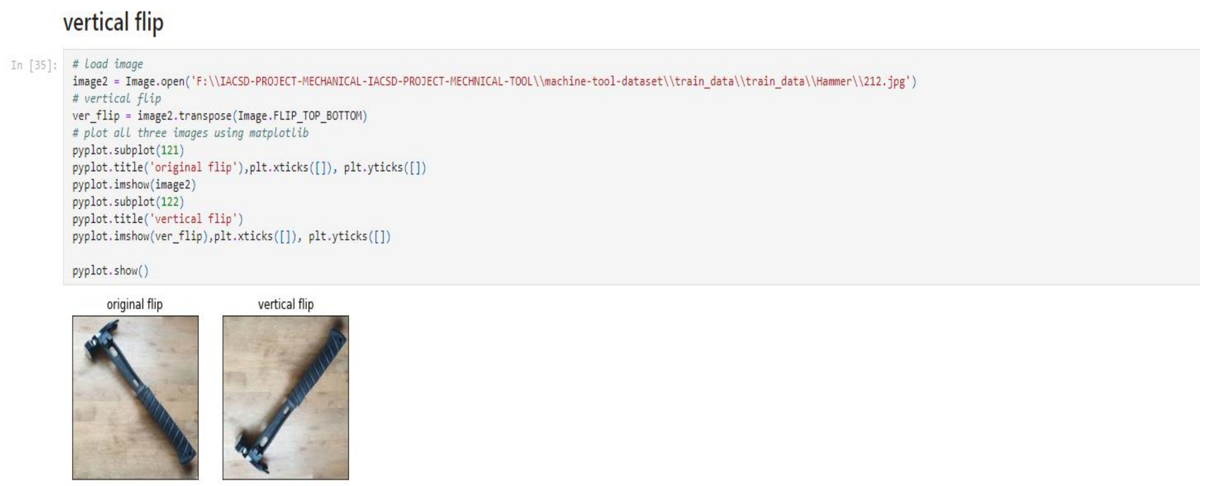
6.1.1 Horizontal and Vertical flip

For flipping the image horizontally (means left to right image flip). Flips it horizontally, and displays the original and flipped image using standard JPG display. For flipping the image vertical (means right to left image flip). Flip it vertically, and displays the original and flipped image using standard JPG display.


horizontal flip

```
In [36]: # Load image
image1 = Image.open('F:\\IACSD-PROJECT-MECHANICAL-IACSD-PROJECT-MECHNICAL-TOOL\\machine-tool-dataset\\train_data\\train_data\\Hammer\\212.jpg')
# horizontal flip
hoz_flip = image1.transpose(Image.FLIP_LEFT_RIGHT)
# plot all three images using matplotlib
pyplot.subplot(121)
pyplot.title('original flip'),plt.xticks([]), plt.yticks([])
pyplot.imshow(image1)
pyplot.subplot(122)
pyplot.title('horizontal flip'),plt.xticks([]), plt.yticks([])
pyplot.imshow(hoz_flip)

pyplot.show()
```

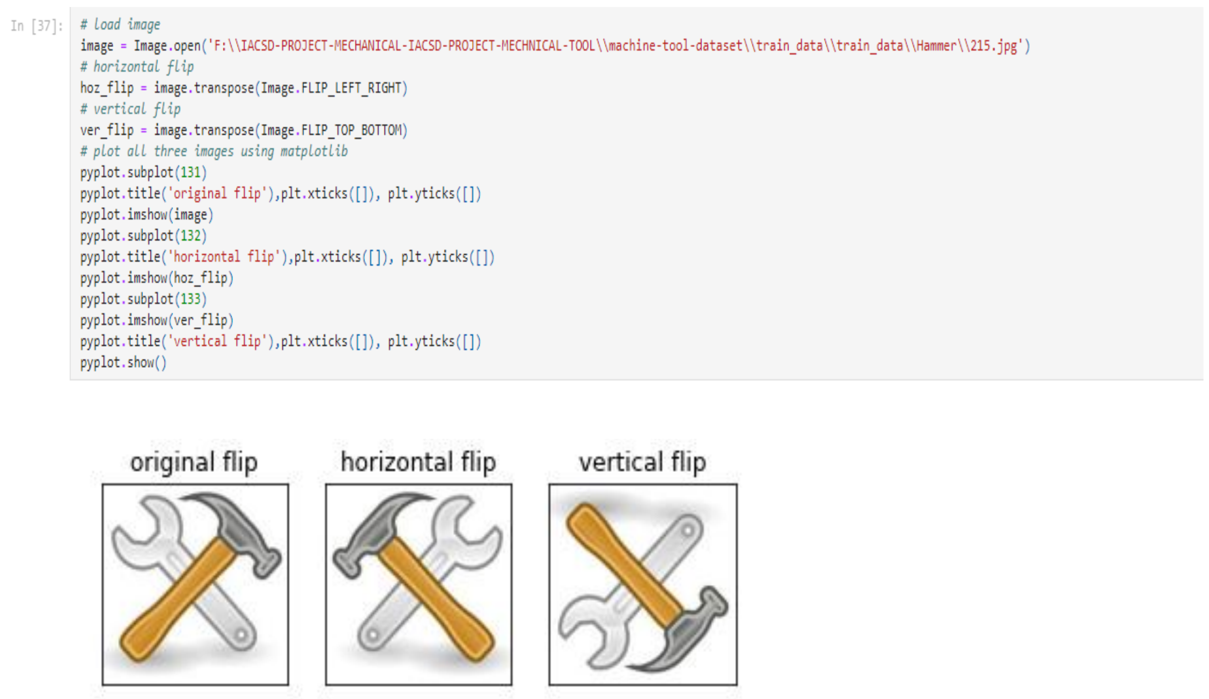
Fig6.1.1: - Horizontal Flip



vertical flip

```
In [35]: # load image
         image2 = Image.open('F:\\IACSD-PROJECT-MECHANICAL-IACSD-PROJECT-MECHNICAL-TOOL\\machine-tool-dataset\\train_data\\train_data\\Hammer\\212.jpg')
         # vertical flip
         ver_flip = image2.transpose(Image.FLIP_TOP_BOTTOM)
         # plot all three images using matplotlib
         pyplot.subplot(121)
         pyplot.title('original flip'),plt.xticks([]), plt.yticks([])
         pyplot.imshow(image2)
         pyplot.subplot(122)
         pyplot.title('vertical flip')
         pyplot.imshow(ver_flip),plt.xticks([]), plt.yticks([])

         pyplot.show()
```

Fig 6.1.2: - Vertical Flip

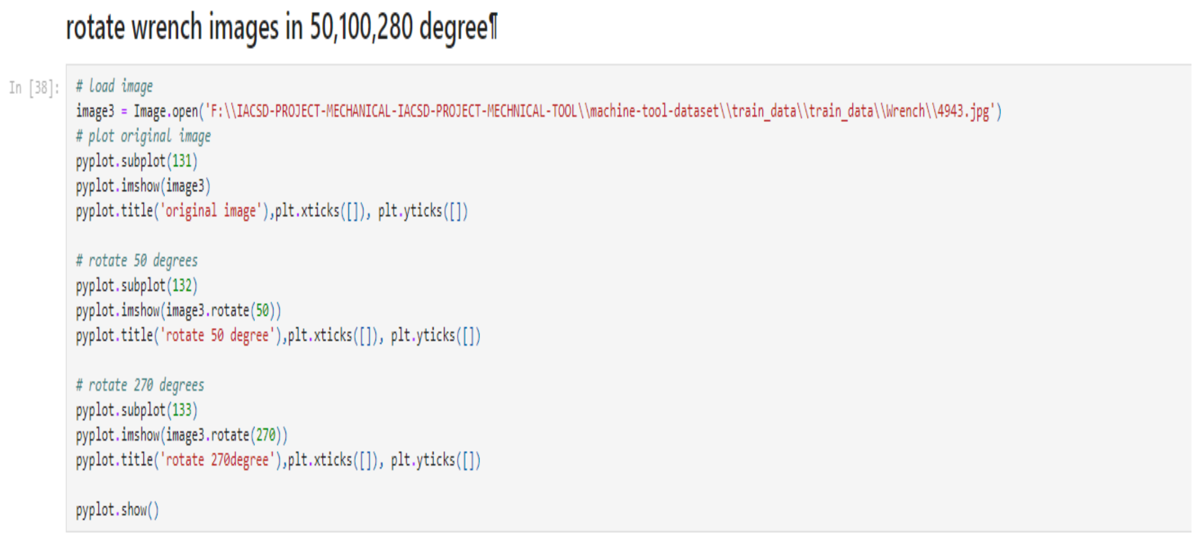


```
In [37]: # load image
         image = Image.open('F:\\IACSD-PROJECT-MECHANICAL-IACSD-PROJECT-MECHNICAL-TOOL\\machine-tool-dataset\\train_data\\train_data\\Hammer\\215.jpg')
         # horizontal flip
         hoz_flip = image.transpose(Image.FLIP_LEFT_RIGHT)
         # vertical flip
         ver_flip = image.transpose(Image.FLIP_TOP_BOTTOM)
         # plot all three images using matplotlib
         pyplot.subplot(131)
         pyplot.title('original flip'),plt.xticks([]), plt.yticks([])
         pyplot.imshow(image)
         pyplot.subplot(132)
         pyplot.title('horizontal flip'),plt.xticks([]), plt.yticks([])
         pyplot.imshow(hoz_flip)
         pyplot.subplot(133)
         pyplot.imshow(ver_flip)
         pyplot.title('vertical flip'),plt.xticks([]), plt.yticks([])
         pyplot.show()
```

Fig 6.1.3: - Horizontal, vertical flip

6.1.2 Rotate

For rotating the image by specifying degree. Rotates to a specified degree, and displays the original and rotated image using standard JPG display.



rotate wrench images in 50,100,280 degree¶

```
In [38]: # load image
         image3 = Image.open('F:\\IACSD-PROJECT-MECHANICAL-IACSD-PROJECT-MECHNICAL-TOOL\\machine-tool-dataset\\train_data\\train_data\\Wrench\\4943.jpg')
         # plot original image
         pyplot.subplot(131)
         pyplot.imshow(image3)
         pyplot.title('original image'),plt.xticks([]), plt.yticks([])

         # rotate 50 degrees
         pyplot.subplot(132)
         pyplot.imshow(image3.rotate(50))
         pyplot.title('rotate 50 degree'),plt.xticks([]), plt.yticks([])

         # rotate 270 degrees
         pyplot.subplot(133)
         pyplot.imshow(image3.rotate(270))
         pyplot.title('rotate 270degree'),plt.xticks([]), plt.yticks([])

         pyplot.show()
```

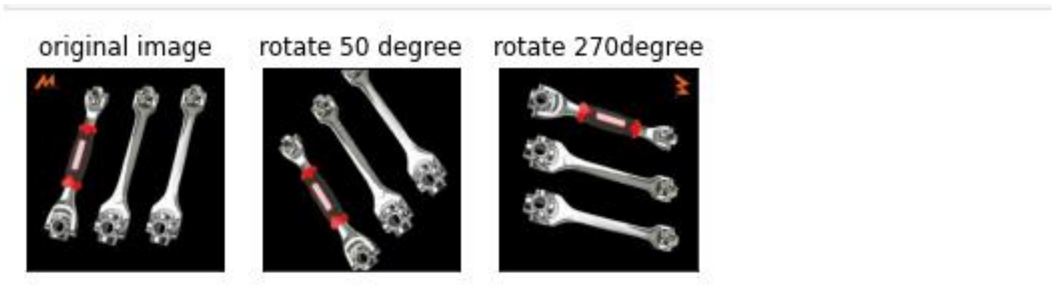
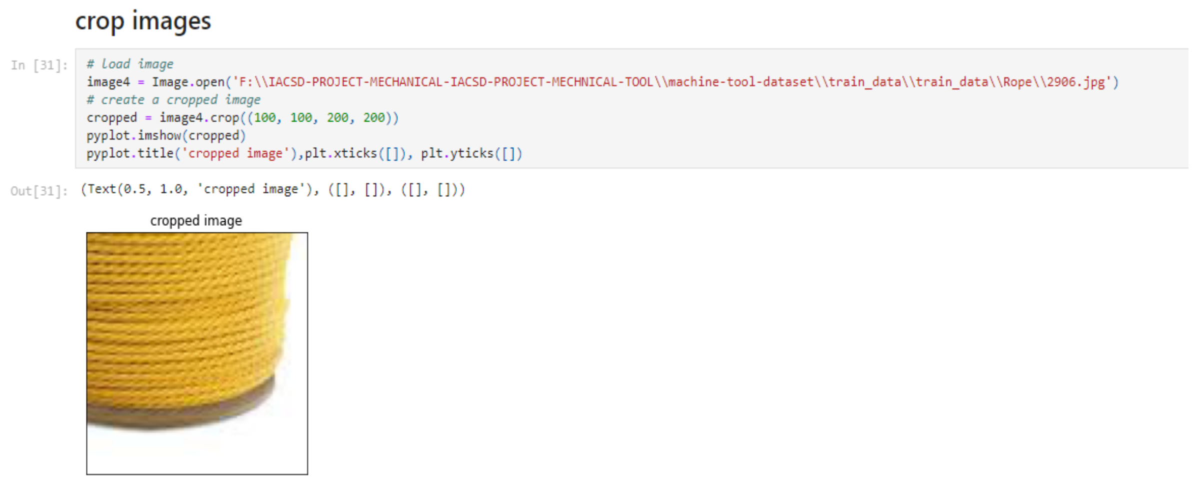


Fig 6.1.4: - Rotate image

6.1.3 Crop

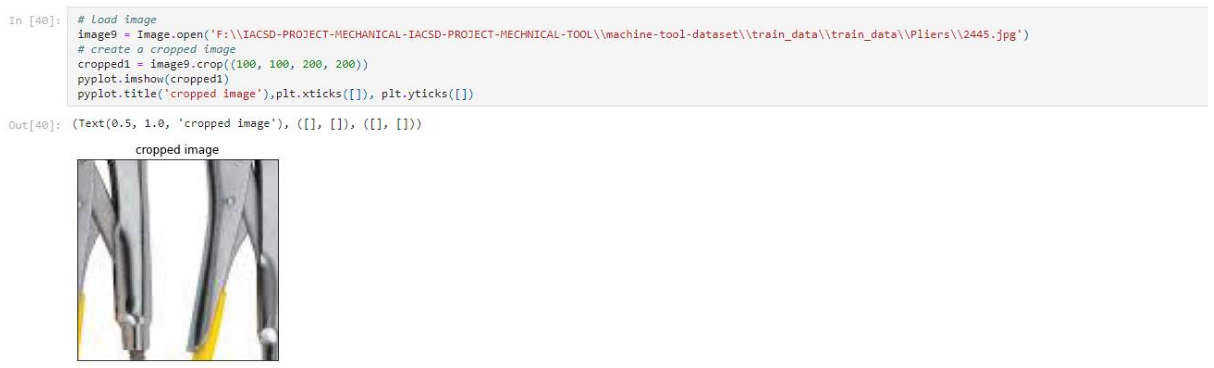
PIL is the Python Imaging Library which provides the python interpreter with image editing capabilities. PIL.Image.crop() method is used to crop a rectangular portion of any image.
Syntax: PIL.Image.crop(box = None)
Parameters:
box – a 4-tuple defining the left, upper, right, and lower pixel coordin ate.
Return type: Image (Returns a rectangular region as (left, upper, rig ht, lower)-tuple).
Return: An **Image** object.



crop images

```
In [31]: # Load image
         image4 = Image.open('F:\\IACSD-PROJECT-MECHANICAL-IACSD-PROJECT-MECHNICAL-TOOL\\machine-tool-dataset\\train_data\\train_data\\Rope\\2906.jpg')
         # create a cropped image
         cropped = image4.crop((100, 100, 200, 200))
         pyplot.imshow(cropped)
         pyplot.title('cropped image'),plt.xticks([]), plt.yticks([])

Out[31]: (Text(0.5, 1.0, 'cropped image'), ([], []), ([], []))
```

Fig 6.1.5: - crop image Rope



```
In [40]: # Load image
         image9 = Image.open('F:\\IACSD-PROJECT-MECHANICAL-IACSD-PROJECT-MECHNICAL-TOOL\\machine-tool-dataset\\train_data\\train_data\\Pliers\\2445.jpg')
         # create a cropped image
         cropped1 = image9.crop((100, 100, 200, 200))
         pyplot.imshow(cropped1)
         pyplot.title('cropped image'),plt.xticks([]), plt.yticks([])

Out[40]: (Text(0.5, 1.0, 'cropped image'), ([], []), ([], []))
```

Fig 6.1.6: - crop image pliers

6.1.4 Thresholding

In digital image processing, thresholding is the simplest method of segmenting images. From a grayscale image, thresholding can be used to create binary images. The simplest thresholding methods replace each pixel in an image with a black pixel if the image intensity $I_{(i,j)}$ is less than some fixed constant T (that is, $I_{(i,j)} < T$), or a white pixel if the image intensity is greater than that constant.

In the example image on the right, this results in the dark tree becoming completely black, and the white snow becoming completely white. The thresholding process is sometimes described as separating an **image** into foreground values (black) and background values (white).

thresholding

```
In [41]: image5 = pyplot.imread('F:\\IACSD-PROJECT-MECHANICAL-IACSD-PROJECT-MECHNICAL-TOOL\\machine-tool-dataset\\train_data\\train_data\\Gasoline Can\\22.jpg')
         print(image5.shape)
         pyplot.imshow(image5)
         pyplot.title('original image'),plt.xticks([]), plt.yticks([])
         (194, 259, 3)
Out[41]: (Text(0.5, 1.0, 'original image'), ([], []), ([], []))
```

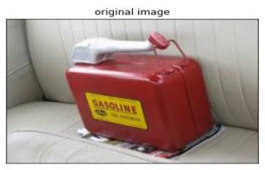


Fig 6.1.7: - Original image

```
In [17]: def thres_seg(image5):
             gray = rgb2gray(image5)
             print(gray.shape)
             pyplot.figure() # To create new frame for the image to be displayed
             pyplot.imshow(gray, cmap='gray')
             gray_r = gray.reshape(gray.shape[0]*gray.shape[1])
             gray_r [ gray_r > gray_r.mean() ] = 1
             gray_r [ gray_r <= gray_r.mean() ] = 0
             gray_r = gray_r.reshape(gray.shape[0],gray.shape[1])
             pyplot.figure()
             pyplot.imshow(gray_r, cmap='gray')

In [18]: thres_seg(image5)

         (194, 259)
```

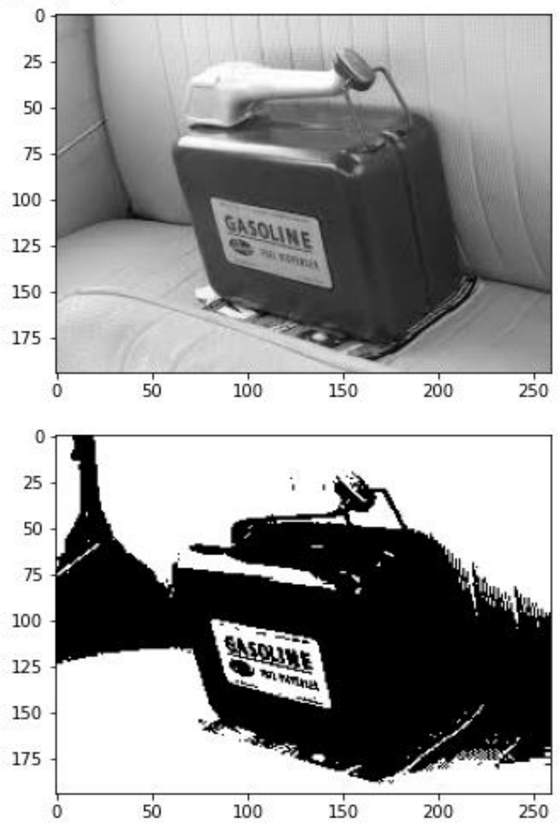


Fig 6.1.8: - Thresholding image

6.1.5 Gaussian-Blur

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (e.g., noise, edges) from the image resulting in edges being blurred when this is filter is applied. Image Blurring (Image Smoothing). OpenCV provides mainly four types of blurring techniques. Averaging, Gaussian Filtering, Median Filtering, Bilateral Filtering instead of a box filter consisting of equal filter coefficients, a Gaussian kernel is used.

It is done with the function, **cv2.GaussianBlur()**. We should specify the width and height of the kernel which should be positive and odd. We also should specify the standard deviation in the X and Y directions, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as equal to sigmaX. If both are given as zeros, they are calculated from the kernel size. Gaussian filtering is highly effective in removing Gaussian noise from the image.

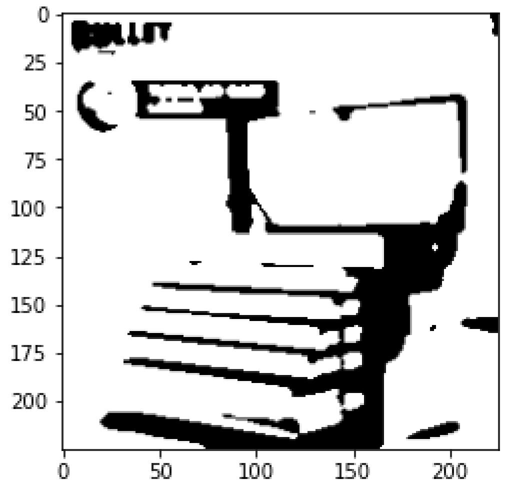


Fig 6.1.9: - Gaussian Blur

6.1.6 Canny Edge detection

Noise reduction using Gaussian filter. Gradient calculation along the horizontal and vertical axis, Non-Maximum suppression of false edges, Double thresholding for segregating strong and weak edges, Edge tracking by hysteresis
Noise reduction using gaussian filter: -
It uses a Gaussian filter for the removal of noise from the image, it is because this noise can be assumed as edges due to sudden intensity change by the edge detector. The sum of the elements in the Gaussian

kernel is 1, so, the kernel should be normalized before applying as convolution to the image.

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, we can find edge gradient and direction for each pixel as follows:

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$
$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, *minVal* and *maxVal*. Any edges with intensity gradient more than *maxVal* are sure to be edges and those below *minVal* are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to “sure-edge” pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the image below:

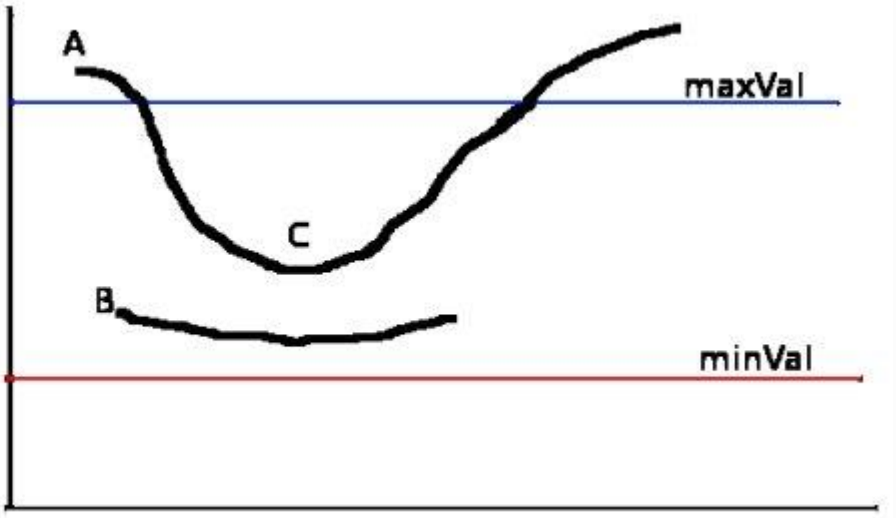
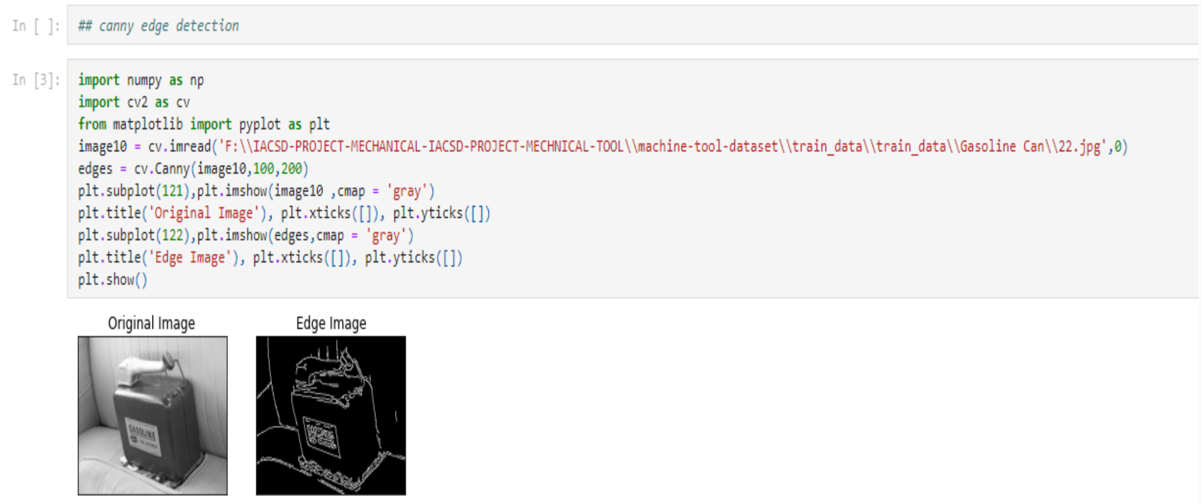


Fig 6.1.10: - Sure-edge

The edge A is above the maxVal, so considered as “sure-edge”. Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above minVal and is in same region as that of edge C, it is not connected to any “sure-edge”, so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result. OpenCV puts all the above in single function, **cv2.Canny()**. First argument is our input image. Second and third arguments are our minVal and maxVal respectively. Third argument is aperture_size. $Edge\_Gradient\ (G) = |G_x| + |G_y|$.



```
In [ ]: ## canny edge detection
```

```
In [3]: import numpy as np
        import cv2 as cv
        from matplotlib import pyplot as plt
        image10 = cv.imread('F:\\IACSD-PROJECT-MECHANICAL-IACSD-PROJECT-MECHNICAL-TOOL\\machine-tool-dataset\\train_data\\train_data\\Gasoline Can\\22.jpg',0)
        edges = cv.Canny(image10,100,200)
        plt.subplot(121),plt.imshow(image10 ,cmap = 'gray')
        plt.title('Original Image'), plt.xticks([]), plt.yticks([])
        plt.subplot(122),plt.imshow(edges,cmap = 'gray')
        plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
        plt.show()
```

Fig 6.1.11: - Canny edge detection

6.1.7 Salt-pepper noise

Noise: Noise means random disturbance in a signal in a computer version. In our case, the signal is an image. Random disturbance in the brightness and color of an image is called Image noise.

Salt-and-pepper: It is found only in greyscale images (black and white image). As the name suggests salt (white) in pepper (black)–white spots in the dark regions or pepper (black) in salt (white)–black spots in the white regions.

In other words, an image having salt-and-pepper noise will have a few dark pixels in bright regions and a few bright pixels in dark regions. Salt-and-pepper noise is also called impulse noise. It can be caused by several reasons like dead pixels, analog-to-digital conversion error, bit transmission error, etc.

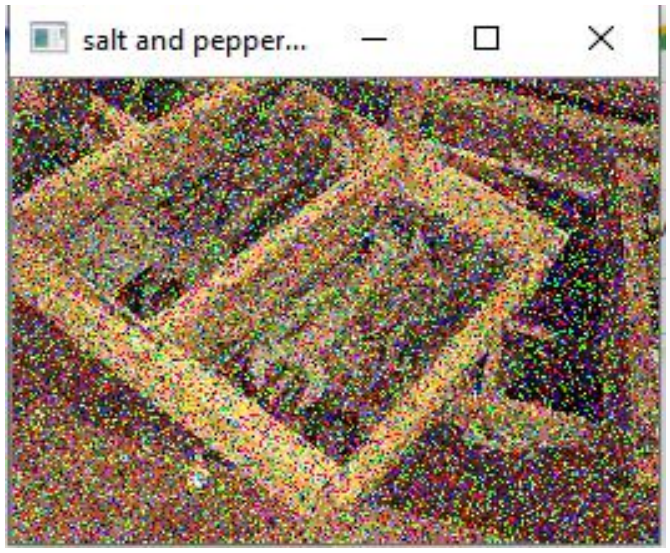


Fig 6.1.12: - Salt-pepper noise

6.1.8 Speckles noise

Speckle is a granular noise that inherently exists in an image and degrades its quality. Speckle noise can be generated by multiplying random pixel values with different pixels of an image.

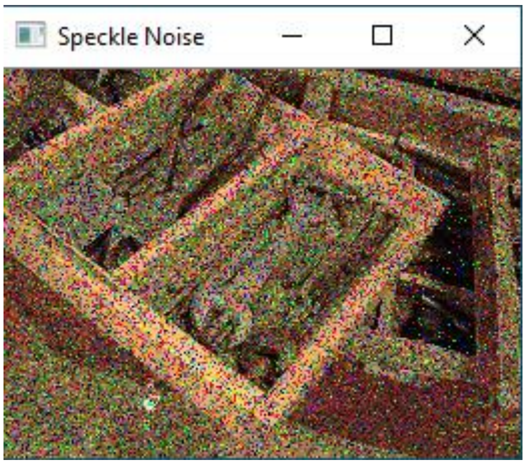


Fig 6.1.13: - Speckles noise

6.1.9 Gaussian noise

Gaussian Noise is a statistical noise having a probability density function equal to normal distribution, also known as Gaussian Distribution. Random Gaussian function is added to Image function to generate this noise. It is also called as electronic noise because it arises in amplifiers or detectors. The side image is a bell-shaped probability distribution function which have mean 0 and standard deviation(sigma) 1.

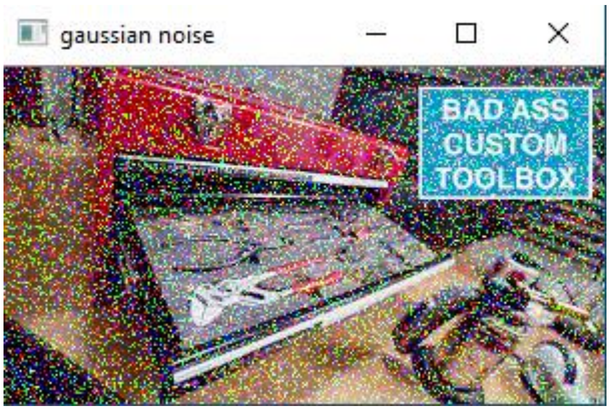


Fig 6.1.14: - Gaussian noise

Chapter 7

7.1 Future Scope

The purposed system can be used for the real time mechanical tool image and video frame. Images also apply different methods, confusion matrix, AUC and ROC curve. Flask will also create.

Chapter 8

CONCLUSION:

This system has utilized deep neural network learning capabilities to mechanical tools images system. This system is based on a simple classification mechanism which exploits the feature extraction functionalities of CNN, VGG16 transfer learning model. For prediction, the transfer learning model utilizes the fully connected layers. The system has achieved an overall 86% testing accuracy on publicly accessible dataset.

It is concluded from accuracy that VGG16 is highly suitable for accuracy. This system can be feature selection uses different images for blurring, noise, grayscale, thresholding, rotate and crop are applying different type image and show effect of images.

Chapter 9

References:

- Machine Learning with Python Cookbook: Practical Solutions from Pre-processing to Deep Learning.
- Hands-On Machine Learning with Scikit-Learn and Tensor-Flow: Concepts, Tools, and Techniques to Build Intelligent Systems.
- https://en.wikipedia.org/wiki/Convolutional_neural_network
- https://towardsdatascience.com/convolution-neural-network-for-image-processing-using-keras-dc3429056306
- Dataset link: -
  https://www.kaggle.com/salmaneunus/mechanical-tools-dataset?select=Mechanical+Tools+Image+dataset
- CNN:-
  https://towardsdatascience.com/convolution-neural-network-for-image-processing-using-keras-dc3429056306
  https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets/
- VGG16: -
  https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c