**Objective:**

Building an automated visionary bot with precise positioning facility for picking up and dropping objects in manufacturing industries or similar scenarios.

# **Introduction:**

The essence of neural network aids machine learning, in which a computer learns to perform some tasks by analyzing training examples.

The examples are typically pre-labeled by hand. The system would then look for visual patterns in the photos that were consistently associated with certain labels.

After passing an image through the neural network, a process called forward propagation occurs. The image is represented as a set of input values (pixels), and these values are transformed as they pass through the network layers.

Neural nets, which are highly interconnected networks of thousands or even millions of simple processing nodes, are loosely modeled after the structure of the human brain. Today's neural nets are primarily composed of layers of nodes, and they are "feed-forward," which means that information flows through them only in one direction. Multiple nodes in the layer below it may be connected to a single node, which receives data from them, and multiple nodes in the layer above it, which it sends data to.

Every incoming connection that a node receives will be assigned a number called a "weight." Over each of its connections, the node receives a unique data item—a different number—when the network is active, which it then multiplies by the corresponding weight. After that, it totals the output products to get a single number. The node does not send any data to the following layer if that number is less than a certain threshold. The node "fires" if the value surpasses the threshold, which in modern neural nets typically entails sending the value—the total of the weighted inputs—along all of its outgoing connections.

The output of the neural network (predictions) are compared with the actual labels of the images. This comparison is quantified using a loss function that measures the difference between the predicted and true values. Then the calculated loss is used to update the model's parameters (weights and biases) through a process called backpropagation. This involves calculating the gradient of the loss with respect to each parameter and adjusting the parameters in the direction that reduces the loss.

Next an optimization algorithm is used to iteratively adjust the model parameters to minimize the loss on the training data. The process of forward propagation, loss calculation, backpropagation, and optimization are repeated for multiple iterations called epochs. Each epoch involves going through the entire dataset.

Nowadays, the deep learning process mentioned above has become much simpler using Python or more specifically Tensorflow library’s Keras API. This contains many built in functions and other features which makes it easier to compile such Deep learning models for image classifiers such used in our project.

A 3D-printed robotic arm with three degrees of freedom (3DOF) is a versatile tool that can be used for a variety of tasks, such as pick-and-place, assembly, and inspection. The arm is typically controlled by an Arduino microcontroller, which sends signals to the servo motors that actuate the joints of the arm. The 3D-printed design of the arm makes it lightweight and easy to customize.

The arm typically consists of three main components:

* Base: The stationary part of the arm that is attached to the ground or another surface.
* Shoulder joint: The first rotational joint of the arm, and it allows the arm to move up and down.
* Elbow joint: The second rotational joint of the arm, and it allows the arm to bend and straighten.
* End effector: The part of the arm that interacts with the environment. It can be a gripper, a tool, or any other type of end effector that is needed for the task at hand.

3DOF robotic arms are a powerful tool that can be used for a variety of applications. They are a popular choice for hobbyists and educators, and they are also used in industry for tasks such as:

* Pick-and-place: Moving objects from one location to another.
* Assembly: Putting together components to create a product.
* Inspection: Checking for defects or flaws in products.

3D-printed robotic arms with three degrees of freedom (3DOF) are a versatile and powerful tool that can be used for a variety of applications. Their lightweight design, ease of customization, and affordability make them a valuable tool for anyone interested in robotics.

# **Description:**

Our project focuses on developing an automated system for object retrieval using an image classifier and a robotic arm. The goal is to recognize the position of an object among nine predefined locations and instruct a robotic arm to pick up the identified object.

# **Keywords:**

Image classifier, Deep learning, Tensorflow, OpenCV, 3 DOF ( Degree of Freedom) Robotic Arm, Serial Communication.

# **Operations:**

Initially, we will place our object in one of the 9 blocks in the plane board. The standby camera will be positioned such that the whole operation region will be within the field of view of the camera. When we run our python script, the camera will capture an image which will be loaded in our IDE. Then we will use our trained model to predict the result for the position of our object. Our image classifier will compare with the images of 9 different positions, then return the predicted position.

The returned position value will be passed to the arduino through serial communication, causing the 3 servo motors attached to the mechanical arm to rotate based on the predefined angles for each of the 9 position. After reaching the desired position, the arm will return back to its default state (0).

# **Problem Space Overview:**

Before jumping to the working process and algorithm we would like to give the idea about the problem space’s scenario.

### **Plane Board:**

Our operations will be done on a plane board whose dimensions are 9x9 inch. This board will be divided into 3x3 grid or total 9 squares. The positions are numbered serially from pos\_1 to pos\_9.

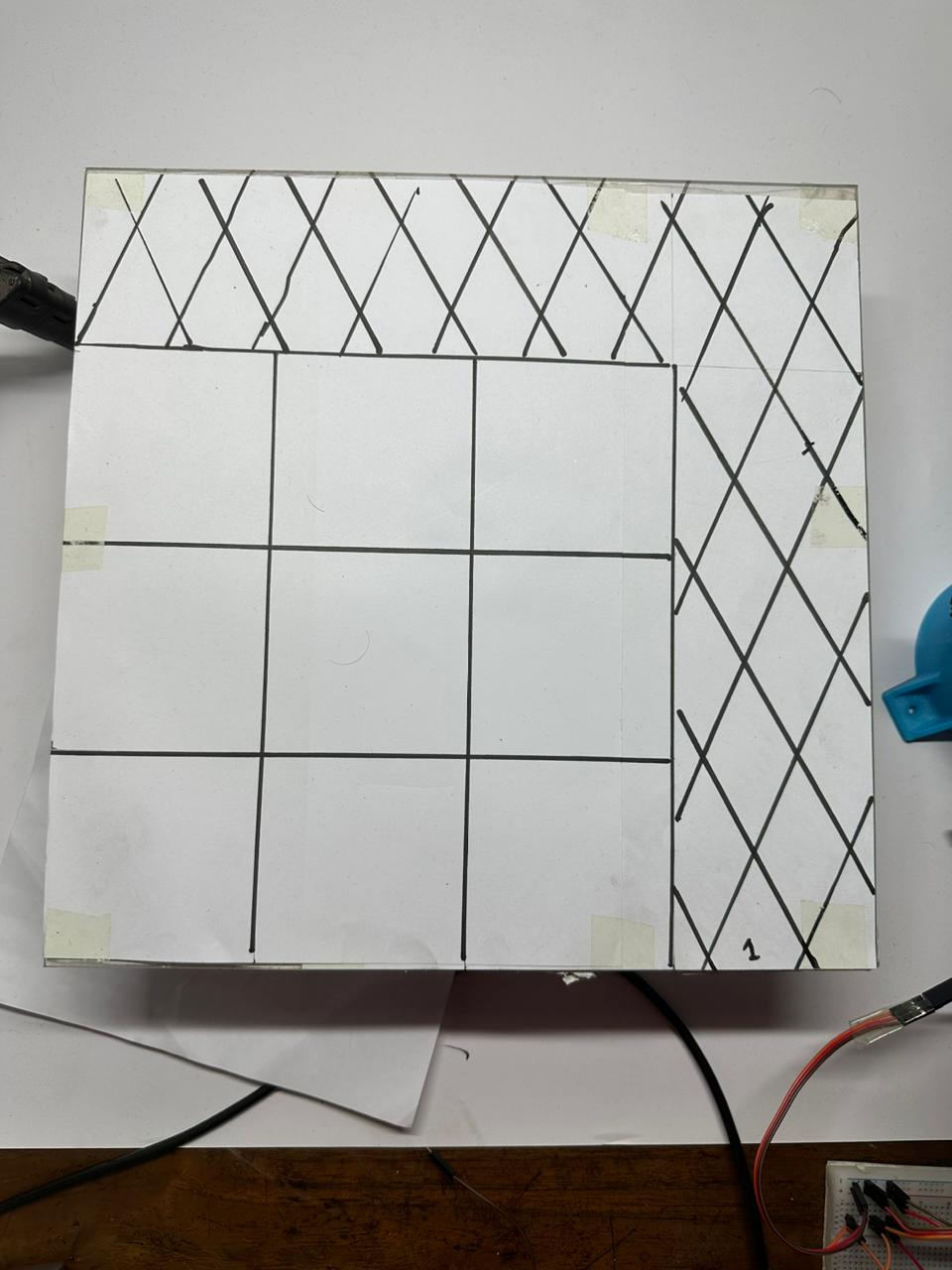
****

Fig:01

**Object:**

We have used a small servo motor as our test object.

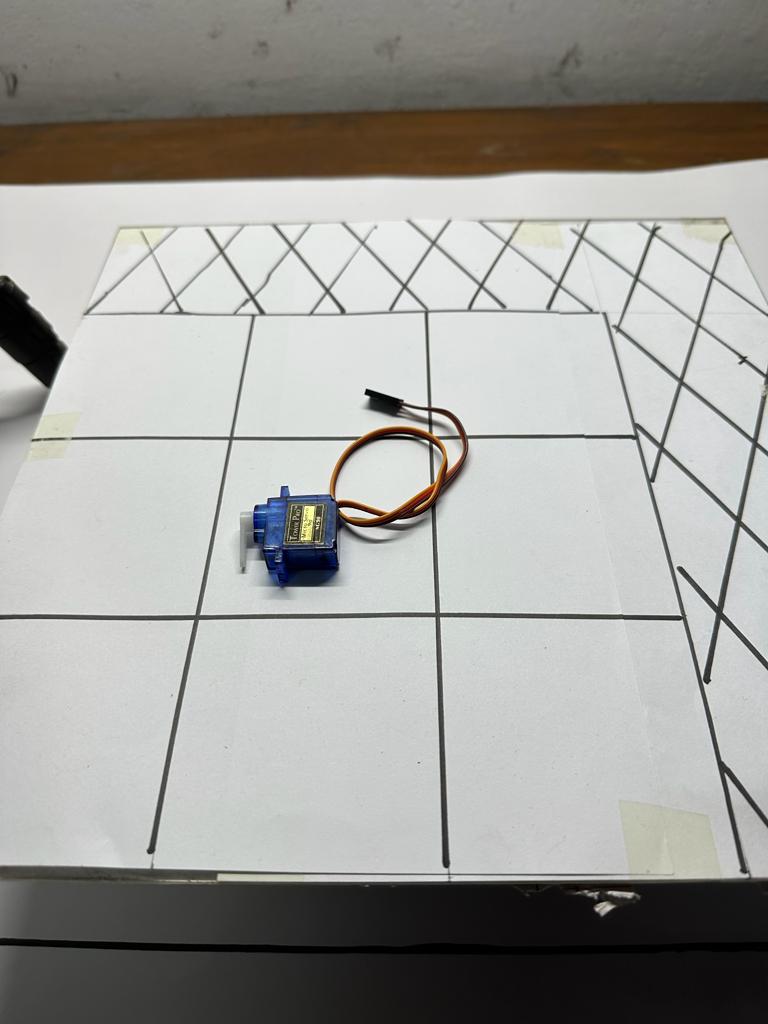


Fig:02

**Camera:**

In order to create the dataset as well as capturing a single image for detection, we utilized a webcam.



Fig:03

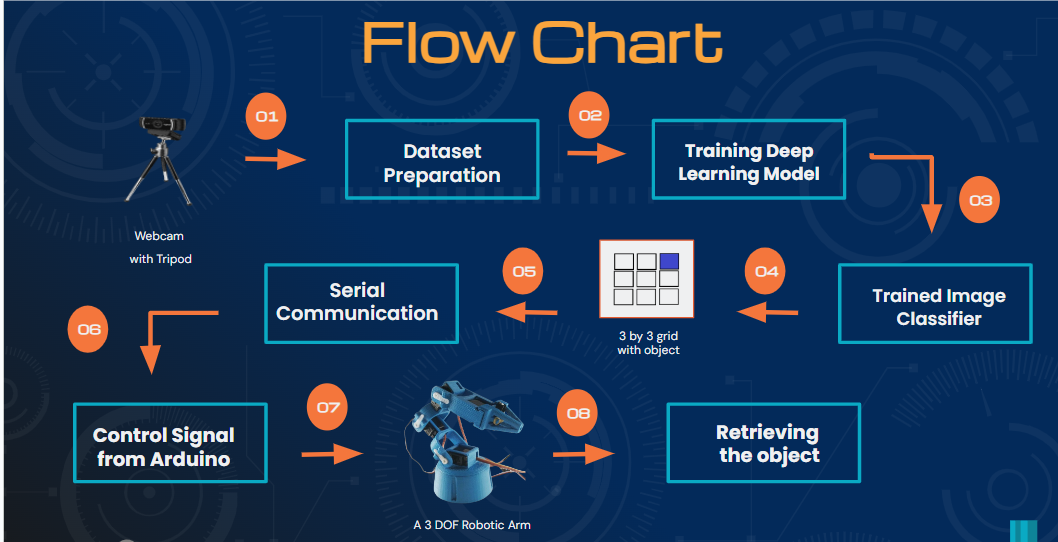
**Robotic Arm:**

For reaching the desired position and retrieving the object.

### 

Fig:04

# **Algorithm:**



## **Data Collection:**

For creating the dataset, we created 9 sub-folders, comprising 100 images for each of the positions. In order to capture 100 images, we used opencv libraries and segmenting 100 images frame-by-frame from a live video feed. The following code describes the above scenario

**Dataset Preparation:**

Inside the images folder, there were 9 sub-folders each named according to the class or in our case the name of the positions for the object. Inside them the pictures for the corresponding position were stored. A helper function was created to make a directory named file. Using os.walk method all the sub directories and files inside them from the original image folder were parsed and the image names were stored in lists. Inside the newly created file directory 3 sub directories were created named test,train and validation. In each of them 9 sub directories were created according to the class names. Then using random shuffling the images were copied to the new directories according to their class names using the names from the list of files or image names. Images were distributed in such a way that 70% data were in the training set, 20% images were in the test set and 10% of them were in the validation set.

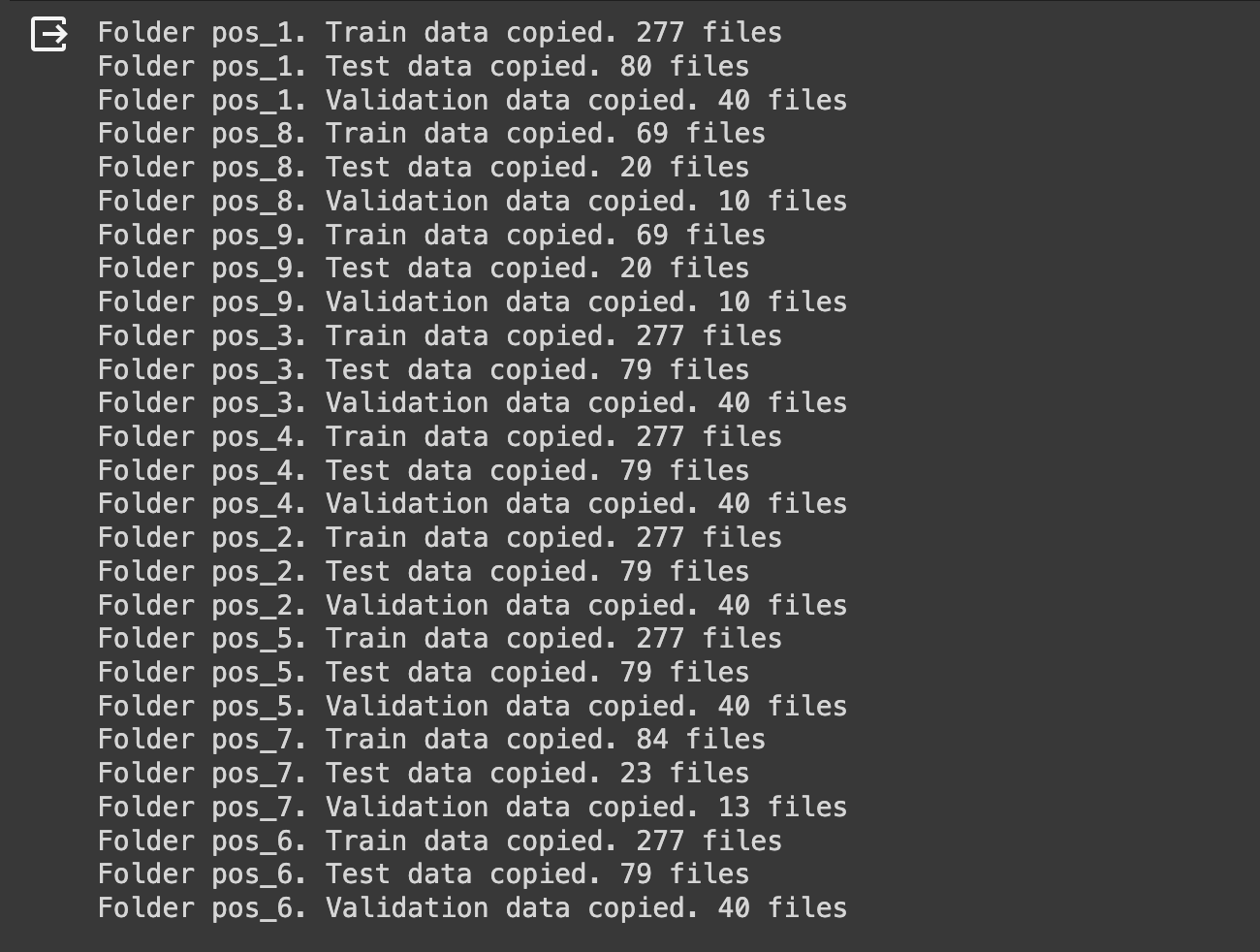


Fig:05

Another helper function was also created to plot a random image from a specific class to check whether the entire data set preparation was done successfully or not.

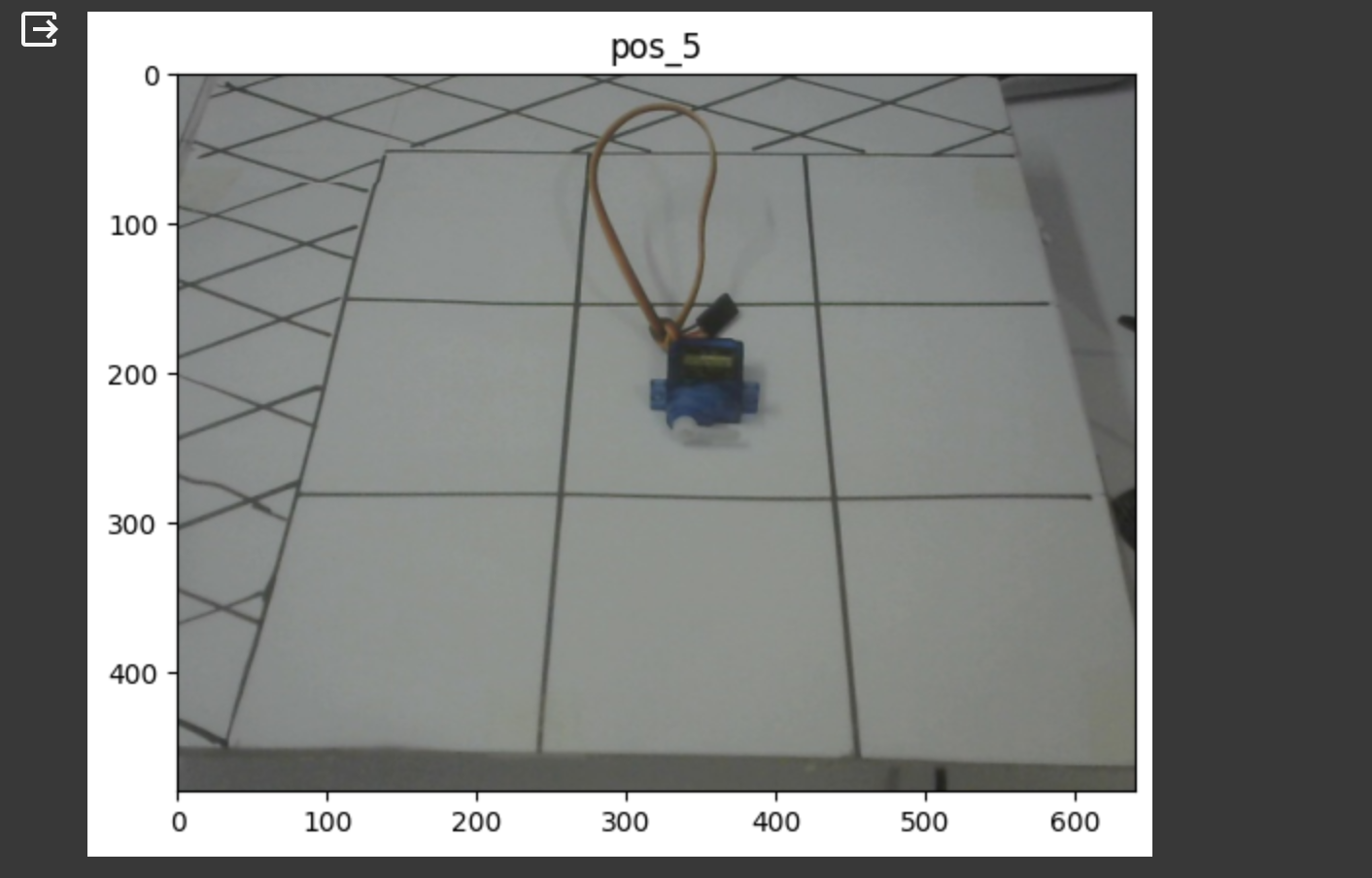


Fig:06

## **Setting ImageData Generators:**

Then from Tensorflow’s Keras Preprocessing, the ImageDataGenerator class was used. We configured 3 different image data generators for train, test and validation using the flow from directory method. The images were normalized before. Also, they were bit shifted along length and width for better framing.

After creating the image data generators, we created the train\_data, test\_data and validation data. On the fly, they were resized as 240x240 pixels and 20 images were processed at a time. We used class-mode categorical as we are dealing with multi-class image classification.

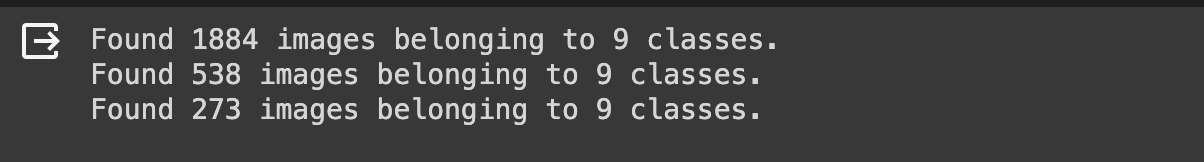


Fig:07

## **Designing and Compiling the Model:**

After preparing the data for test, train and validation we moved towards designing the model which will be used for implementing the image classifier for our project.To design the model, we have used Keras’s Sequential or layer by layer API. Besides the input and output layer, we have used 3 hidden layer sets.

Our input layer takes a 240x240 image of 3 channel color(RGB). First CNN layer uses 16 filters of 3x3 shape and uses the ReLu activation function. These filters use kernels which have specific transfer functions. If we move the kernel over the entire image, this filter will extract features from the image.

Then a Maxpooling operation was done by another layer of 2x2 size. Maxpooling runs over all the pixels of an image. It takes a 2x2 matrix of pixels and keeps only the maximum pixel value and discards the rest of the pixels. This makes the image sharp.

Next 2 CNN and Maxpooling layer sets do the same job as the first hidden layer set. Then a flatten layer was used to flatten the 2d features from previous layers to 1D. The final output layer is a dense layer with 9 neurons as we are classifying between 9 classes of images using softmax activation which is suitable for multi-class image classification.

The layers of the model are shown with their input / output below:

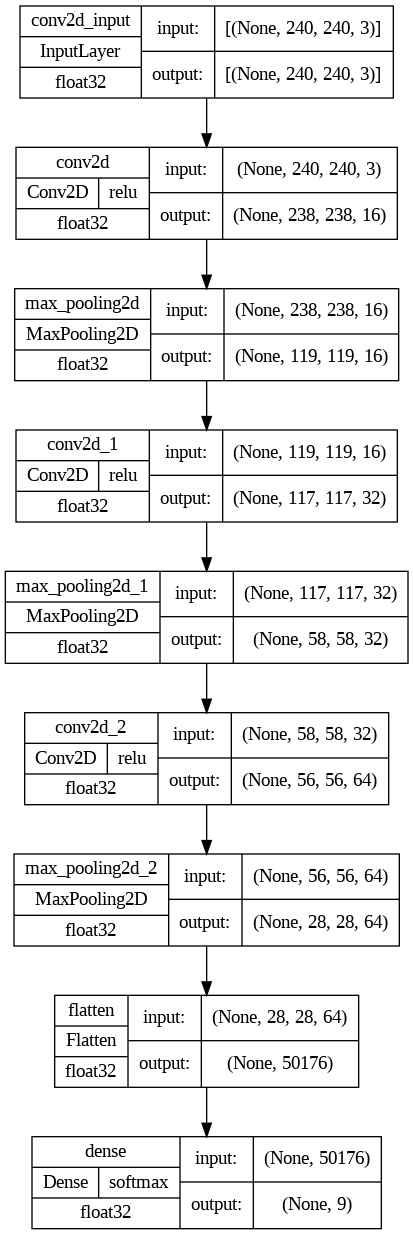


Fig:08

The model summary is shown below:

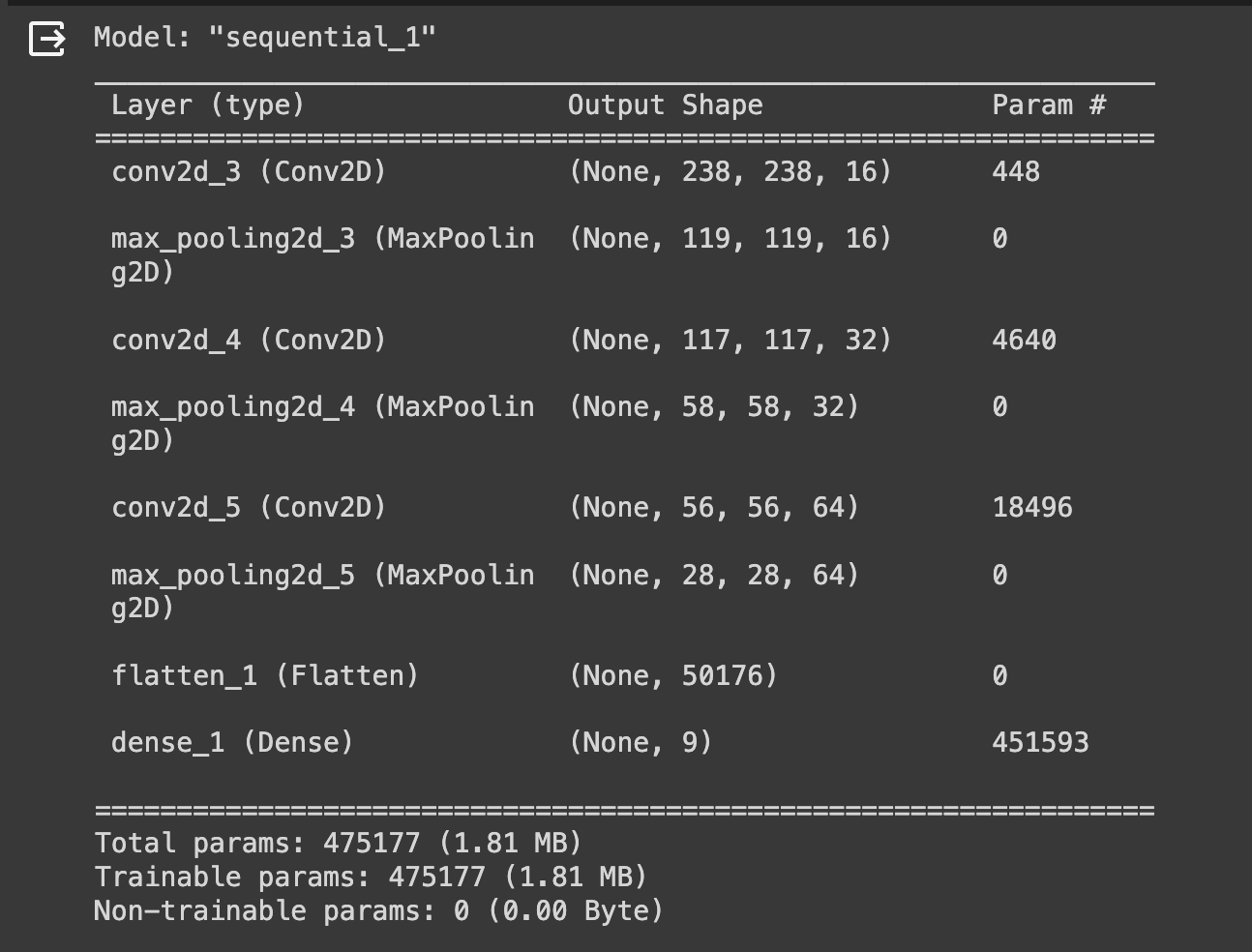
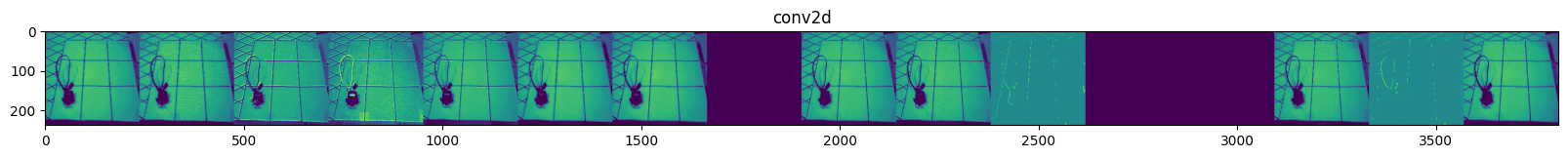


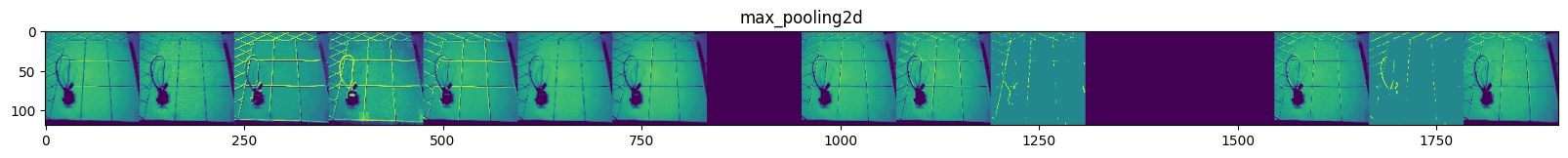
Fig:09

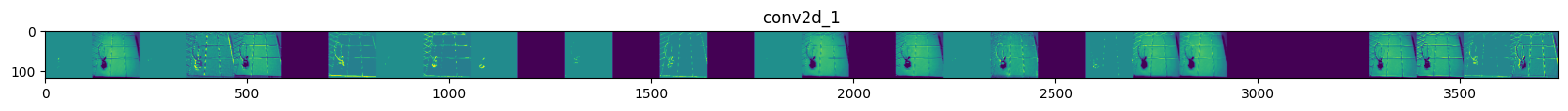
## 

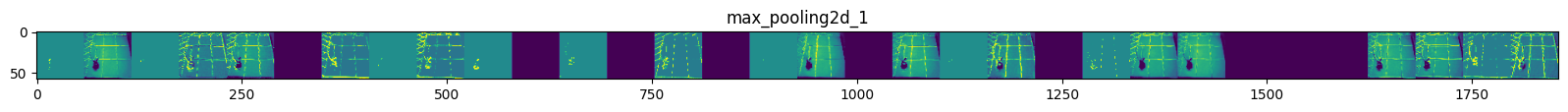
## **Fitting and Evaluating the Model:**

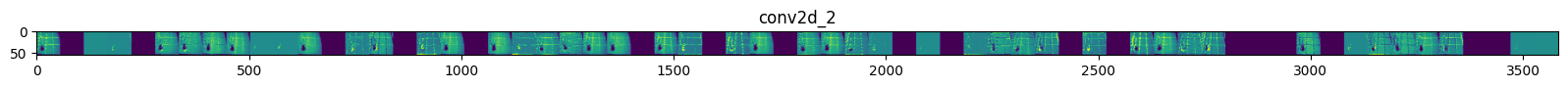
The model was fitted using 30 epochs and batch size of 20. The intermediate results from convolution and maxpooling for a random position is shown below:











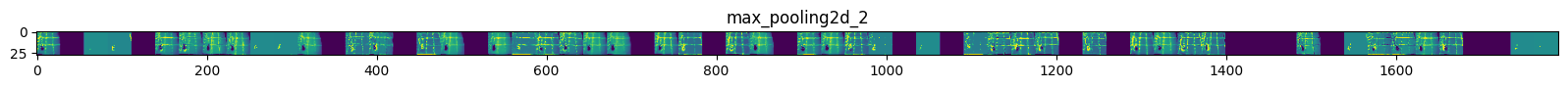


Fig:10

The model was tested on the test dataset and it got an accuracy of 100%. From the evaluation results we can clearly see that our model has been k because after 6 epochs, it reached an accuracy of 98%.

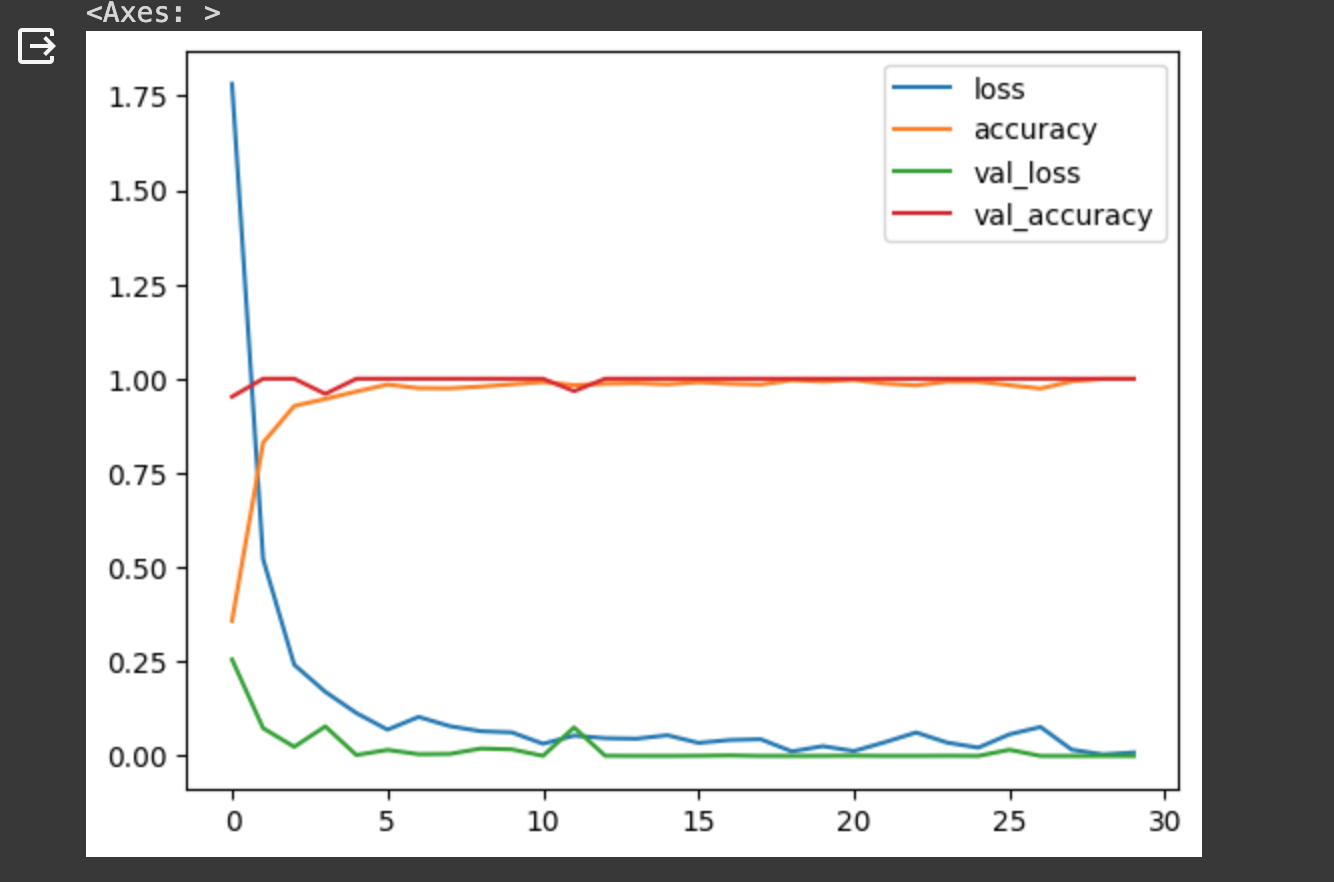


Fig:11

In general, this is a problem with deep learning models. But as the scenario for our case is very static, this overfitting problem won’t have any big impact and we decided to keep going on with this model.

## **Saving and Reusing the model:**

The model was saved as an h5 file in the current directory and another notebook was written to use this model to test on various images. And the model was able to predict classes for different images successfully.

## **Creating an image classifier tester:**

It first loads the model which was previously saved as a h5 file in the same directory.

A helper function was created to load a processed image tensor. This function first reads an image using tf.io.red\_file which results in a tensor containing the raw binary data of the image. Then it decodes the image into a specific datatype. It also resizes the image into a specific size based on the input argument of the function. Finally, it normalizes the image and returns the processed image tensor.

Finally, the tester script uses the model to predict output on the loaded image. The output returns an array containing the matching probabilities with each class. To find out the matched position or class name, we used argmax() function to get the index and used that index on the class name array to get the desired class name or position number.

## **Serial communication:**

In order to establish a communication channel between our python scripts in VScode and arduino IDE, we utilize python built in library Pyserial. The code sets up a serial communication object named ‘SerialObj ’using the ‘serial’ module. The object is configured with parameters such as the COM port (COM3 in this case). The code sends a command to the microcontroller based on the value of ‘new\_pred\_name’. The value of ‘new\_pred\_name’ represents a position, and a corresponding byte is sent to the microcontroller using SerialObj.write(b'X'), where 'X' is a digit associated with the position. There's a 3-second delay using time.sleep(3) which allows the Arduino to stabilize after the serial port is opened. Once the connection between the terminal has been established, the desired position number is printed across the arduino terminal.

## **Arduino codes for controlling the arm segments:**

The mechanical design of the arm can be broken down to 3 parts: 1)Base 2)Shoulder 3)Elbow. Each of these parts is separately controlled by servo motors. The servo motors are in turn controlled by arduino codes based on the specified angles for each position. The specified angles were found applying the equations of inverse kinematics as well as by using potentiometers. Then a function for each position was defined such as pos1.

## **Extra Features:**

The script that we used initially was for creating a model from the captured RGB images. And we used colored images for the model compilation and training. We also kept a section for using grayscale images for training. If we run those sections which are specifically for grayscale training, it will generate grayscale images on the fly while using imageData generator. Then it will also finish the training on those grayscale images.

Also, in the tester script, we have kept the option for loading the captured image on grayscale and using that converted grayscale image for predicting the class or position number.

# **Limitations:**

Currently our model can classify a single object positioned at one of the 9 places in the plane. If we place our object in the middle of any two boxes or if we place the object anywhere out of that 3x3 grid, our model will fail to give correct position prediction to the arm. Also, if we try to place 2 objects at a time in the place, our model will fail to recognize two different objects.

As we progressed, new challenges emerged (such as unavailability of components) requiring constant adaptation. Issues ranging from algorithmic intricacies to real-world variability demanded a debugging approach. For instance, we were facing severe issues with controlling the servo motors via potentiometers and calibrating the servo angles for each of the position. Furthermore, Integration of software and hardware was a very challenging task. We needed serial communication to implement this process. While doing trial and error method needed to be performed.

# 

# **Problems Faced:**

Initial problems that we faced during the start of this project was lack of knowledge about OpenCv, machine learning and Deep neural networks specifically using deep learning to make image classifiers. Our first 3-4 weeks were spent solely on finding and trying out different resources and getting ideas on how to use those ideas to build our project. That’s the reason we had to modify our initial proposed project idea as we realized it would be very much complicated if we try to stick to our initial proposed idea. Thus, we modified our original plan and simplified the working algorithm so that we can successfully finish our project within the allocated time.

For the hardware section the troubles were worse than that faced in the software section.

Challenges faced:

1) Dealing with variations in lighting conditions, perspectives, and backgrounds proved to be a challenge during the training phase. Fine-tuning the image classifier to handle these variations enhanced its robustness.

2) Ensuring precise and accurate movements of the robotic arm to reach the identified position required careful calibration and testing. Achieving a balance between speed and precision was crucial.

# **Future Work:**

In this project, we have built a deep learning model to identify the position of a single object positioned at one of the 9 distinct boxes. Based on the prediction of the image classifier, our robotic arm will reach the location of the object and move the object to a predetermined place. Some tasks can be done in future which are:

* We can expand the dataset to include a more diverse set of scenarios, backgrounds, and object orientations which would eventually enhance the model’s ability to generalize to different conditions.
* Building an object recognition algorithm instead of image classifiers to identify more than 1 object from the given frame.
* Finding out the distance between two objects from a given frame so that we don’t need to position our object in predetermined locations.
* Instead of using static images, we can use real time video footage
* Integrating additional sensors, such as depth sensors or cameras with 3D capabilities, could provide richer data for better understanding the environment and object positioning.
* We could also incorporate features that enable human interaction, such as voice commands or gesture recognition, these would make the system more intuitive and user-friendly.

In the hardware portion we can develop the motion algorithm. Currently our moving criteria is only limited in the nine predefined positions. In the future we want to eliminate this limitation by expanding the movement criteria to any points in three dimensional space. We are also thinking about expanding the degrees of freedom of the arm to 6dof. Besides, involving the PID motion algorithm along with stepper motor based motion will bring swift and precise movement. We are also thinking about making the arm voice activated. Furthermore, we are trying to build the deep learning model completely microcontroller based with a user defined data set. If we can successfully implement this, then anyone can use this independently for their own function.

# **Guideline:**

To switch from image classifier model to object recognition we can use some models such as bounding box models. If we can identify the object from our frame, we can compare it with our trained model if it is our desired object or not. Also, to find out location of an object or distance between two objects in a frame, we can use OpenCV which will allow us to place our object anywhere within the captured frame. And using real time video footage is very similar to processing static images. The main difference is that we have to process multiple static images very fast which will consume much more processing resources compared to static images.

To implement our hardware improvement, the first step is to expand the working area to 3D space by applying inverse kinematics. Second step will be to implement PID controlled motion. Next, we can make the deep learning model completely microcontroller based. Next we can upgrade our degrees of freedom and change the movement to stepper motor based by transferring the algorithm.

# **Acknowledgement:**

For hardware: Our design is based on a open project done by theGhizmo from EEZYrobots

[Link](https://www.instructables.com/EEZYbotARM-Mk2-3D-Printed-Robot/)

Besides some tutorials from coursera and youtube:

Inverse kinematics Angela Sodermann: [[link]](https://www.youtube.com/watch?v=D93iQVoSScQ&list=PLUbz9cJKt-%20lYIUTBKy5pIF4DlVSozHOZG&index=9&pp=gAQBiAQB)

Easy inverse kinematics for robot arms:[[link]](https://youtu.be/Q-UeYEpwXXU?si=9wFmiWy9mQlD4anl)

Robot arm with inverse kinematics tutorial: [[link]](https://youtube.com/playlist?list=PLajy7T24Oh4QsDclNPTJ1gMSQfBioqb3&si=fyN_S7o0NBRTJcPy)

ROS for beginners: Basic, Motion and openCV [[link]](https://www.udemy.com/share/101XvY/)

# **Conclusion:**

To summarize, our study successfully created an integrated system that automates object retrieval by combining robotic control with image classification. We have illustrated a useful use of deep learning and robotics in a real-world setting by creating and training the image classifier to recognise nine distinct positions and using serial communication to send judgements to a robotic arm.