4

Methodology

This chapter describes the proposed approaches and the state-of-the-art uncertainty estimation methods used in this research work.

4.1 Testing based approach

In this work we have proposed a tool for generating synthetic datasets under different test constraints for evaluating the performance of the state-of-the-art uncertainty estimation techniques. The approach consists of four major steps. The approach begins by defining a set of requirements for different test cases like, lighting, distance, blur, deformation etc., These requirements allow the developer to check if the blender software is capable enough to generate the variation in the dataset. After formulating the requirements and analyzing the capabilities of blender, we create a configuration file for the modifiable parameters which helps in generating synthetic datasets for different test constraints.

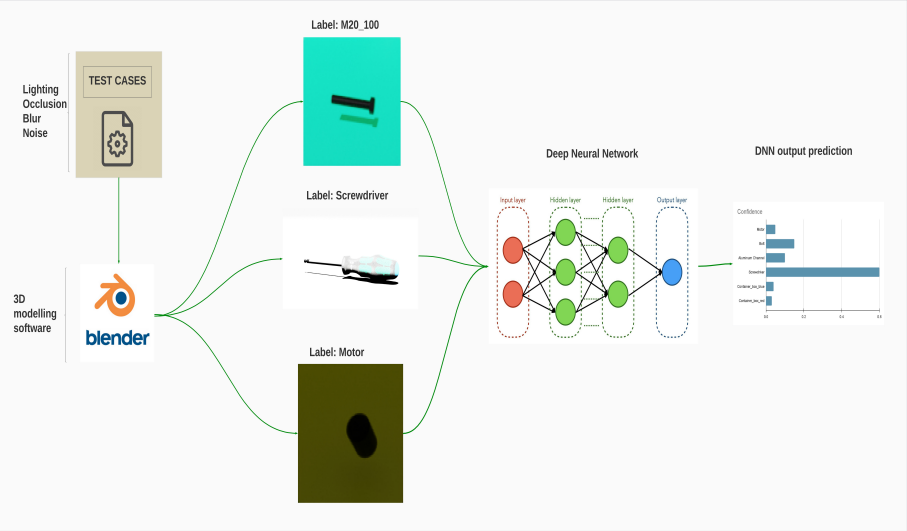


Figure 4.1: Testing based approach

After identifying the modifiable parameters of the software for a test constraint we need the 3D cad models (real world representation of an object in the 3D modeling software). These models can be either

directly created in blender software or can be imported from the existing designs or other platforms in .obj and .stl formats. Some of the sources like YCB household dataset and ShapeNet ect, provide several 3D representations of real world objects which can be used for generating the datasets for several applications like classification, object detection, semantic segmentation etc. The main idea for developing this tools is to generate synthetic datasets in different test constraints by modifying very few parameters in the configuration file. Hence initially a dataset for the required objects will be generated in normal conditions or a base scenario and later by modifying the existing parameters of the software, we introduce different variations and generate dataset for distinct scenario or test constraint datasets which are not similar to the base scenario. The key idea in this approach is to train the deep learning models on a base scenario or on normal conditions dataset and test their performance in the distinct scenario or different test constraints. For example, we generate a dataset in which the objects appear clearly in visible range and train the DNN model, but in real world the scenario might be different and the objects may be far or near. So, by adjusting the position of the objects or the camera's parameters we create test constraints for far and near to generate datasets in which the objects appear very far or very near in the images. Thus by this approach we can test the performance of different DNN models in different real world scenarios and check if the models are uncertain in the test constraints.

With the help of the proposed tool in this work we have generated datasets for different test constraints like, bright lighting, dark lighting, near distance, far distance, blur, deformation of the objects, distractor objects, and background textures which we will discuss in the next chapter 5.

4.2 Expeted value based approach

In deep learning entropy is considered as a measure of uncertainty or randomness and it is used for evaluating the performance of a DNN model. Generally entropy is obtained from the output predictions of a deep learning model and higher the confidence for a particular class lower will be the entropy. In general, humans have a little knowledge of uncertainty or randomness in some real world scenarios. To represent this uncertainty in a mathematical form fuzzy logic can be used as it can make decisions based on imprecise or incomplete data. Fuzzy logic is used in a variety of applications like, data analysis,artificial intelligence and control systems. Fuzzy logic uses a set of fuzzy rules which are defined as if-then rules and these rules can be defined in natural language to produce a mathematical output. Hence in this approach we are interested to create an expected value for the entropy of an image even before passing the image through the DNN model. Based on the previous approach presented in 4.1 , we extend this approach to generate an expected value of entropy using fuzzy logic.

Since we are using a tool for generating synthetic datasets, we have more precise control of the characteristics of the generated data and it is possible to specify the exact distribution of the data and the specific properties of the generated samples. These characteristics are then used by the uncertainty generator which is a fuzzy inference system(FSI) to generated expected entropy values. The main idea in this approach is to check if the model's output entropy distribution is similar to the entropy distribution generated using fuzzy logic. For example, in the distance, as a human we know as the distance of objects increases from the camera the DNN model should be uncertain in its predictions and hence the entropy of

the DNN model should be high. We consider this human knowledge and the camera's parameters such as position or focal length and use it for creating the fuzzy rules, if the focal length value is high or lower then the entropy should be high. If the focal length is in medium or normal range then the entropy should be low. These rules are determined by the membership functions in the fuzzy logic to obtain final output values. We will discuss this approach in more detail in the next chapter 5 .

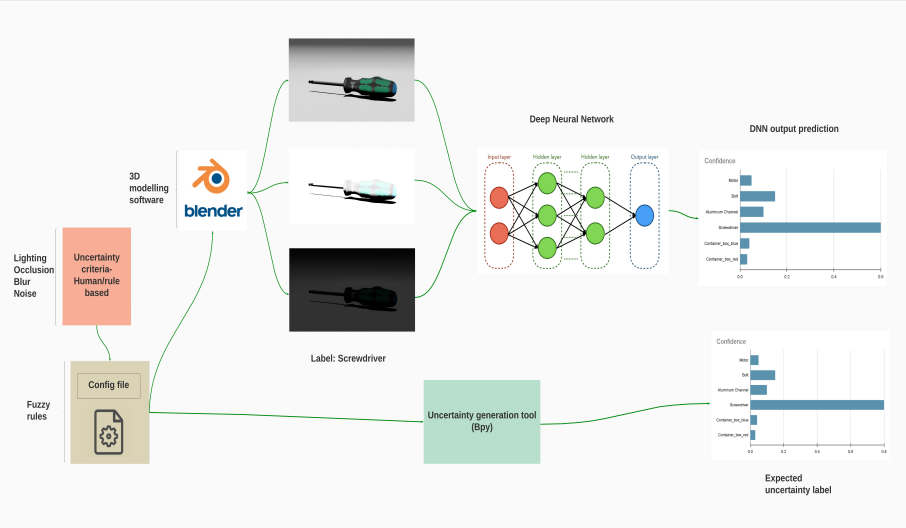


Figure 4.2: Expected value approach

4.3 Uncertainty estimation techniques

Since we are interested for evaluating the performance of deep learning models and the uncertainty of their predictions on different test constraints, it is important to select the uncertainty estimation techniques which can be incorporated to different DNN architectures easily. Hence in this work we have used the following uncertainty estimation methods for evaluating the performance of DNN models in different test constraints.

4.3.1 Softmax-Cross Entropy Loss

Although cross entropy loss is not typically used as a measure of uncertainty similar to other uncertainty estimation techniques like, evidential deep learning, bayesian approaches, it the most commonly used loss function in the classification task. It can also be used in conjunction with other techniques like dropout and ensembles to help quantify the uncertainty of a model's prediction. Hence in this work, we are interested to check the performance of the DNN models trained with cross entropy loss.

Cross entropy loss is used to measure the difference between the predicted probability distribution and the true probability distribution. The cross-entropy loss is often used in conjunction with the softmax function, which converts a vector of real values into a probability distribution. The output of the neural

network is passed through the softmax function to produce a probability distribution over the possible classes.

4.3.2 Evidential Deep learing

Evidential deep learning is a single-pass uncertainty estimation technique, which can provide both model and data uncertainties. It is based on the idea of evidential reasoning, which is a type of reasoning that takes into account of the evidence or data that is available in order to make a decision or prediction. The output prediction of a DNN model trained on evidential loss is obtained as a dirichlet distribution which is often used to represent the distribution of categorical variables. The dirichlet distribution is defined by a set of k positive parameters, known as "concentration parameters" α, which represent the relative weight of each category. Unlike probability distribution, the dirichlet distribution provides a value greater than or equal to one in its predictive distribution. The maximum value of dirichlet distribution is determined by the shape of the distribution and the concentration parameters α .

4.3.3 Monte-Carlo Dropout

Monte Carlo dropout is a technique for approximating Bayesian inference in deep neural networks. During the training of a deep neural network, dropout layers are used as a regularization technique to avoid over-fitting. When the dropout layers are activated, some of the neurons in the hidden layers of the DNN model will be dropped randomly. This has the effect of forcing the remaining units to learn more robust and generalizable features, as they must compensate for the missing units and their contributions to the prediction. The dropout layers of a DNN model are activated using a drop rate or probability p. The value of p ranges from zero to one and this value represents the percentage of neurons to drop. This process of dropping certain number of predictions is repeated multiple times which are termed as number of farward passes . The resulting predictions from all the farward passes are then averaged to produce final prediction. The number of forward passes in Monte Carlo dropout is determined by the number of times that the model is sampled and used to make a prediction. A larger number of forward passes will result in a more accurate estimate of the uncertainty or confidence of the model's predictions, but it may also increase the computational cost of making the prediction. The number of farward passes depends on the specific characteristics of the model and the training data. Also, enabling dropout during test time and passing the data multiple times allows to estimate the predictive uncertainty.

4.3.4 Deep Ensembles

Deep ensembles is a method for improving the performance and uncertainty estimation of deep learning models. This method involves training of multiple deep learning models on the same data, and combining the output predictions of all the models in order to make a final prediction. One of the common ways to combine the outputs of multiple DNN models is to compute to the average of all the model's output predictions. Proper scoring rules are required for training the ensemble models and in the case of multi

class classification, the softmax cross entropy loss can be used as proper scoring rule [9]. Different DNN models provide different outputs based on the same loss function and by considering the variance or disagreement between the predictions of the different models helps to provide more accurate estimate of the confidence of a models's prediction or the uncertainty.

5

Solution

This chapter presents the the experimental setup for generating different types of synthetic datasets for testing the performance of the deep learning models using the state-of-the-art uncertainty estimation techniques discussed in the chapter4. and the datasets used in this research work.

Blender has numerous amount of features like modelling, UV unwrapping, texturing, sculpting, rigging, motion tracking, geometry nodes, animation, movie creation, vfx, rendering, and compositing. All these features contain huge amount of parameters to control. The parameters that can be modified or used in different features depends on the application and desired output of the human. Thus based on the limitations discussed in 3, we begin our approach by identifying different modifiable parameters in the blender software that can help to introduce maximum variability in the test data sets which can help to analyze the performance of different uncertainty estimation techniques.

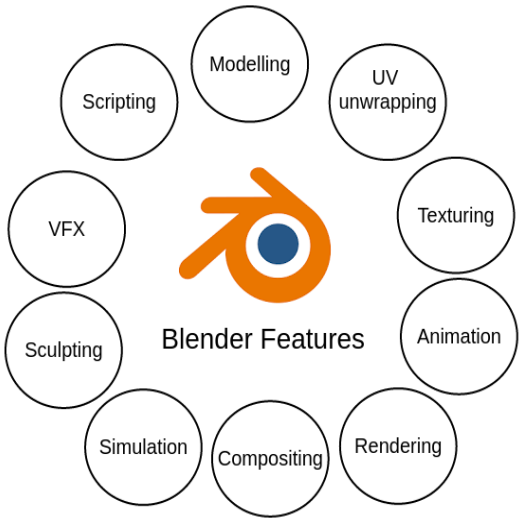


Figure 5.1: Features of blender software

5.1 Modifiable Parameters

Using the Blender's api (Bpy) functionalities, we can control different parameters of the software in the form of python scripts to introduce different variations to the synthetic dataset. Some of the modifiable parameters used in this project are discussed in this section.

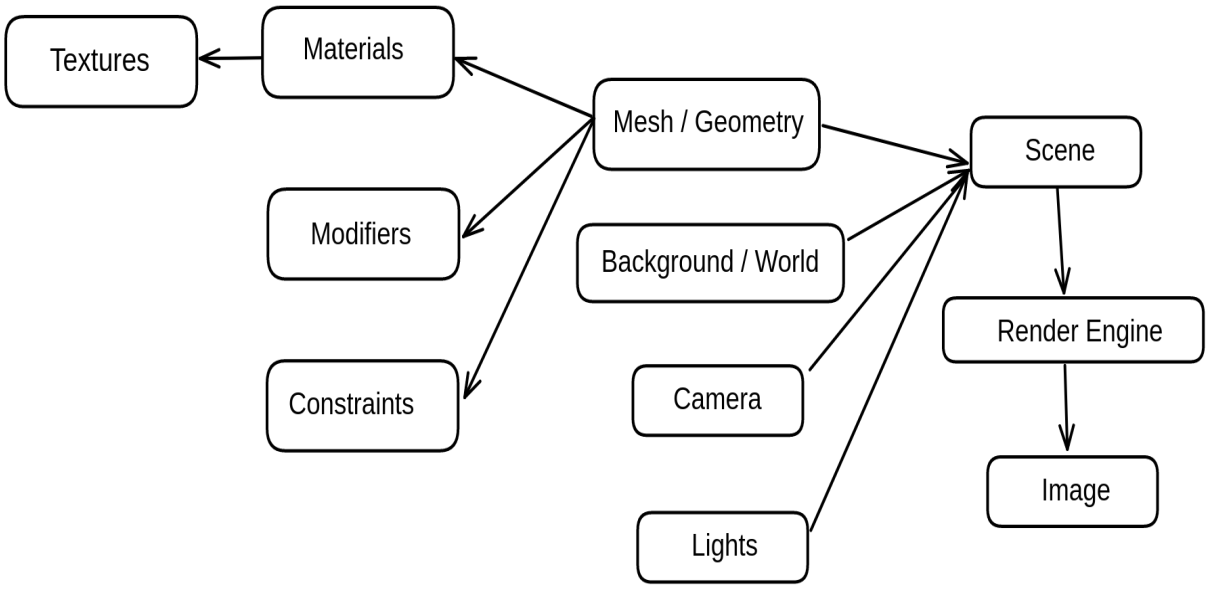


Figure 5.2: Modifiable Parameters in blender

5.1.1 Render Engine

Rendering is a process in which a 3D scene is converted into a 2D image. During rendering, the computer calculates the light in our scene to create our final image or animation. For this calculation, a render engine needs information about the 3D scene which can be obtained from geometry of the objects, materials, textures, light setup, camera setup, world background and many other parameters.

Blender has two inbuilt rendering engines, Evee and Cycles both of them aim to be similar in view but works completely different. Evee is a real time rendering engine and its main priority is speed. Hence it is appropriate for close to real-time rendering performance which means the rendering process in Evee is fast. On the other hand Cycles is a path-traced render engine which is much slower than Evee but has an advantage of producing light conditions similar to the real-world [19]. Hence most of the community in blender uses cycles rendering engine for rendering photo realistic images. Also, other external render engines like OctaneRender, LuxCoreRender, Radeon ProRender can be integrated into the software for rendering the images or animation.

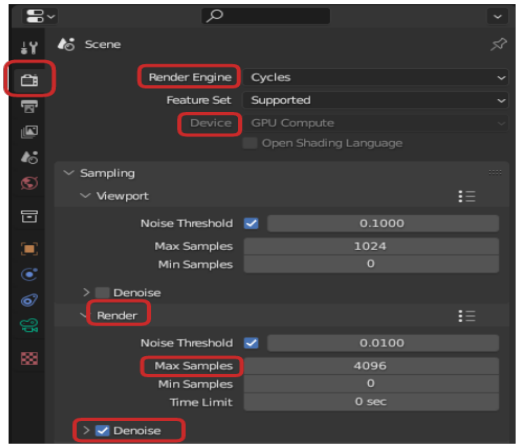


Figure 5.3: Render engine settings

In this work we have used cycles rendering engine as it is mostly used by the community for generating photo realistic images. One of the other parameters which affects the rendering time is the number of samples 5.3. Number of samples is the parameter used to trace the number of pathways in every pixel in the rendered image. Generally it is believed that high number of samples provide more accurate and less noisy images which will also increase the rendering time. More number of samples are used mostly where very high realistic output is required like, movies, animation, games etc. By default blender sets this value to 4096 in the render settings. Instead of using such high number of samples we can reduce the samples to a value between 100 or 200 and turn on the denoise option to achieve good quality images with very less rendering time. Enabling the denoising option removes the noise in the final rendered image. Also, blender provides 2 types of denoisers Open Image Denoise and Optix denoiser. Both of them use artificial intelligence algorithm to remove the noise. Open Image denoiser runs on CPU and it is the default one. Optix is based on Optix-NVIDIA graphics engine and it uses GPU for computation.

5.1.2 Mesh/Geometry

3D modelling is a techniques used to make a surface which represents the human imagination of a abstract objects or resembles the shape of a real world object. This 3D representation of real world objects can be achieved in blender using different methods like mesh modelling, surface modelling, sculpting. With little knowledge in the software users can also import the 3D representations created by other people into the scene instead of modelling them from scratch. Blender supports good range of file formats like, obj, stl, fbx etc, which represents the information of 3D representations of the objects. In general when a 3D model is created or imported from other sources, the objects will have the desired shape but do not contain any color information. They appear to be gray and the users must adjust some settings or create materials by themselves to obtain the desired real world representations which we will discuss in the following sections.

A scene's geometry is made up of one or more objects. These items might be anything from simple 2D and 3D shapes to fill your environment with models, armatures to animate those models, cameras to record it all in the form of images or videos, and even lights to brighten the scene. All of these objects are made up of two components which represent the information of the object in 3D space. The first component is called as object and it represents the information of scale or size, position and rotation of the object. The second component is known as object data and it represents the information like, textures, materials list for the objects, and in case of camera this object data provides the information about the sensor size, focal length, depth of field values etc. In this section we will mainly discuss about the parameters corresponding to the 3D representations of real-world objects.

- Pose of the object
  - Every object in the 3D contains an origin point and the position of the object is determined by the location of this origin point. This origin point also plays a key role in the rotation the object, i.e the axis of rotation depends on this origin point. It is very important to set this origin point before performing any animations or transformations in the scene. In this work we have designed the 3D representations of real world components like, bearing, container box, aluminum channels and 9 other objects for generating the synthetic datasets. To keep the rotations uniform we have set the origin point of each object to its center with using the option set_origin - origin to geometry. The location rotation and size of the object can be then controlled by providing the respective values in x,y,z direction as shown in the fig 5.4

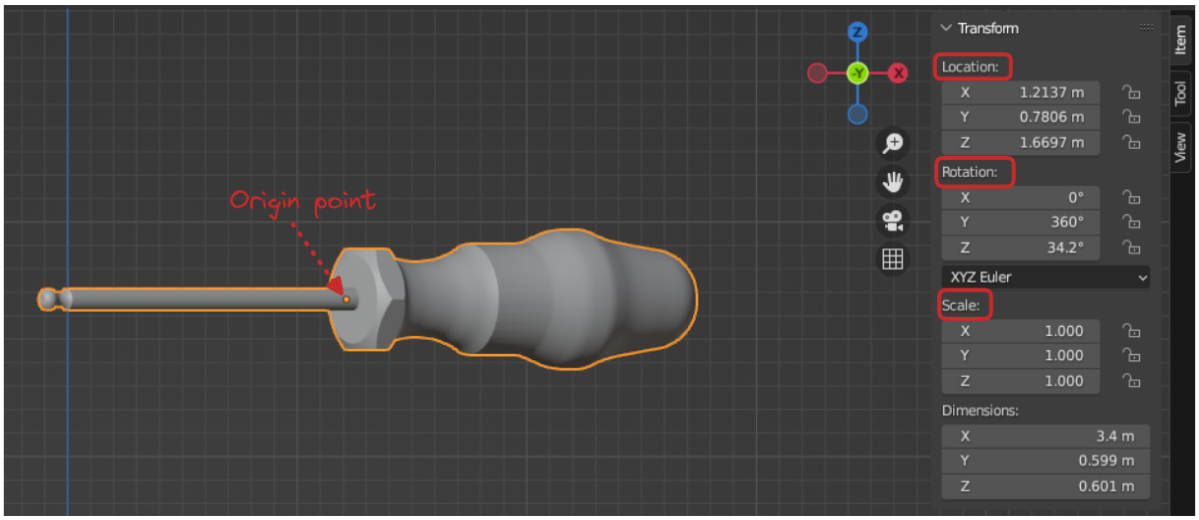


Figure 5.4: 3D representation of a screwdriver

- Modifiers
  - Modifiers are the operations used in blender to change the object's geometry in a non-destructive way. Modifiers help to preform many complex operations which consume lot of time and effort

automatically. Blender has a good range of modifiers built into it and utilizing them depends on the application and desired output. Subdivision surface, Edge split, Bevel, Mirror, and Solidify are some other commonly used modifiers. All these modifiers do not change the geometry of the object directly but they represent this change in the final rendering process.

– In this work we have used two modifiers for all the objects. First modifier is the subdivision surface 5.5. This modifier allows to divide the surface of the mesh into smaller segments and gives more smooth and even look to the object. With the help of this modifier a smooth organic look can be achieved without storing and maintaining large amounts of data. The second modifier used is Edge split modifier 5.5 . This modifier helps in making the edges of the object to appear sharp instead of curve or round surface by splitting the edges based on a threshold edge angle. i.e if the angle of the edge is more than the threshold edge angle it will split the edges and makes them separate which gives the objects a sharp look at the edges. One main thing to consider with the modifiers is that the order in which the modifiers are applied effects the geometry of the object. Hence we have first used subdivision surface modifier to smooth the surface completely and then applied edge split modifier to make the outer edges appear sharp.

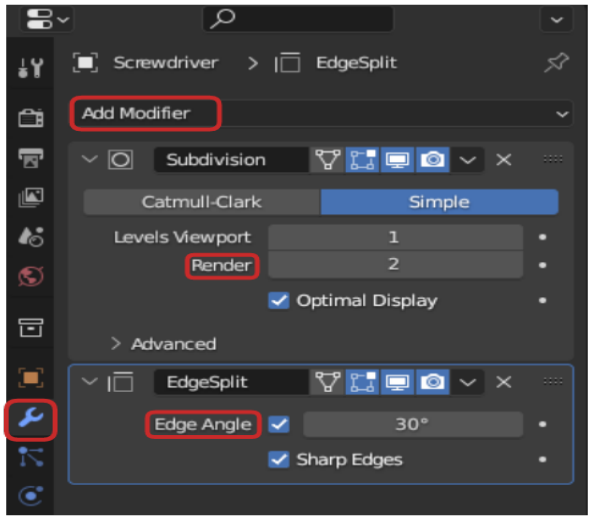


Figure 5.5: Modifiers

- UV unwrapping

– After creating the 3D model of an object, it is important to unwrap the mesh. UV unwrapping is a method in which a 3D shape is projected as a 2D mesh 5.6. The different types of available projections are, cube projection, cylinder projection, sphere projection. The type of projection depends upon the shape of the geometry. Also, instead of choosing the projection type we can use smart uv project option which performs the best operation for complex geometry shapes. Each point present in the UV map represents a vertex point of the mesh and similarly edges

and faces. This process will be very helpful for applying materials and image textures. The materials applied to the object can also be visualized as an image in the uv editor and mesh in 2D space can be adjusted to avoid any distortion of material on the object's surface. Changes in the position, orientation and scale of the mesh in 2D space of UV editor doesn't affect the shape of the object in 3D space, but it helps to avoid the distortion appearance of the material in the final render.

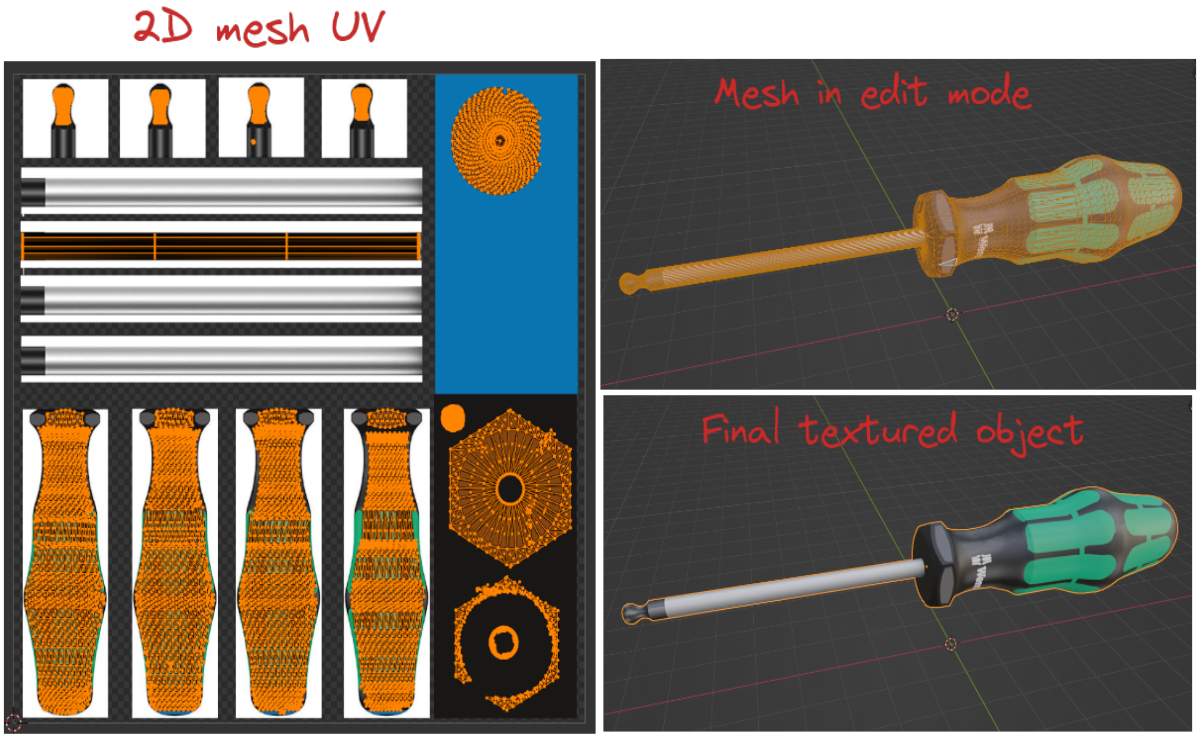


Figure 5.6: UV unwrapping

- Materials and textures
  - Materials describe the color and texture of an object and they are very important for representing the 3D geometry similar to the real-world object. In blender, for creating and applying materials we have a separate workspace called Shader Editor in which we can combine different nodes corresponding to input nodes, output nodes, shader nodes, vector nodes, texture nodes etc to create a realistic material. With the help of these nodes any material can be created in the form of a node graph as shown in the fig 5.7. Shader nodes like noise textures, muskgrave texture node, emission node, glass shader node, image texture node ect, are available in blender and each of them has many properties to vary. By varying different properties of these nodes and connecting them in the form of a node tree a material can be created. This type of approach is called procedural texture creation and the materials created using this approach can also

be stored as image textures which contains the information of the color, metallic properties, roughness properties and the texture properties of an object in individual images. These type of image textures are called as physics based rendered materials or PBR materials. In blender, materials are considered as data-blocks and they can be applied to one or more objects present in the scene.

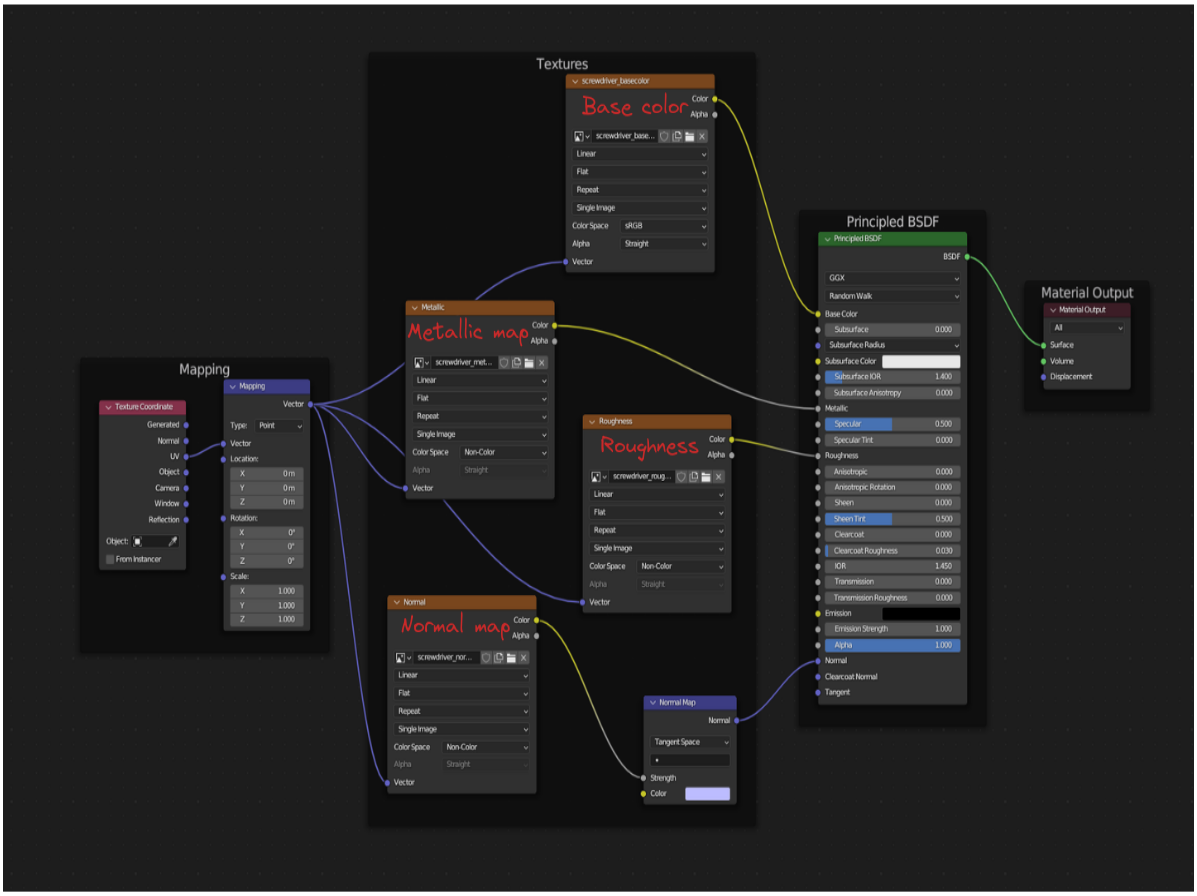


Figure 5.7: Example image of node tree for the material in blender software

– Principled BSDF shader

  * Principled BSDF shader node is one of the most commonly used shader node for creating physics based rendering materials and most of the real world materials can be created by adjusting the parameters of this node. This node defines the properties of the material and how the light interacts with the material i.e reflection and refraction of light on the surface of the object. We will discuss the important parameters 5.8 of this node in detail.
  * The first parameter is the Base color, we can define the color of the object's material in three ways. We can choose the color from either RGB color space or HSV color space. We

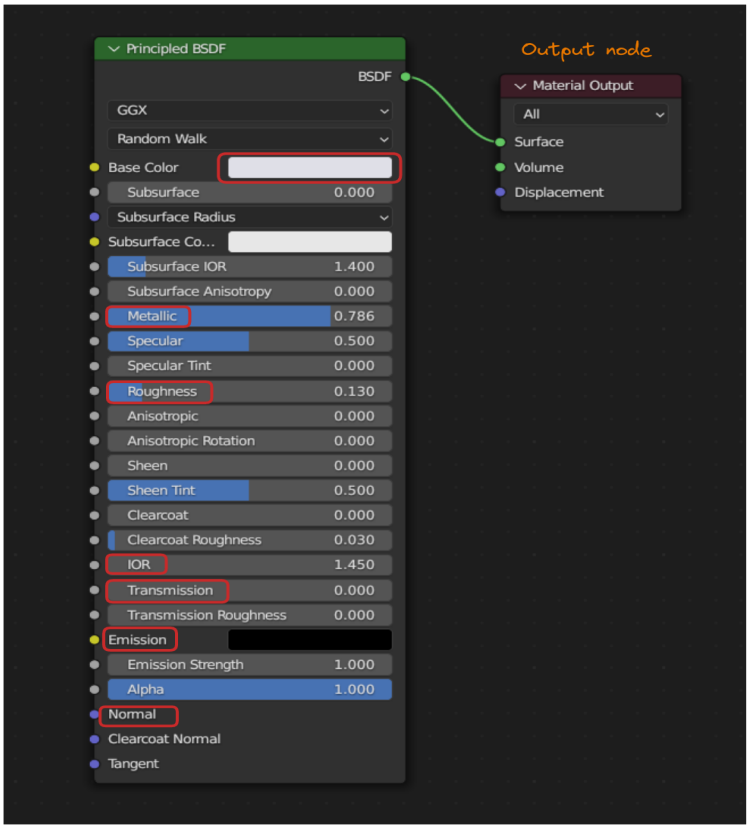


Figure 5.8: Principled BSDF Shader

can also define the color by providing a Hex value which is the color code.

* Metallic value defines if the material is a pure metal or non-metal. This parameter also determines the reflection of light on the surface of the material. If the metallic value is set to one then the material will be considered as a pure metal and thus it has complete reflection.
* Roughness value determines the refraction of light on the object's surface i.e if the value of roughness is set to zero then the surface appears to be smooth and the refraction of light is minimum. If the roughness value is set to one then the surface of the material will appear to be rough and the refraction of light will be high.
* Transmission value determine if the material is a glass type material. If the transmission value is increased to one then the material appears to be a glass material. If the value is low then the material will not look like a shiny glass material. This can also be done using the Glass shader node but there you cannot control the metalic and roughness properties directly.
* IOR value corresponds to Refraction of Index value. Each materials like, water, plastic,

glass etc, have a differnt IOR value and these values can be provided to make the material look more realistic.

* Emission property of this material allows to emit light from an object or only from a part of the object. Both the color and intensity of light to emit can be controlled in this property. This property can also be used to create lights in the scene. Also, the same effect can be achieved by using Emission Shader Node alone.
* Normal value determines the height or bumps in the texture. Not all real world materials will be smooth. They will have some imperfections like groves, bumps, noisy texture, scratches etc, all such data can be represented by this normal property. Although for creating such scratches, ridges or other texture details we need to use Noise texture node, Muskgrave texture node, Wave texture node or other combination of texture nodes and a normal map node and connect the output value of normal map node to the normal property of principled BSDF node.

– Image texture node

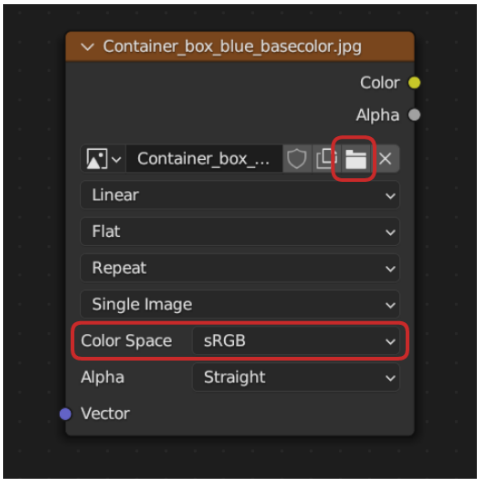


Figure 5.9: Image texture node

* Image texture nodes can be used to import the existing materials which are present in the form of images. These textures are created by a process called Texture baking which we will discuss in the following section. In general the image textures are generated after creating a material procedurally using principled bsdf and other shader nodes and then these materials are baked into individual images which contain the basecolor, metallic, roughness, and normal information separately.
* Different image texture nodes can be used to import the different images containing the information of a material. One important parameter in this node is the color space. Since the base color represents the actual color of the material we use sRGB space and for the

metallic, roughness, and normal images we change this color space parameter to Non-Color. This is because the metallic, roughness and normal images do not represent the color information.

- Texture coordinate and Mapping nodes
  - Texture coordinate node is generally used as a vector input node for vector textures. It has different methods like, generated, normal, UV, object, camera, window and reflection as shown in the fig 5.10. The most commonly used methods are UV and object parameters. This node is often connected to a mapping node which controls the location of these coordinates. The UV parameter uses the textures coordinates directly from the active UV map created in the UV editor. By adjusting the parameters of the mapping node the position, orientation and scale of the texture on the surface of the object can be modified for obtaining the desired output. These two nodes are often used in procedural texture creation and texture baking process. The output vector value from the mapping node is then used in combination of other shader nodes like color ramp, noise texture, image texture etc to provide information to the principled BSDF shader.

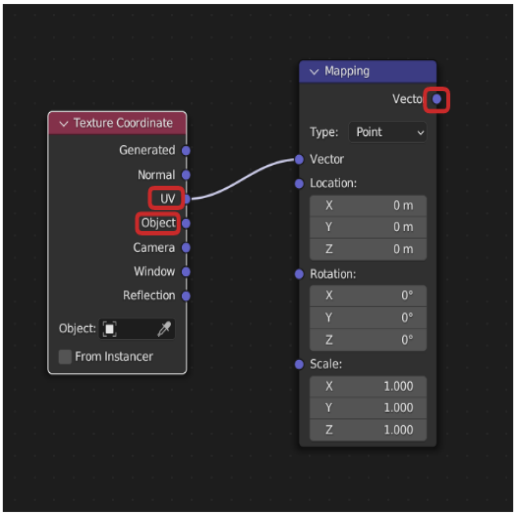


Figure 5.10: Texture coordinate and mapping nodes

- Texture baking
  - The materials created in blender can be exported along with the objects in different formats like, .obj, .fbx etc, which can be again imported into other gaming engines or 3D modeling software. These formats allow to store the material color or appearance alone and properties like, roughness, metallic, lighting etc cannot be transferred. Texture baking process helps to store these kind of information in the form of images which can be later used in the form of image textures. This process is widely used for creating the background textures and also for

the objects present in the scene. Texture baking process is done separately for each object and the important steps for baking textures are discussed in this section.

With the help of texture baking we can store different information like the color, metallic, roughness, normal, lighting, emission etc, in the form of images as shown in the figure 5.12. These kind of image textures can be created only in cycles render engine as it is a ray tracing engine and it calculates the data based on the light paths.

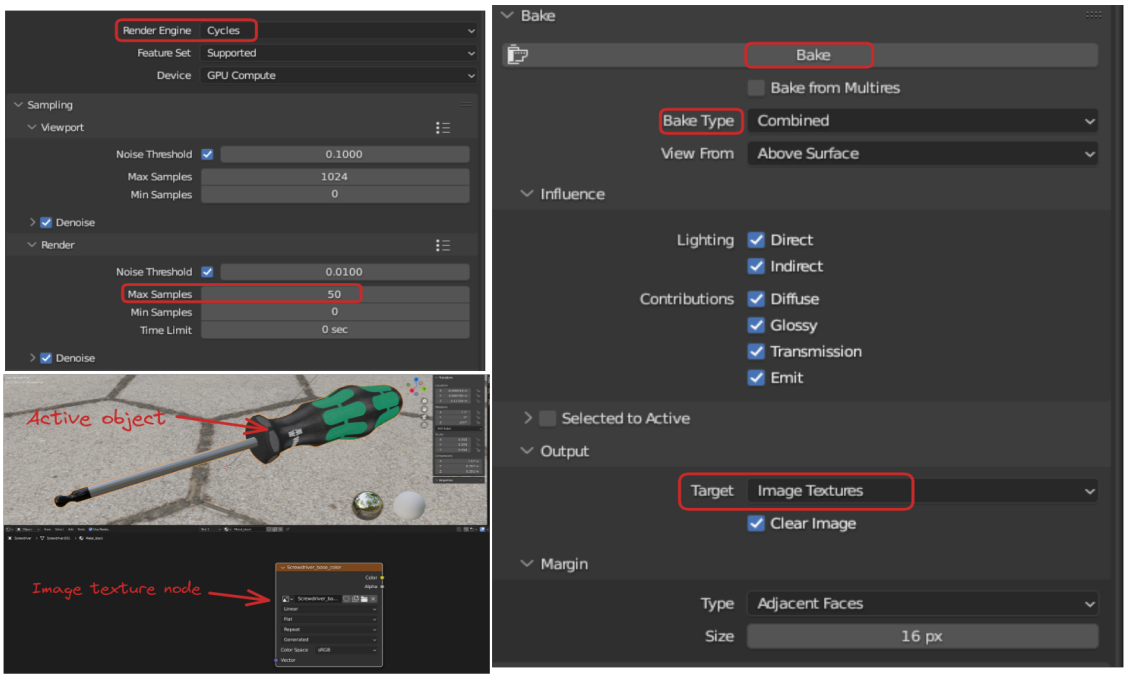


Figure 5.11: Texture baking

– Since we are interested to bake the textures of objects, first we need to select the object and uv unwrap the object. This is very important because all the texture baking process happens based on the uv coordinate maps. Hence before baking the textures, all objects have to be uv unwraped. Then inside the material node editor we need to create a image texture node with a new image. Then both the object and the image texture node has to be selected and the bake option is used for baking the corresponding properties as shown in the figure 5.11. The main important parameters which effects the time for baking the textures are the image resolution and the max number of render samples. For example, to create an image of the object's color or appearance in the scene we need to create a new image with name basecolor and then in the baking properties we set the baking type to diffuse and proceed for baking. Similarly, for roughness we create a new image in the image texture node with the name roughness map and then we set the bake type to roughness. For normal map we create another image with name normal map and set the bake type to normal. For the emissive properties and metallic

properties we create a new image with name metallic map and set the baking type to emssion. Finally, we can also bake the effect of light falling the objects from different light sources present in the scene by creating a new image with name ambient occlusion and setting the bake type to ambient occlusion.

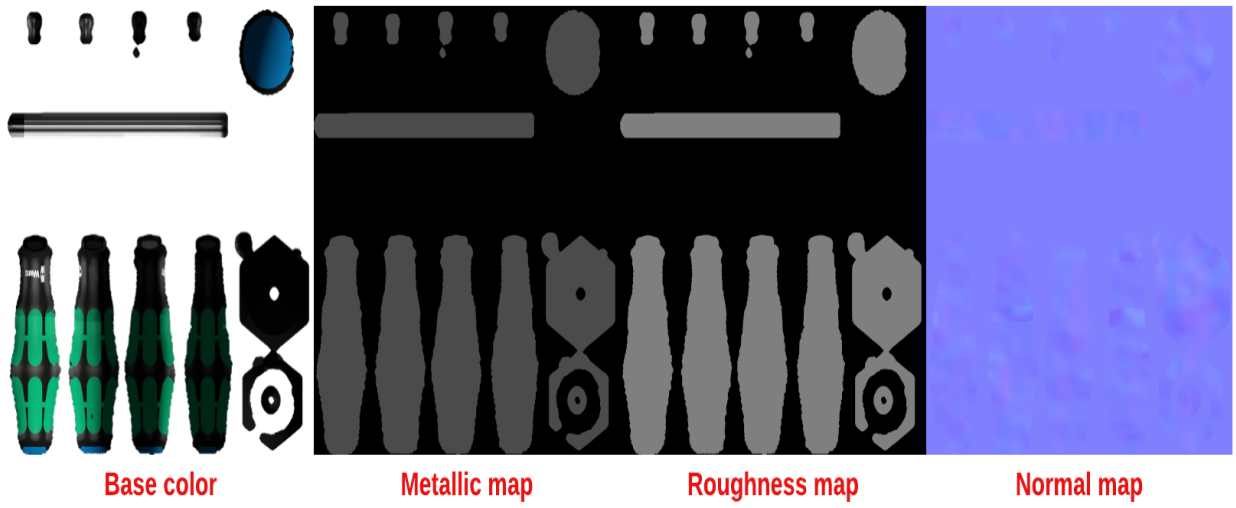


Figure 5.12: Images of basecolor, metallic map, roughness map and normal map of screwdriver object after baking the textures

5.1.3 Camera

- Pose
  - Similar to all other objects, cameras in blender are also considered as objects and the location and rotation of this camera object can be controlled by providing the x,y,z values as shown in the fig 5.13. Since controlling the camera in 3D space and keep the objects inside the camera's frame is difficult, we need to constraint the camera depending on the application. There are many ways to do this, the technique used in this project will be discussed in the later section.
- Lens Type
  - Blender offers three types of lenses: Perspective, Orthographic, and Panoramic. The Orthographic lens does not change the size of objects based on their distance from the camera, which means that objects will appear the same size regardless of their distance. This makes it less suitable for creating variations in distance. The Panoramic lens is also not well suited for generating deep learning datasets. In contrast, the Perspective lens allows for the capture of

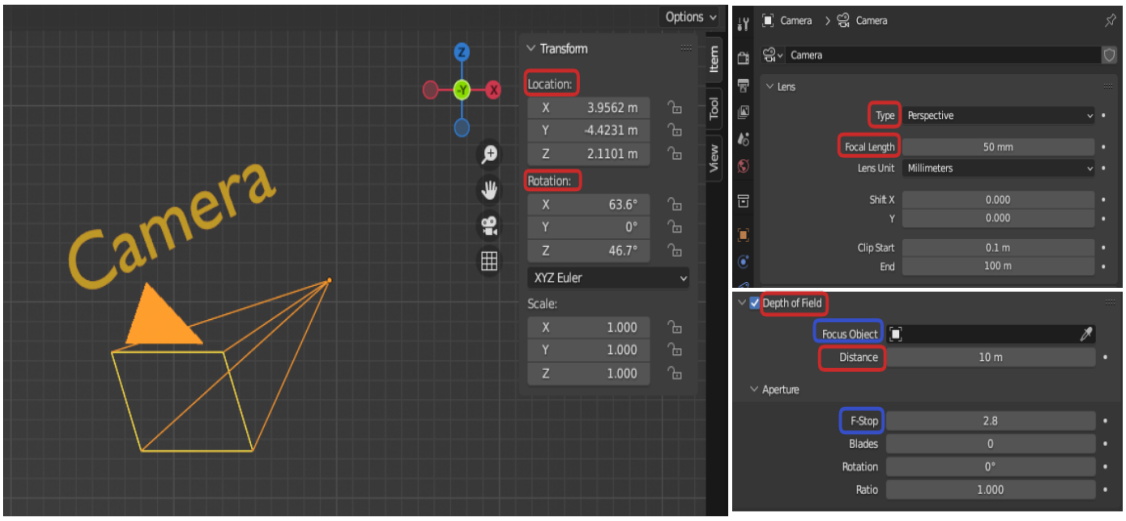


Figure 5.13: Camera settings

distance, such that objects that are farther away appear smaller, and objects that are closer to the camera appear larger. For this reason, we have used the Perspective lens 5.13for generating images for all of the datasets in this project.

- Focal length
  - Depending on the pose of the camera focal length parameter shown in the figure 5.13 can be used to represent zoom in and zoom out. Also this zoom in and zoom out effect can be obtained by changing the position of the camera. Adjusting the position of the camera in 3D space requires little more constraints and thus in this work we have used focal length parameter to create zoom in and zoom out effects.
- Depth of field
  - In real-world cameras, light passes through a lens before being bent and focused onto the sensor. As a result, objects that are closer than a certain distance appear blurry, while objects farther away appear sharp. The focal point, which determines the region that is in focus, can be set either precisely or based on the distance between the camera and a particular object. There are two ways to achieve this blur effect in Blender. The first method is to set a focus object and adjust the value of the aperture's F-stop 5.13, which determines the amount of blurring. The second method is to directly change the "Distance" parameter of the "Depth of Field" effect, as shown in the figure 5.13. In both cases, the numeric values to use depend on the location of the camera. Lower values create a stronger blur effect, while higher values result in a sharper appearance. In this project, we used the second method, which is to specify the distance directly, to create blurry images.

- Resolution

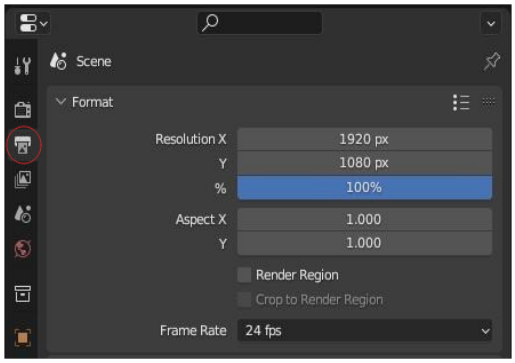


Figure 5.14: Resolution settings

  - The resolution of the rendered image can be changed by adjusting the (x,y) resolution parameters in the "Output Properties" section. The resolution is one of the factors that affects the rendering time of the image. If the resolution is high, the rendering time will be longer. By default, the scene visible in the camera view is rendered, but you can also use the "Render Region" option to render only a part of the scene. Whether or not to use this option depends on the specific application and the user's preferences. In this case, we have used a (96 x 96) resolution to reduce rendering time, considering the available hardware capabilities.

5.1.4 Lights

- Pose
  - Similar to other objects, lights are also considered as an object in blender and their position and orientation affects the quality of the final rendered image. This position and orientation can be controlled by changing the values of location and rotation as shown in the fig 5.15
- Types
  - There are two ways to create lights in Blender. The first method is to create a light object directly, which has four variations: Point, Sun, Area, and Spot. The choice of which variation to use depends on the specific application, and they can also be used in combination to illuminate the scene. The second method is to create a mesh object, such as a plane, sphere, cube, or other shape, and apply an emission shader to it. This approach is often used when certain parts of an object should emit light. The position and orientation of the light object determines the amount of light present in the final render. For example, if the light source is very close to the 3D object, the intensity will be high and the rendered image will be very bright. However, if the same intensity of light is moved farther away from the object, the final rendered image

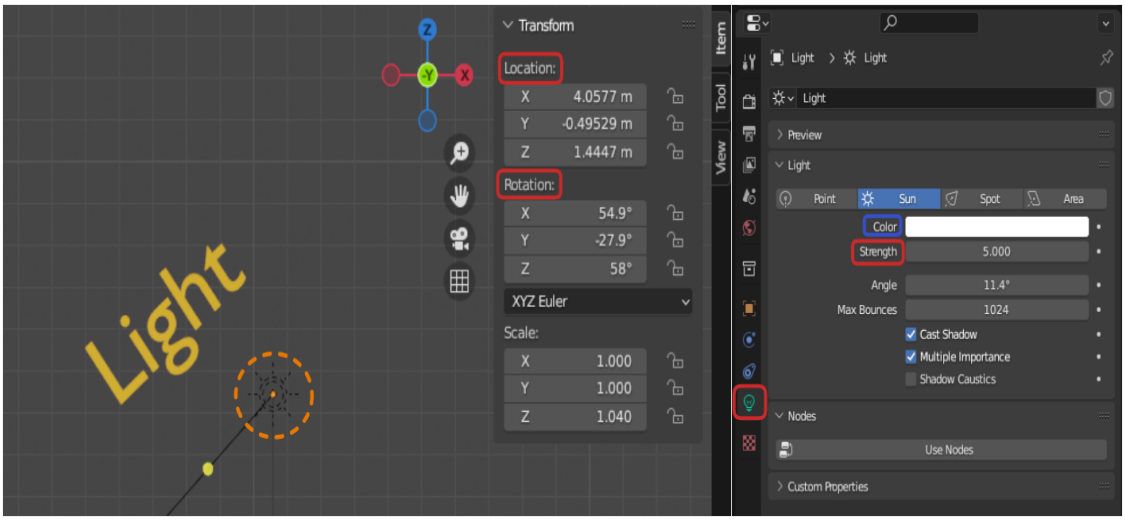


Figure 5.15: Light settings

will appear dark. The color of the emitted light can also be controlled in the RGB color space to produce different colors of light. We can also control the amount of light present in the scene by fixing the position of the light and adjusting its strength value. By enabling the ”Cast Shadows” option, we can show the shadow of an object based on the intensity and angle of the light source. 5.15

5.1.5 World/Background

- There are two ways to create a background for a scene in Blender. One way is to use HDRI (High Dynamic Range Image) backgrounds and studio lights to replicate the real-world environment using images. This method does not require any geometry or mesh objects and is mainly used for videos and animations. The realism of the render depends on the scene setup, camera size and angle. The other way is to create a mesh object, like a plane or cube, that covers the entire scene and use procedural or image textures on it to create the illusion of a background world. This method is useful for small scene setups and still images and gives the user more control over lighting, allowing them to set it up as needed and potentially reduce rendering times. To use this method, the user can turn off the HDRI background and use the scene lighting and world options as show in the figure 5.16. In this example, a background shader node was used for the world option and the color was set to gray to remove any lighting or color from the world. A plane was then created and image textures were used to create the background for the scene.

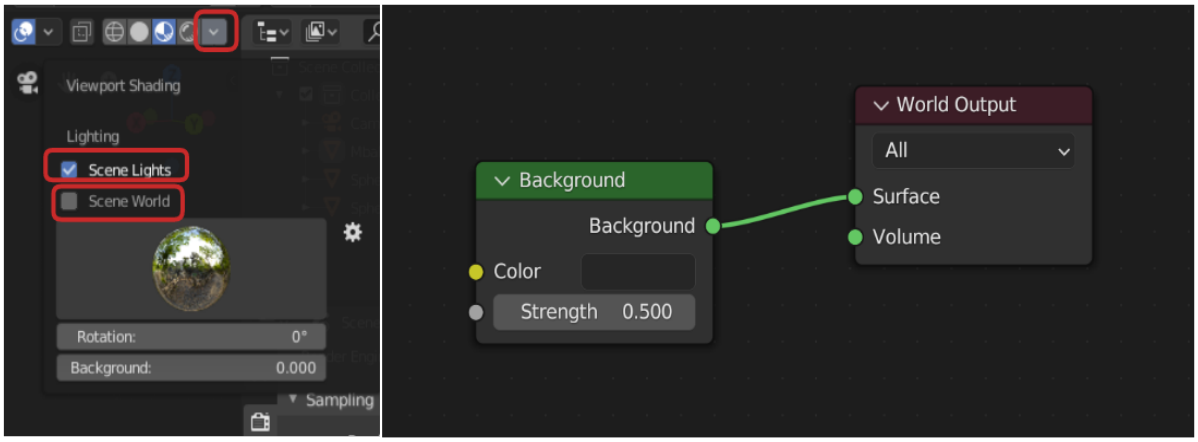


Figure 5.16: World settings

5.1.6 Constraints

- Constraints can be used to control an object's properties, such as its position, rotation, and size. These constraints can be applied to a single object by using its inherent values, or they can be based on the properties of other objects, known as the target. This allows us to control an object's behavior in relation to other objects in the scene, giving more flexibility in creating animations and other visual effects. The figure 5.17, illustrates the constraints used in this project. Constraints are often used in animation to make the movement of one object depend on the movement of another object. The choice of constraints to use depends on the specific application, and in this case, we have used the "Follow Path," "Track To," "Limit Location," and "Limit Rotation" constraints. These constraints allow us to control the movement and orientation of the objects in the scene to create the desired output. We use the "Follow Path" constraint on the camera object and the light object to make them move along a specific curve or path. We have also applied the "Track To" constraint to the camera object and the light object, with the target object representing the object that the camera and light should track. When this constraint is applied, the camera and the light source will always orient themselves towards the target object. This ensures that the objects are always included in the frame of the rendered image.

- We apply "Limit Location" and "Limit Rotation" constraints to various objects in the scene to keep them within the boundaries of the scene. As we are working in 3D space and creating datasets, it is important to restrict the positions and orientations of the objects to prevent them from overlapping or going below the background plane during rendering. These constraints ensure that the objects stay in their designated locations and orientations within the scene.

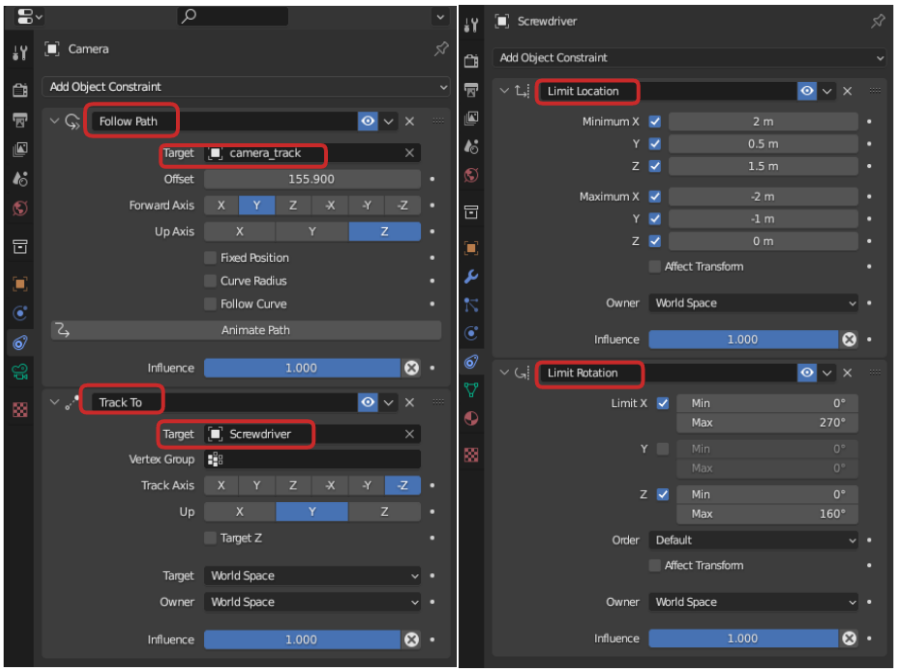


Figure 5.17: Blender constraints

5.2 Scene Setup

In this work, we have used the features and customizable settings of Blender 5.1 , to create a basic scene that can be useful for creating deep learning datasets for various test situations. By utilizing the capabilities of Blender, we were able to create a scene that can be modified to fit different test scenarios like, lighting, distance, textures, deformation, blur,distractor etc,. Our scene consists of a background plane, two curved circles (made using a Nurbs curve), a camera, and a light source, as depicted in the figure. 5.18 . For the camera object, a ”Follow path” constraint 5.1.6 is applied and the one of the circle is used as a path so that the camera will always move in this path alone. Similarly, for the light object a ”Follow path” constraint is applied and the second circle is used as a path. We name these two circles as camera track and light track as shown in the figure 5.18.

For generating a deep learning dataset, numerous amount of images have to be rendered and it is important to make sure that the objects are present in the rendered frame. So we have used a ”Track to” constraint for the camera object, so that the camera will align itself to the target object by which we can ensure that the object is in the render frame. The target objects are the CAD models corresponding to the classes in deep learning dataset. In the case of generating a classification dataset each class will be considered as a target object. Depending on the test constraint, if required then we turnoff the track to constraint and change the position and orientation of the object to introduce variation to the dataset. All these modifications are done using the python based api of blender(Bpy). While changing the position and orientation of the objects, and still ensuring that the objects are present in the render frame, we use

"Limit location" and "Limit rotation" constraints for the objects separately.

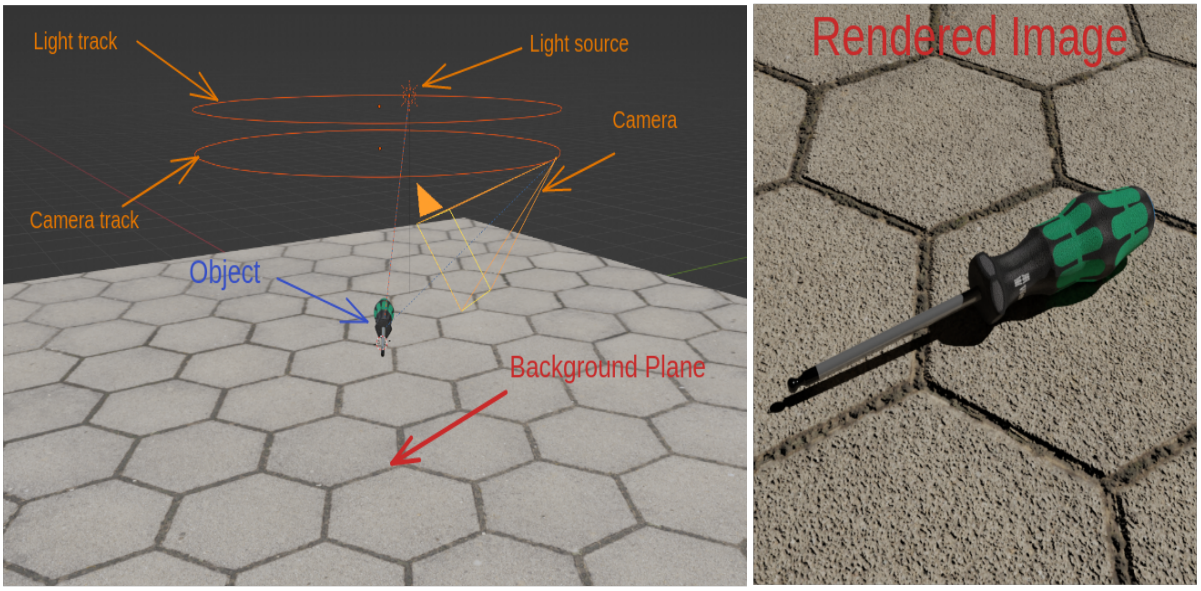


Figure 5.18: Scene setup

For the background, we have created a plane object and used a image texture node for creating the material of the object based on image textures. Depending on the test constraint the image of the background will be changed and thus creating a new background in the rendered image. For all the objects present in the scene, the origin point is set to the center of the object so that the position and orientation of the objects will be around the object's axis and not about the world origin. This helps in rotating the objects with out changing the position. If the rotation is about the world origin, then position of the object will also be changed and they may not be visible in the render frame.

5.3 Synthetic datasets

This work mainly focuses on generating synthetic datasets for the classification tasks in deep learning. By adjusting and varying the parameters discussed in the section modifiable parameters 5.1 , we generate classification datasets in different test constraints like, lighting, distance, textures,deformation, distractor, blur etc,. For creating the dataset, we have designed the 3D cad models of the 15 classes of RoboCup@work components present in figure 5.19a and applied appropriate procedural textures so that the representation of the objects will be close to the real world. Figure 5.19b shows the 3D cad models of 15 classes of Robocup @work components. These models were then used to generate different synthetic datasets using the capabilities blender's python based api (Bpy).

All the objects are placed at the global origin in the scene setup 5.2 and initially all objects are set to hidden mode while rendering. At this point if we render an image no object will be present in the image. Later in the code while generating the dataset,we will consider one object at a time and make it visible

(a) RoboCup@work objects (b) CAD models of robocup @work components

Figure 5.19: Figure representing real world objects and blender designed objects for synthetic datasets.

in the rendered images and generate images for one class. After generating the images for one class, we hide the present object and make the next object visible and continue the process of rendering.This loop continues for all the classes/objects present in the scene. Due to the hardware capabilities and rendering time, we have images with a resolution of (96 x 96) in all the training and test datasets.

5.3.1 Normal training dataset

As discussed in the chapter methodology 4, we have created a dataset for training by considering the parameters in normal range such the the objects in the images appear in good conditions. Based on the scene setup, for normal lighting, we have defined a range of (lx,ly) and a random value is sampled for the strength of the light source while rendering each image. Where lx is the minimum allowable value and ly is the maximum allowable value. We do this to introduce a little variation in the with in the normal conditions. We discuss more about this range of lighting in the following section of lighting constraint 5.3.2.

Similarly based on the scene setup, the location and orientation of the camera is fixed to the camera track path. Hence we use the focal length parameter of the camera to introduce the variation in the distance of the objects. For the normal training dataset, we have defined a range of (dx,dy) as normal range and a random value will be sampled from this range for the focal length parameter. Where dx is the minimum allowable value and dy is the maximum allowable value. The random value sampled from this range produces images in which objects are clearly visible in normal conditions. We discuss more

about the range of distance in the following section of distance constraint 5.3.3.

All the objects present in the scene are randomly rotated so that there will be some variation in the object's pose. All the orientations takes place locally with respect to the axis of the object and not with respect to the global origin. This kind of rotation is considered because of the scene setup in which we have created as a background plane with objects on the top. Here if the objects are rotated with respect to the global origin then they take large rotation angles which results in a scenario where the objects go behind the background plane and will not be visible in the rendered images.

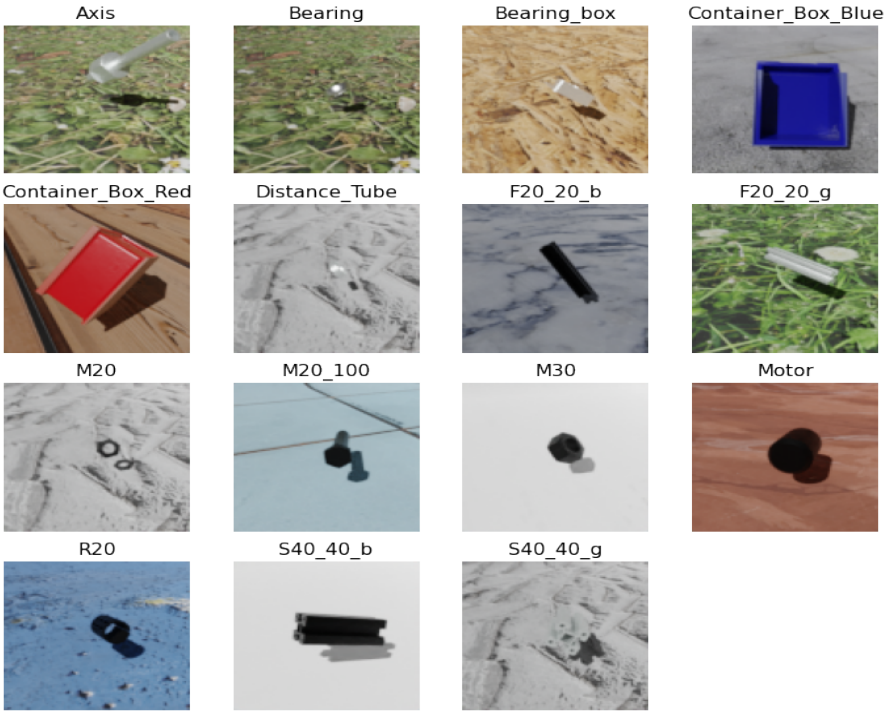


Figure 5.20: Traing dataset with normal conditions

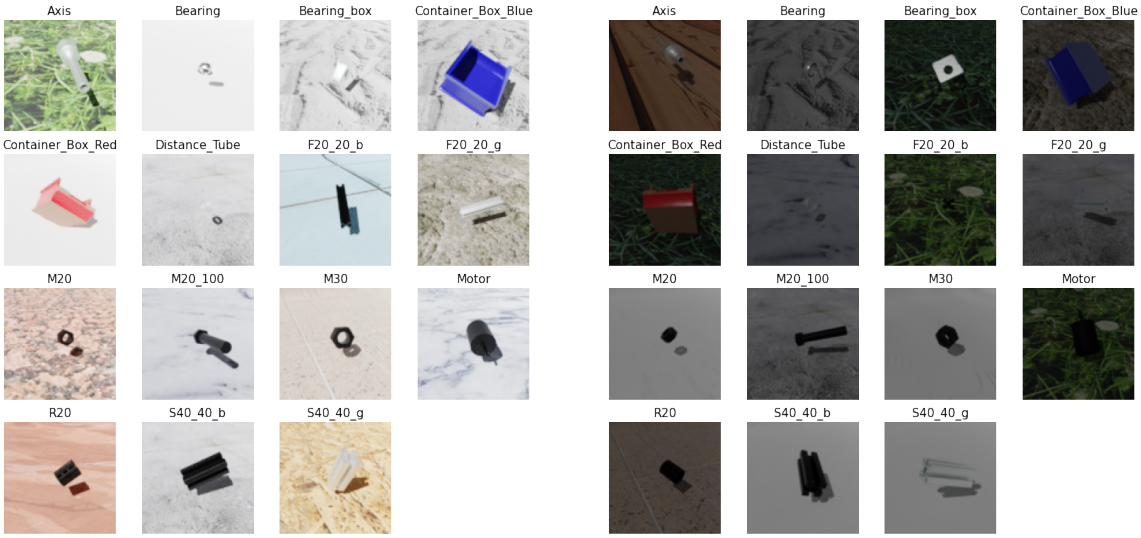
For the better performance of the model, we have defined a set of images belonging to a particular class (A) for the background textures. A random image is then sampled from class A and used an image texture for the background plane. Different background textures belonging to class A are depicted in the figure 5.20. Here we have defined a class of textures (A) so that we can create a test constraint for the textures in which we have images belonging to class B. We will discuss more about these classes in the section textures constraint 5.3.4 .

Finally, using the above discussed parameters we have generated a dataset under normal conditions in

which each class contains 1000 images. Hence for the robocup @work objects we have created a normal conditions dataset of 15000 images. This dataset is used for training different deep learning models. The models trained on normal conditions are then used for testing in different test constraints.

5.3.2 Dataset for testing the impact of lighting

The primary goal of creating this type of dataset is to test the performance of different deep learning models in different lighting conditions such as bright, normal and dark. Here the datasets are created as 3 different classification datasets with images and class labels in two scenarios of lighting (bright and dark). To achieve this the lighting setup discussed in the section scene setup 5.2 is used. Among the available light types of Point, Area, Spot and Sun we have considered the light type of Sun. The default range of the intensity of light (Sun object) in blender is (0-10). Here the value 0 produces completely dark image and the values above 10 produces completely bright images.



(a) Bright lighting constraint

(b) Dark lighting constraint

Figure 5.21: Datasets for lighitng constraints

As discussed in the section 5.3.1, we have defined the normal lighting range as $(lx, ly) = (3, 7)$. For generating images for dark lighting constraint we have defined the light range as $(lx_1, ly_1) = (0 - 2)$. Similarly, for bright lighting constraint, we have defined the range as, $(lx_2, ly_2) = (15, 20)$. With in these ranges a random value is choosen for each image while rendering. If the value of the light in blender is less than or equal to 2 then the lighting is considered as dark. If the light value is greater than or equal to

15 then the lighting is considered as bright. The values between 3 and 7 is considered as normal lighting region. The threshold here is considered by observing the scene in different light values and the value at which the images render as dark choosen as min threshold and values at which images appear bright choosen as max threshold.

$$lx_1 < lx < lx_2$$

$$ly_1 < ly < ly_2$$

- $(lx, ly) = Normal\ lighting\ range$
- $(lx_1, ly_1) = Dark\ lighting\ range$
- $(lx_2, ly_2) = Bright\ lighting\ range$

In this approach the dataset generated under normal conditions as shown in the figure 5.20 is used for training and the datasets generated in the bright constraint 5.21a and dark constraint 5.21b are used for testing. Both the dark and bright lighting datasets contain 100 images for each class i.e, bright constraint corresponds to 1500 images and dark constraint corresponds to 1500 images.

5.3.3 Dataset for testing the impact of distance

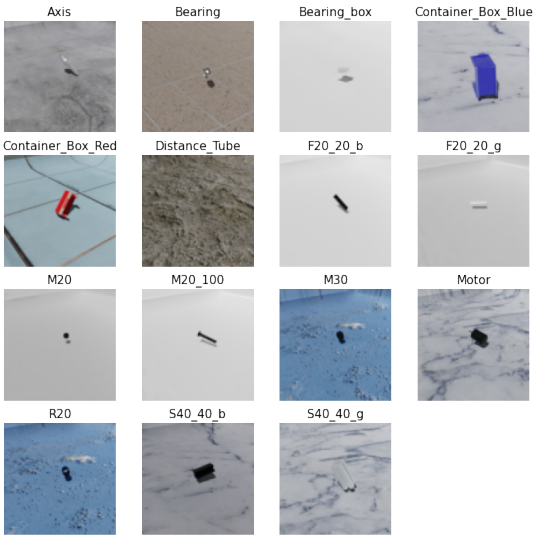
For testing the performance of different uncertainty estimation methods, under distance constraint, we have generated two datasets with near and far constraints. As discussed in the section 5.3.1, we have defined the normal distance range as $(dx, dy) = (50, 80)$. for the focal length parameter of the camera. By modifying the range of focal length we have created variations in distance and define a threshold region $(dx_1, dy_1) = (20, 30)$ for far distance constraint and a range of $(dxx_2, dy_2) = (115, 140)$. The ranges for all the three constraints, far,normal and near constraints is considered based on the size of the objects present in the scene. From the defined ranges a random value is sampled for the focal length value for the corresponding constraints and the images are renderd. The main idea in the distance constraint is to train the deep learning models on normal distance and test them with the images in which objects appear much far to the camera and very close to the camera. Some of the example images for the far constraint and near constraint are depicted in the figure 5.22 . For the test datasets we have generated 1500 images for each constraint (far and near) in which each class corresponds to 100 test images.

$$dx_1 < dx < dx_2$$

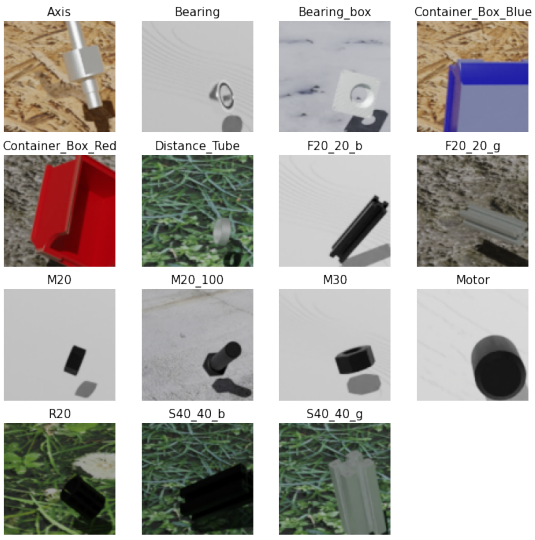
$$dy_1 < dy < dy_2$$

- $(dx, dy) = Normal\ distance\ range$

- $(dx_1, dy_1) = Far\ distance\ range$
- $(dx_2, dy_2) = Near\ distance\ range$



(a) Far distance constraint



(b) Near distance constraint

Figure 5.22: Datasets for distance constraints

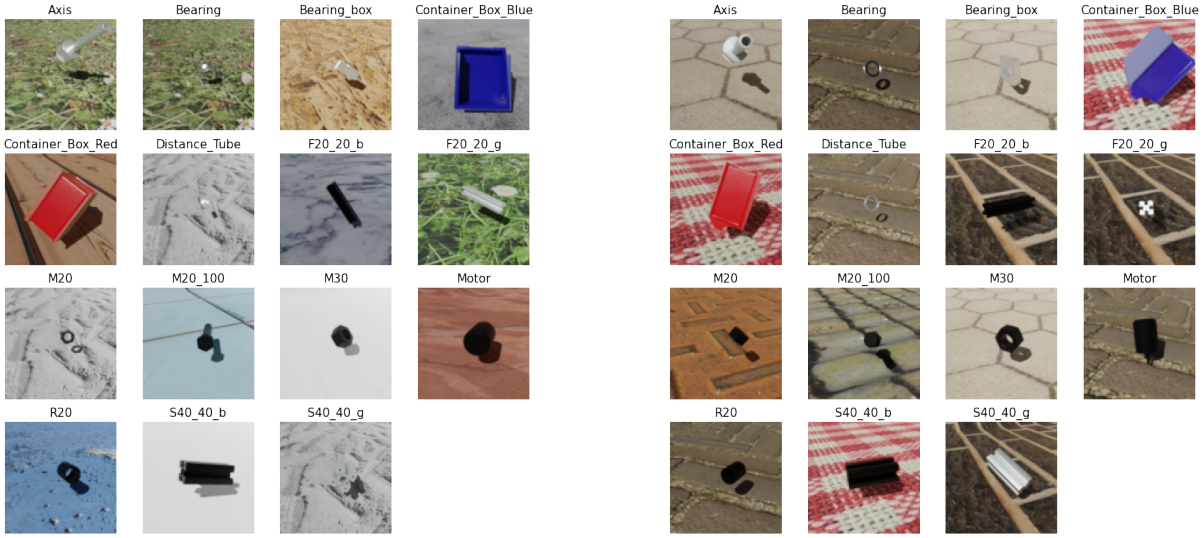
5.3.4 Dataset for testing the impact of background textures

The dataset for this test constraint is generated to check the impact of unseen textures on the deep learning models. As discussed in the section normal training dataset 5.3.1 , initially we generate a normal training dataset in which the background image textures belong to a particular class A. Then for the textures constraint, we generate a test dataset in which the image textures belong to class B. The images present in class B are not present in class A and thus they are not seen by the DNN models during training. Similar to the previous test constraints, we have generated 1500 images for the textures constraint dataset.

$$(Training\_textures \in A)$$

$$(Constraint\_textures \in B, B \notin A)$$

The figure 5.23 depicts the different textures used for normal and texture constraint data sets.



(a) Normal training textures

(b) Constraint textures

Figure 5.23: Datasets for Texture constraints

5.3.5 Dataset for testing the impact of blur images

In real world scenarios, the focus of the camera may not be adjusted properly and in some cases we may get some blurry images. So for testing the impact of blur images we have created a constraint for blur using the depth of field parameter of the camera. The DNN models are trained on normal conditions in which the objects in the images appear clearly and then we test with the blur constraint dataset in which the objects are blurred and not visible properly. Some of the example images of blur constraint dataset are depicted in the figure 5.24 . Since we have already restricted the position and orientation of camera using Follow path and Track To constraints, we just adjust the value of distance in the depth of field parameter. Based on the scene setup we have observed that the distance value of 0 produces completely blur images in which objects are not identifiable and a value of 2 produces less blurry images. Hence we have used a range $(bx, by) = (0, 2)$ for the blur constraint. Where bx is the minimum allowable value and by is the maximum allowable value for the producing the blur images.

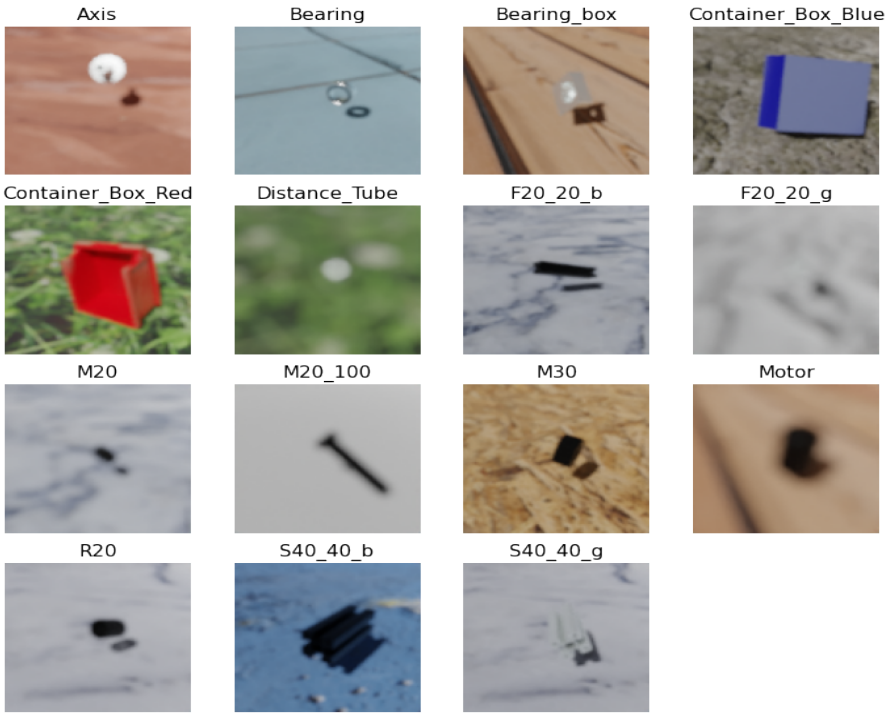


Figure 5.24: Dataset for Blur constraint

5.3.6 Dataset for testing the impact of object's deformation

Generally while training a deep learning model, users collect the dataset in which all the objects will be in good shape and size. But in real world scenarios the objects might get deformed due to several external factors and in some of the cases the DNN models may fail to classify the deformed objects. Hence for testing the performance of different deep learning models under the deformation of objects, we have created a constraint for the deformation. For generating deformation dataset, we have used simple deform modifier of blender software for all the objects present in the scene. The parameters bend and twist 5.5 allows the user to bend or twist the object with respect to x,y,z axis. The axis of deformation is selected based on the shape of the object and using the parameters of bend and twist we have deformed the 15 objects of robocup@ work components and generated a 1500 images for the deformation constraint. In this dataset each class corresponds to 100 test images in which objects are deformed. The figure 5.25 depicts some of the example images generated for deformation constraint.

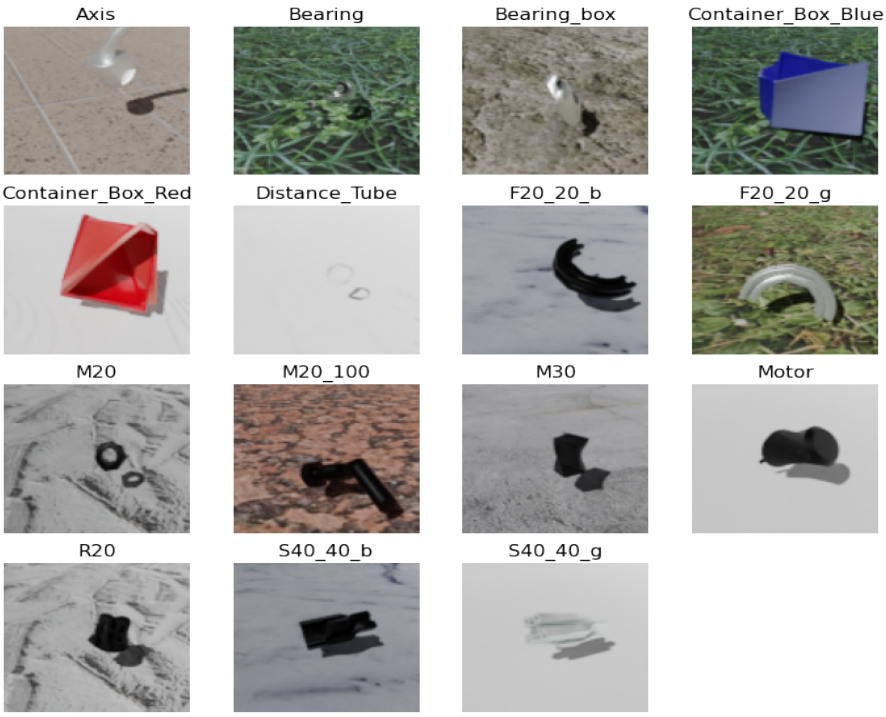


Figure 5.25: Dataset for Deformation constraint

5.3.7 Dataset for testing the impact of distractor objects

Distractor objects in a classification dataset are objects that do not belong to any of the classes being considered for classification task. They are included in the dataset to make the classification task more challenging and to better evaluate the performance of the classifier. Hence in this work, we have considered the state-of-the-art YCB dataset models as distractor objects for robocup @work components. 15 YCB models which are not similar to the objects present in @work components are choosen and a random ycb object is selected and placed in the scene such that the rendered image contains both robocup@work component and YCB object. This helps to ensure that the classifier is not simply memorizing the training data, but rather is able to generalize and correctly classify new, unseen data. Some of the test images generated for distractor constraint are depicted in the figure 5.26 . This dataset is again generated under normal conditions along with ycb objects.

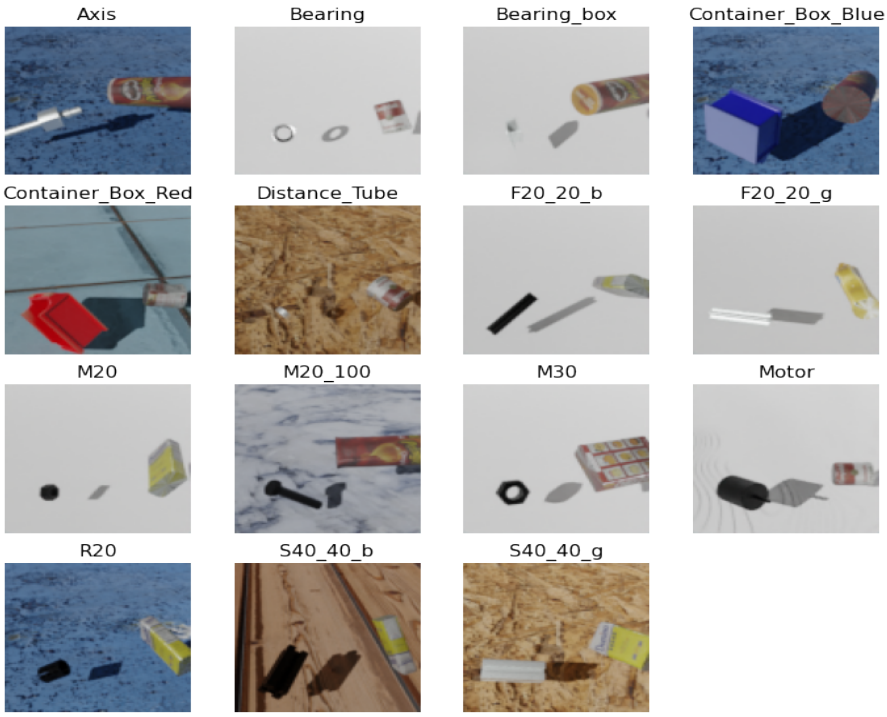


Figure 5.26: Dataset for Distractor constraint

5.3.8 Dataset for training on all variations

After checking the performance of the DNN models in all constraint datasets like lighting, distance, blur, textures, deformation and distractor objects, we then combine different constraints like lighting, distance and blur and generate a training dataset. We we consider the constraints of textures then the model would be learning only the corresponding textures. Hence we exclude the textures constraint while generating the all variations dataset. Similarly deformation and distractor constraints have some ambiguity and they may confuse the DNN models in the classification task. Hence we also exclude the distractor and deformation constraints. Only bright, dark, far, near, blur and normal constraints are combined together to form one single training dataset. This dataset contains 1000 images for each class of which atleast 15 to 20 percent of the images corresponds to the lighting,distance and blur constraints. The figure 5.27 depicts some of the rendered images for all variations dataset in which we can observe the images corresponding to normal, bright, dark, far, near and blur constraints.

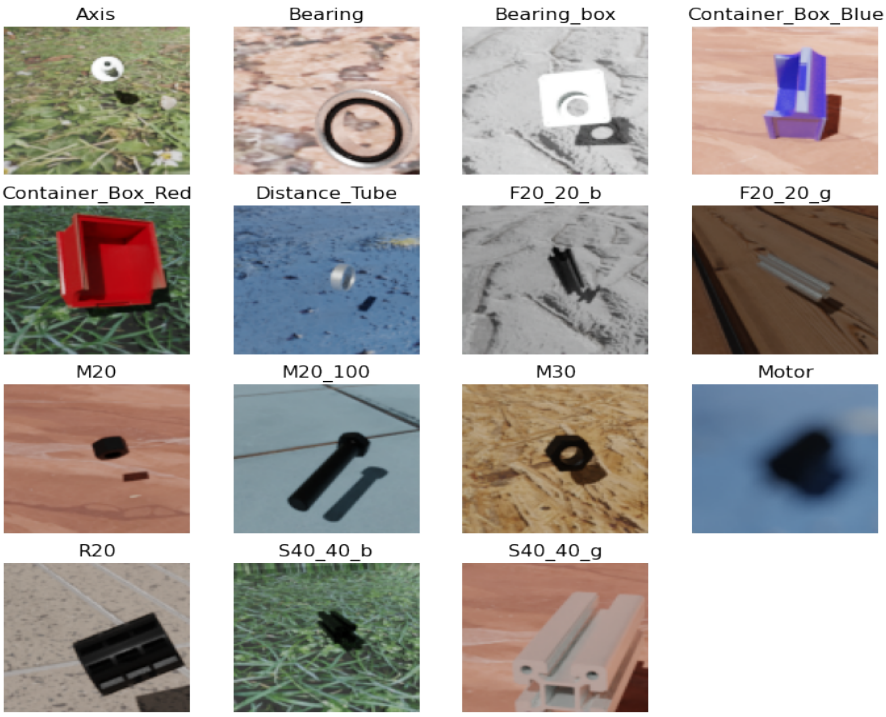


Figure 5.27: Dataset for All constraints

5.4 Computing expected uncertainty value

This section describes the second approach for evaluating the performance of different uncertainty estimation methods discussed in chapter methodology 4 . We begin the approach with a brief introduction to fuzzy inference system and then discuss how to generate dataset using blender. In general the output of a deep learning model is obtained as distribution of values. Each value in the distribution represents the probability of the image belonging to a particular class. The values in this distribution are obtained based on the type of loss function used. A DNN model trained on cross entropy loss and softmax layer produces a probability distribution corresponding to the number of classes present in the dataset. For example the output of a crossentropy model trained on 5 classes can be represented as, $Probability\ distribution = [0.02, 0.1, 0.6, 0.08, 0.2]$. The sum of all the values in this distribution will be equal to one.

Similarly for the models trained on evidential loss function the output distribution is represented as a dirichlet distribution in which the values do not lie in the range of zero to one. The values of

dirichlet distribution starts from one and can reach any value higher and one. An example for the dirchlet distribution of 5 classes can be obtained as, $Dirichlet\ distribution = [1, 1, 1050, 1, 1]$. Unlike probability distribution, the sum of values in dirichlet distribution do not sum up to one. In both the distributions the maximum value is considered as the confidence of the DNN model and corresponding class is considered as the predicted label.

While generating synthetic data in different constraints, we have a precise control of the objects and the rendered images and there will not be any miss labeled data. Hence we can try to model the confidence for the rendering image using fuzzy logic as an expected value even before passing the image through the model. For example, if the expected value from the fuzzy logic is 0.8 for an image then we can use this value as a confidence and distribute the remaining 0.2 uniformly among the other classes and create a multinomial distribution. After passing the image through the DNN model, we get a distributional output and then we can compare weather the distribution of model's output and the distribution based on fuzzy logic are similar or not. But this method has a limitation for various distributions like dirichlet distribution which is not a probability distribution and the values in dirichlet distribution may go beyond hundreds. So, instead of modelling the confidence value for an image we can use the knowledge of entropy from previous methodology and model the entropy for the rendered image using fuzzy logic. In deep learning uncertainty of the predictions of a DNN model can be measured using entropy. The entropy of a certain or high confidence prediction will be low and the entropy for the uncertain predictions will be high. For example, if we consider the distance constraint, we know that as the object goes far away from the camera the uncertainty in the predictions of the model should be high and thus the value of entropy will be high. This information can be used to formulate the rules for the fuzzy inference system and provide the camera's focal length parameter as an input to model the entropy value.

In general a fuzzy inference system(FSI)is a computational system which uses fuzzy logic to make decisions. The fuzzy inference system uses a set of if-then rules to make the decisions, which allows them to mimic the way humans reason and make decisions. A fuzzy inference system consists of four components, fuzzification unit, rule base, inference system and the defuzzification unit.

- **Fuzzification:** The fuzzification unit takes some crisp values as an input and converts them into fuzzy subsets which will be in the range of zero and one. In this case the input value of focal length will be converted to a value between zero and one. The input values are converted into fuzzy sets with the help of membership functions.
  - **Membership function:** In a fuzzy inference system, a membership function is used to map the crisp input data to fuzzy sets. They help to convert the input values to a fuzzy set value i.e value between zero and one.
  - In this case, the fuzzy inference system can contain five membership functions for the distance as far,little far, normal, little near and near. All these membership functions are called as fuzzy subsets and each membership function would define a range of values for the distance variable
  - The degree of membership of a particular distance value (focal length value) in each set would represent a value between 0 and 1.

Also more number of membership functions can be used to produce accurate output values. But with the increase in membership functions the rule base will also increase and so does the computation.

- **Rule base:** Rule base consists of a set of if-then fuzzy rules. These fuzzy rules describe the relationship between the input and output variables. In this case an example fuzzy rule can be defined as, If the focal length value is High then the entropy value shall be High.

- **Inference system:** The inference module uses the fuzzy rules which are defined in the rule base and the fuzzy input data to generate the fuzzy output data. It works by evaluating the rules one by one and combining the results of the individual rules using fuzzy logic operators such as **and**, **or**. The output of the inference system is is a fuzzy set that represents the overall output of the FSI.

- **Defuzzification:** Defuzzification unit converts the fuzzy output of a inference system into crisp output values which have more meaning and can be used in practical applications. Defuzzification uses different methods like, center of gravity, mean of maximum and smallest of maximum etc, to convert the fuzzy output to final crisp output. The method of defuzzification depends on the application and most commonly used defuzification method is the center of gravity method, which calculates the centroid of the fuzzy output set from the inference system. This centroid value is then mapped back to the crisp output value whcih is the entropy.

Let us consider an example to understand the fuzzy inference system for distance and entropy relation in detail. Here we want to define a fuzzy inference system which takes in the distance(focal length) as an input and produces a value for the entropy as output. Both the inputs and the outputs of the fuzzy inference system have a universe of discourse, which defines the range of values for the inputs and outputs. In this case the universe of discourse for the distance can be written as, $distance \in (40, 110)$ and universe of discourse for the entropy can be written as $entropy \in (0, 2.7)$. This universe of discourse is represented by the x-axis in the figure 5.28 . The universe of discourse for entropy is defined based on the number of classes present in the dataset. In this case the maximum value for the entropy for a multinomial distribution of 15 classes is 2.7. A random value for the distance will be obtained from the focal length parameter of blender's camera object, which will used as an input to the fuzzy inference system. The value obtained is in the range of the universe of distance and this universe is defined based on the human visualization of objects being far and near to the camera and based on the scene setup in the bender software.

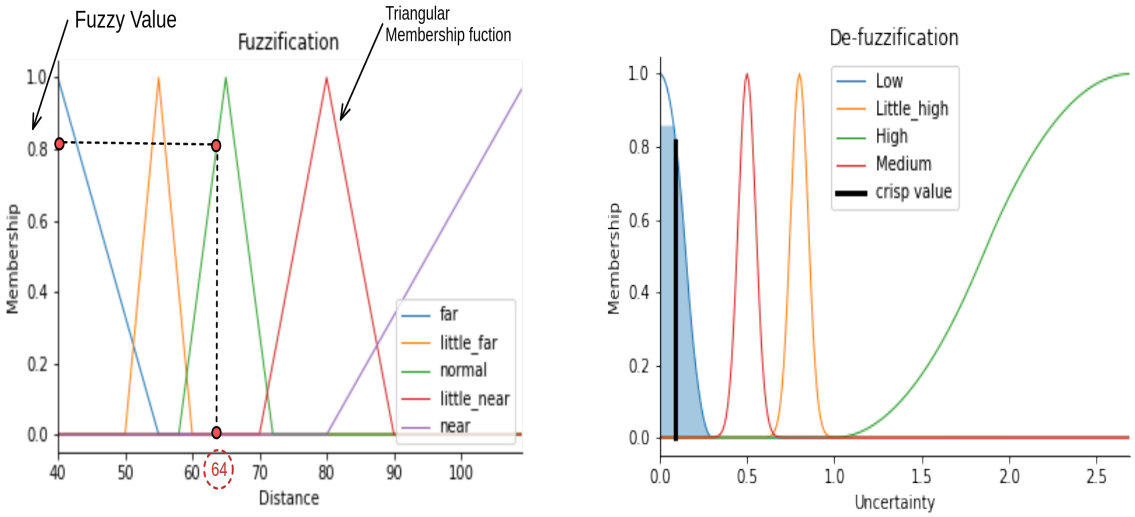
The next step is to convert the input value to a fuzzy set using membership functions and creating the rules. In this work we have chosen a triangular membership function for the distance variable as depicted in the figure 5.28a. Since we need to form fuzzy rules and obtain entropy in different test scenarios like far,normal and near, we have defined a total of five membership functions for the distance as, **far,little far, normal, little near and near**. The ranges of these membership functions are defined based on the human observation of relative position of objects with respect to the camera in scene. Similarly for the entropy we have defined four membership functions which are represented as **low , medium, little_high**

and high in the figure 5.28b. The number of membership functions and their range depends on the application and more the number of membership functions more will be the accuracy and so does the computation.

As part of fuzzy rule base, we need to define the fuzzy rules for the created membership functions such that the inference system can compute a output value.The fuzzy rules defined for the inference system of distance are,

- **Rule-1:** If the distance is Far then entropy is High.
- **Rule-2:** If the distance is Little Far then entropy is Little_high.
- **Rule-3:** If the distance is Normal then entropy is Low.
- **Rule-4:** If the distance is Little Near then entropy is Little_high.
- **Rule-5:** If the distance is Near then entropy is High.

As we can observe in the figure fuzzification of distance 5.28a , an input value of **64** is mapped to a fuzzy input value of **0.82** on the y-axis with the help of corresponding triangular membership function based on rule three. This fuzzy value of **0.82** is used as an input to the entropy on the y-axis of defuzzification and it is mapped to the corresponding Gaussian membership function based on the fuzzy rules. Finally the shaded area depicted in the figure 5.28b , is used for defuzzification and the value is again mapped back to the crisp output value of **0.09** on the entropy universe represented by the x-axis to get the final output of the fuzzy inference system.



(a) Fuzzification of distance

(b) Defuzzification of entropy

Figure 5.28: Fuzzy inference system for distance

5.4.0.1 Challenges

- The accuracy of the fuzzy inference system depends on the number of membership functions used for both input and output and the rules defining the relation between the inputs and output.
- Change in membership function for an input or the output will change the output crip value and choosing the correct membership function for representing the entropy is challenging.
- Since with change in membership functions provide different results for entropy, it challenging to decide to trust the output of the fuzzy inference system.
- For defining the range for a membership function, knowledge of the respective domain is required and it can be different for different domain experts. Similarly defining the range for the low, medium and high entropy can be different and it depends on the number of classes.
- As number of classes increases the universe of discourse for the entropy increases and so does the range for the entropy membership functions.

5.4.1 Expected value dataset

The main aim of creating this dataset is to define a expected value for the uncertainty in the image based on different test constraints like distance,lighting and blur. To achieve this, we use the methodology discussed in the section 5.4 which is based on fuzzy inference system. In this section we will discuss about the dataset generated for distance constraint. Initially we have imported the 3d models of all the required objects along with their materials and textures into the scene created in blender 5.2. Since we are creating the expected values for the entropy only for distance constraint we use a single image texture for the background so that the uncertainty in the textures will not be included to the DNN models. Also, we set the lighting strength of the sun object to be in normal region as our main focus to get expected value for distance alone. If we want to include the lighting constraints like bright and dark then we may need to add more membership functions for the lighting and define the corresponding fuzzy rules as well. The camera object was placed in the scene such that all the objects will be visible in the render frame and all the objects are rotated randomly so that there will be some variations in the dataset. After setting up the camera parameters and light parameters, the focal length parameter for each image is sent as an input to the fuzzy inference system created in the section 5.4 and the expected entropy value is computed. Then finally we save this expected entropy value in a JSON file as an uncertainty label for each image. Thus the final dataset will be a set of images, class labels and uncertainty labels. Finally we have generated the dataset for the distance constraint with 1000 images for each class along with the corresponding uncertainty labels which also can be said as 1000 json files for each class. Thus in this approach we have created a dataset in the form of $[data = (image, class\_label, uncertaitny\_label)]$. An example image of the uncertainty label can be seen in **??** and the rendered images along with their class labels can be seen in the figure 5.29

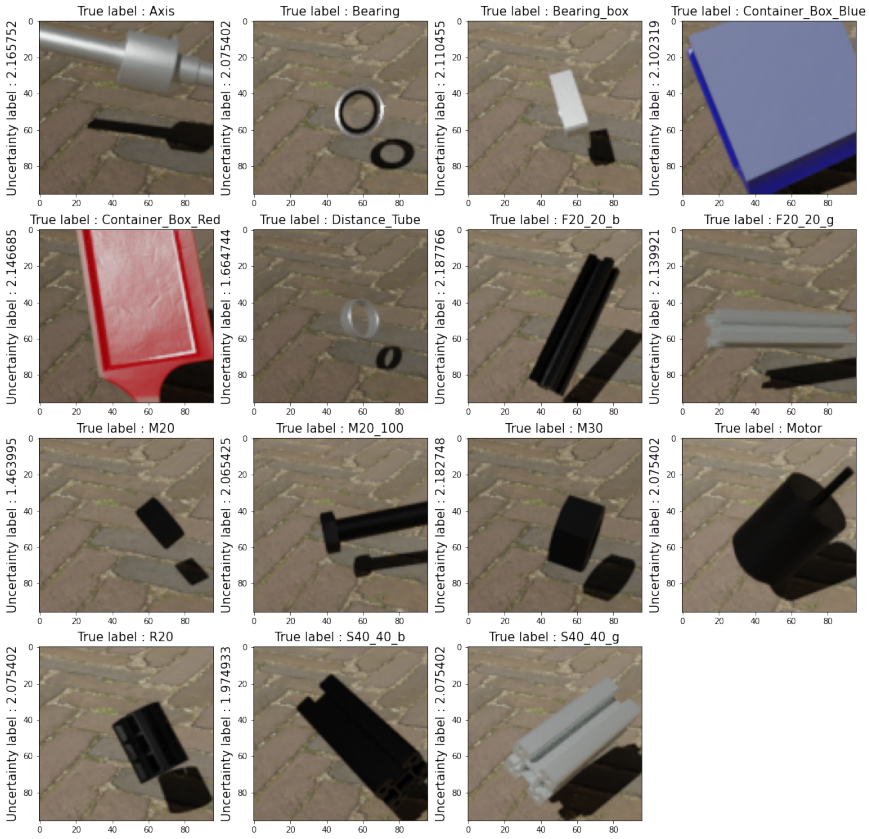


Figure 5.29: Example images in expected value approach


```
{
'Total_num_classes': 15,
'focal_length': 103.0,
'high_threshold': 110,
'image_path': 'data_dir/class_name/image_name.png',
'low_threshold': 40,
'obj_name': 'Container_Box_Blue',
'uncertainty_label': 2.165752
}
```


Figure 5.30: Example of uncertainty label in json format

6

Experimental setup and Evaluation

In this work, we have performed two different experiments to test the performance of deep learning models. The first experiment is using the proposed approach of testing the performance of deep learning models under different constraints like, lighting, distance, blur, deformation, textures and distractor objects. In the second experiment we define the expected uncertainty value using fuzzy inference system and compare the uncertainty of the deep learning model with the fuzzy generated expected uncertainty.

6.1 Experimental Setup

The main objective of all the experiments conducted in this project is to test the performance of the deep learning model in a classification task using different uncertainty estimation techniques discussed in the section 4.3. All the synthetic datasets used in this work are generated using the software blender (version 3.2.0) and its rendering capabilities. All the images are rendered with a resolution of 96 x 96 and 300 samples for each point in cycles render engine. In the following experiments, the dataset generated under normal conditions 5.3.1 , is used for training the deep learning models and different test constraint datasets are used for the evaluation of the performance of the trained DNN models. Resnet18 architecture from the pytorch pretrained models is used as a base model and the model is trained using cross entropy loss and evidential loss using the hyper parameters mentioned in the table below 6.1. A StepLR learning rate schedular is used and the step size of the schedular is adjusted for different models to increase the training performance. All the models are checked for over or underfitting and the models with best weights are used for testing different test constraint data sets.

Hyperparameters	
Model	ResNet18
Number of epochs	20
Batch size	64
Learning Rate	1e-3
Weight Decay	1e-5
Optimizer	Adam

Table 6.1: Hyperparameters

6.2 Data sets used

The synthetic datasets discussed in section 5.3 are used for the training and testing of crossentropy, evidential, dropout and ensemble models. The table 6.2 represents the different training and test datasets considered in this work.

S.no	Dataset name	Condition
1.	Normal dataset	Training and testing
2.	Bright lighting	Testing
3.	Dark lighting	Testing
4.	Far distance	Testing
5.	Near distance	Testing
6.	Blur constraint	Testing
7.	Textures constraint	Testing
8.	Deformation constraint	Testing
9.	Distractor constraint	Testing
10.	Expected value dataset (Fuzzy logic)	Training and Testing

Table 6.2: List of datasets used in the experiments

6.3 Training of deep learning models

The normal training dataset mentioned in the section 5.3.1 is initially split into three parts in which 70% of the data is considered for training, 20% for validation and 10% for final testing. A set of augmentation techniques like random rotation, horizontal flip and vertical flip were used while creating the data loaders to help improve the performance. Then both cross entropy loss and evidential loss are used for training the ReseNet 18 DNN model. Only the number of output neurons in the final layer is changed to the number of classes present in the dataset. i.e for RoboCup@work components - 15 classes and no dropout is added during the training process. The validation dataset is used to predict the output of the DNN model and also to check if the model is over-fitting or under-fitting. The hyper parameters mentioned in the table 6.1 were used for training the two models and the models with best weights and the 10% test data is used for checking the performance of the models.

For training the dropout model, a ResNet18 architecture is considered with a dropout rate of $p = 0.5$ and the normal training dataset 5.3.1 is used for training, validation and testing. The dropout model is trained using the cross entropy loss and the model with best weights is saved. The dropout model with best weights is tested on the 10% test data for five number of farward passes and a dropout rate of $p = 0.5$. The output predictions of the five farward passes are then averaged to get a single output prediction for each image.

In the case of ensembles model, we have considered a total of 4 different deep learning architectures like, ResNet18,ResNet34,MobileNetV2 and MobileNetV3_small and trained the four models individually using the normal training dataset and the hyperparameters provided in the table 6.1. During testing the 10% test data is passed through all the four models individually and the resulting output predictions are averaged to obtain as a single prediction for the input image.

6.4 Classification metrics

For evaluating the performance of deep learning models in the task of multi class classification, we have used the general classification metrics like, accuracy score, precision score, recall and f1-score. All these metrics are calculated using the sklearn.metrics library [20].

6.4.1 Accuracy score

Accuracy score is the most commonly used metric for measuring the performance of a deep learning model in classification task. The metric is computed based on the ratio of number of correct predictions of the DNN model to the total number of predictions. The metric takes into account of true labels and the predicted labels alone and do not consider the confidence of the model.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{6.1}$$

6.4.2 Precision score

Precision score is considered as a measure of the exactness of the predictions of classifier. It can be computed from the ratio of correct predictions to the all positive predictions made by the model. Precision describes about how often the classifier is correct in making a positive prediction.

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

6.4.3 Recall

Recall score is considered as a measure of the completeness of the model's predictions. A classifier with high recall is good at identifying all of the positive instances in the dataset, Recall score can be computed from the ratio of true positives which are correct predictions to the total number of actual positive instances.

$$Recall = \frac{TP}{TP + FN} \tag{6.3}$$

6.4.4 F1-score

F1-score is considered as the combination of precision and recall and it is a measure of the balance between precision and recall. F1-score is often used as a single metric to evaluate the performance of a classifier. A classifier with a high f1-score is able to make precise and complete predictions. It can be computed from the harmonic mean of precision and recall.

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{6.4}$$

6.5 Evaluation metrics for uncertainty

In deep learning for evaluating the performance of a classifier, considering model's accuracy is alone not sufficient as it does not represent the reliability of the model's predictions. Accuracy score do not provide any information about how certain the predictions are as it considers only the true labels and the predicted labels. So, for explaining the reliability of the DNN model, calibration metrics which measures the reliability of the model are necessary. In this work, we have used Brier score and Expected calibration error metrics for evaluating the performance of different deep learning models. Both of the metrics are calculated using the approaches presented in [21] .

6.5.1 Brier score

Brier score measures the mean squared distance between model's predicted probabilities and the actual expected outcome which is one. Brier score is influenced by both discrimination and calibration simultaneously and small values of brier score indicates the good performance of the model [22]. Brier score determines the accuracy of the probability estimates and a good brier score value will be close to zero. A higher brier score indicates that the model is poorly calibrated and the accuracy of the probability estimates is bad. Brier score can be formulated as,

$$BS\left(p,\hat{p}\right)=\sum_{i=1}^{c}\left(P_i-\hat{P}_i\right)^2=1-2^n p_j+\sum_{i=1}^{c}\hat{p}_i^2 \tag{6.5}$$

Brier score considers only the mean squared error between the predicted probabilities and the true labels, and does not take into account other aspects of the predicted probabilities, such as their distribution or their calibration making its application limited to binary classification. Another limitation of the Brier score is that it is sensitive to the number of classes in the dataset. In multiclass classification tasks, the Brier score may be biased towards models that predict a larger number of classes, even if their predictions are less accurate. This can make it difficult to compare models with different numbers of classes [22] [8].

6.5.2 Expected calibration error

Expected calibration error (ECE) is a metric used to evaluate the calibration of a classifier or the deep learning model. is defined as the difference between the average predicted probability of a class and the average true probability of that class, over all classes and over all test examples. The lower the ECE, the better the calibration of the model. Therefore, models with lower ECEs are considered to be more reliable and trustworthy, as they produce predictions that are more closely aligned with the actual outcomes. ECE can be computed using the formula 6.6, in which B is the number of histogram bins, n_b is the number of probabilities in bin b and N is the total size of the data [21].

$$ECE\left(B\right)=\sum_{b=1}^{B}\frac{n_b}{N}\left|acc\left(b\right)-conf\left(b\right)\right| \tag{6.6}$$

$$= \frac{1}{N} \sum_{b \in B} | \sum_{(\widehat{p}_i, \widehat{y}_i)} 1(y_i = \widehat{y}_i) - \widehat{p}_i | \tag{6.7}$$

In case of expected calibration error the result depends upon the distribution of data across the number of bins used for calculation and the number of bins chosen affects the ECE algorithm [23]. This means that models with low ECE can still have poorly calibrated predictions or be overconfident in its predictions.

6.5.3 Entropy

Entropy is a measure of uncertainty or randomness of a distribution and it can be used to measure the uncertainty of the model's predictions. The value of entropy depends on the type of distribution and deep learning models provide different types of output distributions like, multinomial and dirichlet distributions. In practice, entropy is often used as a post-processing step to calibrate the predicted probabilities of a deep learning model, in order to obtain more meaningful and accurate measures of uncertainty. For example, it can be used to identify misclassified or out-of-distribution examples. For example, if a model is highly confident in its prediction, the entropy of the predicted class distribution will be low. On the other hand, if the model is uncertain about its prediction, the entropy of the predicted class distribution will be high. Entropy is a general measure of uncertainty that can be applied to any classification task, regardless of the number of classes or the distribution of the data. It can be also used to compare the uncertainty of different models or different sets of predictions, which makes it useful for evaluating and comparing the performance of different uncertainty estimation methods. Hence in this work we mainly focus on evaluating the performance of different uncertainty estimation methods like evidential deep learning, MC-dropout and ensembles using the entropy of output predictions.

6.5.3.1 How to compare boxplot of entropy distributions

A box plot represents the distribution of data and each segment in the box plot contributes to 25% of the data. Which means the region between Min and Q1 corresponds to 25% of the data and similarly other regions, The region between Q1 and Q2 is called as interquartile region and it represents 50% of the data. The region between the median value and Q1 is called as lower quartile region and the region between the median value and Q2 is called as the upper quartile region. Box plots can be used to compare different distributions or the outputs of different deep learning models. By comparing the box plots for different datasets or models, it is possible to identify differences in the distribution of the data or the performance of the models. This can be useful for evaluating the performance of different models or for identifying potential problems with the data.

6.5.3.2 Comparison of box plot based on entropy of correct and incorrect predictions of a test constraint

We know that the entropy value of a distribution is dependent on the number of classes, which means as the number of classes increases the max value for the entropy increases, but the minmum value will still

remain at zero. In ideal case an entropy value of zero for an input represents that there is no uncertainty in the output predictions of the model. This can also be said as only one particular class has high probability value than the other classes in the distribution. The max entropy value for an input represents that the model is uncertain about its predictions. This can also be represented as the probability of the input belonging to a particular class is equal for all the number of classes and thus the model is confused or uncertain about its predictions.

The entropy which we discussed above comes from the model's output distribution and not the actual class label. But in deep learning, we have both true labels and predicted labels. In some cases the models may have high confidence value of 0.9 to a particular class (A) but in reality the input passed to the model might belong to class (B). If the true label do not match with the predicted label, then they are consider as incorrect predictions. But if we check entropy for this output then the value will still be low and close to zero. This is because the entropy takes only number of classes into consideration and not the actual labels and predicted labels. Hence for comparing the entropy distribution we need to consider the correct and incorrect predictions seperately. Thus in this case we expect the entropy of correct predictions to be lower than the entropy of incorrect prediction. We expect this because in ideal case we want the model to be more confident only for correct predictions and for incorrect predictions we expect some uncertainty in the distribution and not a high confidence value to a particular class.

In the following experiments, inorder to infer that the perfomace of the models in differentiating the correct predictions and incorrect predicitions, we create a rule stating, **The uncertainty of correct predictions shall be lower than the uncertainty of incorrect predictions**. Which we can also say from a box plot that the difference of medians of correct predictions and incorrect predictions shall be high.

6.5.3.3 Comparison of box plot for the entropy of correct predictions of normal constraint and correct predictions of test constraint

Since we are testing the performance of DNN models with nearly similar or unseen data (test constraints), we expect some uncertainty in the predictions of the DNN models as the test data is not seen during training process. In different test constraints like lighting, distance etc, even for the correct predictions, ie, $truelabel = predictedlabel$, we expect the model to have some confusion. If the model has high confidence in the correct predictions of test constraint, then we can say that the model is not differentiating between the training and test constraint and the learning is not very good. We still expect the models to give correct predictions but with high entropy rather than assigning high confidence to one particular class. Hence in the following experiments, we create a rule stating that, **The entropy of correct predictions of a test constraint shall be higher than the entropy of correct predictions of normal constraint**. Which from a box plot we can also say that the difference of medians of correct predictions of a test constraint and the correct predictions of normal constraint shall be high.

One main thing to remember in a box plot is that the lower and upper quartile region represents 50% of the data and we expect that most of this interquartile region for the correct predictions of a test constraint not to lie in the range of interquartile region of normal constraint(training data). If the

interquartile region of a test constraint is in the range of interquartile region of normal constraint, then we can say that the models are giving high confidence to predictions of test constraints like lighting, distance, blur, deformation etc, which is not expected. We expect the models to have some uncertainty in their predictions for a test constraint since the test constraint data is not seen during the training process.