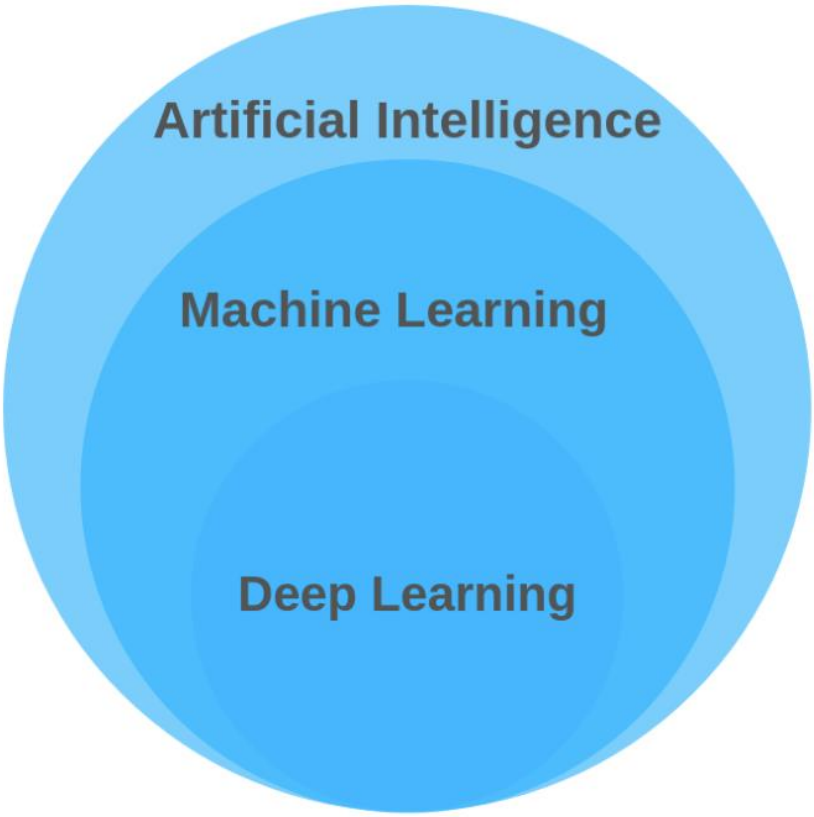
Multi-class Image classification model using CNN

What is Deep learning?

Deep Learning is the subfield of machine learning inspired by the structure and function of the brain called artificial neural networks. It has been quite a topic in the market now; let's deep dive in and know why it is picking up even though it has been around for quite some time.

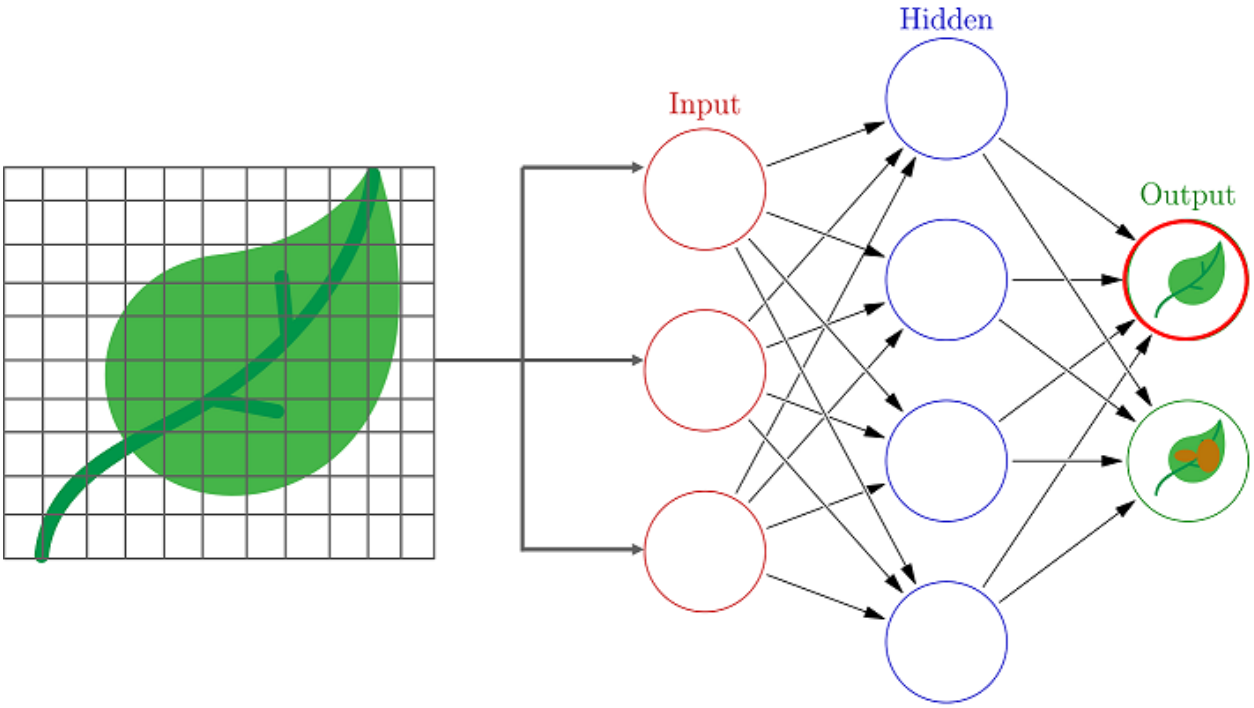
Deep Learning needs massive data and strong storage units (which were expensive earlier). Earlier neural networks were not famous when it came to application because the technology back then was not strong enough to support this concept, but now organizations are increasingly turning to deep learning as it allows computers to learn independently. Unlike other machine learning, deep learning attempts to simulate how our brains remember and process information.



How does Deep Neural Network work?

Think of the human brain comprising billions of neurons connected and orchestrated to adapt new learnings. For example, when a child sees a dog and says, “Dog,” the parent’s acknowledgment saying, “Yes, it is a dog” or “No, it is not a dog,” helps the child to learn and distinguish between the type of animals. So, the next time he sees an animal, he can differentiate whether the animal is a dog or not. A neural network is based on the idea of mimicking the human brain’s biological process to adapt to learning new things. Some examples of neural networks are Image classification, facial recognition, object detection, Speech recognition, NLPs, and many more.

The below image shows how a neural network differentiates between types of leaves.



Architecture:

A basic neural network architecture has interconnected artificial neurons in three layers – The input layer, the Hidden layer, and the output layer.

But wait, what is a neuron?

Neurons are nodes like humans in their brains through which data and computations flow. They receive the input signals, perform some calculations, and send some output signals to neurons deeper in the neural net through a synapse.

Input layer: This layer accepts the data and passes it to the rest of the network

Hidden layer: This layer can be more than one, and they are responsible for the performance of the neural networks. Hidden layers perform multiple functions simultaneously, such as data transformation, automatic feature creation, etc.

Output layer: The last layer holds the result of the problem statement.

For example, we have a bunch of vehicle pictures at the input layer, the hidden layer performs the function and selects the relevant features from the image to classify the image, then sends this output to the output layer, which tells the user what kind of vehicle it is.

So, what's the purpose of this built prototype?

While volunteering for one of the volunteer-based non-profit organizations Ewa(Earthwise Awareness), I realized that volunteers struggle with classifying the images they captured to upload them to the iNaturalist website as observations. Taking the deep learning course made me realize I could use the knowledge and some online resources to solve this problem, which takes them hours to complete. For now, I have created the prototype, which can be expanded by adding more features like image enhancement, image to text features and making it handy for people with zero or fewer technical skills. The model is made reusable to support N number of classes as the insect community is massive.

Let's dive deep in.

To avoid the hassle of downloading anaconda, python and setting it up, we will use "Google Colab" to build the model.

First thing first, install the following libraries in your notebook.

```
[1]  #install the below set of libraries

     !pip install tensorflow tensorflow-gpu opencv-python matplotlib
```

Then, import it

```
[4]  import tensorflow as tf
     import os
```

To get started, we need some data. So, we download a bunch of images from google for insect categories – bumble bee, honeybee, beetle, snail, butterfly, grasshopper, ants, milkweed bug, and dragonfly, and save it in the following pattern. (These images will be used to train the data)

The directory for training images is – real_data

The images can be of any format or extremely small, which does not help train the model. So, we do some cleaning as follows:

```
[10] #removing images of low quality or unsupported extension

     for i in os.listdir(data_dir):
       print(f"Removing dodgy images from folder {i}")
       for image in os.listdir(os.path.join(data_dir,i)):
         image_path = os.path.join(data_dir,i,image)
         try:
           img = cv2.imread(image_path)
           tip = imghdr.what(image_path)
           if (round(os.stat(image_path).st_size/1024) < 10) or (tip not in image_ext):
             os.remove(image_path)
         except:
           print("Image is corrupted: {}".format(image_path))
```

Here, “imghdr” is the library in python which helps to determine the type of image.

After cleaning data, the folders only contain images with jpg, jpeg, png, and bmp extensions, and other formats are removed.

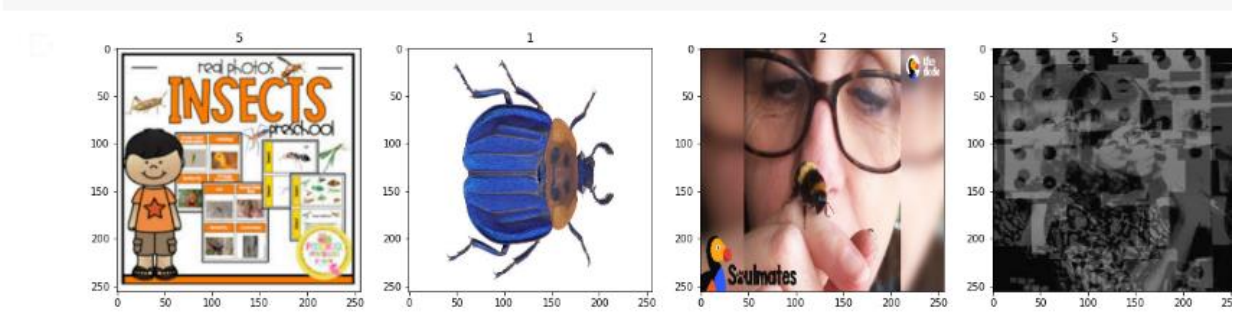
Loading data using Keras utility:

We have used tf.keras.utils.image_dataset_from_directory utility to load these downloaded images from the drive. This utility returns the tf.data. A dataset that yields batches of images from the subdirectories of your folder, labeling them starting from 0 to the n-1 (number of image folders). By default, the batch size is 32.

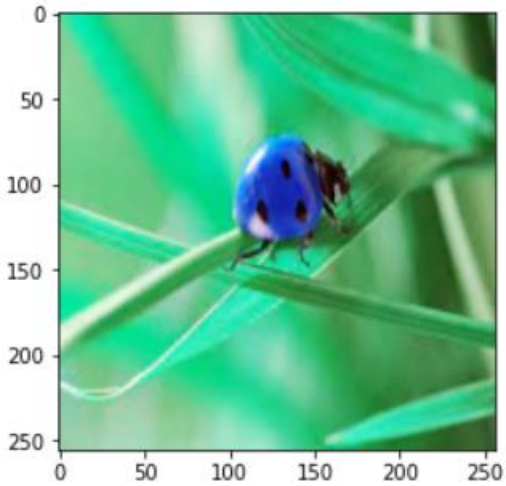
```
#loading data from the given directory using tf.data.dataset

data = tf.keras.utils.image_dataset_from_directory(data_dir)
```

```
Found 1010 files belonging to 9 classes.
```

Once the data is loaded, we will iterate through the batches and plot it to check if it is loaded correctly.



(Note* - Sometimes the image may appear as follows when you try to plot it using matplotlib



This is because the matplotlib expects the image to be in RGB format but receives the image in BGR format as OpenCV reads the image in BGR format.

You can convert the format from BGR to RGB using the cv2 function – cv2.cvtColor, then pass the code cvt.COLOR_BGR2RGB along with the image.)

As the images come in different sizes and are grayscale, all the values range from 0-255. We must scale all the images in one size to get better results and speed up our training process later.

So, to scale the images, we resize them and divide them by 255 to get the values between 0 and 1. This process is also called normalization.

Now, it's time to build the model. But before that, we must split data into three parts – training data to train the model, validation data to validate the performance, and test data to test the unseen data by the model to check its performance. We are following the 80-20 rule here. 80% is training data, 15% is validation, and the rest is test data. All the dataset is stored as a NumPy array.

Out of all the neural networks available, we will use a convolutional neural network (CNN) as they are better suited for image classification. CNN uses filters, also called kernels or feature maps which help the model to capture and learn various characteristics of an image.

Convolution is often represented mathematically with an asterisk * sign. If we have an input image represented as X and a filter represented with *f*, then the expression would be:

$$Z = X * f$$

In Keras, we have two APIs for instantiating the model – Sequential and Functional.

> Sequential allows the user to design a model layer by layer. This is easy to use but doesn't allow the sharing or branching of layers. (We will use this API for now)
>
> Functional API is more flexible as compared to sequential API. This model allows to share or branch the layers. It can also have multiple inputs and outputs.

We instantiate our model with Sequential API, saying that the training process occurs sequentially. So, our first layer is a convolutional layer. Let's break down this in part and understand.

```
model.add(Conv2D(16, (3,3),1, padding='same', activation='relu', input_shape=(256,256,3)))
```

Here, 16 is the number of filters used in the first layer to scan the image and extract as many features as possible.
(3,3) is our filter pixel size and stride of 1, which means that one pixel moves at a time.

In the above line of code, the input_shape attribute comprises our image size – (256,256,3), where 3 is the number of channels. We can increase the number of strides or filters based on architectural requirements. We have also used the activation function here in each layer of the model.

What is the activation function?
The activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a network layer. They are a critical part of the design, as the choice of activation function will influence the model's performance.
We are using the"relu" activation function, which means we will take the output from the Conv2D layer and pass it through the function where all the negative values will be converted to 0, and the positive values will remain the same.

```
[ ]  from tensorflow import keras
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropou
```

```
[ ]  model = Sequential()
```

```
[ ]  model.add(Conv2D(16, (3,3),1, padding='same', activation='relu', input_shape=(25
     model.add(MaxPooling2D())
     model.add(Conv2D(32, (3,3),1, padding='same', activation='relu'))
     model.add(MaxPooling2D())
     model.add(Conv2D(64, (3,3),1, padding='same', activation='relu'))
     model.add(MaxPooling2D())
     model.add(Conv2D(128, (3,3),1, padding='same', activation='relu'))
     model.add(MaxPooling2D())
     model.add(Flatten())
     model.add(Dense(256, activation='relu'))
     model.add(Dense(num_classes, activation='softmax'))
```

```
[ ]  from tensorflow.python import losses
     model.compile('adam', loss = tf.keras.losses.SparseCategoricalCrossentropy(from_
```

The padding attribute tells the model not to change or compress the image size.
We have also used the "softmax" function, which means we are reshaping the output.
MaxPooling2D will scan through the output, take the maximum value and condense the information. We haven't provided any pixel size, so that it will take the pixel size as (2,2).

Now, the process continues until we reach the flattened layer. And we are then condensed into one image via a Dense layer. In the last dense layer, we pass the number of classes to help the model classify images in the appropriate category.

After adding the layers, we compile our model using 'Adam' as an optimizer and loss function.

`SparseCategoricalCrossentropy`, as our classification is multi-class.

Below is our model summary:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 256, 256, 16)	448
max_pooling2d (MaxPooling2D)	(None, 128, 128, 16)	0
conv2d_1 (Conv2D)	(None, 128, 128, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_3 (Conv2D)	(None, 32, 32, 128)	73856
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 256)	8388864
dense_1 (Dense)	(None, 9)	2313

Conv2D: (3x3x1)x16

Seeing the above picture, you may wonder what those numbers are under the Params column. The numbers under the param column tell us whether the model is learning in that particular layer. The layers in CNN can be broken down as follows:

Input layer: The input layer has nothing to learn; at its core, what it does is provide the input image's shape

(28,28,1) i.e., 28*28*1 = 784 with 0 parameters

Convolution layer: This is where CNN learns. A number of parameters in a CONV layer would be : ((m * n * d)+1)* k), added 1 because of the bias term for each filter. The same expression can be written as follows: ((shape of the width of the filter * the shape of the height of the filter * a number of filters in the previous layer+1)*a number of filters).

Pool layer: This has no learnable parameters because all it does is calculate a specific number; no backdrop learning involved! Thus the number of parameters = 0.

Fully Connected Layer (FC): This certainly has learnable parameters; in comparison to the other layers, this category of layers has the highest number of parameters because every neuron is connected to every other neuron.

Here is the calculation behind those numbers:

Conv2d_1: (3*3*16)+1 * 32 = 4640

Max pooling: no parameters

Conv2d_2 : ((3*3*32)+1)*64 = 18496

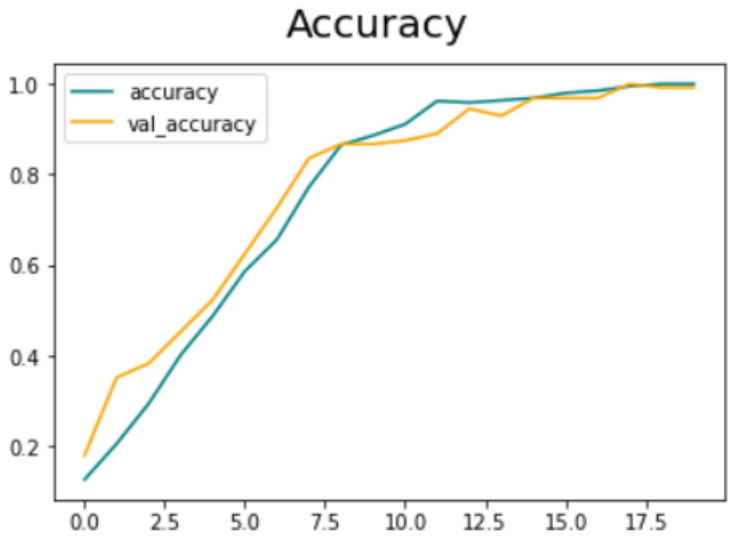
Similarly, we do it for another convolutional layer as well.

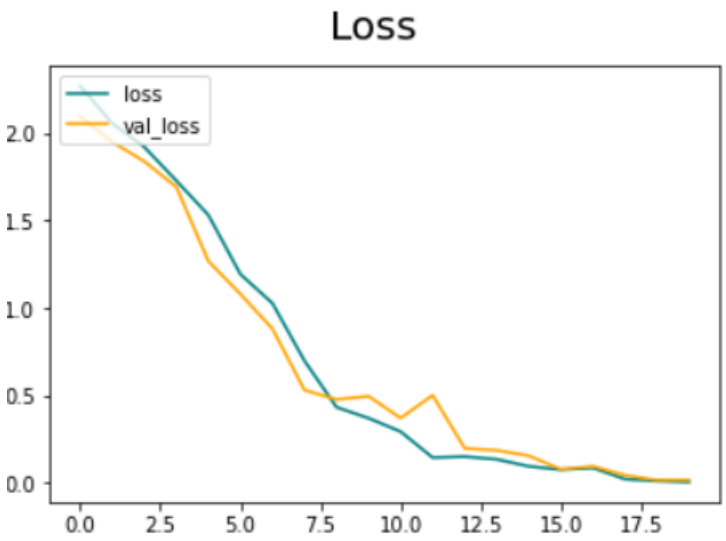
For dense layer:

Dense: 32768*256 = 8388608

Dense_1: (256*9)+9 = 2313

Finally, after the model is trained, we fit the model using the model. fit and plot the accuracy and loss for the trained model compared with validation data. Below are the results:





From the above images, we can conclude that model started learning after epoch 7 by giving accuracy close to 99% for both training and validation data. The loss has decreased from 2 to almost 0 by the end of the epochs.

To evaluate the model's performance, we created the classification report showing the model's precision, recall, and f-1 score. From the below image, it can be concluded that all parameters show high accuracy.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2
1	1.00	1.00	1.00	2
2	1.00	1.00	1.00	6
3	1.00	1.00	1.00	2
4	1.00	1.00	1.00	2
5	1.00	1.00	1.00	3
6	1.00	1.00	1.00	2
7	1.00	1.00	1.00	6
8	1.00	1.00	1.00	7
accuracy			1.00	32
macro avg	1.00	1.00	1.00	32
ghted avg	1.00	1.00	1.00	32

After this, we tested the model with unseen data comprising 4 images, out of which the model guessed all the images correctly. I haven’t found a way to show the name of the classes with the corresponding label numbers. Since the label for each class is encoded in alphabetical order of the subdirectory, I cross-checked the accuracy in such a manner.

Last, we will save the model for future use using the model.save function. The model can be saved using TensorFlow lite or in the pickle file. This model can be picked and converted to a reusable model using TensorFlow lite for mobile applications.