© 2022 IJRTI | Volume 7, Issue 4 | ISSN: 2456-3315

Wildlife Classification Model - Deep Learning in Python with Keras

Mohammed Zidan Vaheed[1], Dr M.N. Nachappa[2]

[1]Post Graduate Student, [2]Professor and Head
School of Computer Science and Information Technology,
JAIN (Deemed to be University), Bangalore

***Abstract*: Deep learning algorithms are a subset of the machine learning algorithms, which aim at discovering multiple levels of distributed representations. Recently, numerous deep learning algorithms have been proposed to solve traditional artificial intelligence problems. This work aims to create a state-of-the-art deep learning model with computer vision at its base by highlighting the contributions and challenges from recent research papers. It first goes over the recent studies on the deep learning topics and the discoveries of other brilliant individuals in this field and highlights their successes and how that has contributed to the creation of our model. The system requirements are elaborated to help understand the intensity of work the system has to handle and what minimum level of computational power is required to create these deep learning models, as we know the models are quite computationally power heavy and need a lot of resources and time to work and create these models. Then we see the overview of the system, analyzing it as well as its designs using pictorial representations to make understanding of the topic much easier. Finally, we summarize the discussion with the future plans for this deep learning model and how it will be upgraded for obtaining greater results that far surpass the results we have obtained from this project now.**

***Keywords*: Artificial Intelligence, Computer vision, Power heavy, Convolution Neural Networks, Class weights, Model Layers.**

INTRODUCTION

Deep learning is a subfield of machine learning which attempts to learn high-level abstractions in data by utilizing hierarchical architectures. It is an emerging approach and has been widely applied in traditional artificial intelligence domains, such as semantic parsing, transfer learning, natural language processing, computer vision and many more. There are mainly three important reasons for the booming of deep learning today: the dramatically increased chip processing abilities (e.g., GPU units), the significantly lowered cost of computing hardware, and the considerable advances in the machine learning algorithms. Deep networks have been shown to be successful for computer vision tasks because they can extract appropriate features while jointly performing discrimination. In recent ImageNet Large Scale Visual Recognition Challenge (ILSVRC) competitions, deep learning methods have been widely adopted by different researchers and achieved top accuracy scores. The current image recognition methods use artificial extraction features. This method is not only time-consuming and laborious, but also difficult to extract. However, deep learning is a kind of unsupervised learning. In the process of learning, it is not necessary to know the tag value of the sample, and the whole process can extract good features without human participation. In the recent years deep learning has become a hot topic of research in the field of image recognition. It has achieved good results and has a broad space for research.

The most famous aspect of computer vision deep learning models is the use of Convolution Neural Networks (CNN). The convolutional neural networks (CNNs) can use the model structure of the convolution layer and the lower sampling layer in turn by simulating the human visual system. The convolution layer enhances the original signal and improves the signal to noise ratio. The lower sampling layer uses the principle of image local correlation to sample the image from the neighborhood. It can extract useful information while reducing the amount of data. At the same time, parameter reduction and weight sharing alleviate the problem of long training time to a certain extent. However, experiments show that the training time of CNNs is still very long.

In this paper we aim to create convolution neural networks that can recognize the images of animals and accurately classify them. Alongside this we aim to reduce the time of training for the CNNs which are famously slow as well as add extra layers into the neural networks to get greater results for our deep learning model.

LITERATURE REVIEW

The use of neural networks to a variety of computer vision subjects has demonstrated that with the correct resources, computer vision can recognize almost everything. Obtaining the required datasets for the project is arguably the most difficult component of a process like this. The dataset will serve as the project's foundation, since neural networks will use it to train and learn how to recognize a specific image. The effort of figuring out and putting up networks, as well as strategies to improve them, will be easier to figure out and implement with the aid of other publications; datasets are irrelevant. What counts are the networks that have been built, the outcomes that have been produced, and how they were developed. In recent literatures there have been several approaches to using deep convolution neural networks for image classification, but since convolution neural networks are more scalable for larger datasets it is more suitable to apply them for our wildlife classification as the datasets being used are enormous.
Convolution neural networks were used to classify food images [1], to be more precise they used an inception v3 model which was pre-trained by ImageNet. The convolutional neural network learns the filters that were previously hand-engineered in existing

methodologies. They conducted their investigation using the Food-11 Dataset, which included 16643 photos divided into 11 categories. Dataset sizes are often in this range, which is one of the key reasons why training neural networks takes so long.
The inception v3 model consists of the following layers:

- Convolution Layer: At the beginning convolution layer with input size 299 x 299 x 3 to create feature maps by convolving input images.
- Max Pooling Layer: Max-pooling is a sample-based discretization process. Max pooling is done by applying a max filter to non-overlapping sub regions of the input matrices. Max-pooling extracts the most important features like vertical edges and horizontal edges.
- Average Pooling Layer: Average pooling layer reduces the variance and complexity in the data. It also divides the input into rectangular pooling regions and computing the average values of each matrix to down sample the input features [10].
- Concat Layer: The Concat layer concatenates its multiple input blobs to one single output blob [10].
- Dropout Layer: The dropout layer randomly drops elements from a layer in the neural network. Dropout is a technique used to improve over-fit on neural networks.
- Fully Connected Layer: The fully connected (FC) layer in the CNN represents the feature vector for the input. This feature vector holds information that is vital to the input.
- SoftMax Layer: The SoftMax assigns decimal probabilities to each class in a multi-class recognition problem. Those decimal probabilities must add up to 1.0. This additional constraint make training to converge more quickly.

The researchers [2] had opted to use the deep belief networks and they had studied the structure of the restricted Boltzmann machines (RBM). They opted to go for a SoftMax classifier for their deep belief network and carried out their experiment on the MNIST library. Their experiment showcased that the recognition rate of the deep belief network was basically flat and slightly decreased, but the training time was greatly shortened. They concluded by mentioning that the combination of random regression and drop sampling obviously improves the recognition rate of the deep belief network system and reduces its training time. The training methods used by DBNs are also very different from those of traditional neural networks. For the three-layer network, the BP algorithm has a good training time and algorithm recognition rate. However, for a network with multiple hidden layers, the training time is too long.
The training process of DBNs is trained by layer by layer, and only one layer of RBM is trained at each time. This process is exactly the same as the RBM training, and the parameters are adjusted separately.
It can be divided into two parts:

- Step 1 - Pre-training First, the parameters of the deep belief network are set up. The number of nodes on each tier, as well as other aspects of the DBNs network, such as Dropout size, sparsity, and noise, are all part of the initialization process. After startup, each layer's RBM may be trained independently. The first level's output h1 becomes the second level's input, and so on, with the weight Wij of each tier being preserved.
- Step 2 - Fine tune in order to improve the performance of the network, the whole network may be changed based on the sample's label value. It is decided to employ the gradient down algorithm. The DBNs network is now an ordinary neural network, comparable to the BP algorithm. The weights of all layers are trained in advance before fine-tuning. Because it is not randomized like a neural network, it just requires a limited number of iterations to achieve decent results.

IMPLEMENTATION

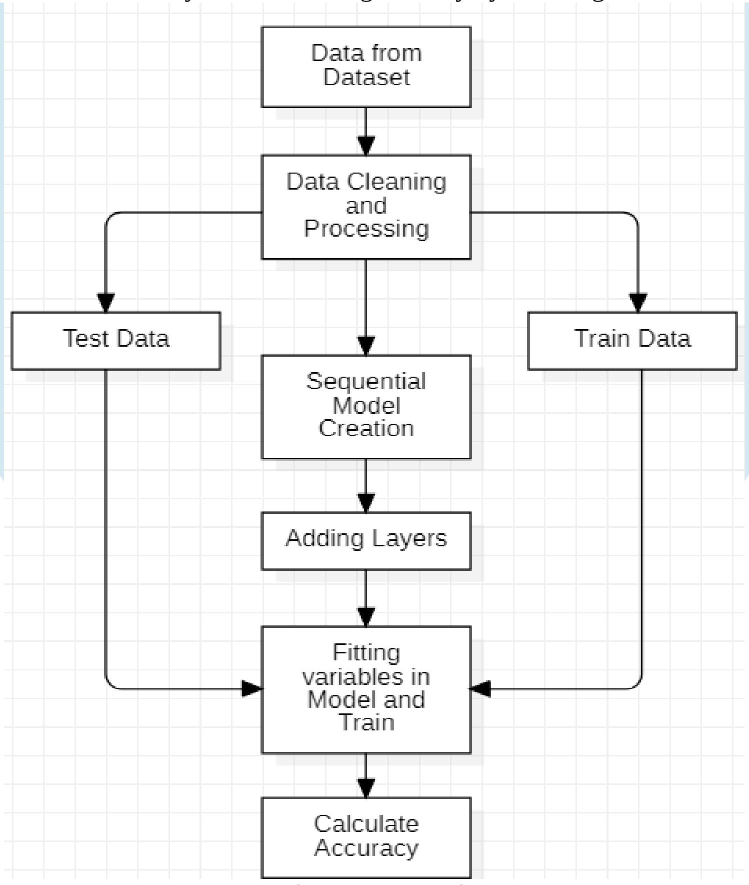
PREPROCESSING SECTION

We start off by bringing the image dataset into the respective integrated development environment (IDE) of choice, here we use Google Colab as it offers free GPUs to use for up to 12 hours to training our models. This option is very useful for individuals who have a hard time obtaining high performance hardware which is highly necessary for training models in a short time. The image dataset was loaded into the google drive from where it is brought into the Colab file. We opted for this method so it would not be necessary to load in an approximately 700mb dataset every time we had to train an iteration of the model or just run the colab file when working on it. Loading it into the google drive allows us to upload it once and then just copy the file into colab without wasting excess data. By going through each section of animal images in the dataset it is quite clear that the dataset holds an unequal number of images for each category. The number of images is as follows:

- Dogs – 4863
- Horse - 2623
- Elephant - 1446
- Butterfly - 2112
- Chicken - 3098
- Cat - 1668
- Cow - 1886
- Sheep - 1820
- Spider - 4821
- Squirrel – 1862

To tackle this issue, we implement the class weights which allow us to adjust which category of images must be trained more than others by assigning weights to each category so as to make the training more balanced. If the dataset of images are not properly balanced the model will not be able to train for each category properly. We start off by reading all the images from the dataset and reshaping them into a desired height and width. The size that has been opted for this project is a uniform size of 128 for both height and width. Once the images have been reshaped, we return the images along with an index value for each category they belong to

and store it in a dictionary. This dictionary is then shuffled and we split the dictionary into 2 lists which hold the images in one list and the labels of the respective images in the other list. Both these lists are then converted into arrays using the NumPy package. The image list is also reshaped into the previously decided height and width along with a few more parameters. This is done so that the images can be properly loaded into the model. When the model is created it is setup to only take in images that follow a certain shape. This shape is the shape we have just converted it into.
We then move onto the splitting of the data into training data and testing data. This is the data we will be implementing into our model as data to train the model as well as data used to test the model. Once the training and testing data is ready. We implement one final pre-processing step onto the images. An important step to always implement in image classification models is to make sure our images are rescaled to a uniform range. Image pixels range from 0-255 and this includes all images. What it means is that some images are high pixel range images while others are low pixel range images. So, if the images are implemented into the model directly without rescaling them it will cause issues. Therefore, we rescale them from the range of [0, 255] to the uniform range of [0, 1]. Doing so will allow the model to treat all images in the same manner and all the images contribute more evenly to the total loss, as well as allow us to use a single learning rate for all images. Higher pixel range images result in higher loss and should use a smaller learning rate, while lower pixel range images will need a larger learning rate. To implement the rescaling of our images we implement a function called the ImageDataGenerator from Keras which allows us to rescale the images by saving lots of memory as well as perform Data Augmentation on our data. The main reason for the use of the ImageDataGenerator was to rescale all the images while saving a lot of memory. There were roughly 26 thousand images in the dataset and attempting to rescale the images directly would require more than 12 gigabytes of RAM which was over the limit of Google Colab's free virtual hardware. So, implementing the ImageDataGenerator was very useful in saving memory by rescaling all the 26 thousand images.


Data from Dataset
Data Cleaning and Processing
Test Data
Train Data
Sequential Model Creation
Adding Layers
Fitting variables in Model and Train
Calculate Accuracy
Fig 1. Data flow diagram of the system

The training and testing data is ready to be implemented into our model for training, but before doing so we must create the class weights we need to balance the data accordingly. There exists an inbuilt function we can use to implement class weights once it has been imported from the sklearn.utils package. By using the compute_class_weight function we can create the necessary weights needed for each category of images in our dataset. All we need to provide the function is the labels of our dataset and the function will figure out the necessary weights to be provided to each class depending on the number of images present inside it. This result is returned as a list and to be able to use it we must convert it into the dictionary format as the model only takes class weights as a parameter in the form of a dictionary. Fig 1, shows a proper representation of all the steps included in the whole process.

MODEL CREATION AND TRAINING

We import all the required packages for model creation through TensorFlow and Keras. The model we use is a Sequential model. Sequential is the easiest way to build a model in Keras. It allows us to build models by adding layers that we deem necessary. Our model consists of 23 Layers and an output layer of different layers which are explained as follows:

- Convolution2D - Sliding convolutional filters are applied to 2-D input by a 2-D convolutional layer. By moving the filters along the input vertically and horizontally, computing the dot product of the weights and the input, and then adding a bias term, the layer convolves the input. The first Convolution2d layer takes the input size of our data. Previously we had reshaped our data into a specific shape and those exact parameters are what we input here, specifically (128, 128, 3) where the parameters are height, width and channels.

- Batch Normalization - The batch normalization layer applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. It works differently during inference and training. The mean and standard deviation of the current batch on inputs are used to normalise the layer's inputs for training, whilst the mean and standard deviation of the batches it has seen during training are used to normalise the layer's output for inference.
- Dropout - Dropout layers are critical in CNN training because they prevent the training data from being overfit. If they aren't there, the first batch of training data has a disproportionately large impact on learning. As a result, learning of traits that only occur in later samples or batches would be prevented. Throughout our model we use a standard value of 0.2 for the dropout layer.
- Max Pooling - Max-pooling is a discretization method based on samples. By applying a max filter on non-overlapping sub regions of the input matrices, max pooling may be achieved. The most significant characteristics, such as vertical and horizontal edges, are extracted via max-pooling.
- Flatten - Flattening is the process of turning data into a one-dimensional array for use in the following layer. To construct a single lengthy feature vector, we flatten the output of the convolutional layers. It's also linked to the final classification model, which is referred to as a fully-connected layer.
- Dense - The dense layer is a simple layer of neurons in which all the neurons of the previous layers pass on its inputs to the new layer of neurons, thus the name. Dense Layers are used to identify images based on convolutional layer output.
- Output Layer (Dense) - The final output layer which is a dense layer as well intakes the number of classification categories we have present as well as an activation function which is set to SoftMax. The reason of using SoftMax activation function in the final output layer is because the SoftMax function is used as the activation function in the output layer of neural network models that predict a multinomial probability distribution.

Throughout the creation of the model the layers that use activation functions are all using the ReLU activation function because this activation function has a few advantages over the sigmoidal functions those being, ReLU is very simple to calculate, since it only involves the comparison of the input and the value 0. It also has a derivative of either 0 or 1, depending on whether its input is positive or not. The use of ReLU helps to keep the compute required to run the neural network from growing exponentially. The computational cost of adding more ReLU's increases linearly as the size of the CNN grows.

Before moving onto training the created model we also implement the optimizer we will be using for training the model. Optimizers are useful in model training as they modify attributes to help reduce overall loss and improve the accuracy. The Optimizer we have chosen is the Adam optimizer with a learning rate of 1e-3 or 0.001. This value for the learning rate is at an optimal range and increasing or decreasing it could result in faster or slower training time, but it will also sacrifice accuracy based on the way our model is built. Optimizers are compiled along with the model along with its loss function. The loss function used is the sparse categorical cross entropy function. It is a different version of the categorical cross entropy loss function which is used in multi-class classification models except the data being trained must be in a one hot encoded form, whereas the sparse categorical cross entropy loss function does not require the data to be one hot encoded. The one hot encoding process can be too memory heavy for multi-class classification models especially when there are 10 categories in our classification categories.

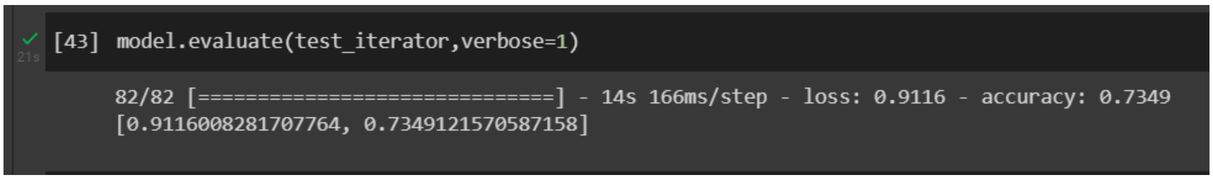
The compiled model is then placed into training phase using the fit method where we add the parameters we have prepared up until now. The training data, the testing data and the class weights are added into the fit method as parameters. Alongside these parameters we also set the number of epochs to 20 and the batch size to train as 64. An epoch is a unit of time used to train a neural network using all of the training data for a single cycle. We use all of the data exactly once in an epoch. A forward and backward pass are combined to make one pass: An epoch is made up of one or more batches in which we train the neural network using a portion of the dataset. We make use of the GPU that Google colab provides to fasten the time taken to train our model. The time taken on our devices hardware was 2 and a half hours for a single epoch which would result in around 50 hours of total training time. By making use of Google colab GPU we were able to drastically reduce the time taken to 1 min and 30 seconds for a single epoch, which resulted in a completion time of around 40 – 50 minutes for a single version of our model. During the training of the model, it allows us to see the exact accuracy values for each epoch shifting as the epochs complete the training. To be specific it shows us the training loss and accuracy as well as the validation loss and accuracy. The validation values are what we want to focus on as those values are what we ultimately use to judge our model's performance. Once the model completes the training phase, we can save the model using the save function so our model is saved, then we copy it into our system or into the google drive so that it can be used later on whenever we want to make predictions.

RESULTS

After the training was completed, the model showed an accuracy value of 73 percent for the validation accuracy and 87 percent for the training accuracy. From this result we can conclude that the model has achieved a decent accuracy value considering the parameters we had to work with. Along with the restricted hardware and time the value of 73 percent is a decent level to achieve for the model, so we move on with the prediction results.

Training Loss	Training Accuracy	Validation loss	Validation Accuracy
0.3629	0.8707	0.9116	0.7349

Table 1. Various Results from the model.



```
[43] model.evaluate(test_iterator,verbose=1)

82/82 [==============================] - 14s 166ms/step - loss: 0.9116 - accuracy: 0.7349
[0.9116008281707764, 0.7349121570587158]
```

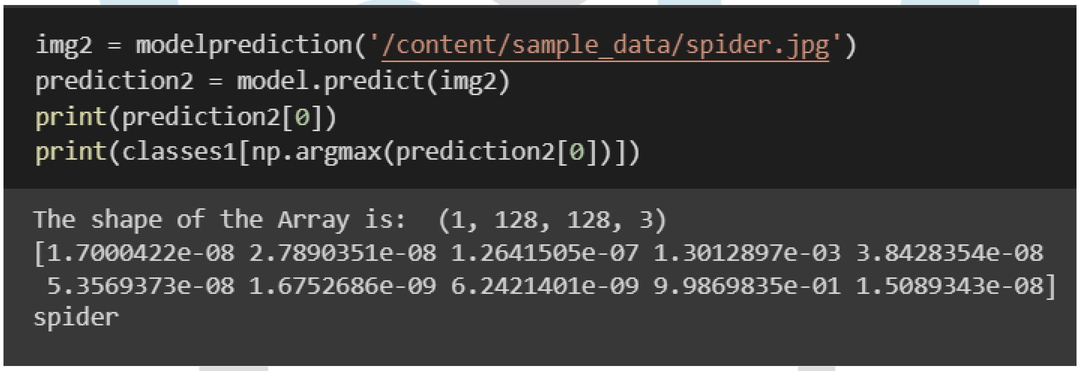
Fig 2. Validation Accuracy of the model.

The images fed into the model do show correct prediction for some of the pictures and it does show wrong prediction for a few other images which is expected for a model of accuracy in the range of 70-75 percent. The images that do show few errors are in the categories of cows and elephants which leads us to believe that the class weights implemented might not have been perfect as these 2 classes seem to show errors in predictions compared to the other classes. So, the class weights would have to be revised in order to obtain a better result for the model.
The images being fed into the model for predictions must be put through the same pre-processing steps that the trained images have been put through or else the prediction results will be drastically different and wrong. So, each image is put through a function that takes the image and converts it into the target size of 128 height and width, then converts it into a NumPy array, shifts its dimensions into the necessary dimensions that the model requires it to be in (i.e. (-1, 128, 128, 3)) and finally it rescales the image by 255 so as to convert the pixels of the image from the 0-255 range to a uniform range of 0-1 to consider all images equally. Once the image has been passed through this pre-processing function it will be put into the model.predict() function. The obtained value is an array of values from which we must get the maximum argument value as that is the value which indicates how close the result is to the categories. So, we have a categories list ready which holds all the categories and when we implement the category along with the maximum argument of the predicted value it should give us the correct category. This list is shown in Table 2.

Dog	Horse	Elephant	Butterfly	Chicken	Cat	Cow	Sheep	Spider	Squirrel

Table 2. All categories of the animals.

One of the results obtained is displayed in Figure 3, from which we can see the prediction values in a list and how we obtain the final result from the respective prediction.

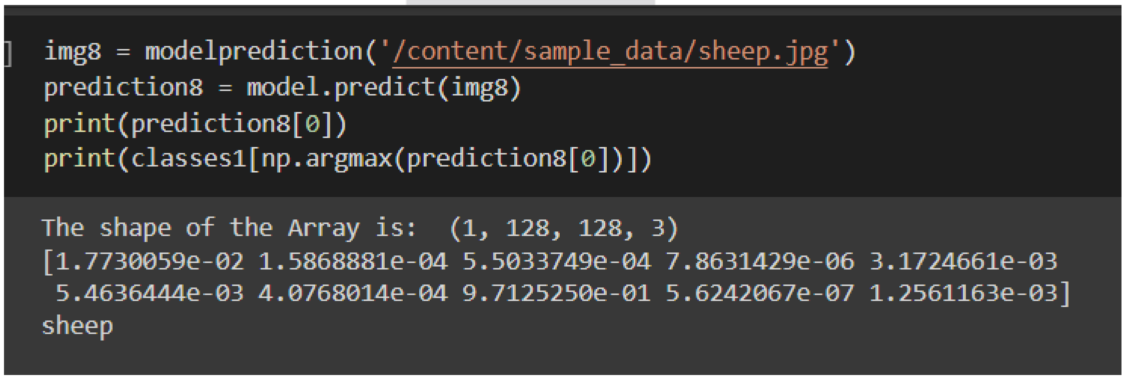


```
img2 = modelprediction('/content/sample_data/spider.jpg')
prediction2 = model.predict(img2)
print(prediction2[0])
print(classes1[np.argmax(prediction2[0])])

The shape of the Array is:  (1, 128, 128, 3)
[1.7000422e-08 2.7890351e-08 1.2641505e-07 1.3012897e-03 3.8428354e-08
 5.3569373e-08 1.6752686e-09 6.2421401e-09 9.9869835e-01 1.5089343e-08]
spider
```

Fig 3. Prediction result on the image of a spider.

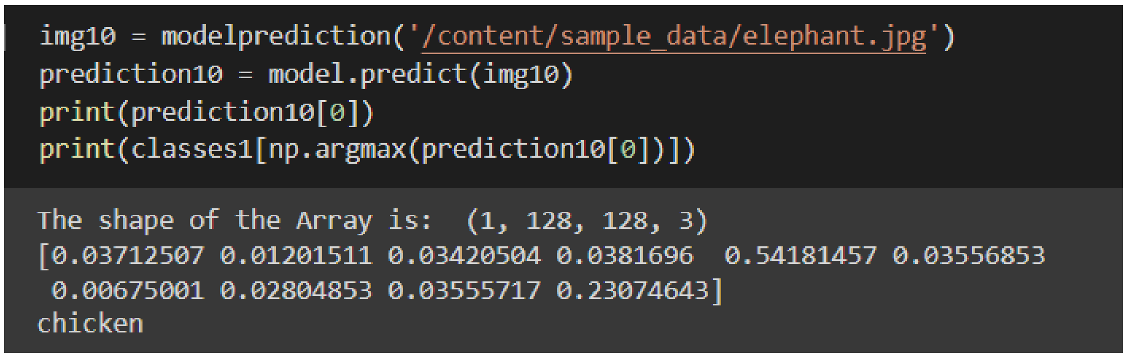
Figure 3 shows the result to an image of a spider, the model returns the correct prediction for the image, Figure 4 shows the prediction result for the image of a sheep, while Figure 5 shows how the model has predicted the wrong value for the input image.



```
img8 = modelprediction('/content/sample_data/sheep.jpg')
prediction8 = model.predict(img8)
print(prediction8[0])
print(classes1[np.argmax(prediction8[0])])

The shape of the Array is:  (1, 128, 128, 3)
[1.7730059e-02 1.5868881e-04 5.5033749e-04 7.8631429e-06 3.1724661e-03
 5.4636444e-03 4.0768014e-04 9.7125250e-01 5.6242067e-07 1.2561163e-03]
sheep
```

Fig 4. Prediction result on the image of a sheep



```
img10 = modelprediction('/content/sample_data/elephant.jpg')
prediction10 = model.predict(img10)
print(prediction10[0])
print(classes1[np.argmax(prediction10[0])])
```

```
The shape of the Array is:  (1, 128, 128, 3)
[0.03712507 0.01201511 0.03420504 0.0381696  0.54181457 0.03556853
 0.00675001 0.02804853 0.03555717 0.23074643]
chicken
```

Fig 5. Prediction result on the image of an elephant.

CONCLUSION

This project showcases the strengths and weaknesses of the convolution neural networks in the field of computer vision under image recognition and classification. The model created has showcased decent results and works as intended for the image inputs it receives to analyse upon. The model has trained with a good accuracy level of 73 percent, compared to the previous iterations of the model which were capped at a 60 – 65 percent accuracy level, this model shines in comparison to those. The future of this project lies in the improvement and advancement of the neural network to make it provide a stronger and more accurate result for a larger dataset as well as improves the time taken to train the network. The goals at the moment are to identify how to make the model better through changes in the layers of the model as well as implementation of other factors such as usage of different optimizers, better class weights for the dataset, and so on. The biggest challenge for all deep learning models is to solve the time-consuming issue it possesses. So far advancements have not been made into tackling this issue, but there are tremendous amounts of research being aimed in these sectors to solve these issues to make the use of neural networks more viable and easier for anyone who wishes to implement these neural networks into their work.

REFERENCES

[1] Md Tohidul Islam, B.M. Nafiz Karim Siddique, Sagidur Rahman and Taskeed Jabid, Image Recognition with Deep Learning, ICIIBMS 2018, Track 2: Artificial Intelligent, Robotics, and Human-Computer Interaction, Bangkok, Thailand.

[2] Fuchao Cheng, Hong Zhang, Wenjie Fan and Barry Harris, Image Recognition Technology based on Deep learning, Springer Science+Business Media, LLC, part of Springer Nature 2018.

[3] Anurag Arnab, Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Mans Larsson, Alexander Kirillov, Bogdan Savchynskyy, Carsten Rother, Fredrik Kahl and Philip Torr, Conditional Random Fields Meet Deep Neural Networks for Semantic Segmentation, IEEE Signal processing magazine, VOL. XX, NO. XX, January 2018.

[4] Luyu Dong, Fan Chen, Xin Li and Mengting Li, Improved image classification algorithm based on convolutional neural network, Proceedings of 2020 3rd International Conference on E-Business, Information Management and Computer Science (EBIMCS 2020). ACM, Wuhan, China, 4 pages. https://doi.org/10.1145/3453187.3453403

[5] Xin Jia, Image Recognition method based on deep learning, 2017 IEEE

[6] Casey Breen, Latifur Khan and Arunkumar Ponnusamy, Image classification using neural networks and ontologies, Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA'02) 1529-4188/02 $17.00 © 2002 IEEE

[7] Meiyin Wu and Li Chen, Image Recognition based on Deep Learning, 978-1-4673-7189-6/15/$31.00©2015 IEEE

[8] Dwiretno Istiyadi Swasono, Handayani Tjandrasa and Chastine Fathicah, Classification of Tobacco Leaf Pests Using VGG16 Transfer Learning, l2th International Conference on Information & Communication Technology and System (lCTS) 2019

[9] Myeongsuk Pak and Sanghoon Kim, A review of deep learning in Image recognition, This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2015R1D1A1A01057518).

[10] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich, Going deeper with Convolutions, 978-1-4673-6964-0/15/$31.00 ©2015 IEEE

[11] Wang Hao, Garbage recognition and classification system based on convolutional neural network VGG16, 2020 3rd International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)

[12] Y. Wang et al., Unsupervised local deep feature for image recognition, Information Sciences (2016), http://dx.doi.org/10.1016/j.ins.2016.02.044

[13] Xulei Yang, Zeng Zeng, Sin G. Teo, Li Wang, Vijay Chandrasekha and Steven Hoi, Deep Learning for Practical Image Recognition: Case Study on Kaggle Competitions, KDD '18, August 19–23, 2018, London, United Kingdom © 2018 Association for Computing Machinery. ACM ISBN 978-1-4503-5552-0/18/08. . . $15.00 https://doi.org/10.1145/3219819.3219907