International Journal of Data Informatics and Intelligent computing (IJDIIC)
Vol. 1, No. 1, September 2022, pp. 39~52
DOI: 10.59461/ijdiic.v1i1.12

An Intelligent Dog Breed Recognition System Using Deep Learning

E.S.Madhan[1], Arihant Kaushik[2],Rohit Raju[2]
[1]School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India.
[2] Department of Computational Intelligence, SRM Institute of Science and Technology, Kattankalatur, India.

Article Info

Article history:

Received August 14, 2022
Revised September 09, 2022
Accepted September 15, 2022

Keywords:

Cloud Computing
Computer Vision
Deep Learning
Micro Services
Machine Learning
MobileNetV2

ABSTRACT

Image processing has been getting great attention recently in the field of machine learning and deep learning. This technique can be used to process an image in such a way that the computer understands the features of the image and classifies it. Our study focuses on building an efficient CNN model to predict the breed of the dog using its image, giving the best accuracy possible with the least amount of computing resources involved. This CNN model is deployed on cloud service, Google App Engine which identifies certain characteristics or features in an image such as the paw, nose, stout, and ears of a dog, employing a dataset containing 10222 images of different dog breeds or classes of dogs and opening a wide scope for future developments.

Corresponding Author:

E.S. Madhan
School of Computer Science and Engineering
Vellore Institute of Technology
Vellore.
India
Email: esmadhan@gmail.com

1. INTRODUCTION

Image Processing is a process in which some algorithms are applied to digital images to enhance them or to extract some useful information from them [1]. During Image Processing, the image is provided as an input and useful information regarding the features of the image is obtained as output [12, 13]. Image processing is normally performed in three steps:

1. Importing the image
2. Analyzing and manipulating of image
3. Obtaining the output regarding features and information which can be used to study the image.

1.1. Keypoint Detection

Keypoints are the features of an image which are used to identify or recognize certain characteristics of an image. For example, the features of dogs like paws, ears, nose and more are used to identify the breed of dog in an image. The process of identifying such features is called Keypoint Detection which is performed by Deep Learning models using algorithms like SIFT descriptors, etc. [2]

1.2. Deep Learning

Deep Learning is a subset of machine learning with some layers of neural networks [17,18] which have the potential to mimic a human mind. Ideally, a deep learning model consists of three layers: The Input Layer, The Hidden Layer, and The Output Layer, although more layers could be added as per the requirements. The purpose of neural networks is to stimulate the human brain and help the model learn features and classify them. Convolutional Neural Network (CNN) is a deep learning algorithm that is commonly used for image classification problems. [23] Convolution is a mathematical operation where the operation on two functions produces a third function that defines how the shape of one is affected by another. [3] [4]

The image is first processed and converted into a pixel matrix, following which the matrix is passed to a CNN model. CNN model [15] has several layers, the first layer the is input player which usually extracts the features from an image using SIFT descriptors and the output is sent to consecutive layers where more complex features are recognized. The *activation function* in CNN makes the decision of which neuron to activate or not, by calculating the sum of the weights and biases together [16]. A model without an activation function will only serve as a linear model [14] which cannot predict data or values in complex predictions. Types of Activation functions are:

- Sigmoid Function
- Tanh Function
- ReLu Function

1.3. MobileNetV2 Model

MobileNets are small, low-power, low-latency models parameterized to meet the resource constraints of a variety of use cases. MobileNetV2 improves the performance of MobileNets on multiple benchmarks. Earlier, MobileNetV1 used depth-wise [24] separable convolutions whereas MobileNetV2 is built on top of it and provides linear bottlenecking in between the layers to prevent non linearity to destroy the performance of the model [5].

1.4. Microservices

Microservices also called microservice architecture is an architectural style which helps in structuring different components or features of an application as separate independent services. Such a structure helps in allowing continuous deployment, preventing bottlenecks in database and improving business capabilities along with allowing scalability at a high level where each service can operate independently without affecting the performance of other services.

To understand why microservice architecture is better than monolithic architecture, we need to first understand the latter and it’s disadvantages due to the current software demands.[1] Microservices are better than monolithic architecture where all the processes are tightly coupled and run as a single service [26]. Here, one service’s unavailability or scalability status can affect the same for other services. This kind of a structure was prevalent a decade ago. But now the increased demands and increased complexity of code in addition to more importance being given to code reusability and better code understanding this structure is no longer relevant and microservice architecture aptly aims to solve these drawbacks. Nevertheless, microservices tend to solve this problem. [6] [7]

1.5. Cloud computing:

Cloud computing is a way of providing services to the customers on demand. Different available cloud services are Microsoft Azure, Amazon’s AWS, and Google Cloud Platform [8]. Earlier companies used to deploy their applications on physical servers at their centers. The main drawback of this approach was that the scaling and maintenance costs were high and difficult to do because, in order to add additional resources or detaching a malfunctioning resource the companies used to shut down the servers, leading to losses during downtimes. In order to tackle these issues, the concept of serverless computing was introduced where human interactions were kept minimum to maintain and scale the server. [9]

The different types of services provided by cloud providers are:

● Infrastructure as a service: Where the customer can demand the infrastructures like compute engines, networks with firewall rules, databases etc.

● Platform as a service: Where customers can solely focus on code and the resources on the basis of the dependencies required which is done automatically (handled by the platform).

● Software as a service: Where customers need not install the software on their system, they can directly use the software on a browser via the internet.

The biggest advantage of Cloud computing is that it is cheap and flexible than hosting your application on physical servers because payment is done only on use basis. [10]

2. SYSTEM ARCHITECTURE AND DESIGN

2.1. Requirements

In order to train the model and learn about different features of dogs of various breeds, a sufficiently large dataset is needed so it learns the features of the dataset and not simply overfits. Therefore, the dataset provided by Kaggle was used, which is an open source dataset and is present on ImageNet. This is a huge database containing images of almost everything and was initially provided by Stanford. For our study, 10222 images of 120 different breeds have been used. The model called mobileNetV2 was used which is a new model that has the ability to extend its use in mobile applications as well. It is a CNN model [21] which is 52 layers deep and an initial fully convolutional layer with 32 filters along with 19 residual bottleneck layers following them.

The model has a prerequisite image size of 224 * 244 * 3 pixels (244 represents 244 pixels for length and breadth respectively; 3 represents the color channel) and commonly used color channel in processing images is RGB (Red, Green, Blue)

The software requirements and specifications for this study include:

● Python 3.x
● Jupyter
● TensorFlow library ver 2.x
● TensorFlow Hub
● Numpy
● Pandas
● Skskit ML library
● Google Cloud Platform
● Flutter
● Flask

2.2. Processing layer:

The images need to be preprocessed before feeding them to the model, because a model cannot recognize raw images. The images have to be converted into in the digital, analog form or numerical form. In our study, one such preprocessed layer has been added to convert the images to its resultant tensor. A tensor is the numerical equivalent of an image resized to have the prerequisite size of the model which can be understood by the computer.

2.3. Modelling Structure:

MobileNetV2 has 2 convolutional blocks: the first is a residual block of stride 1 and the second is stride 2 for the purpose of downsizing. Both convolutional blocks contain 3 layers:

● The First layer is a 1x1 convolutional layer that uses ReLU6. ReLU6 is known for its robustness when used with low precision calculations or computations. The usage of the 1x1 convolution layer is to expand the low-dimensional input feature map to a higher-dimensional space suited to non-linear activations.

● Second layer is a depth-wise convolution layer which uses 3x3 kernels. It applies one convolutional filter per input channel and performs lightweight filtering. Thus, it achieves spatial filtering of the tensor of higher dimensions.

● Lastly, the third layer is a 1x1 convolutional layer without non-linearity. It is because if we use ReLU again it will only be a linear classifier on the non-zero volume part of the output domain.

2.4. Dropout layers:

Overfitting is a phenomenon where a model trains perfectly with the training data and the activation function is too closely aligned to the data points. In simple words, the model tends to learn the details and noise of the training data to such an extent that it is unable to predict accurately when given data outside of the training data's domain. The model tends to learn the slightest idiosyncrasies of the training data. So, whenever a data is fed which has even the slightest change of detail, like lighting or position or a different angle, the performance of the model will get affected.

To avoid overfitting, dilution has been added or more popularly known as dropout layers. They prevent complex co-adaptation of the model on the training data. Dropouts randomly set the outgoing edges of hidden layers to 0. The dropout layers require a rate i.e they set the values of the edges to 0 at the rate of rate. The edges that are not set to 0 are scaled up to 1/(1-rate).

Dropout layers are added in accordance to the validation accuracy of the model. In this project, the maximum validation accuracy which could be achieved was with 2 dropout layers.

2.5. Augmentation layers:

As discussed above, preventing the model from overfitting is very crucial. There is another way to add randomized features on the existing dataset which can make the model perceive the image as a different image and learn its features, but accurately. The augmentation layers can be added which could provide diversity to our training dataset by adding random but realistic transformations to the dataset. Such transformations can be flipping, rotating, resizing and scaling. We can add random horizontal and vertical flips and even rotate the images to create a transformed image.

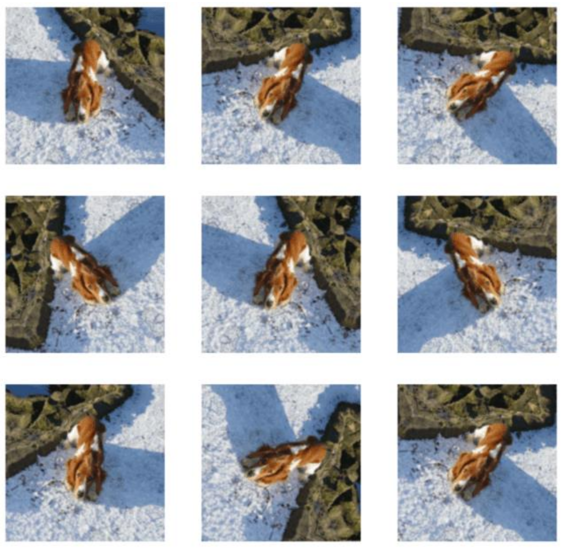


Figure 1. The image has been flipped horizontally and rotated by an angle of 0.2

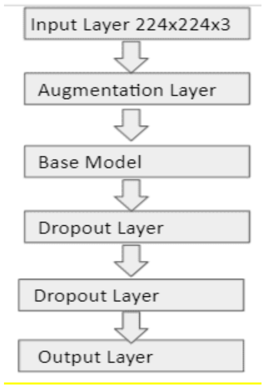


Figure 2.Block diagram of our proposed model where base model is MobileNetV2

2.6. Cloud Architecture:

Flask framework is a python framework used for making web applications or creating microservice. The model was loaded in the Flask application which exposed the '/prediction' api endpoint which takes images using multipart requests and returns prediction in Json format to the front end. The focus of the study was more on model optimization and deployment rather than the infrastructure needed. Since, the model was optimized multiple times it needed to be continuously deployed. So, the model was deployed on GAE which is a PaaS providing auto scaling and easy deployment.

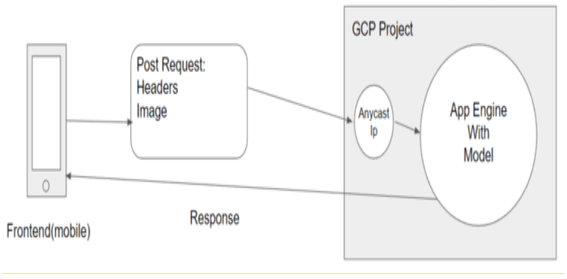
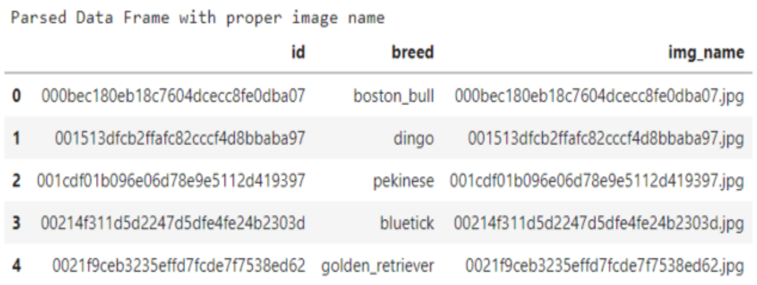


Figure 3.Cloud architecture

3. METHODOLOGY

3.1. Dataset

There are 10222 images of 120 breeds of dogs required to train the model was collected. The dataset was initially separated as images and labels, subsequently, associating each image with its corresponding label. A data frame was created where 3 columns represented the image id, location, and its label. For testing, a separate dataset was provided which was not exposed to our model at the time of training.



```
Parsed Data Frame with proper image name
```

	id	breed	img_name
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull	000bec180eb18c7604dcecc8fe0dba07.jpg
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo	001513dfcb2ffafc82cccf4d8bbaba97.jpg
2	001cdf01b096e06d78e9e5112d419397	pekinese	001cdf01b096e06d78e9e5112d419397.jpg
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick	00214f311d5d2247d5dfe4fe24b2303d.jpg
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever	0021f9ceb3235effd7fcde7f7538ed62.jpg

Figure 4. (a) Data frame containing raw training data (before preprocessing)

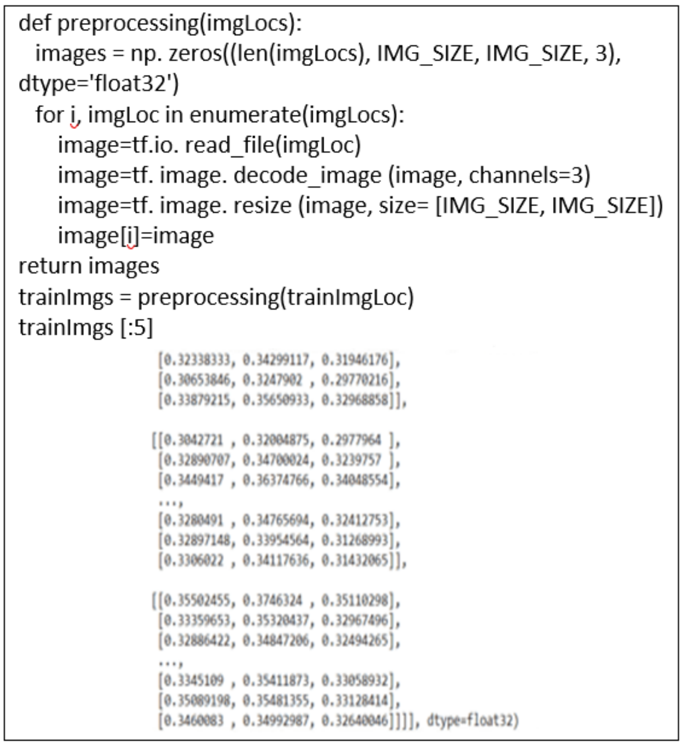
Figure 5. Golden retriever

3.2. Preprocessing Data

3.2.1. Converting Images to Matrices

For the model to extract features and classify the images based on the features, the data needs to be preprocessed. Deep learning models [22] recognize the images as a matrix or grid of pixels from which Key points (points of interest) are determined to extract features. Similarly, our data was converted into numpy matrices and the images were resized into 224 * 224 * 3 matrices.

[19, 20] Colored pictures were chosen instead of greyscale because few dogs are very similar physically but their color is one of the identification features.



```
def preprocessing(imgLocs):
  images = np. zeros((len(imgLocs), IMG_SIZE, IMG_SIZE, 3),
dtype='float32')
  for i, imgLoc in enumerate(imgLocs):
     image=tf.io. read_file(imgLoc)
     image=tf. image. decode_image (image, channels=3)
     image=tf. image. resize (image, size= [IMG_SIZE, IMG_SIZE])
     image[i]=image
return images
trainImgs = preprocessing(trainImgLoc)
trainImgs [:5]
```

```
                    [0.32338333, 0.34299117, 0.31946176],
                    [0.30653846, 0.3247902 , 0.29770216],
                    [0.33879215, 0.35650933, 0.32968858]],

                   [[0.3042721 , 0.32004875, 0.2977964 ],
                    [0.32890707, 0.34700024, 0.3239757 ],
                    [0.3449417 , 0.36374766, 0.34048554],
                    ...,
                    [0.3280491 , 0.34765694, 0.32412753],
                    [0.32897148, 0.33954564, 0.31268993],
                    [0.3306022 , 0.34117636, 0.31432065]],

                   [[0.35502455, 0.3746324 , 0.35110298],
                    [0.33359653, 0.35320437, 0.32967496],
                    [0.32886422, 0.34847206, 0.32494265],
                    ...,
                    [0.3345109 , 0.35411873, 0.33058932],
                    [0.35089198, 0.35481355, 0.33128414],
                    [0.3460083 , 0.34992987, 0.32640046]]]], dtype=float32)
```

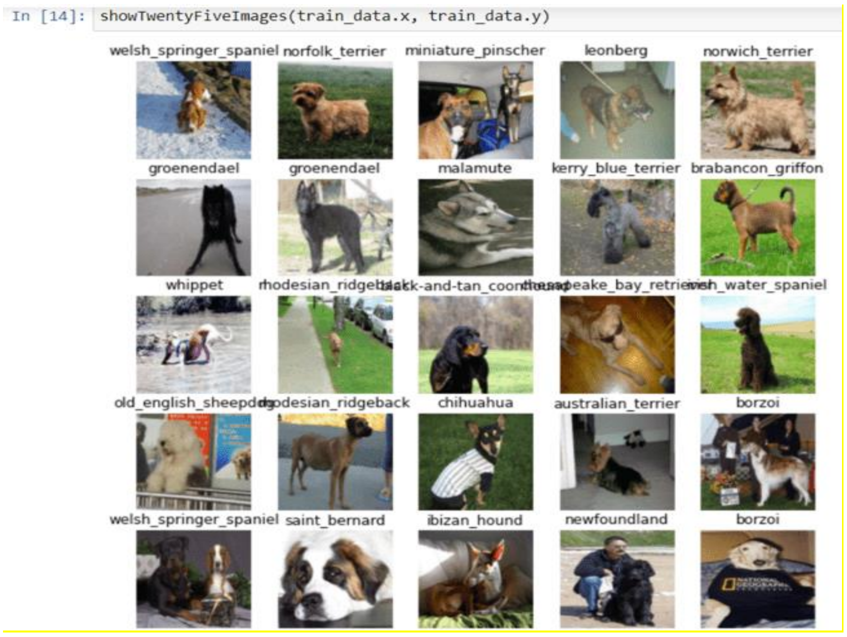
Figure 6. Images in the form of numpy matrices

3.2.2. Converting Labels into Boolean Matrix

The labels were converted into Booleans, because categorical cross entropy works on multiple class classification and it basically works on possibility of answer based on probability distribution. Hence, the boolean matrix gives yes or no relation for each class and encodes the labels for loss function to work with.

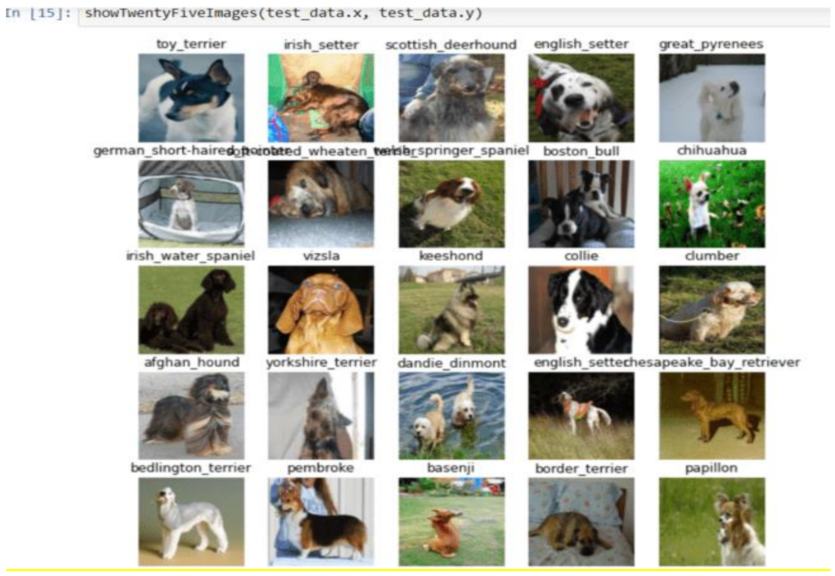
3.2.3. Splitting Data into Train Dataset and Validation Dataset

Our data was split into a training dataset and validation dataset. The ratio of train into validation was 80:20



```
In [14]: showTwentyFiveImages(train_data.x, train_data.y)
```

Figure 7. Training Data



```
In [15]: showTwentyFiveImages(test_data.x, test_data.y)
```

Figure 8. Validation data

3.2.4. Preparing Tensors

Tensor is a mathematical function which describes the relationship between nonlinear objects into a vector form. Our images and labels were converted into single tensors so that model could understand the provided information from which it is to study.

3.3 Preparing the Model

3.3.1 Attempting to Build Cnn Model from Scratch

Originally, the model had Conv2D as the first layer with ReLu activation function for input layer. The input layer was followed by Max Pooling layer of size 2x2. To sum it up, the structure of our original model is described in the following image:

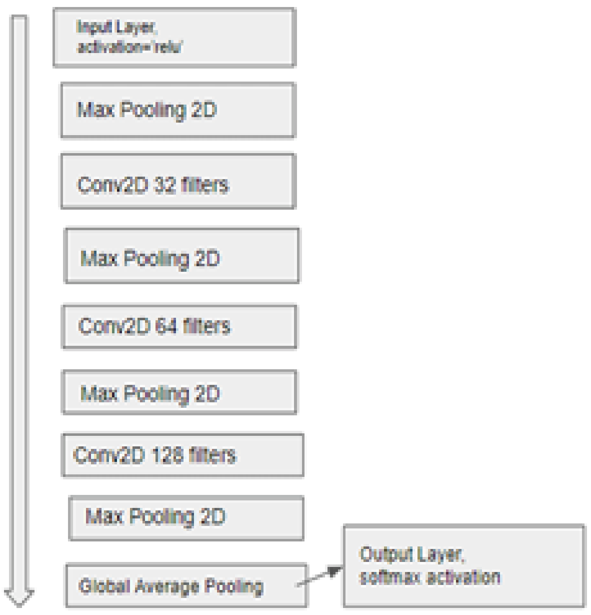


Figure 9. Initial Model

The mentioned model was built with RMSprop optimizer with learning rate of 0.001 and rho=0.9. The loss function used was sparse categorical cross entropy. The accuracy achieved was 64% with loss of 0.84.

3.3.2. Transfer Learning

Transfer learning is a technique which is most commonly used in machine learning in which pre-trained model's layers are used to reduce the training time and minimize the generation of errors. The model used for transfer learning was mobileNetV2. All the trainable layers were frozen. The top layer was replaced with our input layer and was kept trainable. Then, a 2D convolution layer with 128 filters along with max pooling layer followed by a dense (fully connected) layer for output with the shape equal to the number of breeds and 'softmax' activation function was created.

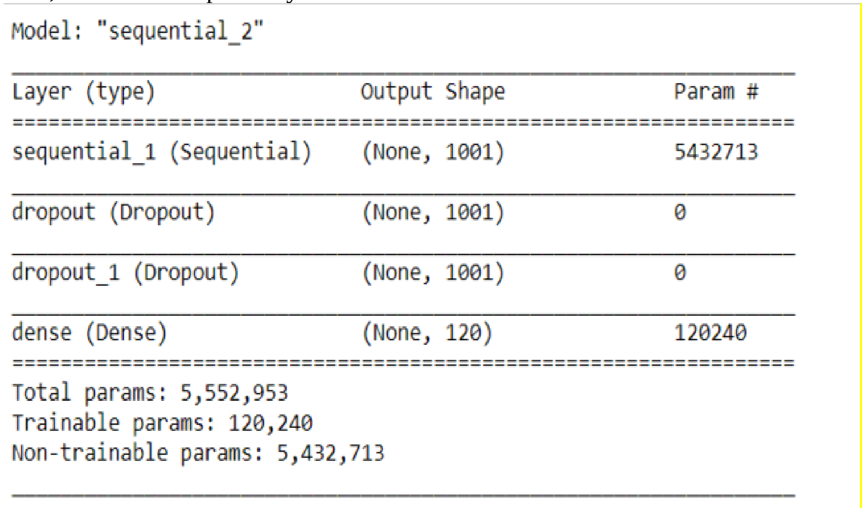
The mentioned model was compiled with the original model's configuration RMSprop optimizer (rho=0.9 and learning rate=0.001) and sparse categorical cross entropy. The new accuracy was 77% with loss of 0.84 and the training time was longer than the previous approach. The previous time was 60 minutes while new time was 90 minutes. It was observed that after a certain poch, the accuracy was fixed at 77% along with the loss, hence the model was over fitting.

In order to overcome over fitting, new changes were introduced. For simplicity, this model has been considered as a base model.

3.3.3. Fine Tuning the Model

The same model was compiled with new configuration. The optimizer was replaced with Adam Optimizer and the loss function was replaced with categorical cross entropy. Moreover, labels were converted to Boolean matrix to indicate the probability matrix. To avoid overfitting, we augmented the input images at random angels and used them as an additional input for the model.

A new sequential model was created with first layer as input to which augmentation layer was added, following which the next layer (base model) got output from the augmentation layer as input and to dilute the model further, two more drop out layers were added.



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential_1 (Sequential)    (None, 1001)              5432713
_________________________________________________________________
dropout (Dropout)            (None, 1001)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1001)              0
_________________________________________________________________
dense (Dense)                (None, 120)               120240
=================================================================
Total params: 5,552,953
Trainable params: 120,240
Non-trainable params: 5,432,713
_________________________________________________________________
```

Figure 10. Abstract of our model where sequential model 1 is base model input layer ->augmentation layer->base (MobileNetV2)

3.4. Training the Model

To train the model, Adam optimizer was used with 0.001 Learning Rate and Categorical Cross Entropy as the loss function. A call back function for early stopping was used when the model had reached the peak accuracy and was not learning any further for three epochs. Each epoch was 128 steps and training completed with a batch size of 64.

3.5. Preparing the Model for Deployment

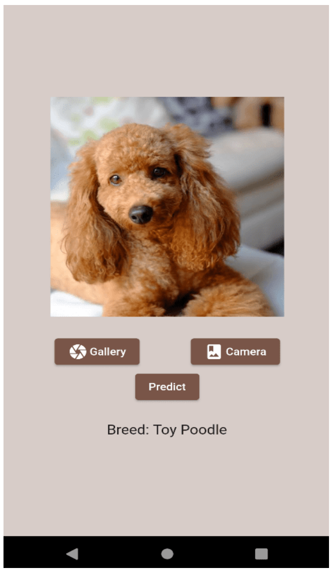
For deployment, the model was saved in. pb format along with the weights. The flask framework was used to write a backend script which loaded the models and took the input image through multipart request. Thereafter, it was stored in a temporary location and then preprocessed. The loaded model was used to generate the prediction and return it in json format.

3.6. Deployment of the Model

The flask code was pushed to a GitHub repository along with the app.yaml containing the dependencies required to run the application. The GitHub repository was used as a pipeline to deploy the code on Google App Engine with F4 instances having 1GB memory. The technique of branch and pruning was used to cut down unnecessary weights and reduce the size of the model to make it run more efficiently under 1 GB [18].

3.7. Front-End (client)

For the front-end, the mobile app was made using the Flutter framework in which the image can be taken using the mobile camera or can be selected from the gallery. The app sends the image using a multipart request to the flask app deployed on GAE, through 'API endpoint'.

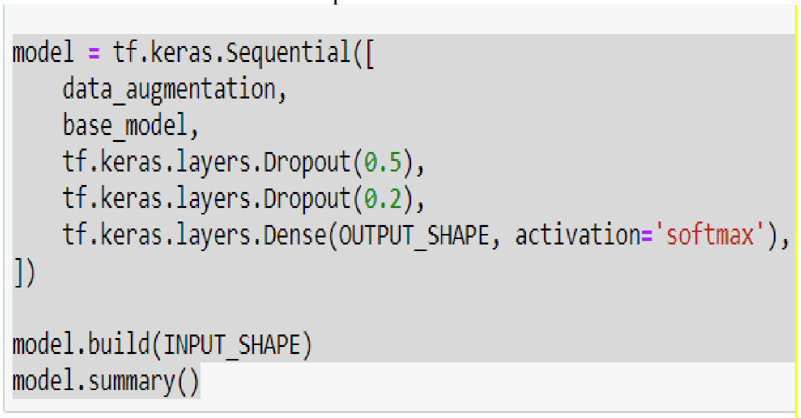


Breed: Toy Poodle

Figure 11. Prediction of image using mobile app

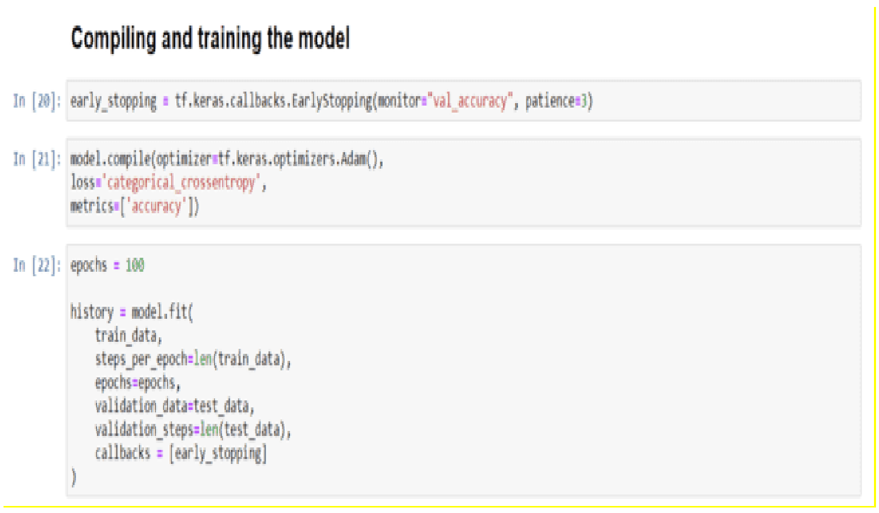
3.8. Coding and Testing

Lastly, TensorFlow API was used to implement the model.



```
model = tf.keras.Sequential([
    data_augmentation,
    base_model,
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(OUTPUT_SHAPE, activation='softmax'),
])

model.build(INPUT_SHAPE)
model.summary()
```

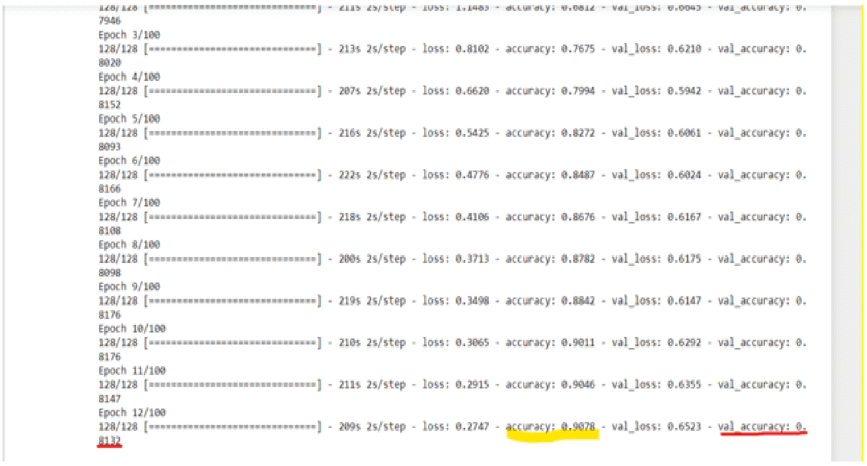
Figure 12. Code Snippet of Model



Compiling and training the model

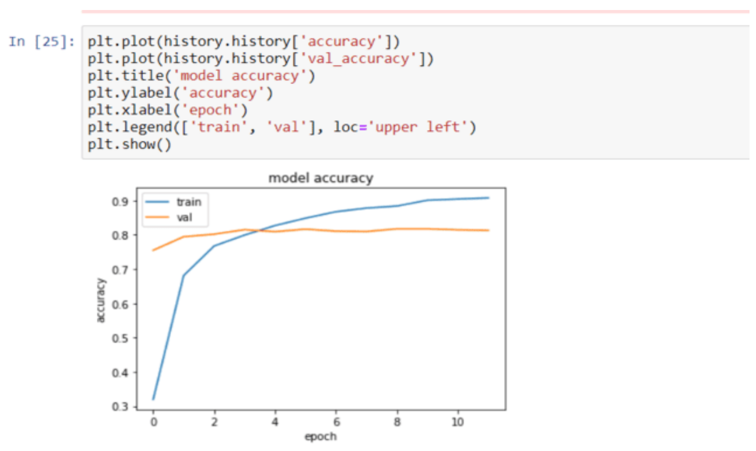
```
In [20]: early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_accuracy", patience=3)

In [21]: model.compile(optimizer=tf.keras.optimizers.Adam(),
         loss='categorical_crossentropy',
         metrics=['accuracy'])

In [22]: epochs = 100

         history = model.fit(
             train_data,
             steps_per_epoch=len(train_data),
             epochs=epochs,
             validation_data=test_data,
             validation_steps=len(test_data),
             callbacks = [early_stopping]
         )
```

Figure 13. Training configuration of model as per section IV



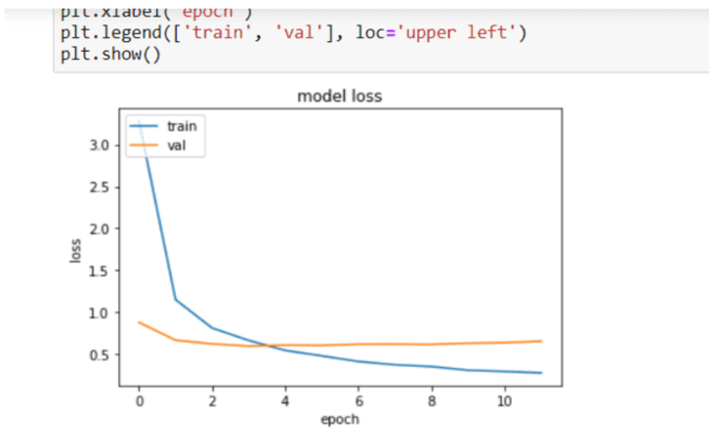
```
128/128 [==============================] - 211s 2s/step - loss: 1.1483 - accuracy: 0.6812 - val_loss: 0.6645 - val_accuracy: 0.
7946
Epoch 3/100
128/128 [==============================] - 213s 2s/step - loss: 0.8102 - accuracy: 0.7675 - val_loss: 0.6210 - val_accuracy: 0.
8020
Epoch 4/100
128/128 [==============================] - 207s 2s/step - loss: 0.6620 - accuracy: 0.7994 - val_loss: 0.5942 - val_accuracy: 0.
8152
Epoch 5/100
128/128 [==============================] - 216s 2s/step - loss: 0.5425 - accuracy: 0.8272 - val_loss: 0.6061 - val_accuracy: 0.
8093
Epoch 6/100
128/128 [==============================] - 222s 2s/step - loss: 0.4776 - accuracy: 0.8487 - val_loss: 0.6024 - val_accuracy: 0.
8166
Epoch 7/100
128/128 [==============================] - 218s 2s/step - loss: 0.4106 - accuracy: 0.8676 - val_loss: 0.6167 - val_accuracy: 0.
8108
Epoch 8/100
128/128 [==============================] - 200s 2s/step - loss: 0.3713 - accuracy: 0.8782 - val_loss: 0.6175 - val_accuracy: 0.
8098
Epoch 9/100
128/128 [==============================] - 219s 2s/step - loss: 0.3498 - accuracy: 0.8842 - val_loss: 0.6147 - val_accuracy: 0.
8176
Epoch 10/100
128/128 [==============================] - 210s 2s/step - loss: 0.3065 - accuracy: 0.9011 - val_loss: 0.6292 - val_accuracy: 0.
8176
Epoch 11/100
128/128 [==============================] - 211s 2s/step - loss: 0.2915 - accuracy: 0.9046 - val_loss: 0.6355 - val_accuracy: 0.
8147
Epoch 12/100
128/128 [==============================] - 209s 2s/step - loss: 0.2747 - accuracy: 0.9078 - val_loss: 0.6523 - val_accuracy: 0.
8132
```

Figure 14. Accuracy of 90 and validation accuracy of 81



```
In [25]: plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.title('model accuracy')
         plt.ylabel('accuracy')
         plt.xlabel('epoch')
         plt.legend(['train', 'val'], loc='upper left')
         plt.show()
```

Figure 15. Accuracy Graph



```
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```

Figure 16. Loss graph

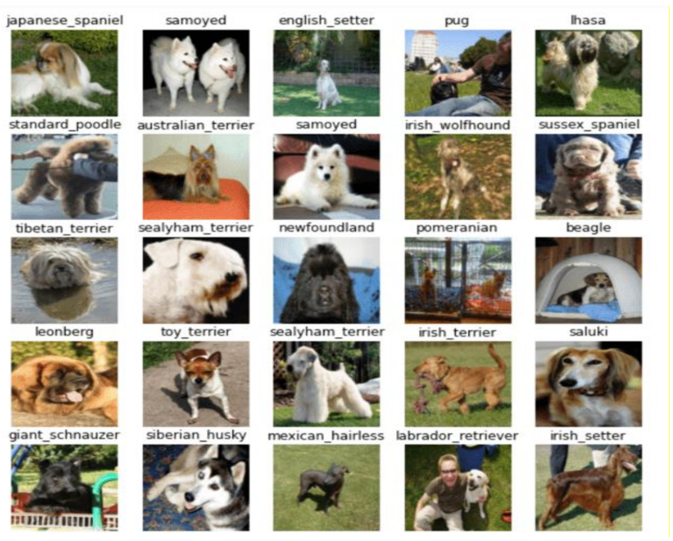


Figure 17. First 25 Predictions by the model

4. RESULTS AND DISCUSSION

Initially, the original model without any transfer training as giving an accuracy of 64%. So, in order to increase the accuracy, the transfer learning was applied and to improve the accuracy further Adam optimizer, dropout layers and augmentation layers were used. With the mentioned changes, the model was able to predict the dog's breed with an accuracy of 81%. The recorded training accuracy of the model was 90% along with the validation accuracy of 81%.

The accuracy of the model could not be increased any further, because the dataset itself was not large enough for the model to provide a higher accuracy. The portion of images used were sufficient to derive the results, given the resources available. The model was successfully deployed on cloud (GAE) and the Image was sent to it using a mobile app as a front-end and the communication was done through 'API endpoint'. The front-end sent the image through multipart request and received prediction in the form of json from the flask app deployed on GAE. Blurred images or images taken during night time, having less light were not represented in our results.

Table 1. Comparison of the models created in our study

Model	Optimizer	Loss Function	Accuracy	Time taken for training (min)
Basic CNN	RMSProp (rho=0.9)	sparse categorical cross entropy	64%	60
Model Combined with MobileNetV2	RMSProp (rho=0.9)	sparse categorical cross entropy	77%	90
Fine Tuned Model	Adam Optimizer	Categorical Cross entropy	81%	60

Table 2. Comparison between original size and compressed size

Model version	Memory size
Original Memory Taken by Model	1.2 GB
After branch and pruning optimization	965

5. CONCLUSION

This study shows that CNN can be used to perform image classification problems using some data polishing techniques and fine-tuning. CNN has the ability to properly identify facial key points and can be employed in other image classification tasks as well, showing the wide scope in future applications. This work also reveals that CNN models can be deployed on cloud services as a microservice to provide much wider applications as a single model deployed on the cloud can be used by multiple clients at the same time. The accuracy of the model can be improved by more research in this area, and by working on a larger dataset. Moreover, a model can be created where the dogs could be classified on major factors like height, color, age and similarities between breeds. Lastly, a similar model could be prepared for distinction between plants, flowers, fruits and other animals.

REFERENCES

[1] Baoliang, Wang; A Automatic Animal Species Identification Based on Camera Trapping Data; thesis submitted in partial fulfillment of the requirements for the degree of Master of Science Department of Computing Science University of Alberta;2016.

[2] Willi M, Pitman RT, Cardoso AW, et al. Identifying animal species in camera trap images using deep learning and citizen science. Methods Ecol Evol, British Ecological Society;10:80–91; 2019.

[3] Tabak MA, Norouzzadeh MS, Wolfson DW, et al. Machine learning to classify animal species in camera trap images: Applications in ecology. Methods Ecol Evol. British Ecological Society; 10:585–590; 2019

[4] Sakshi Indoliaa, Anil Kumar Goswamib , S. P. Mishrab , Pooja Asopa, Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach.

[5] J. Liu. et al. Dog breed classification using part localization. Computer Vision: ECCV 2012, pages 172–185, 2012.

[6] N. Gupta, K. Anantharaj and K. Subramani, "Containerized Architecture for Edge Computing in Smart Home : A consistent architecture for model deployment," 2020 International Conference on Computer Communication and Informatics (ICCCI), pp. 1-8, 2020.

[7] M. Villamizar et al., "Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud," 2015 10th Computing Colombian Conference (10CCC), pp. 583-590, 2015.

[8] Dylan Rhodes, Automatic Dog Breed Identification CS231n, Stanford University; 2020.

[9] IOSR Journal of Computer Engineering (IOSRJCE) ISSN : 2278-0661 Volume 1, Issue 1; PP 38-45; 2012.

[10] Paul, P. K., & Ghose, M. K. Cloud Computing: possibilities, challenges and opportunities with special reference to its emerging need in the academic and working area of Information Science. Procedia engineering, 38, 2222-2227; 2012.

[11] S.K. Sowmya et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (3), 4477-4480; 2014.

[12] Kumar, N., Belhumeur, P. N., Biswas, A., Jacobs, D. W., Kress, W. J., Lopez, I. C., & Soares, J. V. Leafsnap: A computer vision system for automatic plant species identification. In European conference on computer vision, Springer, Berlin, Heidelberg; pp. 502-516; 2012

[13] Kovásznay, L. S., & Joseph, H. M. Image processing. Proceedings of the IRE, 43(5), 560-570; 1955

[14] Boonsivanon, K., & Meesomboon, A. IKDSIFT: an improved keypoint detection algorithm based-on SIFT approach for non-uniform illumination. Procedia Computer Science, 86, 269-272; 2016.

[15] Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629; 2018.

[16] Menglong Zhu Andrey Zhmoginov Liang-Chieh Chen Google Inc. MobileNetV2: Inverted Residuals and Linear Bottlenecks Mark Sandler Andrew Howard; 2019.

[17] Willi, M., Pitman, R. T., Cardoso, A. W., Locke, C., Swanson, A., Boyer, A., ... & Fortson, L. Identifying animal species in camera trap images using deep learning and citizen science. Methods in Ecology and Evolution, 10(1), 80-91; 2019.

[18] Zhang, Ziming, Yuanwei Wu, and Guanghui Wang. "Bpgrad: Towards global optimality in deep learning via branch and pruning."Proceedings of the IEEE Conference on Computer Vision and Pattern recognition; 2018.

[19] Zalan R ´ aduly ´ Babes ̦–Bolyai, Csaba Sulyok Babes ̦–Bolyai, Zsolt Vadaszi, Attila Zolde, Dog Breed Identification Using Deep Learning; 2018

[20] Mohamed Sultan M1, Naveen S2, Praveen Kumar C3, Arun Manicka Raja; Dog Breed Identification Using Convolution Neural Network and Web Scraping M4 Department of Computer Science and Engineering, RMK College of Engineering and Technology(RMKCET) R.S.M. Nagar, Puduvoyal, Thiruvallur Dist. 601206, Tamilnadu, India Paper ID: SR20307193747 DOI: 10.21275/SR20307193747 Volume 9 Issue 3, 2020.

[21] Tibor TRNOVSZKY, Patrik KAMENCAY, Richard ORJESEK, Miroslav BENCO, Peter SYKORA, Animal Recognition System Based on Convolutional Neural Network Department of multimedia and information-communication technologies, ADVANCES IN ELECTRICAL AND ELECTRONIC ENGINEERING; 2017.

[22] Whitney LaRow, Brian Mittl, Vijay Singh, Dog Breed Identification; 2018.

[23] Top 10 Deep Learning Algorithms You Should Know in 2022 (simplilearn.com)

[24] Mr. Renji Roy Isac, A Study on identification of dog breeds through multi-class classification using Deep Learning techniques Dissertation submitted in part fulfilment of the requirements for the degree of [MSc in DATA ANALYTICS] at Dublin Business School; 2018.

[25] Özkan İNİKa*, Bülent TURAN, Classification of Animals with Different Deep Learning Models Department of Computer Engineering, Gaziosmanpaşa University, Tokat/ Turkey. JOURNAL OF NEW RESULTS IN SCIENCE (JNRS); 2018.

[26] Zhang, Ziming, Yuanwei Wu, and Guanghui Wang. "Bpgrad: Towards global optimality in deep learning via branch and pruning." Proceedings of the IEEE Conference on Computer Vision and Pattern recognition. 2018.

BIOGRAPHIES OF AUTHORS

E. S. Madhan received the bachelor’s degree in computer science from Anna University, Chennai in 2011 and a Master’s degree in Computer Science and Engineering from Anna University, Tamil Nadu, Chennai, in 2013 and the Doctoral in Computer science and Engineering from Anna University, Tamil Nadu, Chennai, in 2019. He is currently working as an Assistant Professor at the Vellore Institute of Technology, Vellore. His research interest includes Deep Learning, machine learning, big data analytics, Cloud Computing, and the Internet of Things. He has published articles in National and International Indexed journals, including SCI, WoS, and SCOPUS. He is a reviewer of various springer and Elsevier Journals. He can be contacted at email: esmadhan@gmail.com

Arihant Kaushik studying Final year in the Department of Computer Science and Engineering at SRM Institute of Science and Technology, Chennai. He presented various papers at national and International Conferences. He can be contacted at email: arihantkaushik@gmail.com

Rohit Raju studying Final year in the Department of Computer Science and Engineering at SRM Institute of Science and Technology, Chennai. He participated in Hackathon projects. He presented various papers at national and International Conferences. He can be contacted at email: rohitraju@gmail.com.