DOG BREED CLASSIFICATION

Mini Project submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Kasarla Sahith Reddy (221710308025)

Under the esteemed guidance of

Mr. D. Srinivasa Rao

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM

(Deemed to be University)

HYDERABAD

DECEMBER 2020

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM

(Deemed to be University)

DECLARATION

We hereby declare that the mini project entitled "**DOG BREED CLASSIFICATION**" is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 12-12-2020

Kasarla Sahith Reddy (221710308025)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that Mini Project entitled "**DOG BREED CLASSIFICATION**" is submitted by Kasarla Sahith Reddy (221710308025) in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering. The Mini Project has been approved as it satisfies the academic requirements.

Mr. D. Srinivasa Rao
Assistant Professor
Dept. of Computer
Science and Engineering

Prof. S. Phani Kumar
Head of the Department
Dept. of Computer
Science and Engineering

ACKNOWLEDGMENT

Our Summer Internship would not have been successful without the help of several people. We would like to thank the personalities who were part of our seminar in numerous ways, those who gave us outstanding support from the birth of the seminar.

We are extremely thankful to our honourable Pro-Vice Chancellor, **Prof. N. Siva Prasad** for providing necessary infrastructure and resources for the accomplishment of our seminar.

We are highly indebted to **Prof. N. Seetharamaiah**, Principal, School of Technology, for his support during the tenure of the seminar.

We are very much obliged to our beloved **Prof. S. Phani Kumar**, Head of the Department of Computer Science & Engineering for providing the opportunity to undertake this seminar and encouragement in completion of this seminar.

We hereby wish to express our deep sense of gratitude to **Dr. S Aparna**, Assistant Professor, Department of Computer Science and Engineering, School of Technology and to **Mr. D. Srinivasa Rao**, Assistant Professor, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the summer Internship.

We are also thankful to all the staff members of Computer Science and Engineering department who have cooperated in making our seminar a success. We would like to thank all our parents and friends who extended their help, encouragement and moral support either directly or indirectly in our seminar work.

Sincerely,

Kasarla Sahith Reddy (221710308025)

CONTENTS

ABSTRACT

Dog breed classification is essential for many reasons, particularly for understanding individual breed's conditions, health concerns, interaction behaviour and natural instinct. In the project the method is based on deep learning approach which is used in order to recognize their breeds. Dog Breed identification is a specific application of Convolutional Neural Networks. Though the classification of Images by Convolutional Neural Network serves to be efficient method, still it has few drawbacks. Convolutional Neural Networks requires a large amount of images as training data and basic time for training the data and to achieve higher accuracy on the classification. To overcome this substantial time we use Transfer Learning. In computer vision, transfer learning refers to the use of a pre-trained models to train the CNN. By Transfer learning, a pre-trained model is trained to provide solution to classification problem which is similar to the classification problem we have. In this project we are using pre-trained model InceptionV3 to train over our dataset which is covering different breeds of dogs. So, that it reduces time in training process and gives efficient output.

We have used Google's Inception-v3 model trained on 100,000 images covering 1000 different categories. We retrained the Inception model to classify the dog images, using the Tensorflow Library and achieved an overall 97.47% of accuracy.

LIST OF FIGURES

CHAPTER-1
INTRODUCTION

The World Canine Organization (FCI) is currently listing more than 300 officially recognised dog breeds. Over thousands of years, mankind has managed to create an impressive diversity of canine phenotypes and an almost uncanny range of physical and behavioural characteristics of their faithful four-legged friends. However, apart from cynology scholars, dog breeders and some proven dog lovers most people shrug their shoulders in a clueless gesture, when asked to name the breed of a randomly presented dog, at least when it is not exactly a representative of one of the most popular and well known breeds like Dachshund, German Shepard or pug. If you are one of the few people who finds it slightly embarrassing not being able to identify dogs like a cynologist, you are probably pleased to learn that there might be a technical solution. Because thankfully, the aspiring and astonishing field of Deep Learning and artificial neural networks provides powerful concepts and methods for addressing this sort of classification tasks.

In this project we will develop ideas for a dog identification using deep learning concepts. The software is intended to accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. We are going to use CNN with Transfer Learning.

1.1 MOTIVATION

Dog breed classification is a problem well defined in the project description, but to recap: majority of dogs are often difficult to classify by simply looking, and breed identification is important when rescuing dogs, finding them forever homes, treating them, and various other furry situations. This project classifies purebreds since mixed breeds are, as mentioned, often indistinguishable from purebreds, but the hope is that even classifying a purebred dog or a mixed breed one as one of the breeds it belongs to will help give more information about the dog's personality, full-grown size, and health. We were motivated by our love of puppies to choose this project. Breed predictions may also help veterinarians treat breed specific ailments for stray, unidentified dogs that needs medical care.

This problem is not only challenging but also its solution is applicable to other fine-grained classification problems. For example, the methods used to solve this problem would also help identify breeds of cats and horses as well as species of birds and plants - or even models of cars. Any set of classes with relatively small variation within it can be solved as a fine-grained classification problem. In the real-world, an identifier like this could be used in biodiversity studies, helping scientists save time and resources when conducting studies about the health and abundance of certain species populations. These studies are crucial for assessing the status of ecosystems, and accuracy during these studies is particularly important because of their influence on policy changes. Breed prediction may also help veterinarians treat breed specific ailments for stray, unidentified dogs that need medical care. Ultimately, we found dogs to be the most interesting class to experiment with due to their immense diversity, loving nature, and abundance in photographs, but we also hope to expand our understanding of the fine-grained classification problem and provide a useful tool for scientists across disciplines.

1.2 OBJECTIVE

In this project we will develop ideas for a dog identification app using deep learning concepts. The software is intended to accept any user-supplied image as input. If a dog is detected in the image, it will provide an estimate of the dog's breed. We are considering 7 dog breeds to classify. So, we are going to implement this project by using CNN with the help of transfer learning.

Developed a dynamic Deep Learning Model that accurately predicts the breed of a specific dog upon the feeding the Neural Network with the image containing a dog breeds, but the model prerogatively predicts the breed of the dog if the being specified in the image is a dog. The core theme of the project lies in the application of Convolutional Neural Networks where the images are been fed into the most complex network of the CNN's that eventually generates a dynamic inference upon the images fed to the network.

1.3 PROBLEM DEFINITION

This project is to identify dog breeds from images. This is a fine-grained classification problem: all breeds of Canis lupus familiaris share similar body features and overall structure, so differentiating between breeds is a difficult problem. Furthermore, there is low inter-breed and high intra-breed variation; in other words, there are relatively few differences between breeds and relatively large differences within breeds, differing in size, shape, and color. In fact, dogs are both the most morphologically and genetically diverse species on Earth. The difficulties of identifying breeds because of diversity are compounded by the stylistic differences of photographs used in the dataset, which features dogs of the same breed in a variety of lightings and positions.

This great variety poses a significant problem to those who would be interested in acquiring a new canine companion, however. Walking down the street or sitting in a coffee shop, one might see a friendly, attractive dog and wonder at its pedigree. In many situations, it is impossible to ask an owner about the breed, and in many cases, the owner themselves will be either unsure or incorrect in their assessment. Unless the dog falls into one of a few very widely known and distinctive breeds such as the golden retriever, Siberian husky, or daschund to name a few, it might prove difficult to identify ones ideal companion without a great deal of research or experience.

In this project we use use keras and tensorflow to build, train, and test a Convolutional Neural Network capable of identifying the breed of a dog in a supplied image. Success will be defined by high validation and test accuracy, with precision and recall scores differentiating between models with similar accuracy. This is a supervised learning problem, specifically a multiclass classification problem, and as such the solution may be approached with the following steps:

1. Amass labelled data. In this case, that means compiling a repository of images with dogs of known breeds.
2. Construct a model capable of extracting data from training images, which outputs data which may be interpreted to discern a breed of dog.
3. Train the model on training data, validate performance during training with validation data

4. Evaluate performance metrics, potentially return to step 2 with edits to improve performance
5. Test the model on test data

1.4 LIMITATIONS

- We had a total of 391 dog images, which we then sorted into 7 different breeds. Due to the small sample size, there were a limited number of training images per class, which made identifying smaller nuances between similar breeds more difficult.(ex. Norfolk vs Norwich Terriers)
- We found that multi-coloured dogs were the most difficult to predict and were often predicted wrong, most likely due to limited training images for their breed.
- Additionally, dogs proved to be a fine-grained image recognition subject because, in real-world, the plethora of mixed breed dogs and similarities between breeds.
- We need minimum of i5 and 8^{th} gen laptop to do this project and 8GB RAM, so it can be expensive for buying such laptop and this project is big and time taking. We need much time for implementation of such project.

1.5 APPLICATIONS

- Medical Purposes like treating dog according to its breed.
- Useful in finding legal ownership.
- Due to a large number of dogs, there are several issues such as population control, decreases outbreak such as Rabies and vaccination control. Because of this classification it can be very helpful.
- It can be used in pet centres.

CHAPTER-2

LITERATURE SURVEY

2.1 REFERENCE PAPER 1- STANFORD

This project uses computer vision and machine learning techniques to predict dog breeds from images. First, we identify dog facial keypoints for each image using a convolutional neural network. These keypoints are then used to extract features via SIFT descriptors and color histograms. We then compare a variety of classification algorithms, which use these features to predict the breed of the dog shown in the image. Our best classifier is an SVM with a linear kernel and it predicts the correct dog breed on its first guess 52% of the time; 90% of the time the correct dog breed is in the top 10 predictions.

Technical Solution Summary-To solve this fine-grained classification problem, we developed the analysis pipeline seen in figure 1. First, we trained a convolutional neural network on images of dogs and their annotated facial keypoints (right eye, left eye, nose, right ear tip, right ear base, head top, left ear base, and left ear tip). We used this network to then predict keypoints on an unseen test set of dog images. These predicted keypoints were then fed into a feature extraction system that used these keypoints to create more meaningful features from the image, which could later be used to classify the image. The primary features extracted were grayscale SIFT descriptors centered around the each eye, the nose, and the center of the face (the average of these 3 points). These features were then used for a variety of classification algorithms, namely bag of words, K-nearest neighbors, logistic regression, and SVM classifiers. These algorithms classify the images as one of 133 possible dog breeds in our dataset and complete the analysis pipeline.

Details-To best identify dog breeds, the first step of analysis is key point detection. Dog face keypoints are defined and annotated in the Columbia dataset as the right eye, left eye, nose, right ear tip, right ear base, head top, left ear base. An image representation of our analysis pipeline ear tip. Thus, the goal of this part of the analysis pipeline is defined as follows: given an unseen image from the testing set, predict the keypoints of the dog face as close as possible to the ground truth points in terms of pixels. Convolutional neural networks have been shown to perform quite well on a variety of image detection and classification tasks, including human

face detection, but previous literature on dog face detection had not used neural networks; hence, we decided to tackle a novel approach for dog face keypoint detection. To solve this problem, we trained a fully connected convolutional neural network on 4,776 training images, which was used to predict the keypoints of 3,575 testing images. Before training, the images were all scaled to be 128x128 pixels; furthermore, the pixel intensity values were scaled to be in the range [0,1]. The ground truth keypoints were also scaled accordingly. The neural network was constructed using the nolearn lasagne API. [10] It performed regression using a mean squared error loss function defined as: MSE = 1 n Xn i=1 (Yˆ i − Yi) 2 The network was trained using batches of 180 images for 4,000 epochs. In each batch, half of the images and their corresponding keypoints were flipped. This augmentation created a larger training set, which helped prevent the network from overfitting the data given. The architecture of the network went through several iterations, and the final construction.

Classification

Bag of Words Model

We decided to use a bag of words model for our base3 line because it is a simple model that is frequently used for object classification. The bag of words model ignores our detected facial keypoints and instead just finds a visual vocabulary based on the cluster centers of SIFT descriptors obtained from the training set of images. Because bag of words models perform better classification for objects with very high inter-class variability, we did not expect it to work very well for our fine-grained classification problem. For our bag of words model, we first used OpenCVs FeatureDetector and DescriptorExtractor to extract SIFT descriptors. We then used the K-means algorithm, from Pythons scipy library, to extract a visual vocabulary. We used this visual vocabulary to create feature histograms, preprocessed and scaled these histograms to fit a gaussian distribution, then used them to train an sklearn LinearSVC model.

SVM

The first real classification model we tried was an SVM. SVMs are commonly used machine learning models that perform well at multi-class classification. To date, few other supervised learning algorithms have outperformed SVMs, which is why we thought this would be a good place to start. We tried a few different types of SVMs in order to determine which worked best for our use case. The first we used was the same LinearSVC model from sklearn that we used for our bag of words model. We then tried a normal SVM with a linear kernel and a OneVsRestClassifier, all using the same sklearn library for consistency. All SVMs used the

grayscale SIFT features centered at facial keypoints and the color centers histogram calculated around the dogs face. In our experimental results section, you can see that the normal SVM with linear kernel outperformed the other two SVMs while keeping the features consistent.

Logistic Regression

As we learned in class, linear classifiers are a commonly used discriminative classifier for categorizing images. Unlike a standard linear regression problem, we have a finite number of discrete values (the various possible dog breeds) that our predicted value y can take on. To make our prediction, we would use the following hypothesis function: $h\theta(x) = g(\theta T x) = 1\ 1 + e{-}\theta T x$ Note, however, that the standard application of the sigmoid function for logistic regression is used for binary classification (where y can take on only two discrete values). As we have 133 possible breeds, we extrapolate the above logic to handle multiple classes, also known as multinomial logistic regression. Instead of having a single logistic regression equation, we have $J - 1 = 132$ equations, where J represents the total number of breeds (we only need $J - 1$ since one of the breeds will serve as a reference). Analogous to the standard logistic regression model, our model's probability distribution of response is multinomial instead of binomial, and we use standard Maximum Likelihood Estimation to predict the most likely breed. While such applications of logistic regression have been used before in fine-grained classification, our literature review showed no prior use in dog breed classification specifically. To implement, we used scikit-learn's linear model module in Python.

K-Nearest Neighbors

Finally, we also attempted a nonlinear discriminative classifier using a K-nearest neighbor classifier. A dog is classified by a majority vote of its neighbors, with the dog being assigned to the breed most common among its k nearest neighbors. As noted in lecture, this method depends on training set being large enough to generate enough meaningful votes and does not always produce the optimal results. As such, we anticipated one of the linear classifiers performing better.

Conclusion: Overall, we consider our results to be a success given the high number of breeds in this fine-grained classification problem. We are able to effectively predict the correct breed over 50% of the time in one guess, a result that very few humans could match given the high variability both between and within the 133 different breeds contained in the dataset.

2.2 REFERENCE PAPER 3- DYLAN RHODES CS231N, STANFORD UNIVERSITY

Approaches

First Approach -As a first attempt, I reimplemented Liu's facial detection algorithm with a few minor modifications, since the breed identification model does not require the exact location of the keypoints, just that of the face as a whole, along with its orientation and scale. First, the average displacement of the nose and eyes relative to the center of the face, here defined as their mode, the orientation of the line connecting the eyes, and the interocular distance were all calculated over the dataset's ground-truth labels. With these measurements, the normalized, geometric displacement of the dogs noses and eyes from the center of their faces could be computed and stored. Next, grayscale SIFT descriptors centered at these locations were collected and scaled by the interocular distance for each of the positive training samples.

Negative samples were constructed by randomly sampling a section of the training images outside of the ground-truth facial boxes under a scale and rotation drawn from a normal distribution around the mode of their values in the positive training set. Since it was cheap to generate additional negative samples, the final training set contained 4776 positive, normalized examples of canine faces as well as 13,000 negative examples. With this data in hand, a binary classifier was trained in the form of a linear SVM. In a process identical to that of training set generation, a test set was created - this time including an equal number of positive and negative samples. The positive samples were scaled and rotation normalized as in training based on their ground-truth keypoint labels. Unfortunately, I found that this classifier achieved only 55 percent accuracy on the binary discrimination task between canine faces and non-faces. It is not totally clear why this model performed worse than that of Liu, as his paper does not provide quantitative results of the generic facial detection problem. It is possible that the models are roughly equivalent, since his evaluation procedure consists of pooling scores for candidate windows to select an area for finer grained analysis in order to generate part locations, whereas mine was evaluated on the binary classification task over normalized samples. Regardless, though, I considered this performance unsatisfactory, especially given its near total failure on non-frontal images and opted to start over with a different method.

Second Approach-The second approach to facial detection proved much more successful at the task. This procedure consisted of a convolutional neural network directly regressed on the

ground-truth part locations and therefore able to predict them directly. A fairly shallow convolutional neural netFigure 2. Predicted facial keypoint locations (red) and ground truth labels (green) for three dogs of varying pose under the final model. work was trained with a mean squared error loss function on the location of all eight facial keypoints. The output of even the prototype network seemed workable, so I settled on that type of model over competing approaches such as deformable parts models and poselets, which had been suggested during office hours. The keypoint localization network went through several rounds of improvements over the course of the project, culminating in a model which could successfully identify the facial region, orientation, and scale for all images with precision comparable to that of Liu et al. Briefly, dropout, color and contrast jitter, leaky ReLu, and an extended network architecture and training time were all employed to boost the models accuracy, but as this paper is focused on the breed identification network, which made use of some of the same procedures, they will not be discussed at length here. If anyone is interested in more details of this model and its performance, I encourage them to read my CS231A paper. Figure 2 illustrates some examples of the output of the final model. With this estimator tuned and sufficiently accurate, I turned towards the next phase of breed identification.

A variety of neural networks were implemented, trained, tuned, and evaluated for the breed identification problem over the course of the project. To maintain the ability to directly compare my results with those of Liu et al, I used his original train/test partition of the dataset into 4776 training images and 3575 testing images. Input images of varying content were cropped, rotated, and resized to fit a sixty-four by sixty-four pixel square across three color channels as described in section 4. Figure 7 gives the quantitative results of each model in a single graph. 4 As a baseline, I began with a small, fully connected network. This model consisted of a single hidden layer of one thousand units connected to a softmax layer for prediction over the one hundred thirty-three breeds. Surprisingly, this fairly naive network attained a 9.56 percent classification accuracy, which is impressive for such a simple model. The first convolutional network (Net A in figure 7) improved upon this baseline by a significant margin. A shallow network, it included only three convolutional layers followed by simple rectified linear unit nonlinearities and two by two max pooling.

On top of the convolutions, two fully connected layers of one thousand hidden units each as well as a softmax classifier were trained. The model did not include any tricks, bells, or whistles, and can be considered a translation of my dense network into a convolutional form. This network reached saturation in under an hour of training with a final accuracy score of

21.11 percent. Following these experiments, I decided to expand the network architecture in order to see if its results could be improved. An additional convolution before each pooling layer was included for a total of six and the number of filters available to the original three were increased by a factor of two. Having noticed that the first convolutional model suffered from heavy overfitting with a train/test loss ratio of 0.136, I also added dropout after each pooling layer as well as the first fully connected layer.

These modifications pushed the performance of the network slightly higher but also greatly increased the time to convergence to eight hours. The networks final accuracy score was 24.22 percent. For the final network, I implemented a variety of improvements but concentrated on data augmentation. The size of the training set had been a noticeable hindrance throughout the process, resulting in overfitting and poor performance, so I included a variety of ways to expand it.

Color and contrast jitter were included in this model as well as random horizontal mirroring. The input dataset was also expanded by randomly jittering the theta and center of the facial crops of the original images over a normal distribution centered at their true estimates to produce five transformed copies of each training image (the true estimate was also included for a total expansion factor of six).

Finally, a leaky ReLu nonlinearity was implemented and swapped into the network following each convolutional layer. This final model achieved an accuracy score of 30.6 percent, a substantial improvement over Net B. The network architecture of this model is given in Table 1. As an additional experiment, the performance of Net C's architecture over the raw images in the dataset was also evaluated. Thus, the full training and testing images were scaled down to the same size as the randomized crops of Net C and the same network architecture was trained over them. This procedure resulted in a 1.4 percent accuracy score for breed identification, which while nearly double chance performance, remains much lower than even the dense, naive model over preprocessed inputs. This result illustrates the necessity of effective preprocessing before a complicated image classification task such as breed identification.

Conclusion: Overall, I am disappointed that I failed to match Liu's reported accuracy on the dataset, although my final results are fairly robust for such a difficult problem. At one point, I incorrectly thought that I had succeeded in surpassing 67 percent accuracy, but in reality, my

models do not perform nearly as well as that. Notwithstanding that reality, there are several valuable takeaways from this project as well as remaining avenues of investigation.

2.3. ADVANTAGE OF THE CNN TRANSFER LEARNING

In my approach, which CNN with Transfer Learning. Dog breeds are more accurately classifiedand with the help of Inception pretrained model we can have an effective model which is trained. By using this model weights we are going to implement transferred learning to the custom layes specified, which leads to reduce in the time ting to train the model. The accuracy which is yielding in my project in much greater than the above research papers.

CHAPTER-3

ANALYSIS/PROBLEM ANALYSIS

3.1 PROBLEM BACKGROUND

The project involves Image Processing, classification and detection using Deep Learning. Convolutional Neural Networks along with Transfer Learning is used to deliver the results. Given an image of the dog, the CNN must give out the correct breed of the dog in the image out of 7 classes of dogs.

This Project is based on Image classification. Since this task of recognizing a visual concept (e.g. cat) is relatively trivial for a human to perform, it is worth considering the challenges involved from the perspective of a Computer Vision algorithm. As we present (an inexhaustive) list of challenges below, keep in mind the raw representation of images as a 3-D array of brightness values:

- **Viewpoint variation**. A single instance of an object can be oriented in many ways with respect to the camera.
- **Scale variation**. Visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image).
- **Deformation**. Many objects of interest are not rigid bodies and can be deformed in extreme ways.
- **Occlusion**. The objects of interest can be occluded. Sometimes only a small portion of an object (as little as few pixels) could be visible.
- **Illumination conditions**. The effects of illumination are drastic on the pixel level.
- **Background clutter**. The objects of interest may *blend* into their environment, making them hard to identify.
- **Intra-class variation**. The classes of interest can often be relatively broad, such as *chair*. There are many different types of these objects, each with their own appearance.

Some of the main constraints are:

- Image size is not homogenous, thus image pre-processing will be compulsory.
- There can be more than one dog in an image, hence the learning task of classification model will be more complex.
- Image are provided with white background or with coloured and diverse background, where various objects can be found.
- The task of classification is exceptionally challenging since there is a minimal inter-class between some dog-breed pairs that even a human would have difficulty to distinguish.

3.2 DATASET EXPLORATION

The Data set is taken from kaggle contains 7 Dog breed categories. Which are namely 7 classes.

These Dog Breeds are

- Bull Terrier
- German Shepherd
- Dachshund
- Bull Dog
- Maltese
- Pomeranian
- Poodle

3.3 REQUIREMENTS SPECIFICATIONS

3.3.1 Hardware Requirements

Operating System: Microsoft Windows 10/8/7 or Mac

Ram: Minimum of 8 GB

3.3.2 Software Requirements

Software for Python: Anaconda or Google Colab

Packages: Numpy, Scikit-learn, TensorFlow

CHAPTER-4

DESIGN

4.1 PLAN OF THE PROJECT

Step 1: Loading the data

Step 2: Data Pre-processing

Step 3: Data Visualization

Step 4: Build the Model

Step 5: Training the model

Step 6: Make Predictions

4.2 ALGORITHMS USED:

Convolutional Neural Network (CNN): The CNN algorithm is efficient at recognition and highly adaptable. It's also easy to train because there are fewer training parameters, and is scalable when coupled with backpropagation.

Inception V3: Inception v3 is a widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset. The model is the culmination of many ideas developed by multiple researchers over the years.

4.3 BRIEF ABOUT THE ALGORITHMS USED

4.3.1 Convolutional Neural Networks:

A breakthrough in building models for image classification came with the discovery that a convolutional neural network (CNN) could be used to progressively extract higher- and higher-level representations of the image content. Instead of pre-processing the data to derive features like textures and shapes, a CNN takes just the image's raw pixel data as input and "learns" how to extract these features, and ultimately infer what object they constitute. CNN

receives an input feature map: a three-dimensional matrix where the size of the first two dimensions corresponds to the length and width of the images in pixels. The size of the third dimension is 3 (corresponding to the 3 channels of a colour image: red, green, and blue). The CNN comprises a stack of modules, each of which performs three operations.

1. Convolution:

A convolution extracts tiles of the input feature map, and applies filters to them to compute new features, producing an output feature map, or convolved feature (which may have a different size and depth than the input feature map).Convolutions are defined by two parameters:

- Size of the tiles that are extracted (typically 3x3 or 5x5 pixels).
- The depth of the output feature map, which corresponds to the number of filters that are applied.

During a convolution, the filters (matrices the same size as the tile size) effectively slide over the input feature map's grid horizontally and vertically, one pixel at a time, extracting each corresponding tile.

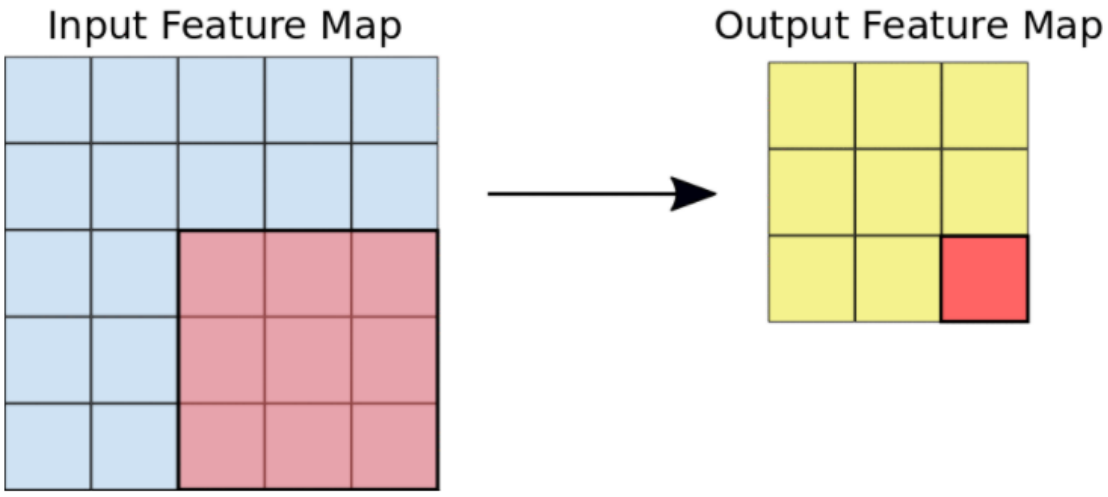

Fig 4.3.1.1: Illustration of convolution over feature map

In fig 4.3.1.1 a 3x3 convolution of depth 1 performed over a 5x5 input feature map, also of depth 1. There are nine possible 3x3 locations to extract tiles from the 5x5 feature map, so this convolution produces a 3x3 output feature map.

In Figure 4.3.1, the output feature map (3x3) is smaller than the input feature map (5x5). If you instead want the output feature map to have the same dimensions as the input feature map, 2you can add padding (blank rows/columns with all-zero values) to each side of the input feature map, producing a 7x7 matrix with 5x5 possible locations to extract a 3x3 tile.

For each filter-tile pair, the CNN performs element-wise multiplication of the filter matrix and the tile matrix, and then sums all the elements of the resulting matrix to get a single value. Each of these resulting values for every filter-tile pair is then output in the *convolved feature* matrix (see Figures 4.3.1.2 and 4.3.1.3).

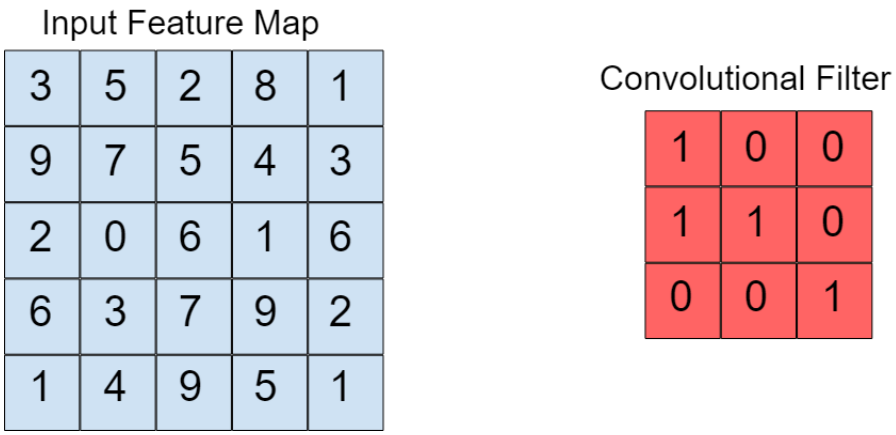

Fig 4.3.1.2: A 5x5 input feature map and 3x3 convolution of deph1

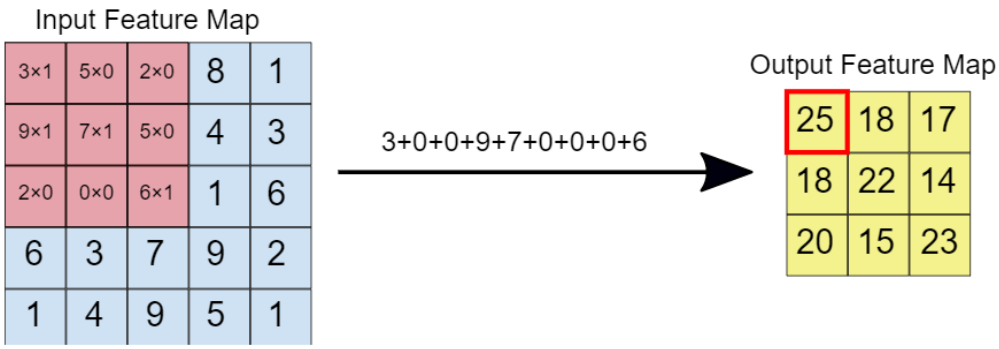

Fig 4.3.1.3: Example of 3 x 3 convolution performed over 5 x 5 input feature map

In Figure:

Left: The 3x3 convolution is performed on the 5x5 input feature map.

Right: the resulting convolved feature.

For each filter-tile pair, the CNN performs element-wise multiplication of the filter matrix and the tile matrix, and then sums all the elements of the resulting matrix to get a single value. Each of these resulting values for every filter-tile pair is then output in the convolved feature matrix.

During training, the CNN "learns" the optimal values for the filter matrices that enable it to extract meaningful features (textures, edges, shapes) from the input feature map. As the number of filters (output feature map depth) applied to the input increases, so does the number of features the CNN can extract. However, the trade-off is that filters compose the majority of resources expended by the CNN, so training time also increases as more filters are added. Additionally, each filter added to the network provides less incremental value than the previous one, so engineers aim to construct networks that use the minimum number of filters needed to extract the features necessary for accurate image classification.

2. ReLU

Following each convolution operation, the CNN applies a Rectified Linear Unit (ReLU) transformation to the convolved feature, in order to introduce nonlinearity into the model. The ReLU function.

$F(x)=\max(0,x)$ returns x for all values of $x > 0$, and returns 0 for all values of $x \leq 0$.

3. Pooling

After ReLU comes a pooling step, in which the CNN down samples the convolved feature (to save on processing time), reducing the number of dimensions of the feature map, while still preserving the most critical feature information. A common algorithm used for this process is called max pooling.

Max pooling operates in a similar fashion to convolution. We slide over the feature map and extract tiles of a specified size. For each tile, the maximum value is output to a new feature map, and all other values are discarded. Max pooling operations take two parameters:

Size of the max-pooling filter (typically 2x2 pixels)

Stride: the distance, in pixels, separating each extracted tile. Unlike with convolution, where filters slide over the feature map pixel by pixel, in max pooling, the stride determines the locations where each tile is extracted. For a 2x2 filter, a stride of 2 specifies that the max pooling operation will extract all non-overlapping 2x2 tiles from the feature map (see Figure 4.3.1.4).

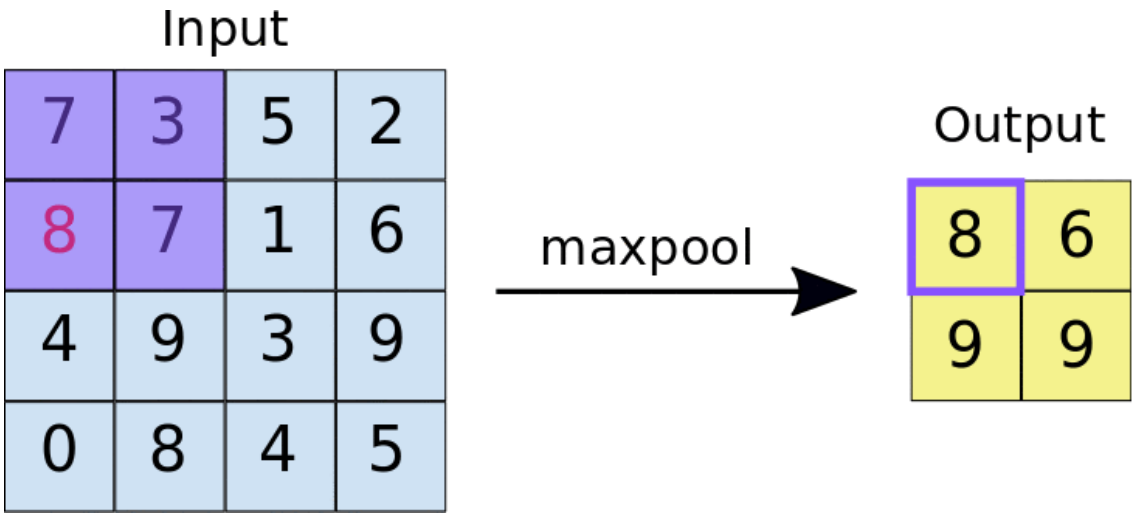

Fig 4.3.1.4: Example of Max Pooling

In Figure 4.3.1.4.

Left: Max pooling performed over a 4x4 feature map with a 2x2 filter and stride of 2.

Right: the output of the max pooling operation. Note the resulting feature map is now 2x2, preserving only the maximum values from each tile.

4. Fully Connected Layers

At the end of a convolutional neural network are one or more fully connected layers (when two layers are "fully connected," every node in the first layer is connected to every node

in the second layer). Their job is to perform classification based on the features extracted by the convolutions. Typically, the final fully connected layer contains a softmax activation function, which outputs a probability value from 0 to 1 for each of the classification labels the model is trying to predict.

Preventing Overfitting

As with any machine learning model, a key concern when training a convolutional neural network is overfitting: a model so tuned to the specifics of the training data that it is unable to generalize to new examples.

Two techniques to prevent overfitting when building a CNN are:

Data augmentation: artificially boosting the diversity and number of training examples by performing random transformations to existing images to create a set of new variants. Data augmentation is especially useful when the original training data set is relatively small.

Dropout regularization: Randomly removing units from the neural network during a training gradient step.

4.3.2 INCEPTION V3 OR GOOGLENET MODEL:

It is basically a convolutional neural network (CNN) which is 27 layers deep**.** Inception v3 mainly focuses on burning less computational power by modifying the previous Inception architectures. This idea was proposed in the paper Rethinking the Inception Architecture for Computer Vision, published in 2015. It was co-authored by Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, and Jonathon Shlens.

In comparison to VGGNet, Inception Networks (GoogLeNet/Inception v1) have proved to be more computationally efficient, both in terms of the number of parameters generated by the network and the economical cost incurred (memory and other resources). If any changes are to be made to an Inception Network, care needs to be taken to make sure that the computational advantages aren't lost. Thus, the adaptation of an Inception network for different use cases turns out to be a problem due to the uncertainty of the new network's efficiency. In an Inception v3 model, several techniques for optimizing the network have been put suggested to loosen the constraints for easier model adaptation. The techniques include factorized convolutions, regularization, dimension reduction, and parallelized computations.

The architecture of an Inception v3 network is progressively built, step-by-step, as explained below:

1. Factorized Convolutions: this helps to reduce the computational efficiency as it reduces the number of parameters involved in a network. It also keeps a check on the network efficiency.

2. Smaller convolutions: replacing bigger convolutions with smaller convolutions definitely leads to faster training. Say a 5 × 5 filter has 25 parameters; two 3 × 3 filters replacing a 5 × 5 convolution has only 18 (3*3 + 3*3) parameters instead.

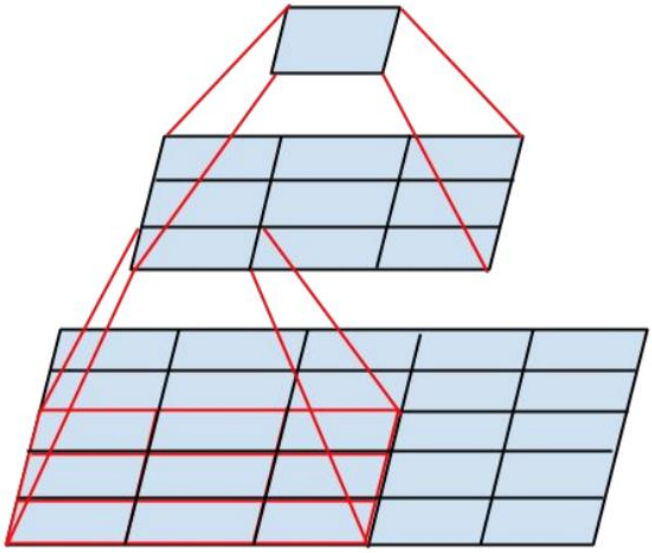

Fig 4.3.2.1 Smaller convolutions

In the middle we see a 3x3 convolution, and below a fully-connected layer. Since both 3x3 convolutions can share weights among themselves, the number of computations can be reduced.

3. Asymmetric convolutions: A 3 × 3 convolution could be replaced by a 1 × 3 convolution followed by a 3 × 1 convolution. If a 3 × 3 convolution is replaced by a 2 × 2 convolution, the number of parameters would be slightly higher than the asymmetric convolution proposed.

Fig 4.3.2.2: Asymmetric convolutions

4. Auxiliary classifier: an auxiliary classifier is a small CNN inserted between layers during training, and the loss incurred is added to the main network loss. In GoogLeNet auxiliary classifiers were used for a deeper network, whereas in Inception v3 an auxiliary classifier acts as a regularizer.

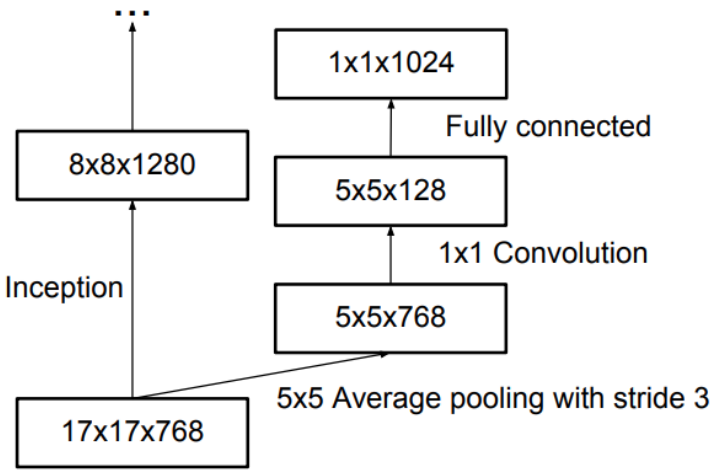

Fig 4.3.2.3: Auxiliary classifier

5. Grid size reduction: Grid size reduction is usually done by pooling operations. However, to combat the bottlenecks of computational cost, a more efficient technique is proposed:

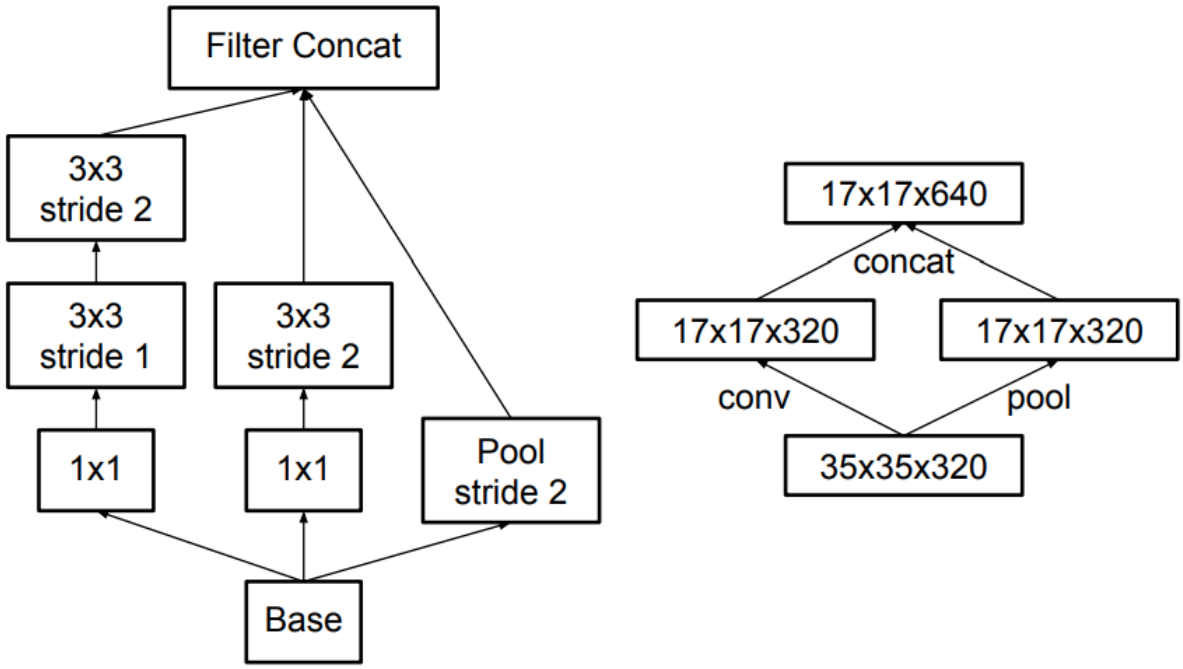

Fig 4.3.2.4: Grid size reduction

All the above concepts are consolidated into the final architecture.

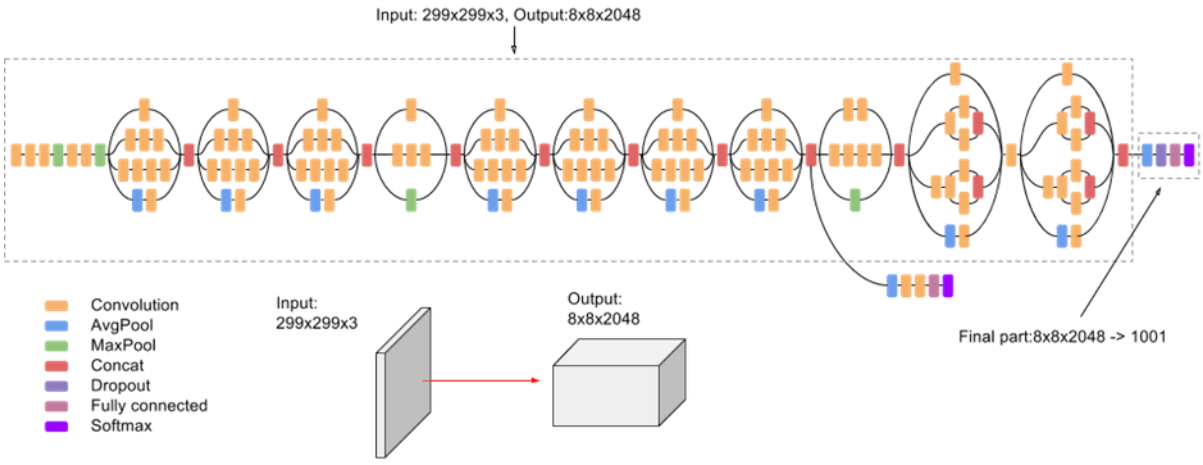

Fig 4.3.2.5: Inception v3 Architecture

Inception Layer is a combination of all those layers (namely, 1×1 Convolutional layer, 3×3 Convolutional layer, 5×5 Convolutional layer) with their output filter banks concatenated into a single output vector forming the input of the next stage.

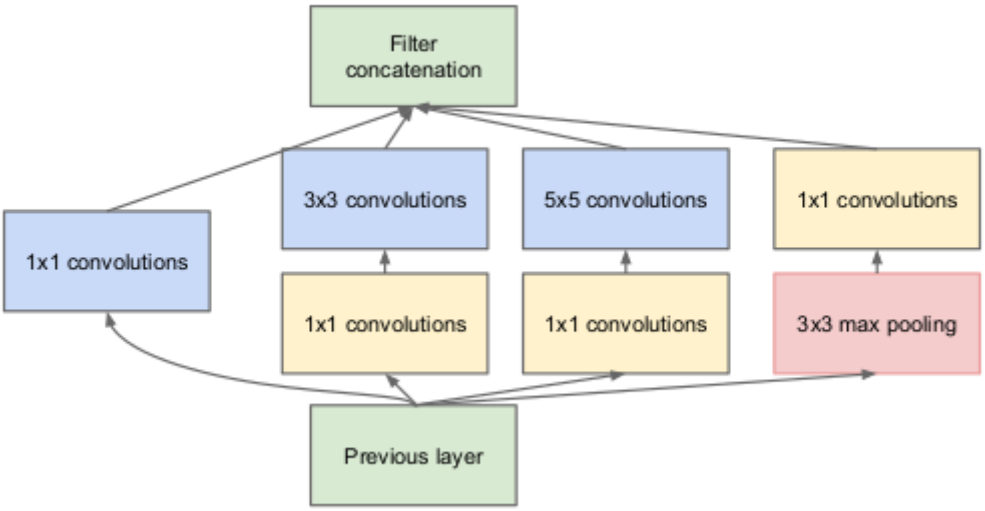

Fig 4.3.2.6: Inception Layer

4.4 COMBINING TRANSFER LEARNING AND CNN

The low-level and high-level features learned by a CNN on a source domain can often be transferred to augment learning in an alternate but related target domain. For target problems with abundant data, we can transfer low-level features, such as edges and corners, and learn new high-level features specific to the target problem. To extract the features, we retrieve the next-to-last layer of the Inceptionv3 as a feature vector for each image. Indeed, the last layer of the convolutional neural network corresponds to the classification step: as it has been trained for the ImageNet dataset, the categories that it will be output will not correspond to the categories in the Product Image Classification dataset we are interested in. The output of the next-to-last layer, however, corresponds to features that are used for the classification in Inception-v3. The hypothesis here is that these features can be useful for training another classification model, so we extract the output of this layer. Nonetheless, if the source and target domain are adequately similar, the feature representation learned by the CNN on the source task can likewise be utilized for the target problem. Deep features extracted from CNNs trained on large annotated datasets of images have been used as generic features viably for an extensive variety of computer vision tasks.

When we have a relatively small dataset, a super-effective technique is to use **Transfer Learning** where we use a pre-trained model. This model has been trained on an extremely large

dataset, and we would be able to transfer weights which were learned through hundreds of hours of training on multiple high powered GPUs.

Many such models are open-sourced such as Inception-v3. They were trained on millions of images with extremely high computing power which can be very expensive to achieve from scratch. We are using the **Inception-v3** model in the project. Transfer learning has become immensely popular because it considerably reduces training time, and requires a lot less data to train on to increase performance.

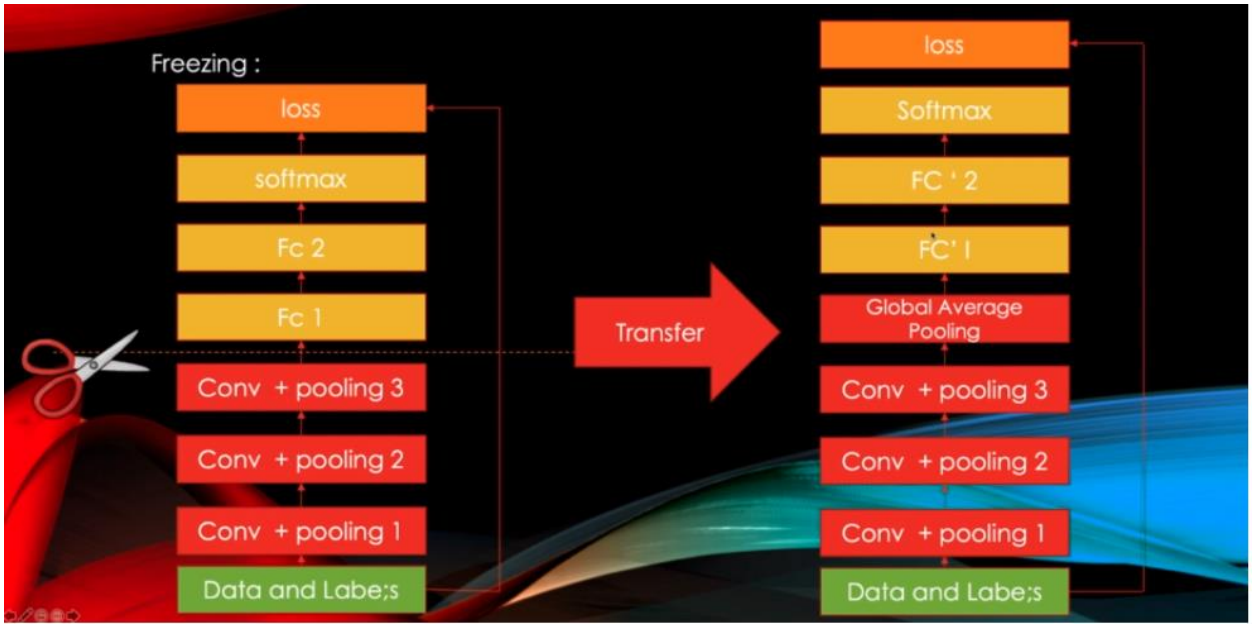

Fig 4.4.1: Transfer Learning

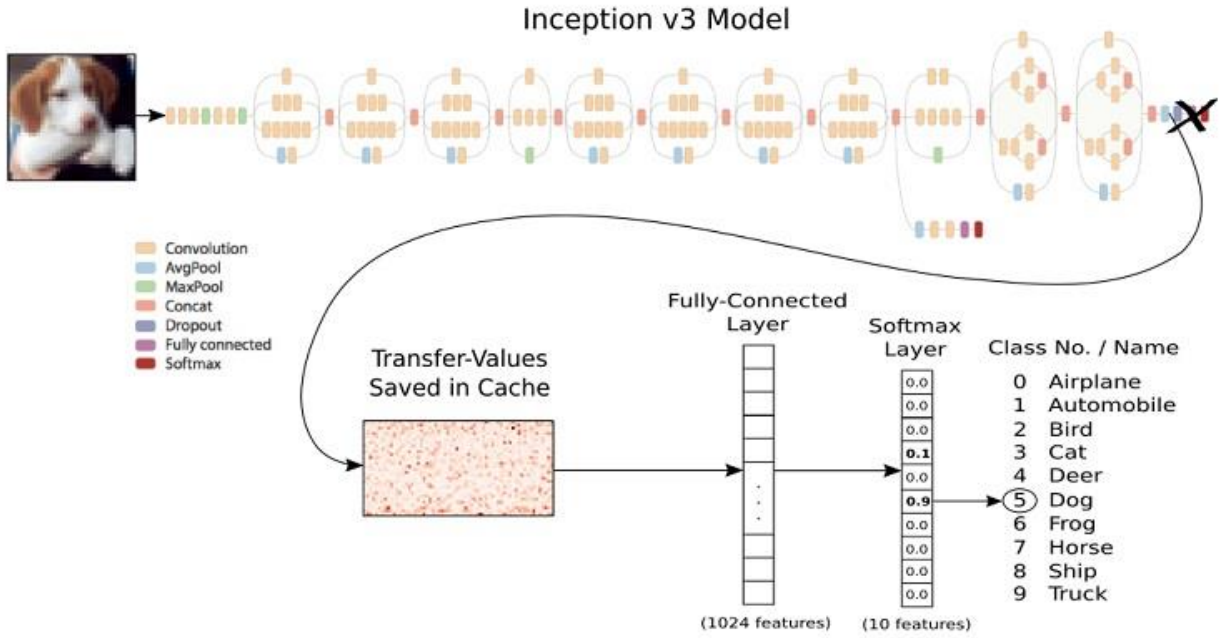

Fig 4.4.2: Transfer Learning with Inception V3

- In Transfer learning using Inception v3 we first cut of the head of the inception v3 model and use pre-trained ImageNet weights and specify custom layers.
- Then we are freezing the top layer and the images are trained for custom layers. So, that it reduces the learning time to learn from scratch.

CHAPTER-5

IMPLEMNTATON

5.1 IMPORTING PACKAGES:

Numpy: In Python we have lists that serve the purpose of arrays, but they are slow to process. Numpy aims to provide an array object that is up to 50x faster that traditional Python lists. The array object in Numpy is called ndata, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

Pandas: Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Matplotlib: Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. Matplotlib is written in Python and makes use of Numpy, the numerical mathematics extension of Python.

Open CV: OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

TensorFlow: TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

OS: The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality.

```
# Importing packages

%matplotlib inline
import pandas as pd
import os,cv2
import random as rn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tqdm import tqdm
from keras.utils.np_utils import to_categorical
from keras import layers
from keras.models import Sequential,Input,Model
from keras.layers import Dense,GlobalAveragePooling2D
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing import image
import tensorflow as tf
import matplotlib.image as mpimg
```

Fig 5.1.1: Packages used

5.2 LOADING DATASET

Python has a built-in open() function to open a file. This function returns a file object, also called a handle, as it used to read or modify the file accordingly. We can specify the mode while opening a file. In mode, we specify whether we want to read, write or append to the file. We can also specify if we want to open the file in text mode or binary mode. The default is reading in text mode. In this mode, we get strings when reading from the file.

```
#Creating directory path for each dog breed

Maltese_dir = '/content/drive/My Drive/dogs datasset/dogs manual/train/101.Maltese'
Bulldog_dir = '/content/drive/My Drive/dogs datasset/dogs manual/train/040.Bulldog'
Bull_terrier_dir = '/content/drive/My Drive/dogs datasset/dogs manual/train/039.Bull_terrier'
Dachshund_dir = '/content/drive/My Drive/dogs datasset/dogs manual/train/056.Dachshund'
German_sheperd_dir ='/content/drive/My Drive/dogs datasset/dogs manual/train/071.German_shepherd_dog'
Pomeranian_dir = '/content/drive/My Drive/dogs datasset/dogs manual/train/123.Pomeranian'
Poodle_dir = '/content/drive/My Drive/dogs datasset/dogs manual/train/124.Poodle'
```

Fig 5.2.1: Creating directories

Creating function by which we it assign dag names for each breed of image read. Here tqdm is used to see the progress of the images loading.

Images are read by using cv2. Which then are resizes every image into 150 x 150 size so that all the images will have same size.

```
X = []
Z = []
IMG_SIZE = 150
```

```
# creating a function that returns which type of dog breed(label)

def assign_label(img,dog_type):
    return dog_type
```

```
#Function creation- which reads images and labels are classified according to images

def training_data(dog_type,DIR):
    for img in tqdm(os.listdir(DIR)):
        label=assign_label(img,dog_type)
        path = os.path.join(DIR,img)
        img = cv2.imread(path,cv2.IMREAD_COLOR)

        #resizing images
        img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))

        X.append(np.array(img))
        Z.append(str(label))
```

Fig 5.2.2: Function Creation

Now we are passing the created directories to the function and every image in the folder is assigned with the specified dog breed name. The variable X stores the every image read in Numpy array format and the variable Z contains labels of the every images of the specifi dog breed.

Loading Data

```
[ ]
    training_data('Maltese',Maltese_dir)
    training_data('Bulldog',Bulldog_dir)
    training_data('Bull_terrier',Bull_terrier_dir)
    training_data('Dachshund',Dachshund_dir)
    training_data('German_sheperd',German_sheperd_dir)
    training_data('Pomeranian',Pomeranian_dir)
    training_data('Poodle',Poodle_dir)
```

```
100%|██████████| 48/48 [00:00<00:00, 87.84it/s]
100%|██████████| 53/53 [00:00<00:00, 96.26it/s]
100%|██████████| 69/69 [00:00<00:00, 88.12it/s]
100%|██████████| 65/65 [00:00<00:00, 101.14it/s]
100%|██████████| 62/62 [00:00<00:00, 80.53it/s]
100%|██████████| 44/44 [00:01<00:00, 41.35it/s]
100%|██████████| 50/50 [00:00<00:00, 84.83it/s]
```

Fig 5.2.3: Loading data

Here the X has the image read values stored in the numpy array format so that it can be useful in classifying breed in CNN.

```
print(X[0])
```

```
[[[ 59 143 169]
  [ 92 171 203]
  [120 195 228]
  ...
  [ 33  24  20]
  [ 36  22  11]
  [ 37  23  11]]

 [[ 46 130 156]
  [ 63 143 174]
  [110 187 220]
  ...
  [ 46  38  34]
  [ 39  26  15]
  [ 41  27  16]]

 [[ 62 147 173]
  [ 52 134 165]
  [ 92 169 202]
  ...
  [ 48  42  37]
  [ 38  27  17]
  [ 39  27  17]]

 ...
```

Fig 5.2.4: Images in Numpy array Format

Next, in Z the all the images converted to numpy arrays for which the each image has identity which are known as labels. For every image labels are assigned, the assigned label values are stored in Z.

```
print(Z)
['Maltese', 'Maltese', 'Maltese', 'Maltese', 'Maltese', 'Maltese', 'Maltese', 'Maltese', 'Maltese', 'Maltese',
```

Fig 5.2.5: Labels

5.3 LABEL ENCODING:

In machine learning, we usually deal with datasets which contains multiple labels in one or more than one columns. These labels can be in the form of words or numbers. To make the data understandable or in human readable form, the training data is often labeled in words.

Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

Example :

Suppose we have a column Height in some dataset

Height
Tall
Medium
Short

Fig 5.3.1: Label Encoder Example1

After applying label encoding, the Height column is converted into:

Height
0
1
2

Fig 5.3.2: Label Encoder Example1 output

Where 0 is the label for tall, 1 is the label for medium and 2 is label for short height.

In the similar manner all the dog breed name are converted into values undwerstandable by machine.

Assiging labels via label encoder

```
[ ]  label_encoder= LabelEncoder()
     Y = label_encoder.fit_transform(Z)
```

Fig 5.3.3: Label Encoder of breed names

Here the below code represents how the each dog breed has value for its breed name. In the Fig 5.3.4 the first cell represents the dog names and the second cell accordingly represents the vales after encoding the label names.

Representation of Dog breeds and its assigning by the label encoder. so that this can be refered during the output prediction

```
[ ]  temp1 = []
     for t in z:
         if t not in temp1:
             temp1.append(t)
     breed_name=temp1
     print(breed_name)
```

```
['Maltese', 'Bulldog', 'Bull_terrier', 'Dachshund', 'German_sheperd', 'Pomeranian', 'Poodle']
```

```
[ ]  breed = Y.tolist()
     temp = []
     for u in breed:
         if u not in temp:
             temp.append(u)
     breed=temp
     print(breed)
```

```
[4, 1, 0, 2, 3, 5, 6]
```

Fig: 5.3.4 Breed names as values

The variable Y which in Label encoded is now having their values of each breed. But in order to classify the dog breeds we need classes. So, the encoded values are categorised into 7 classes which can be easy to predict breeds of the unknown dog s given.

```
Y = to_categorical(Y,7)
X = np.array(X)
```

Fig 5.3.5: Categorization

5.4 SCALING THE DATA

It is a step of Data Pre Processing which is applied to independent variables or features of data. It basically helps to normalise the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm. Real world dataset contains features that highly vary in magnitudes, units, and range. Normalisation should be performed when the scale of a feature is irrelevant or misleading and not should Normalise when the scale is meaningful.

Scaling is done only for X where image data is represented in numpy array format. The items column is divided by 255 because every digital image is formed by pixel having value in range 0~255. 0 is black and 255 is white. For colourful image, it contains three maps: Red,

Green and Blue, and all the pixel still in the range 0~255. Since 255 is the maximin pixel value. Rescale 1./255 is to transform every pixel value from range [0,255] -> [0,1]. Which helps treating all images in the same manner.

```
[ ]  ## scaling
     X=X/255
```

Fig 5.4.1: Scaling

5.5 IMAGE AUGMENTATION

Keras **ImageDataGenerator** class provides a quick and easy way to augment your images. It provides a host of different augmentation techniques like standardization, rotation, shifts, flips, brightness change, and many more. You can find more on its official documentation page.

However, the main benefit of using the Keras ImageDataGenerator class is that it is designed to provide real-time data augmentation. Meaning it is generating augmented images on the fly while your model is still in the training stage. How cool is that!

ImageDataGenerator class ensures that the model receives new variations of the images at each epoch. But it only returns the transformed images and does not add it to the original corpus of images. If it was, in fact, the case, then the model would be seeing the original images multiple times which would definitely overfit our model.

Another advantage of ImageDataGenerator is that it requires lower memory usage. This is so because without using this class, we load all the images at once. But on using it, we are loading the images in batches which saves a lot of memory.

Some of the augmentations:

i. **Random Shifts**

It may happen that the object may not always be in the center of the image. To overcome this problem we can shift the pixels of the image either horizontally or vertically; this is done by adding a certain constant value to all the pixels.

ImageDataGenerator class has the argument height_shift_range for a vertical shift of image and width_shift_range for a horizontal shift of image. If the value is a float number, that would indicate the percentage of width or height of the image to shift. Otherwise, if it is an integer value then simply the width or height are shifted by those many pixel values.

ii. Random Flips

Flipping images is also a great augmentation technique and it makes sense to use it with a lot of different objects.

ImageDataGenerator class has parameters horizontal_flip and vertical_flip for flipping along the vertical or the horizontal axis. However, this technique should be according to the object in the image. For example, vertical flipping of a car would not be a sensible thing compared to doing it for a symmetrical object like football or something else. Having said that, I am going to flip my image in both ways just to demonstrate the effect of the augmentation.

iii. Random Brightness

It randomly changes the brightness of the image. It is also a very useful augmentation technique because most of the time our object will not be under perfect lighting condition. So, it becomes imperative to train our model on images under different lighting conditions.

Brightness can be controlled in the ImageDataGenrator class through the brightness_range argument. It accepts a list of two float values and picks a brightness shift value from that range. Values less than 1.0 darkens the image, whereas values above 1.0 brighten the image.

iv. Random Zoom

The zoom augmentation either randomly zooms in on the image or zooms out of the image.

ImageDataGenerator class takes in a float value for zooming in the zoom_range argument. You could provide a list with two values specifying the lower and the upper limit. Else, if you specify a float value, then zoom will be done in the range [1-zoom_range,1+zoom_range].

Any value smaller than 1 will zoom in on the image. Whereas any value greater than 1 will zoom out on the image.

```
[ ]
    augs_gen = ImageDataGenerator(
            featurewise_center=False,
            samplewise_center=False,
            featurewise_std_normalization=False,
            samplewise_std_normalization=False,
            zca_whitening=False,
            rotation_range=10,
            zoom_range = 0.1,
            width_shift_range=0.2,
            height_shift_range=0.2,
            horizontal_flip=True,
            vertical_flip=False)

    augs_gen.fit(x_train)
```

Fig 5.5.1: Augmentation

5.6 DATA VISUALIZATION

We are displaying 10 random images from the dataset using Matplotlib library. Matplotlib is a library in Python and it is numerical – mathematical extension for NumPy library. Pyplot is a state-based interface to a **Matplotlib** module which provides a MATLAB-like interface.

By using subplots we are specifying positions. The **subplots() function** in pyplot module of matplotlib library is used to create a figure and a set of subplots.

```
fig,ax=plt.subplots(5,2)
fig.set_size_inches(15,15)
for i in range(5):
    for j in range (2):
        l=rn.randint(0,len(Z))
        ax[i,j].imshow(X[l])
        ax[i,j].set_title('Dog: '+Z[l])

plt.tight_layout()
```

Fig 5.6.1 : Input for Visualization of images

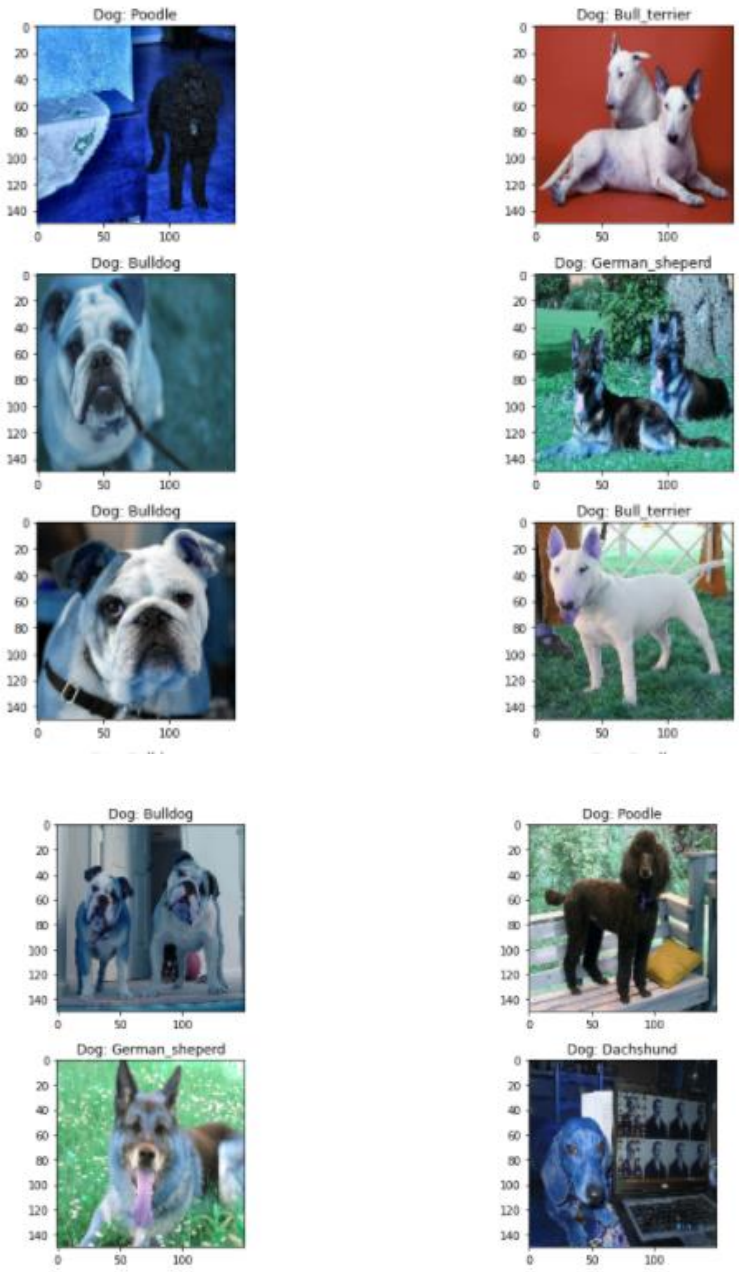

Fig 5.6.2: Output for Visualization of images

CHAPTER-6

TRAINING, TESTING AND VALIDATION

6.1 SPLITTING THE DATA

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for your predictive modelling problem. Although simple to use and interpret, there are times when the procedure should not be used, such as when you have a small dataset and situations where additional configuration is required, such as when it is used for classification and the dataset is not balanced.

It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

- **Train Dataset**: Used to fit the machine learning model.
- **Test Dataset**: Used to evaluate the fit machine learning model.

The objective is to estimate the performance of the machine learning model on new data: data not used to train the model.

This is how we expect to use the model in practice. Namely, to fit it on available data with known inputs and outputs, then make predictions on new examples in the future where we do not have the expected output or target values. The train-test procedure is appropriate when there is a sufficiently large dataset available.

The procedure has one main configuration parameter, which is the size of the train and test sets. This is most commonly expressed as a percentage between 0 and 1 for either the train or test datasets. For example, a training set with the size of 0.67 (67 percent) means that the

remainder percentage 0.33 (33 percent) is assigned to the test set. There is no optimal split percentage.

You must choose a split percentage that meets your project's objectives with considerations that include:

- Computational cost in training the model.
- Computational cost in evaluating the model.
- Training set representativeness.
- Test set representativeness.

Nevertheless, common split percentages include:

Train: 80%, Test: 20%

Train: 67%, Test: 33%

Train: 50%, Test: 50%

Now that we are familiar with the train-test split model evaluation procedure, let's look at how we can use this procedure in Python.

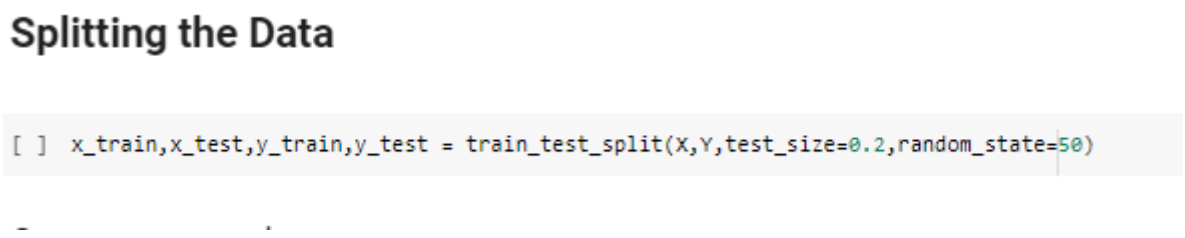

Fig 6.1.1: Splitting Data

Here in the model, we have used 80% of the data as training and 20% of the data as testing data.

6.2 MODEL BUILDING

Firstly, we are importing inception_V3 model from keras. In that the top part is taken as false because we are using transfer flow learning way of approach in which we are not going to train the entire data from scratch. In the Inception v3 model we have to specify the shape of the model which is 150 x 150 x 3. Now the weights are taken from ImageNet where the model is pre trained over millions of different images we are using the final weights of the trained the

model and chopping off the top layer. Here in ImageNet the model has learned few basic features of the dog parts like edges, dog shapes.

Secondly, we are specifying custom layers where this is the main part of the where classification of breed takes place. So the custom layers are trained trained using weights of base inception. As the basic features of the images are pre trained so classification of the breed can be done using custom layers. We are going to freeze the inception model and training takes place only using the custom layers.

- Model is implemented in sequential model. Here I have added 2 fully connected layers by using activation function as relu.
- As the model is multi classification the output dense value is7 and using softmax as activation function for the last dense layer.
- GlobalAveragePooling2D is used to convert data into 1D array.

Output Activation Function

Softmax- Softmax it's a function, not a loss. It squashes a vector in the range (0, 1) and all the resulting elements add up to 1. It is applied to the output scores ss. As elements represent a class, they can be interpreted as class probabilities.
The Softmax function cannot be applied independently to each sisi, since it depends on all elements of ss. For a given class sisi, the Softmax function can be computed as:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

Where sjsj are the scores inferred by the net for each class in CC. Note that the Softmax activation for a class sisi depends on all the scores in ss.

```
base_model = InceptionV3(include_top=False,
                  input_shape = (IMG_SIZE,IMG_SIZE,3),
                  weights = 'imagenet')

# Freezing layers
for layer in base_model.layers:
    layer.trainable = False

model = Sequential()
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dense(512,activation='relu'))
model.add(Dense(512,activation='relu'))
model.add(Dense(7,activation='softmax'))
model.summary()
```

Fig 6.2.1: Model input

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 3, 3, 2048)        21802784
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 3591
=================================================================
Total params: 23,118,119
Trainable params: 1,315,335
Non-trainable params: 21,802,784
_________________________________________________________________
```

Fig 6.2.2: Model Summary

From here only 1,315,335 parameters are trained over 23,118,119 parameters this is because of the Freezing the model. Hence this reduces the time for the model in training.

6.3 COMPILING THE MODEL

Here we need to define how to calculate the loss or error. Since we are using a Multi class classification, we can use categorical crossentropy. With the optimizer parameter, we pass how to adjust the weights in the network such that the loss gets reduced. There are many options that can be used and here I used Adam Optimizer method. Finally, the metrics parameter will be used to estimate how good our model is here we use the accuracy.

```
#----------Compile---------------#
model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

Fig 6.3.1: Compilation of the model

6.3.1 Adam optimizer

Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

Adam is different to classical stochastic gradient descent. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training.

A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds.

The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically:

- **Adaptive Gradient Algorithm** (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).

- **Root Mean Square Propagation** (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).
- Adam realizes the benefits of both AdaGrad and RMSProp.

Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance).

Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages.

6.3.2 Loss function

Categorical Cross-Entropy loss

Also called **Softmax Loss**. It is a **Softmax activation** plus a **Cross-Entropy loss**. If we use this loss, we will train a CNN to output a probability over the CC classes for each image. It is used for multi-class classification.

6.3.3 Accuracy metric

This metric creates two local variables, total and count that are used to compute the frequency with which y_pred matches y_true. This frequency is ultimately returned as binary accuracy: an idempotent operation that simply divides total by count. If sample_weight is None, weights default to 1. Use sample_weight of 0 to mask values.

Arguments

- name: (Optional) string name of the metric instance.
- dtype: (Optional) data type of the metric result.

6.4 TRAINING THE MODEL

In deep learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'validation set'. An important point to note is that during training the classifier only uses the training set. The validation set must not be used during training the classifier. The validation set will only be available during testing the classifier.

- Training set - a subset to train a model.(Model learns patterns between Input and Output)
- Validation set - a subset to test the trained model.(To test whether the model has correctly learnt)

An epoch is a term used in machine learning and indicates the number of passes of the entire training dataset the machine learning algorithm has completed. Datasets are usually grouped into batches (especially when the amount of data is very large). Some people use the term iteration loosely and refer to putting one batch through the model as an iteration.. This helps in attaining good accuracy to the model. I have specified 20 epochs.

Batch size is taken as 16 because instead of sending single image for learning at a time 16 images are send of same time for learning. Verbose is taken as 1 ecause it shows the progress. The model is goimg to fit over the x_train (training phase) and y_train (Testing phase) with their outputs x_ test and y_test respectively for supervised learning.

```
[20]
     #-----------Training-----------#
     history = model.fit_generator(
         augs_gen.flow(x_train,y_train,batch_size=16),
         validation_data  = (x_test,y_test),

         epochs = 20,
         verbose = 1,
     )
```

Fig 6.4.1: Training the model

```
Epoch 1/20
20/20 [==============================] - 3s 142ms/step - loss: 1.1422 - accuracy: 0.7019 - val_loss: 0.3815 - val_accuracy: 0.9114
Epoch 2/20
20/20 [==============================] - 2s 86ms/step - loss: 0.2718 - accuracy: 0.9071 - val_loss: 0.3696 - val_accuracy: 0.9241
Epoch 3/20
20/20 [==============================] - 2s 81ms/step - loss: 0.1871 - accuracy: 0.9455 - val_loss: 0.2936 - val_accuracy: 0.9114
Epoch 4/20
20/20 [==============================] - 2s 82ms/step - loss: 0.1744 - accuracy: 0.9423 - val_loss: 0.2583 - val_accuracy: 0.9494
Epoch 5/20
20/20 [==============================] - 2s 83ms/step - loss: 0.3317 - accuracy: 0.9167 - val_loss: 0.1638 - val_accuracy: 0.9620
Epoch 6/20
20/20 [==============================] - 2s 80ms/step - loss: 0.1336 - accuracy: 0.9647 - val_loss: 0.2660 - val_accuracy: 0.9367
Epoch 7/20
20/20 [==============================] - 2s 82ms/step - loss: 0.1064 - accuracy: 0.9583 - val_loss: 0.1703 - val_accuracy: 0.9494
Epoch 8/20
20/20 [==============================] - 2s 83ms/step - loss: 0.1365 - accuracy: 0.9615 - val_loss: 0.6072 - val_accuracy: 0.9241
Epoch 9/20
20/20 [==============================] - 2s 83ms/step - loss: 0.1374 - accuracy: 0.9487 - val_loss: 0.3288 - val_accuracy: 0.9620
Epoch 10/20
20/20 [==============================] - 2s 80ms/step - loss: 0.0993 - accuracy: 0.9647 - val_loss: 0.3136 - val_accuracy: 0.9620
Epoch 11/20
20/20 [==============================] - 2s 83ms/step - loss: 0.0857 - accuracy: 0.9744 - val_loss: 0.6087 - val_accuracy: 0.9367
Epoch 12/20
20/20 [==============================] - 2s 84ms/step - loss: 0.0561 - accuracy: 0.9776 - val_loss: 0.2827 - val_accuracy: 0.9494
Epoch 13/20
20/20 [==============================] - 2s 85ms/step - loss: 0.0526 - accuracy: 0.9840 - val_loss: 0.5104 - val_accuracy: 0.9241
Epoch 14/20


20/20 [==============================] - 2s 84ms/step - loss: 0.1307 - accuracy: 0.9679 - val_loss: 0.2979 - val_accuracy: 0.9494
Epoch 15/20
20/20 [==============================] - 2s 82ms/step - loss: 0.1078 - accuracy: 0.9647 - val_loss: 0.5850 - val_accuracy: 0.9367
Epoch 16/20
20/20 [==============================] - 2s 81ms/step - loss: 0.1604 - accuracy: 0.9583 - val_loss: 0.4538 - val_accuracy: 0.9367
Epoch 17/20
20/20 [==============================] - 2s 82ms/step - loss: 0.0386 - accuracy: 0.9872 - val_loss: 0.3443 - val_accuracy: 0.9367
Epoch 18/20
20/20 [==============================] - 2s 83ms/step - loss: 0.1588 - accuracy: 0.9551 - val_loss: 0.2733 - val_accuracy: 0.9494
Epoch 19/20
20/20 [==============================] - 2s 82ms/step - loss: 0.0871 - accuracy: 0.9776 - val_loss: 0.5933 - val_accuracy: 0.9367
Epoch 20/20
20/20 [==============================] - 2s 81ms/step - loss: 0.0767 - accuracy: 0.9744 - val_loss: 0.1663 - val_accuracy: 0.9747
```

Fig 6.4.2: Training Summary

CHAPTER-7

RESULT ANALYSIS

7.1 MODEL VALIDATION

Model Validation is the process of evaluating a trained model on test data set. This provides the generalization ability of a trained model. Here I provide a step by step approach to complete first iteration of model validation in minutes.

- The model are validate after completion of training and testing the model.
- Checking the accuracy scores as metrics to validate the models.

7.1.1 TRAIN AND TEST ACCURACY GRAPH

Visualisation of graph between train accuracy and validation accuracy

```
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']
# Visualisation of graph between train accuracy and validation accuracy
plt.figure(figsize=(12,12))
epochs = list(range(1,21))
plt.plot(epochs,train_acc,label='train_acc',)
plt.plot(epochs,val_acc,label='val_acc')
plt.xlabel("epochs",fontsize=20)
plt.ylabel("accuracy pecentage",fontsize=20)
plt.title('Accuracy graph',fontsize=30)
plt.legend()
```

Fig 7.1.1.1: Train accuracy vs Validation accuracy input

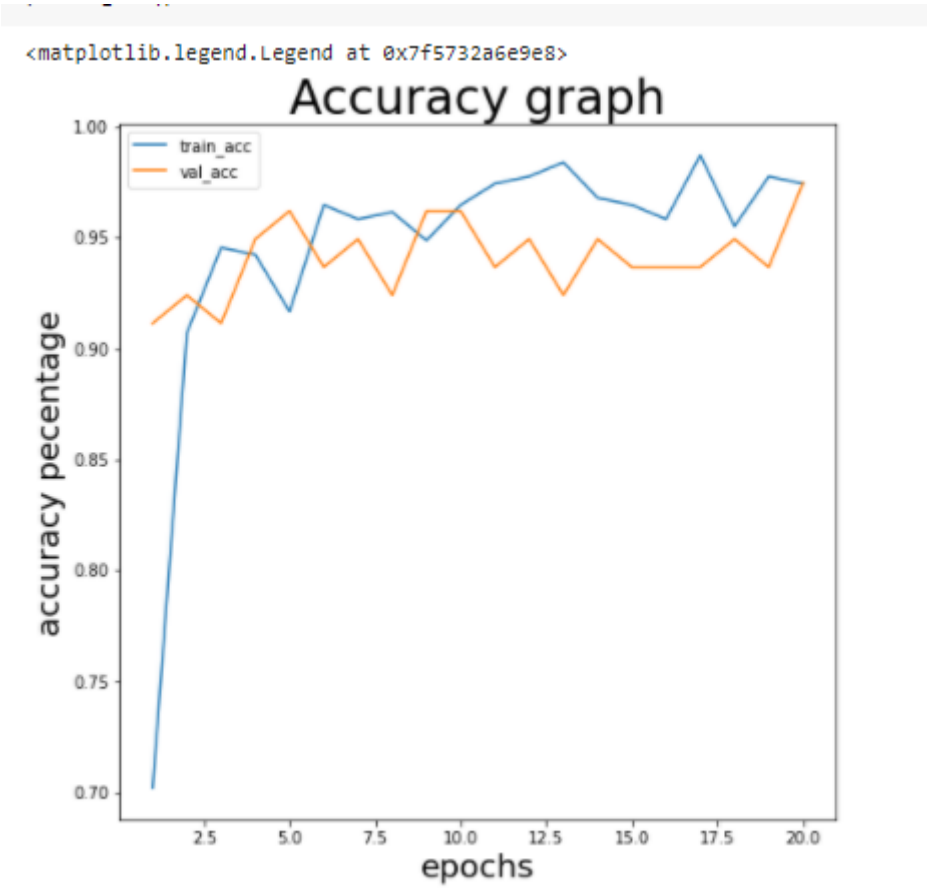

Fig 7.1.1.2: Train accuracy vs Validation accuracy output

7.1.2 TRAIN AND TEST LOSS GRAPH

Visualisation of graph between train loss and validation loss

```
# Visualisation of graph between train loss and validation loss
plt.figure(figsize=(8,8))
plt.plot(epochs,train_loss,label='train_loss')
plt.plot(epochs,val_loss,label='val_loss')
plt.xlabel("epochs",fontsize=20)
plt.ylabel("loss percentage",fontsize=20)
plt.title('loss graph',fontsize=30)
plt.legend()
```

Fig 7.1.2.1: Train loss vs Validation loss input

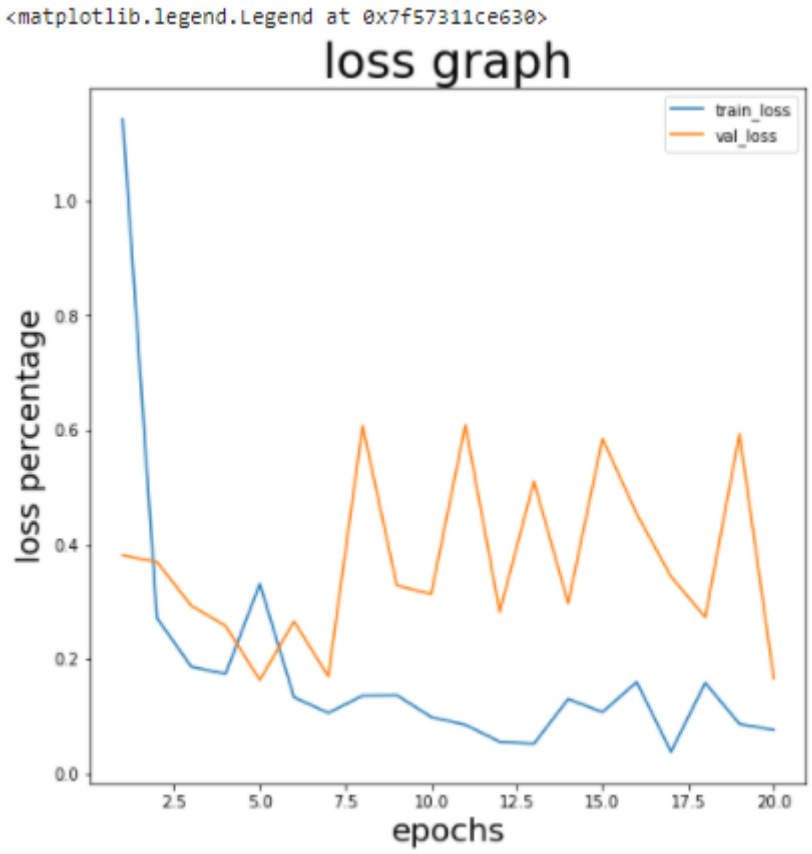

Fig 7.1.2.2: Train loss vs Validation loss output

Observations:

i. Training accuracy is 97.4%
ii. Validation accuracy is 97.4%
iii. Loss percentage is also very less. So, the model has trained well.
iv. By observing from the above graph test accuracy and validation accuracy doesn't have much variation. So, the model doesn't over fit or under fit.

7.2 MAKE PREDICTIONS:

We have a method called predict, using this method we need to predict the output for given input and we need to compare that the model is predicting correctly or not. If the model is predicting correctly then the model is good otherwise some changes should be done.

Now, I am trying to predict an unknown image which is not present in the training or validation set.

Below image is an unknown image of dog taken from internet with a different size. Now it should predict the breed of the dog. In order to predict the new image we need to resize and scale the new image. So, that image is classified without any errors.

```
[ ]  # Predicting unknown dog image
     img_url='/content/drive/MyDrive/dogs datasset/36.jpg'
     new_image = image.load_img(img_url)
     print(type(new_image))

     new_image = tf.keras.preprocessing.image.img_to_array(new_image)
     print("shape of image :",new_image.shape)
     print("Type of image :",type(new_image))

     # resizing
     new_image = tf.image.resize(new_image,(150,150))

     ## Scaling
     new_image = new_image/255
     print('After resizing the image shape :',new_image.shape)
     new_image = np.expand_dims(new_image,axis=0)
     print("image shape",new_image.shape)
```

```
<class 'PIL.JpegImagePlugin.JpegImageFile'>
shape of image : (209, 241, 3)
Type of image : <class 'numpy.ndarray'>
After resizing the image shape : (150, 150, 3)
image shape (1, 150, 150, 3)
```

Fig 7.4.1: Resizing and scaling unknown sample image

- Firstly, we import image from TensorFlow, then we load the image and paste the URL of image we are predicting.
- After loading we convert the image to array format and then check the shape and type of image. Now we will be using this during resizing the image we are predicting on to convert it into required shape.
- Next, we will be applying scaling on the image and to get the dimensions right, expand dims enters the process, finally we check the shape and observe that we got the desired result.
- The predicted values is in the probability wise. So the highest probability given class is considered as the dog breed of that particular assigned label of breed.

```
model.predict(new_image)
```

```
array([[8.3306239e-14, 1.7070663e-12, 2.0833822e-13, 1.3240163e-12,
        1.0000000e+00, 1.5724538e-10, 5.0116415e-14]], dtype=float32)
```

```
array_num = model.predict(new_image)
num_list = array_num.tolist()
x=num_list[0]
item = max(x)
#search for the item
index = x.index(item)
breed_classes=list(zip(breed,breed_name))
q=0
while q<7:
  if index in breed_classes[q]:
    print('Dog Breed: '+ breed_classes[q][1])
    break
  else:
    q=q+1
img = mpimg.imread(img_url)
imgplot = plt.imshow(img)
plt.show()
```

```
Dog Breed: Maltese
```

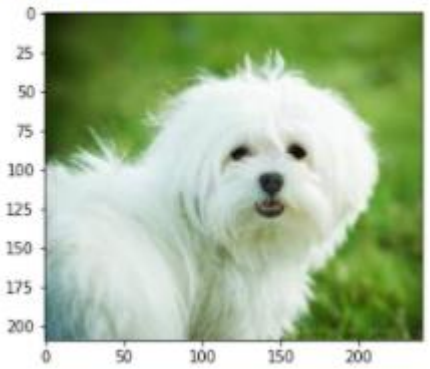

Fig 7.4.2: Predicting dog breed

Observation: The model has predicted the breed of unknown dog image correctly. So the model is well performing in classifying the given class of dog breeds.

CHAPTER-8

CONCLUSION

This project proposes a method to classify dog breeds from various dog images using transfer learning. We demonstrated an application of transfer learning in the context of dog breed classification and introduce the concept of retraining the Inception model released by Google.Inc to classify dog image data. In this experiment, 391 dog images belonging to 7 different dog categories are used to retrain a model. We achieved a remarkable 97.4% accuracy in correctly classifying a new data. In transfer learning, we are mostly concerned with training the fully connected layers because the convolutional layers act as feature extractors; however there have been advantages in also training the final convolutional layer, so this would be an avenue worth exploring.

In future work, we hope to explore some more classification models and make them from scratch. We also wish to do the image processing parts like filter values, feature extraction, edge detection ourselves instead of using a pre-trained model and then compare the results. One of the most exciting things that can be done is Ensemble Learning i.e. combining the results of various classification models and observing the results. In addition, we hope to introduce approaches that can take heterogeneous features and temporal dimensions into account with the techniques mentioned.

REFERENCES

[1] Punyanuch Borwarnginn, “Breakthrough Conventional Based Approach for Dog Breed Classification Using CNN with Transfer Learning”, IEEE, 12 December 2019

Link: https://ieeexplore.ieee.org/abstract/document/8929955

[2] J. Liu, A. Kanazawa, D. Jacobs, and P. Belhumeur, “Dog Breed Classification Using Part Localization”, Computer Vision–ECCV 2012. Springer Berlin Heidelberg, 2012. 172-185.

[3] A. Krizhevsky, I. Sutskever, and G. Hinton. “Imagenet classification with deep convolutional neural networks”, Advances in neural information processing systems. 2012.

[4] Wright, J.C. and Nesselrote, M.S., 1987. Classification of behavior problems in dogs: distributions of age, breed, sex and reproductive status. Appl. Anita. Behav. Sci., 19: 169-178.

[5] Kang, Tim, "Using Neural Networks for Image Classification" (2015). Master's Projects. Paper 395.

[6] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," in IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345-1359, Oct. 2010.

REFERRED WEBSITES

1. https://en.wikipedia.org/wiki/Transfer_learning#:~:text=Transfer%20learning%20(TL)%20is%20a,when%20trying%20to%20recognize%20trucks.
2. http://cs231n.stanford.edu/reports/2015/pdfs/automatic-dog-breed.pdf
3. https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a
4. https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/
5. https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
6. https://keras.io/api/applications/inceptionv3/
7. https://www.kaggle.com/jessicali9530/stanford-dogs-dataset