Code & Module Title

CU6051NA Artificial Intelligence

Assessment Weightage & Type

80% Individual Coursework

Year and Semester

2019-20 Autumn

Student Name: Rajat Shrestha

London Met ID: 17030954

College ID: np01cp4a170021

Assignment Due Date : 7th February 2020

Assignment Submission Date: 7th February 2020

Abstract

This report discusses the details for creating a solution for classifying dogs or cats an exercise released by Kaggle. This can be done by fine-tuning the pre-trained model of VGG-16 which is a popular industry standard architecture of Convolutional Neural Network but since this approch is quite complex a simpler CNN architecture was used. This project discusses the topics such as Computer Vision, Artificial Intelligence, Machine Learning, Deep Learning, Supervised Learning, Classification, Confusion matrix, Perceptrons/Neurons, Activation Function, Artificial Neural Networks, Feed Forward Neural Networks and Multilayer Perceptrons/ Deep Neural Networks. Later Convolution Neural Network, LeNet-5 architecture, CNN layers, CNN operations, CNN architectures, VGG-16 and Image Augmentaiton are also discussed in-depth for creating an effective solution. Various other works solving related were observed to understand how this problem is being tackled and its application. In the last section the elaborate plans, the core solution, pseudocode and flowchart is created to demonstrate how the final artifact is working to solve the problem.

Abbreviations:

Abbreviations	Full Forms
AI	Artificial Intelligence
ML	Machine Learning
NN	Neural Network
ANN	Artificial Neural Network
FFNN	Feed Forward Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit
VGG-16	Visual Geometry Group – 16 (A CNN architecture)
CV	Computer Vision
ASSIRA	Animal Species Image Recognition for Restricting Access
CAPTCHA	Completely Automated Public Turing Test to Tell Computers and Humans Apart
CUDA	Compute Unified Device Architecture
GPU	Graphical Processing Unit
CPU	Central Processing Unit
TPU	Tensor Processing Unit (Specialized device provided by Google in Colab)

Table of Contents

Table of Figures:

Table of Tables:

1. Introduction

1.1. Artificial Intelligence

Artificial intelligence or AI is the process of mimicking intelligence by machines, especially computer systems. These processes include learning, reasoning and self-correction. Artificial intelligence (AI) is a wide-ranging branch of computer science concerned with building machines capable of performing tasks that cannot be explicitly programmed. AI is an interdisciplinary science with multiple approaches, but advancements in machine learning and deep learning are creating a paradigm shift in virtually every sector of the tech industry (Raschka, 2015). There are various proposed definitions on what an AI is during different periods such as:

- Acting humanly: The Turing Test approach
- Thinking humanly: The cognitive modelling approach
- Thinking rationally: The laws of thought approach
- Acting rationally: The rational agent approach

All the definitions suggest AI being a simulation of Intelligence within machines that were thought to be unique to only intelligent living things. Since the unravelling of this discovery, there have been numerous attempts to automating various tasks and enormous efforts in improving them. Now the machines can do various intelligent tasks and are being used to solve various problems such as Knowledge reasoning, Planning, Machine learning, Natural language processing, Computer vision, Robotics, Artificial general intelligence and many more (Norvig & Russell, 2003). Now due to the rise in Big data, improved computing power (advancements in GPU), and improved algorithms have revived the development in AI. In this report, the research and development details will be elaborated on creating an algorithm that can recognize objects from images.

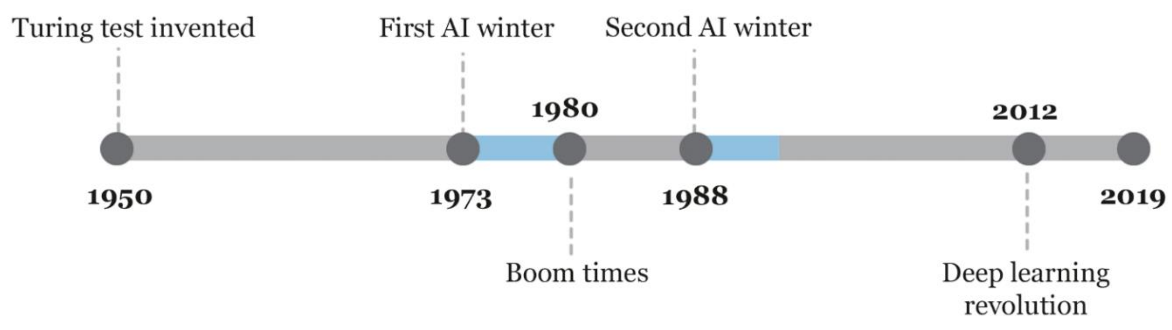


Figure 1: Key events during the development of AI in recent time (Schuchmann, 2019)

1.1.1. Machine Learning

Machine learning is an application of artificial intelligence (AI) that provides systems with the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves (Raschka, 2015). The process of learning begins with training using the data provided to make better predictions for the outcome in the future. The primary aim is to allow the computers to learn automatically without being explicitly programmed and adjust actions accordingly.

There are four types of Machine learning Techniques (Burkov, 2019) which involves different procedures for training machines:

- Supervised Learning
- Unsupervised Learning
- Re-enforcement Learning
- Semi-supervised Learning

1.1.2. Deep Learning

Deep learning is a class of machine learning algorithms that uses deep neural networks (ANN with multiple hidden layers) to progressively extract higher-level features from the raw input. The lower layers may identify edges, while higher layers may identify the concepts relevant to a topic. A convolutional neural network is a class of deep neural networks which is quite effective in analysing digital images. They are also known as shift invariant or space invariant artificial neural networks, based on their shared-weights architecture and translation invariance characteristics (Nielsen, 2015).

Artificial intelligence describes the algorithms that try to mimic human intelligence. Machine learning is one of the AI application where algorithms are designed to make the machine learn rather than to program every possible input, and deep learning is one of those machine learning techniques which uses multi-layer Perceptrons(Hidden Neural Networks) (Goodfellow, et al., 2016).

1.2. Computer Vision

Computer Vision, often abbreviated as CV, is defined as a field of study that seeks to develop techniques to help computers “see” and understand the content of digital images such as photographs and videos by extracting their useful information (Prince, 2012). The problem of computer vision appears simple because but largely remains an unsolved problem based both on the limited understanding of biological vision and because of the complexity of visual perception. Due to the recent development in AI (especially Deep Learning), a vast use cases of computer vision from recognizing faces, identifying handwritten digits, surveillance, and many more have been drastically improved (Szeliski, 2011). Computer Vision discusses a wide range of topics including Image processing, Neural Networks with Multi-Layer Perceptrons and discussion of a wide range of algorithms to detect entities which is crucial for this project.

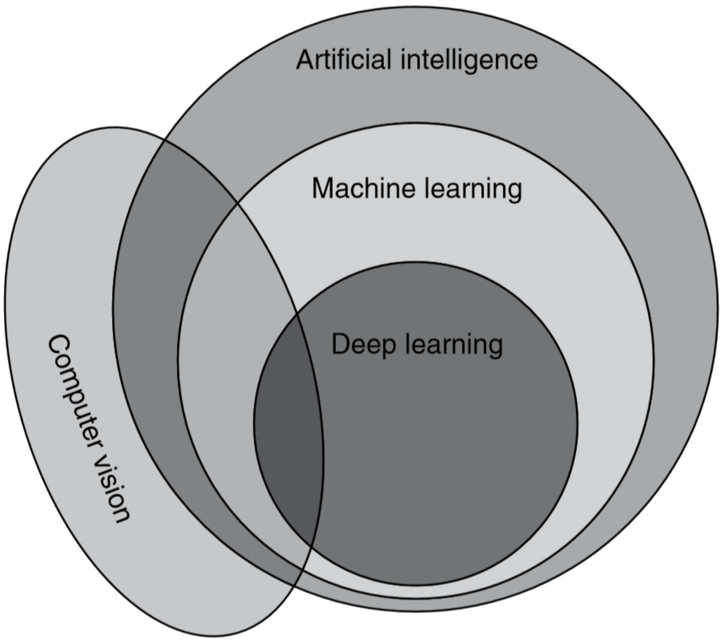


Figure 2: Venn-diagram of AI-nomenclature and Computer vision (Qasim Aziz, et al., 2018)

1.3. Cat-Dog classifying problem

The Dogs vs. Cats classification is a standard computer vision task that involves classifying photos as either containing a dog or cat using the best algorithm to run on computers. Although the problem sounds simple, it was only effectively addressed in the last few years using deep learning convolutional neural networks. While the dataset is effectively solved, it can be used as the basis for learning and practising how to develop, evaluate, and use convolutional deep learning neural networks for image classification from scratch. Also, there are plenty of works done by thousands of people who have been involved in this topic involving detailed information on creating, comparing, and in-depth mechanisms on effectively solving this problem making this topic easy to understand (Brownlee, 2019).

1.3.1. ASIRRA CAPTCHA

Microsoft Research (MSR) proposed the problem of discriminating cats from dogs as a test to tell humans from machines and created the ASIRRA test on this basis. The assumption is that, out of a batch of twelve images of pets, any machine would predict incorrectly the family of at least one of them, while humans would make no mistakes. The complete MSR ASIRRA system is based on a database of several million images of pets, equally divided between cats and dogs (Saul, et al., 2007). A CAPTCHA is simply a program that protects websites against bots by generating and grading tests that filters out computers. CAPTCHAs have been used extensively on the web for Preventing spams/dictionary attacks and protecting various web services prone to brute-force attacks (Carnegie Mellon University, 2010).

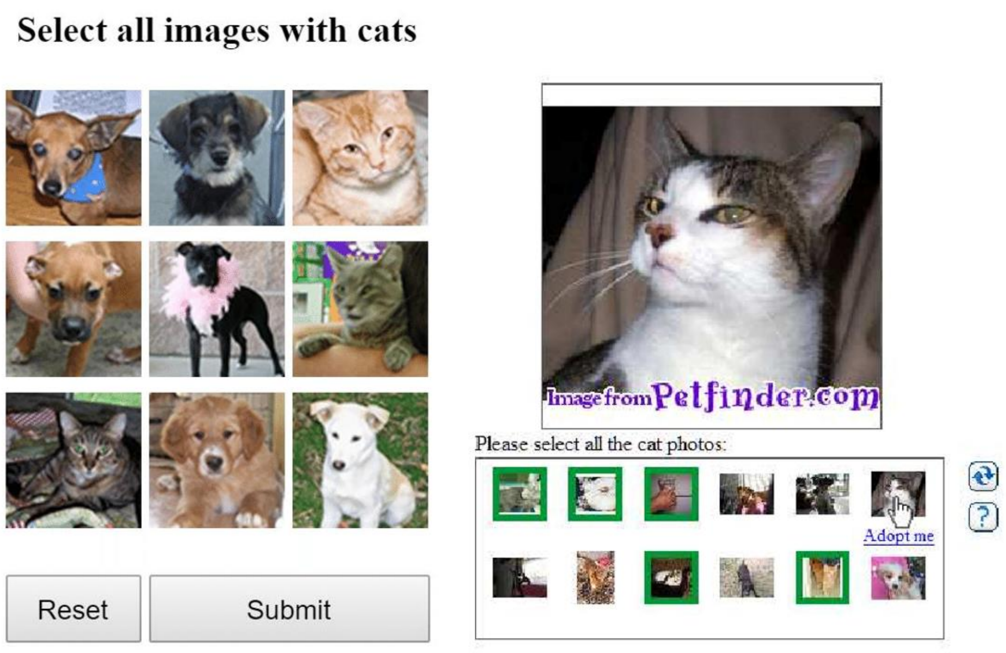
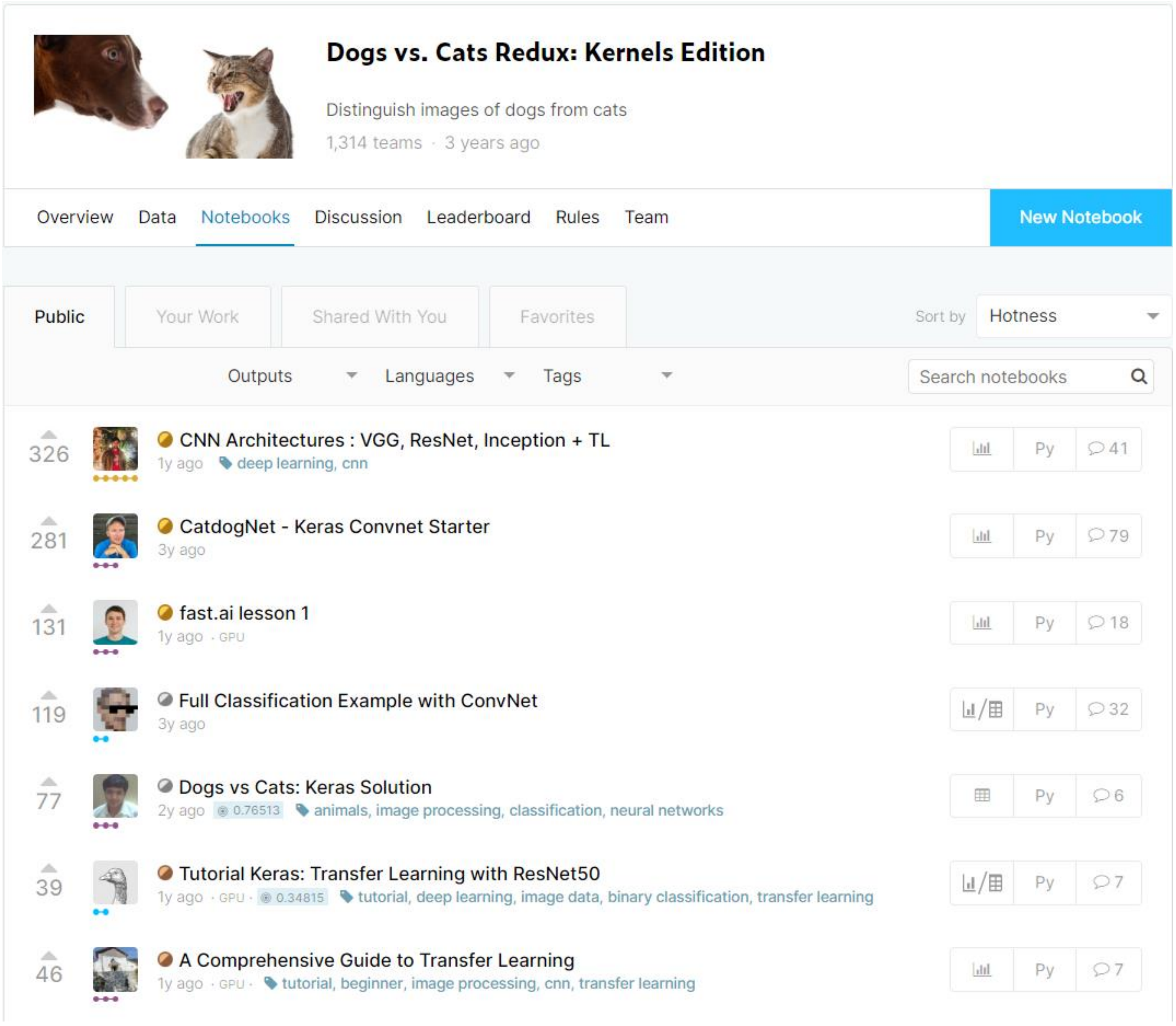

Select all images with cats
Reset
Submit
Image from Petfinder.com
Please select all the cat photos:
Adopt me

Figure 3: Examples of ASSIRA CAPCHA used in throughout the web

1.3.2. Kaggle competition

Kaggle released a competition in 2013 to write an algorithm to classify whether images contain either a dog or a cat. This is easy for humans, but computers cannot distinguish cats and dogs by a picture. The main purpose of this competition is to recognize the limits of such a CAPTCHA system to filter out robots. The current literature suggests machine learning classifiers can score above 80% accuracy on this task so using this metric is not safe for the security of a system (Golle, 2008). Now this challenge has been set to benchmark the latest computer vision and deep learning approaches to this problem so Kaggle has released another challenge after improvement in their platform in 2016. Kaggle provided 25000 labelled pictures of dogs and cats for training and 12500 unlabelled pictures for testing in the form of zipped files. The predicted output should be recorded in a CSV file.


Dogs vs. Cats Redux: Kernels Edition

Distinguish images of dogs from cats

1,314 teams · 3 years ago

Votes	Notebook	Details	Language	Comments
326	CNN Architectures : VGG, ResNet, Inception + TL	1y ago · deep learning, cnn	Py	41
281	CatdogNet - Keras Convnet Starter	3y ago	Py	79
131	fast.ai lesson 1	1y ago · GPU	Py	18
119	Full Classification Example with ConvNet	3y ago	Py	32
77	Dogs vs Cats: Keras Solution	2y ago · 0.76513 · animals, image processing, classification, neural networks	Py	6
39	Tutorial Keras: Transfer Learning with ResNet50	1y ago · GPU · 0.34815 · tutorial, deep learning, image data, binary classification, transfer learning	Py	7
46	A Comprehensive Guide to Transfer Learning	1y ago · GPU · tutorial, beginner, image processing, cnn, transfer learning	Py	7

Figure 4: Kaggle Competition for Dogs vs cats 2016 top public notebooks

2. Background

2.1. Literature Review

To understand the topic and solve the problem various topics were gone through. Since the chosen topic is quite vast and overlaps various other topics only the fundamental basic topics required for solving the problem are described in detail.

2.1.1. Classification (Supervised Learning)

Supervised learning is a type of machine learning where the model is fed with enough information with labelled data for training and validating, so that, based on the learning it has done, it can predict the outcome for a new dataset in the testing phase. Classification is a technique in Machine Learning which labels the new data fed to the machine using the model created by supervised learning using the labelled previous data (Gopalakrishnan & Venkateswarlu, 2018).

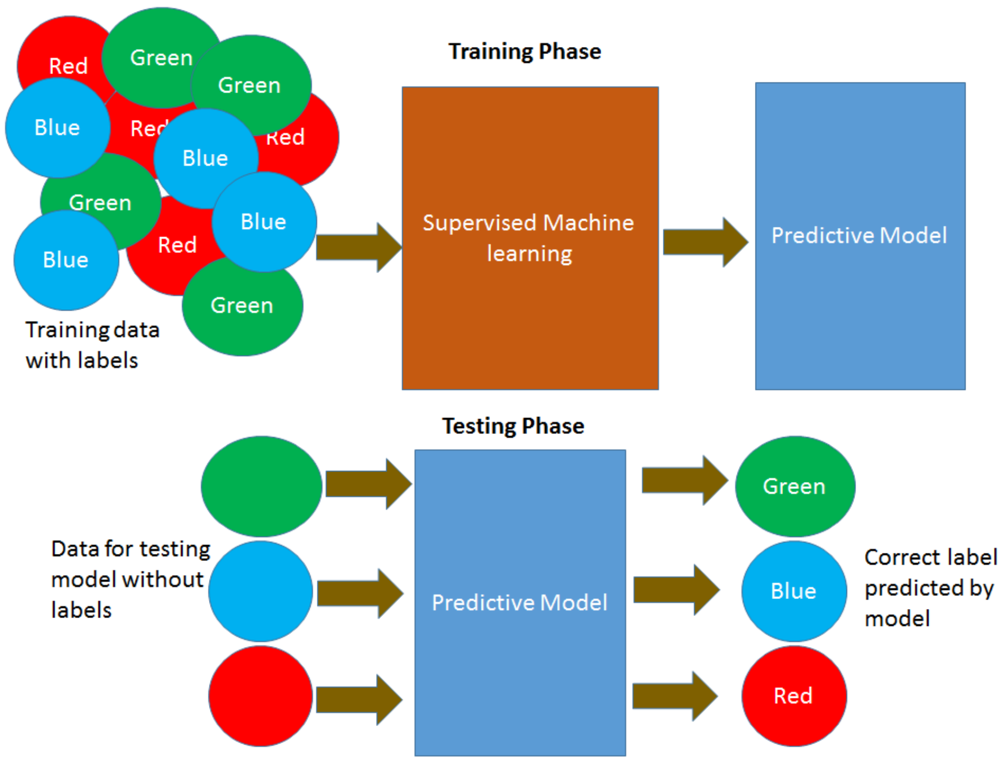

Red
Green
Green
Blue
Red
Red
Blue
Green
Blue
Red
Blue
Green
Training data with labels
Training Phase
Supervised Machine learning
Predictive Model
Testing Phase
Data for testing model without labels
Predictive Model
Green
Blue
Red
Correct label predicted by model

Figure 5: Training and testing phase of the classification model using supervised learning

2.1.2. Perceptron

The perceptron is the simplest form of a neural network also called an artificial neuron or simply neuron because it is modelled after the actual biological neurons found in animals, used for the separation of patterns which is said to be linearly separable (Haykin, 2003). A perceptron consists of a single neuron with adjustable synaptic weights and bias and is limited to performing classification on only two classes but if multiplying the output capability of a perceptron by using more than one this limitation can be removed. Each Perceptron in an artificial neural network has an activation function to give the final output of that neuron (Ding, et al., 2018).

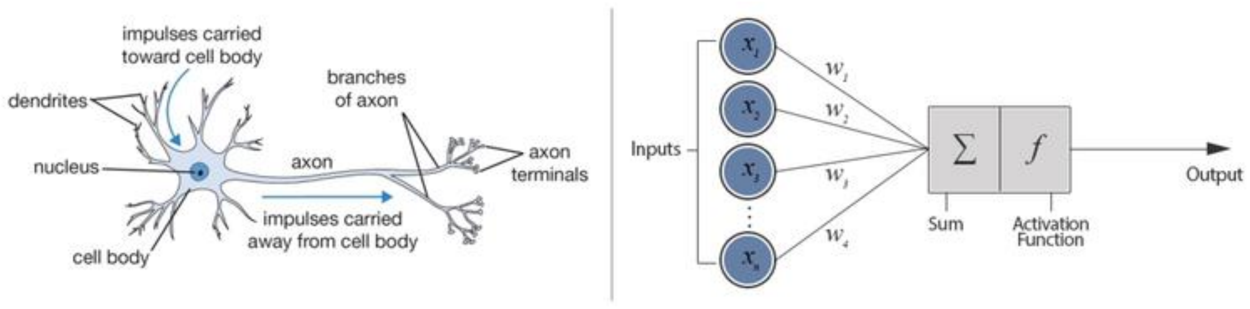


Figure 6: A biological neuron compared with a perceptron (Willems, 2019)

2.1.3. Activation Functions

Activation functions compute the weighted sum of input and biases, which is used to decide the final output of a neuron. It manipulates the presented data through some gradient processing and afterwards produces an output for the neural network, that contains the parameters in the data. These activation functions can be either linear or non-linear depending on the function it represents and is used to control the outputs of out neural networks across different domains (Marshall, et al., 2018). Most activation functions are non-linear, and they are chosen in this way on purpose as it allows the neural networks to compute arbitrarily complex functions.

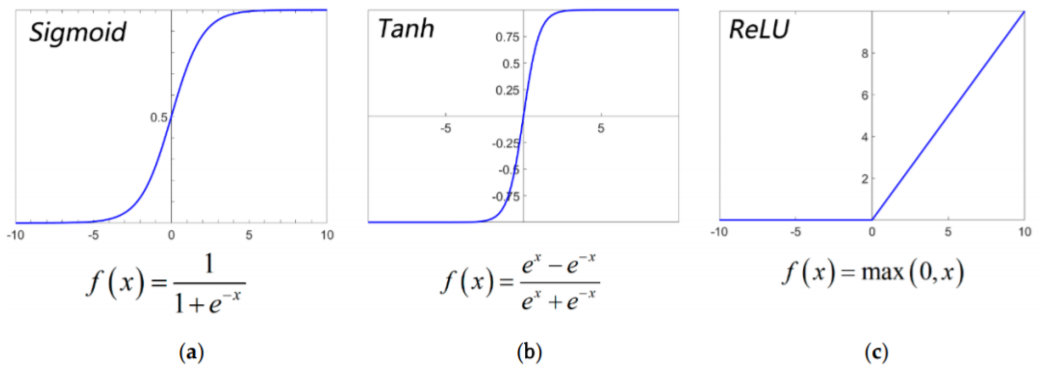


Figure 7:Activation functions and its characteristics: (a) Sigmoid; (b) Tanh; (c) ReLU (Zhang, et al., 2019).

2.1.4. Neural Networks

Neural Networks(NN) also referred to as Artificial Neural Networks(ANN) are simply networks of multiple Perceptrons loosely modelled after a biological brain. A deep neural network is a type of Artificial Neural Network architecture with one or more hidden layers(layer of neurons between the input and output layers) they are also often referred to as Multi-Layered Perceptrons due to their arrangements of neurons (Sarle, 1994). Each connection in a neural network enables the neuron to transmit a signal to other neurons. Neural networks often consist of large numbers of “neurons”.

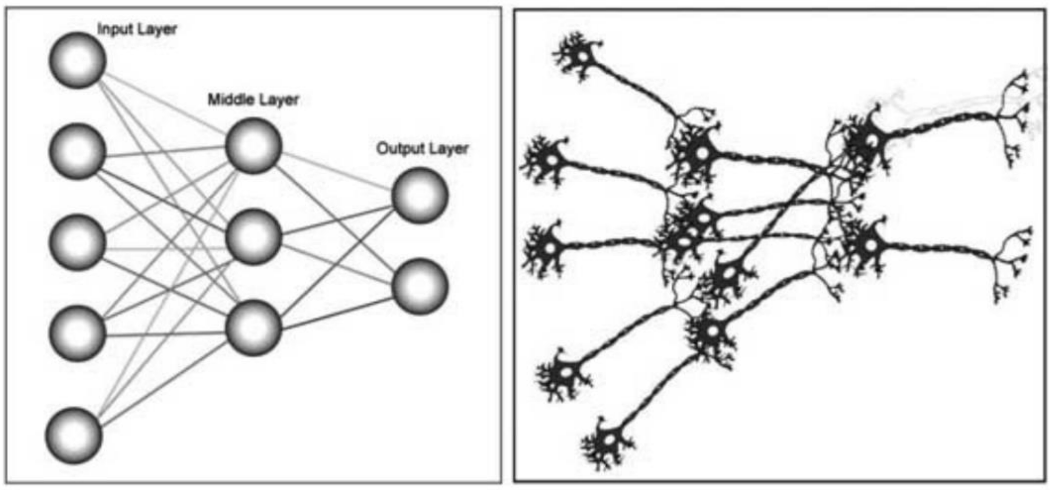


Figure 8: Artificial Neural Network compared to biological neurons (Vaezzadeh, et al., 2011)

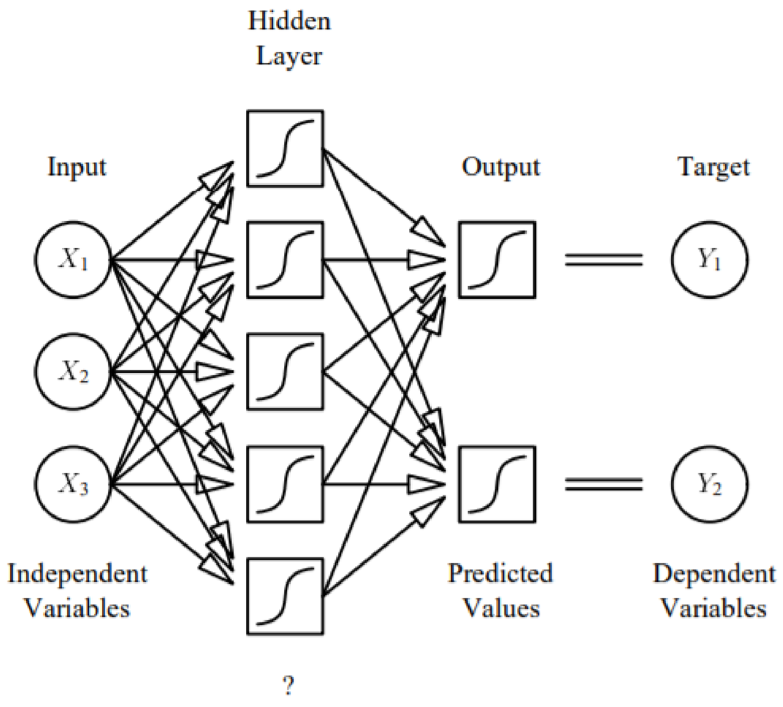


Figure 9: Example of a deep neural network (Sarle, 1994)

2.1.5. Convolutional Neural Networks

A convolutional Neural network or CNN is a special kind of Feed Forward Neural Network(FFNN) With multiple hidden layers it can be classified as a deep neural network (Burkov, 2019). CNN is designed to recognize features with a high degree of invariance to translation, scaling, skewing and other forms of distortion (LeCun & Bengio, 1995). Basically, it reduces the images into a form that is easier to process, without losing features which are critical for getting a good prediction and are specifically designed to take advantage of structure in input data. Therefore, they work so well for image processing and computer vision tasks. CNN involves feature extraction, mapping, and subsampling for recognizing images (Veeranjaneyulu & Bodapati, 2019).

2.1.6. LeNet-5 Architecture

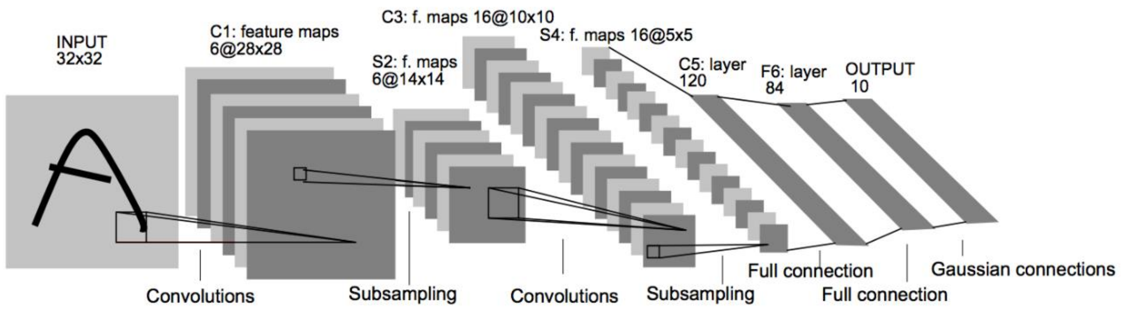
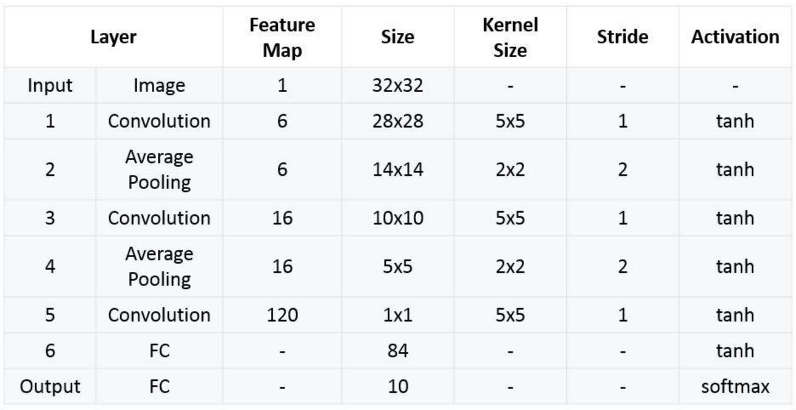


Figure 10: Architecture of LeNet-5, example of Convolutional Neural Network (Lecun, et al., 1998)

The above image represents the architecture of LeNet-5 which was a popular CNN architecture. Yann LeCun, Leon Bottou, Yosuha Bengio and Patrick Haffner proposed this architecture for handwritten and machine-printed character recognition in 1998 in an article released in Proceedings of IEEE. This architecture is straightforward and simple to understand with the following layers:



Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Figure 11: LeNet-5 Architecture Summarized Table (Rizwan, 2018)

2.1.7. AlexNet

AlexNet was developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton in 2012 to compete in the ImageNet competition. The general architecture of AlexNet is quite like LeNet-5, although this model is considerably larger. The success of this model convinced a lot of the computer vision community to take a serious look at deep learning for computer vision tasks. AlexNet used ReLU activation instead of Tanh to add non-linearity accelerating the speed, used dropout instead of regularisation to deal with overfitting (doubled training time with the dropout rate of 0.5) and most effectively used overlap pooling to reduce the size of the network (Krizhevsky, et al., 2012).

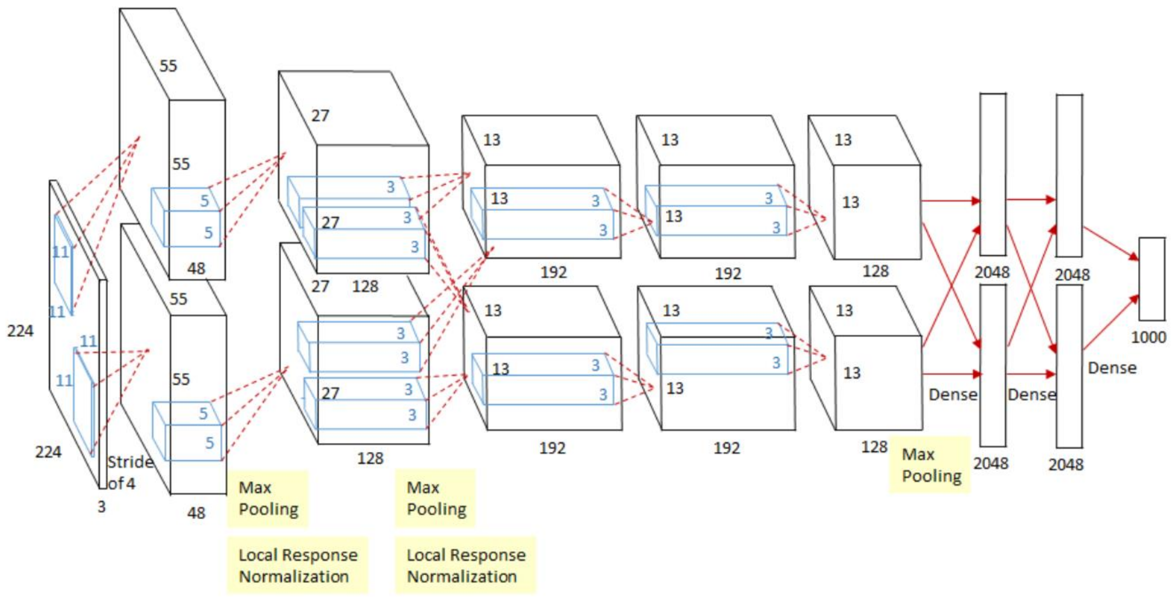


Figure 12: AlexNet 2012 using 2 GPUs(Krizhevsky, et al., 2012)

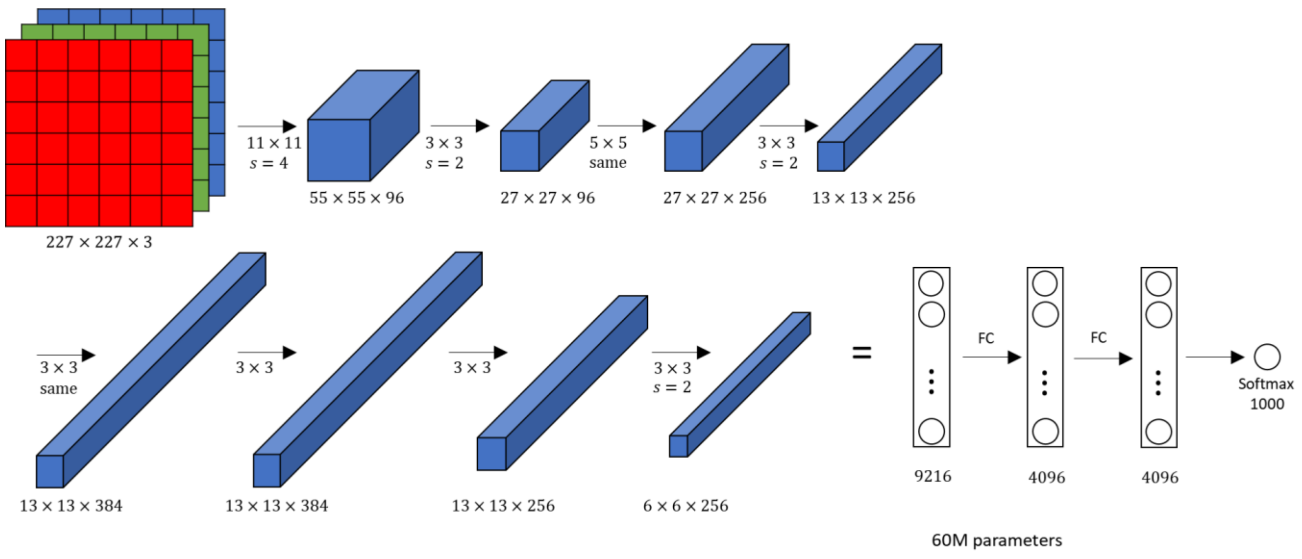


Figure 13: AlexNet mechanism to classify colourful images (Krizhevsky, et al., 2012)

2.1.8. VGG 16 Convolutional Network Architecture

After the success of AlexNet in 2012 VGG-16 or VGG-Net was introduced by Karen Simonyan and Andrew Zisserman to compete in the ImageNet challenge. This architecture was the First runner-up after Google's GoogleNet architecture. Its main contribution was in showing that the depth of the network is a critical component for good performance. VGG-16 contained 13 convolutional layers, 5 pooling layers, and 3 dense layers. VGG-16 features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end. The downsides of the VGG16 are that it is more expensive to evaluate and uses a lot more memory, parameters (140 Million) and produces a model with large size. Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters (Zisserman & Simonyan, 2014).

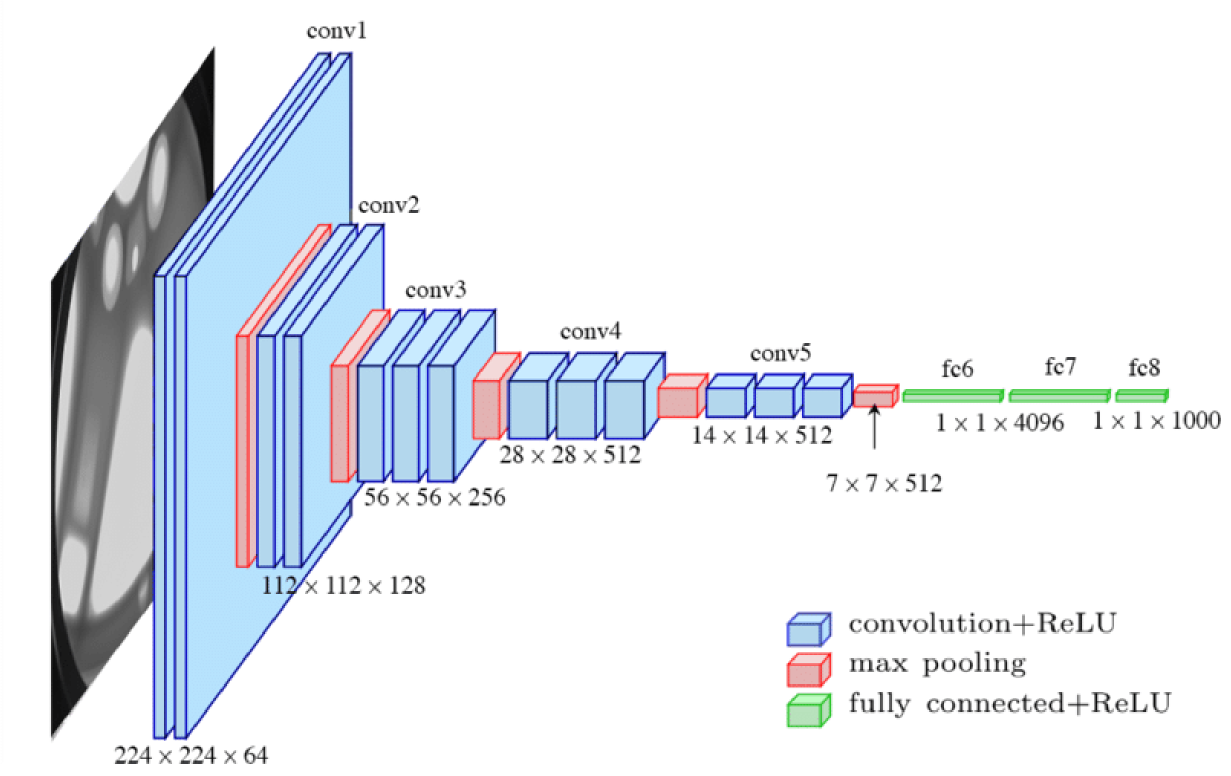


Figure 14: Standard VGG-13 Architecture (Ferguson, et al., 2017)

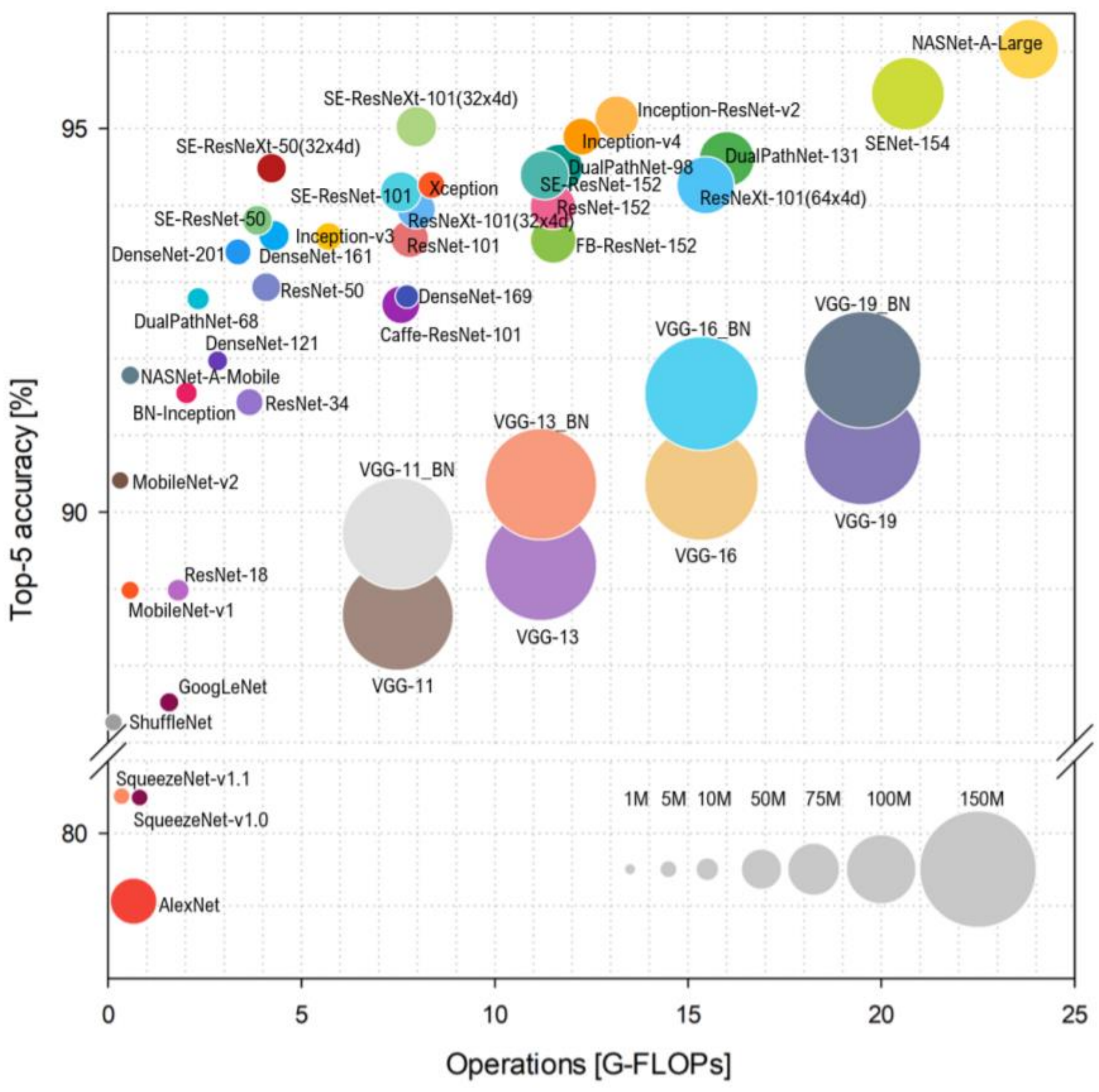


Figure 15: Comparison between various CNN Architectures in terms of top-5 accuracy and operation (Bianco, et al., 2018)

2.1.9. CNN layer types

Any CNN architecture requires to be carefully modelled with various layers which will increase the contrast to the features gradually and carefully reduces the image file to a single dimension to give the final output. The various general layers used in CNN are as follows the convolution operation and max-pooling is detailed after this section:

1. **Input layer**

   The layer takes the input usually in the form of an image.

2. **Convolutional layer:**

   The convolutional layer serves to detect (multiple) patterns in multiple sub-regions in the input field using receptive fields between neurons. Simply, it computes the dot product between the weights and a small patch in the output of the previous layer (Marco & Farinella, 2018).

3. **Rectified Linear Unit layer:**

   This layer applies an activation function to the output of the previous layer to add non-linearity to the network so that it can generalize well to any type of function (Campbell, 2017).

4. **Pooling layer:**

   The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network. The pooling layer operates on each feature map independently. Max pooling is frequently used in the pooling layer the maximum value in a given in height-width measurements (Marco & Farinella, 2018).

5. **Dense layer**

   The dense layers in a CNN are usually 2-3 layers of MLP that that is used to map activation volume from the combination of previous layers into a class probability distribution (Karim, et al., 2018).

2.1.10. Convolutional Operations in Convolution Layer

Convolution is the process of adding each element of the image to its local neighbours, weighted by the kernel and is related to mathematical convolution operation. The convolutional layer simply makes edges and details sharper, by multiplication of the part of the image to a same-sized kernel matrix also known as convolution matrix or mask. A kernel matrix is a matrix which when multiplied with the same sized part of the image will produce the new image with added effects. This effect can be used to sharpen, blur or enhance edges in images. A convolution operation will reduce the size of the image but can easily be mitigated by adding a masking layer to the input image to give an output of the same size (Goodfellow, et al., 2016).

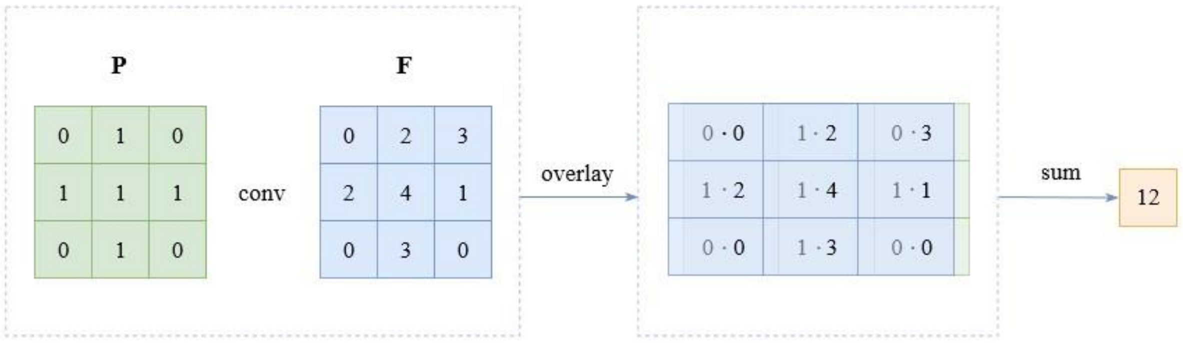


Figure 16: A Convolution Operation (Burkov, 2019)

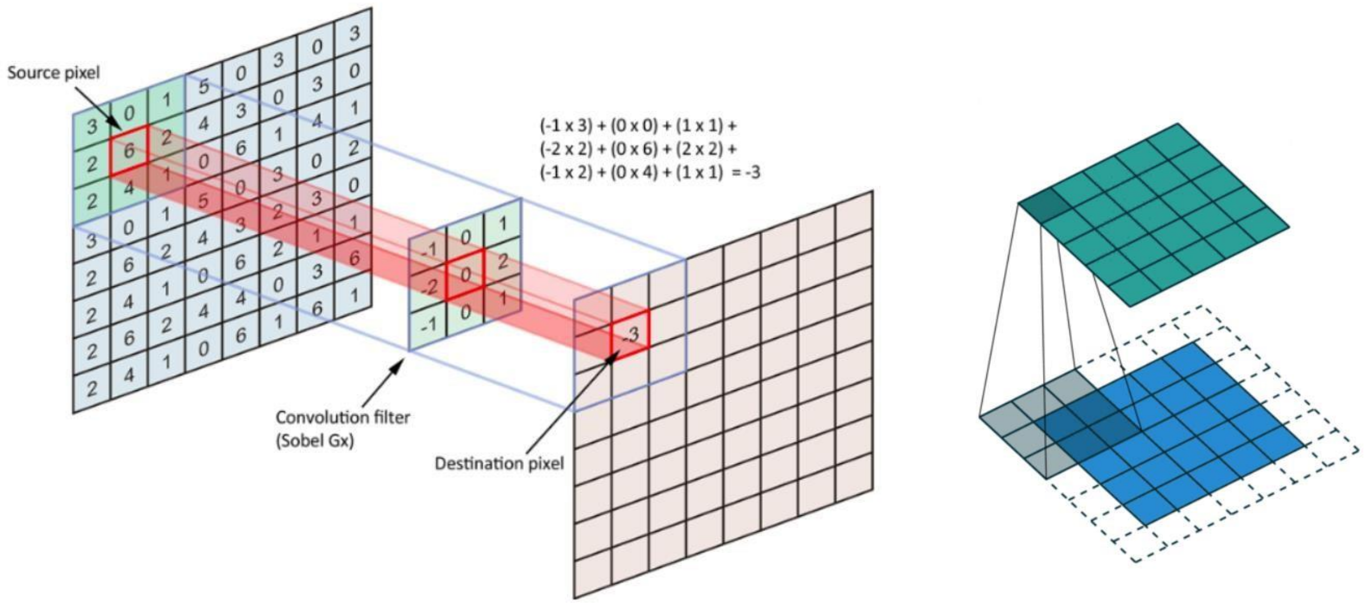

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 17: Convolution filter being applied, with and without padding (Burkov, 2019) (Campbell, 2017)

2.1.11. Max Pooling in Pooling Layer

Pooling is like convolution, as a filter applied using a sliding window approach. However, instead of applying a trainable filter to an input matrix or a volume, the pooling layer applies a fixed operator, usually either max or average. Similarly, to convolution, pooling has hyperparameters: the size of the filter and the stride. An example of max pooling with filter of size 2 and stride 2 is by shrinking the 2 by 2 matrix to a single maximum value within the matrix as shown in figure 19. Usually, a pooling layer is applied after a convolution layer to highlight the main feature. Unlike convolution operation, when pooling is applied to a volume, each matrix in the volume is processed independently of others. Therefore, the depth of the volume is conserved in pooling. Max pooling is used more often than average and often gives better results by highlighting the main feature and decreasing the chance of overfitting. Pooling contributes to the increased accuracy of the model and improves the speed of training due to the reduction in the number of parameters of the neural network (Burkov, 2019).

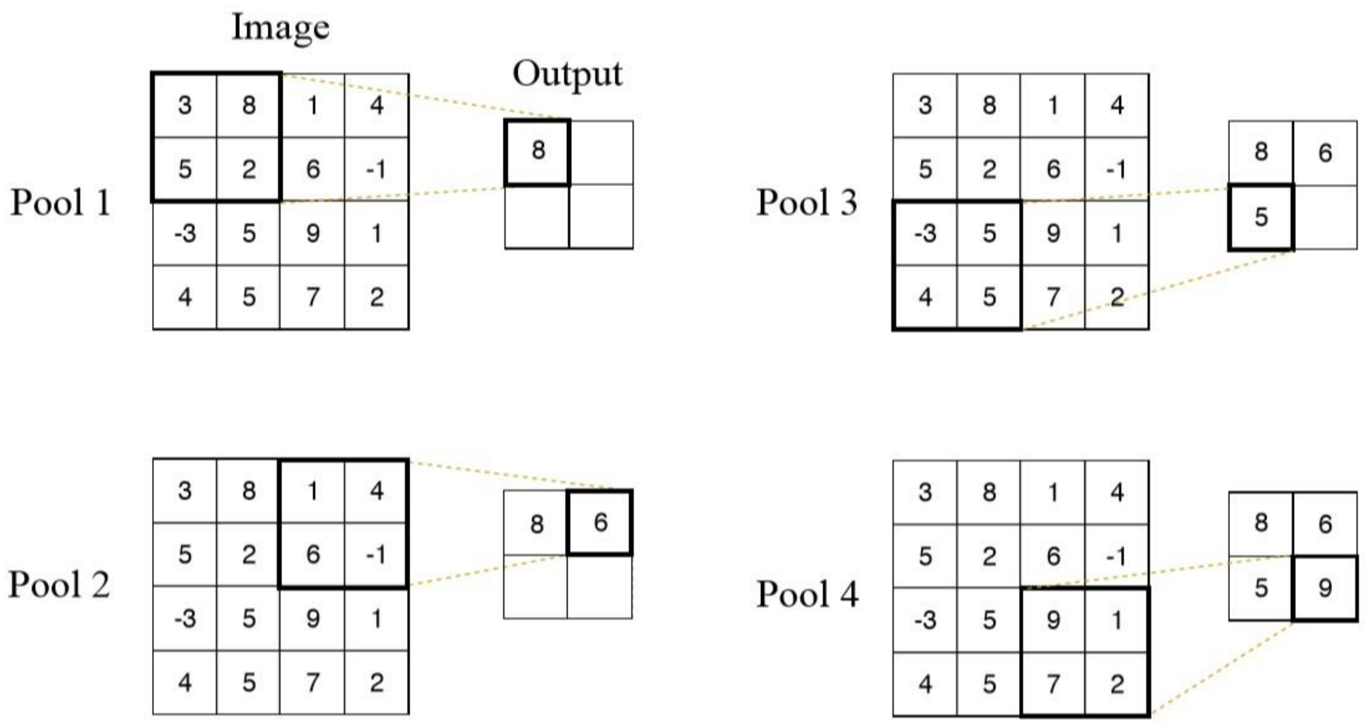


Figure 18: Max-pooling with the size of 2 and stride of 2 (Burkov, 2019)

2.2. Existing Works:

Since the dog's vs cat's classification was a world-wide competition, there are thousands of solutions created. Among them, there were various notebooks that are quite useful for understanding the problem in depth. The most notable source was Kaggle and there are also many blogs and online instructions on solving the problem. The guides and notebooks are discussed more in Appendix (A). The Dog-cat classifier might not seem a serious topic but the underlying mechanism in this classification problem can be used to tackle various image classification problems. CNN can be used in Facial Recognition, analysing medical imagery, understanding Climate, OCR and many more.

2.2.1. ML binary classifier for gastrointestinal disease

. Binary classifiers aim to stratify allocation to a categorical outcome, such as the presence or absence of gastrointestinal disease. This will help in data-driven procedures to detect the disease. Furthermore, such analyses could predict the development of GI disease prior to the manifestation of symptoms, raising the possibility of prevention or pre-treatment. This defines the recent developments in healthcare-based AI/machine learning and its essence in this field. This is one of the many applications of Computer Vision in the medical field (Qasim Aziz, et al., 2018).

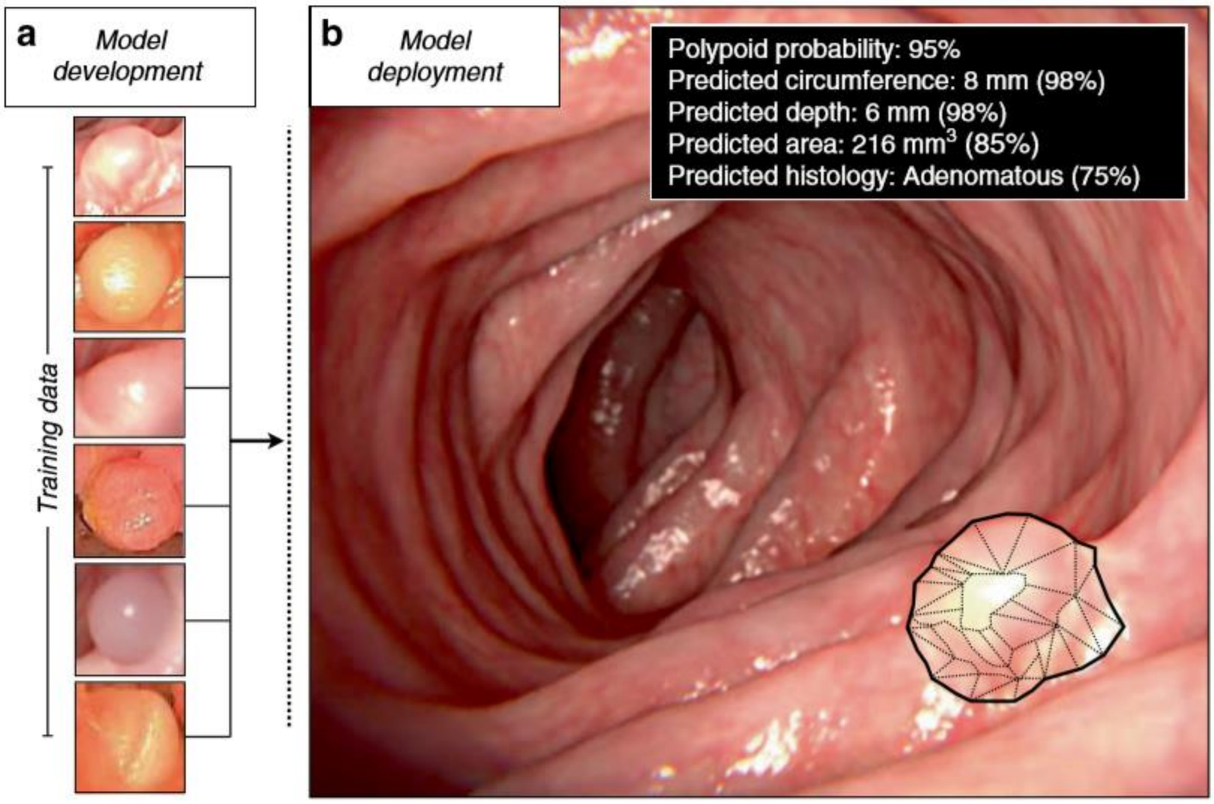

a
Model development
Training data
b
Model deployment
Polypoid probability: 95%
Predicted circumference: 8 mm (98%)
Predicted depth: 6 mm (98%)
Predicted area: 216 mm^3 (85%)
Predicted histology: Adenomatous (75%)

Figure 19: Application of CV to detect GI images (Qasim Aziz, et al., 2018)

2.2.2. CNN for Optical Character Recognition

One of the most frequent use cases of CNN is to digitalize hard copy data to soft copy (digital format). A simple CNN architecture can be used to detect a single letter that can then be used in a loop to detect a whole sentence. This has been used in Text-To-Speech Engines and Translation. This was first proposed by LeCun in his paper "Gradient-Based Learning Applied to Document Recognition" (Lecun, et al., 1998) which is discussed in LeNet-5 CNN architecture.

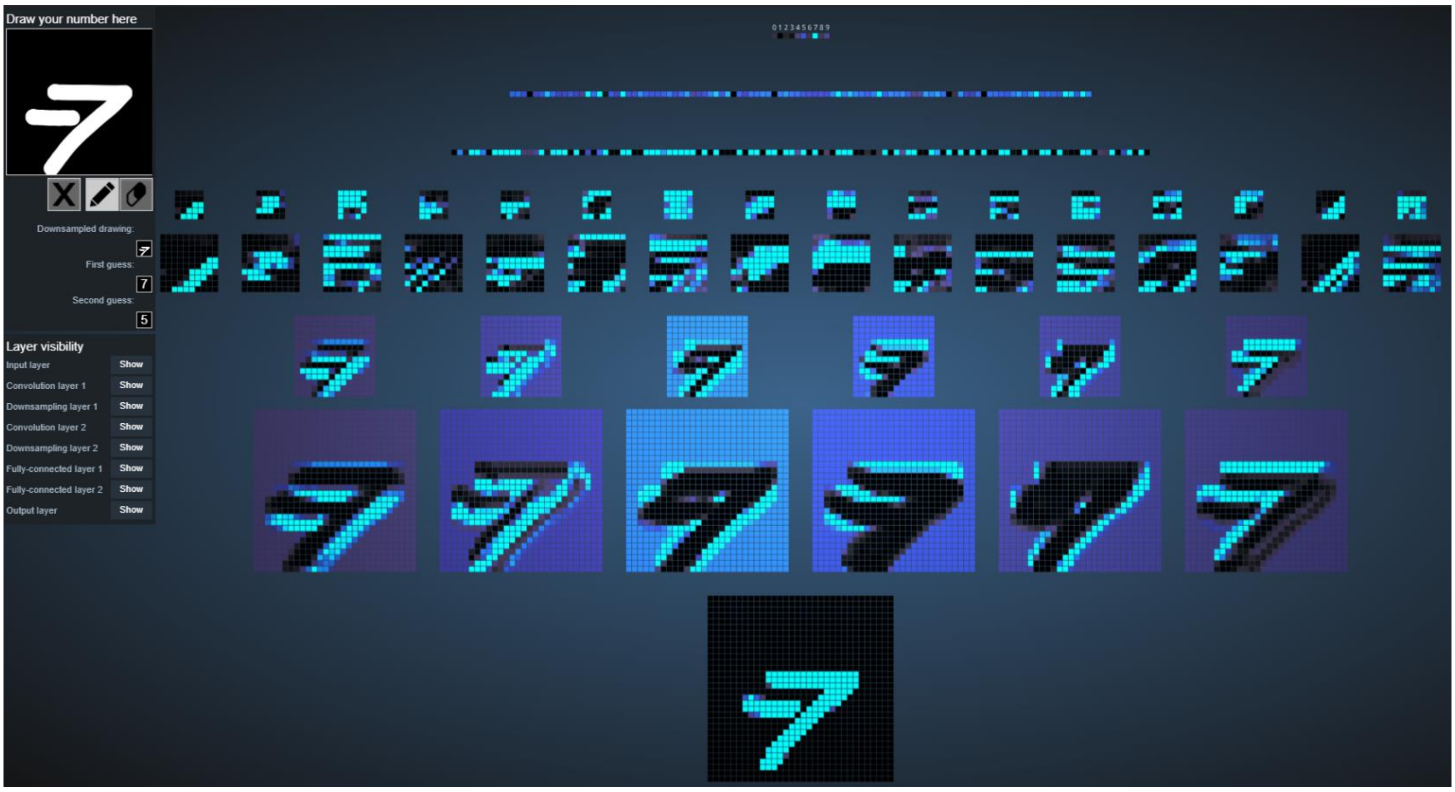


Figure 20: Visualization of a simple CNN architecture for digit recognition

3. Solution

Usually, in the past years, various methods were used to verify humans over automated bots on the internet like the ASSIRA CAPTCHA. But since the rise of the AI techniques, this technique has been obsolete. There have been more advantages of the existence of such an algorithm because now humans can automate machines on doing things that were thought to be impossible. The same algorithm used for classifying cats and dogs apart can be reused to create a more serious application that detects cancer by looking at the images, can identify faults in places humans are unable to reach and many more. The whole problem and the format of the solution are well defined in the previous section. This section describes the plan to fit the research done to tackle the problem by addressing all the concerns.

The tools to be used in the building phase are discussed in Appendix B. The previous proposed solution did not work accordingly so a simpler solution was implemented. The solution implemented in this problem is as follows:

3.1. Solutions Used:

Since the previous proposed solution was too complicated and was hard to implement, a simpler but effective solution is being implemented by training a simple model instead of performing Transfer learning to a pre-trained model. The current solution consists of the following algorithms and processes:

3.1.1. GPU Acceleration for Deep learning

Training Deep neural networks might take a lot of time to be trained but this task can easily be divided into parallel computing tasks and use GPUs for accelerated. Kaggle provides its Kaggle Kernel which allows 30hrs of GPU use per month. Google also provides GPU as well as TPU powered runtimes which is quite useful for deep learning projects. These online kernels were used extensively due to the easy availability and certain features. Due to various limitations and interrupted internet, the GPU acceleration was required to be installed in the local computer. To use TensorFlow GPU in local machine, a Nvidia Graphics card was required which is CUDA compatible with the latest Graphics Driver installed. Then Nvidia CUDA and cuDNN was installed which drastically improved the training time. “CUDA® is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers can dramatically speed up computing applications by harnessing the power of GPUs.” (Nvidia, 2020)

3.1.2. Image Augmentation:

Image augmentation artificially creates training images through different ways of processing or combination of multiple processing, such as random rotation, shifts, shear and flips, etc. This is ten put to a generator which will be fed into the trainer while training the model. This process increases the input training data and prevents overfitting by providing various another version of the same image. This proved quite promising as it increased the accuracy of the model significantly than just using 2000 labelled data (Lau, 2017).

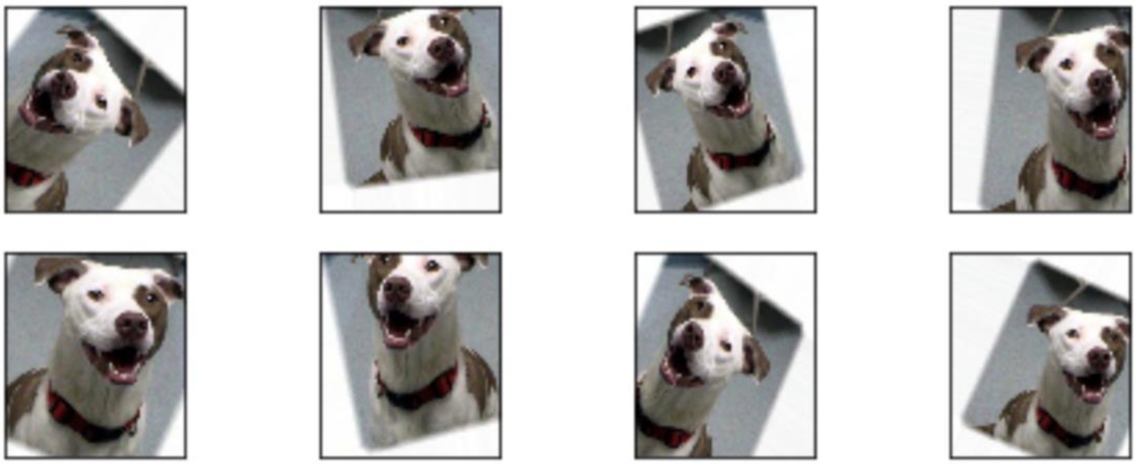


Figure 21: Augmenting an image of a dog in different ways

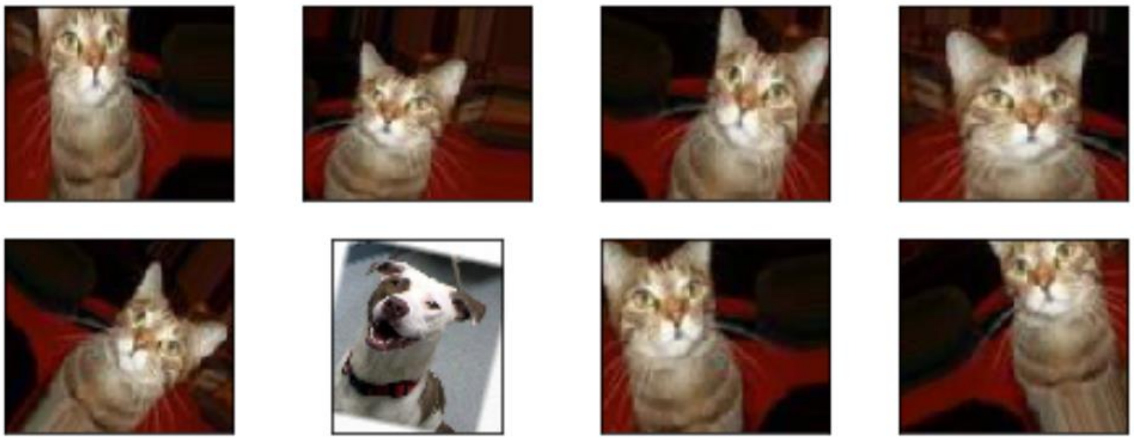
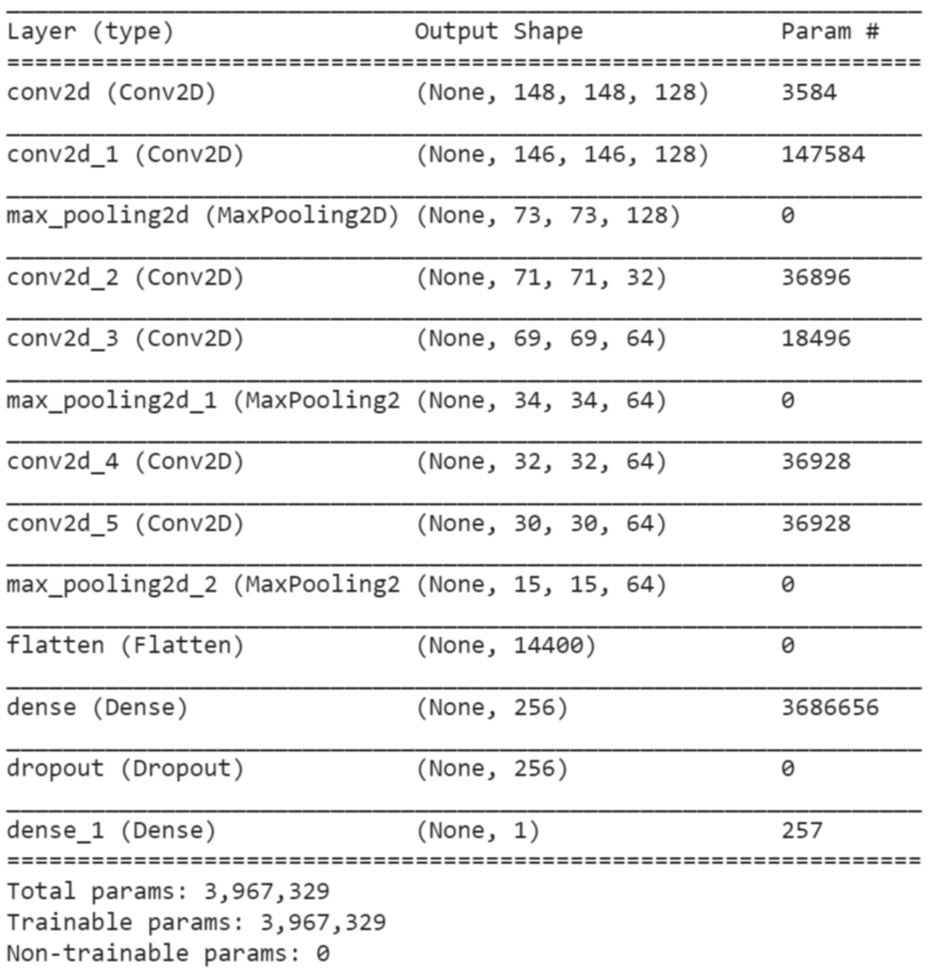


Figure 22: Augmenting another image of a cat in different ways

3.1.3. Designing a Simple Covnet Architecture

The following model was designed for the task of classification:

This model consists of 7 Convolution layers 3 Max pooling layers, and three dense layers for classification. The summary of the model is shown below and then the model was compiled to make it ready for training.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 128)     3584
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 146, 146, 128)     147584
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 73, 73, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 71, 71, 32)        36896
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 69, 69, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 34, 34, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 30, 30, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 15, 15, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 14400)             0
_________________________________________________________________
dense (Dense)                (None, 256)               3686656
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 3,967,329
Trainable params: 3,967,329
Non-trainable params: 0
```

Figure 23: Summary of the CNN Model

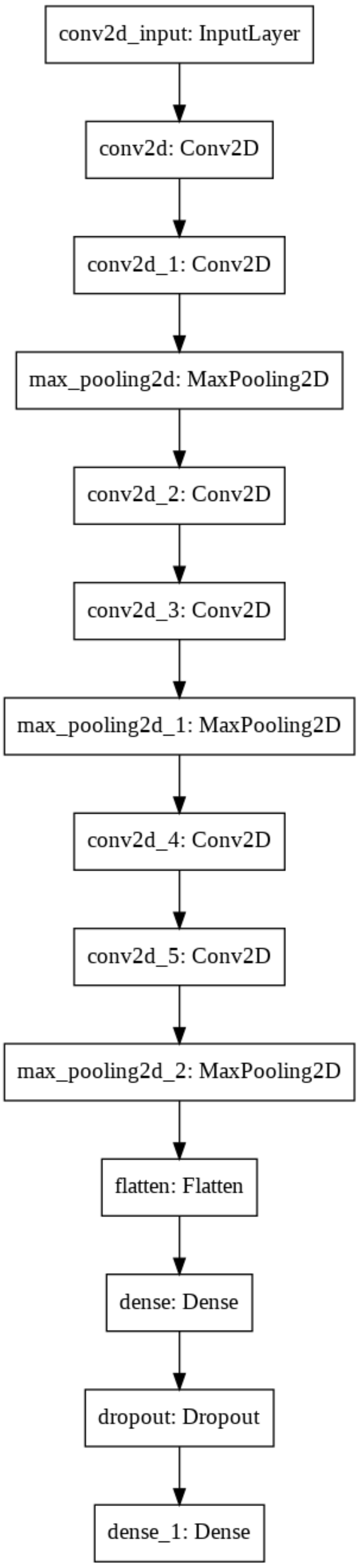
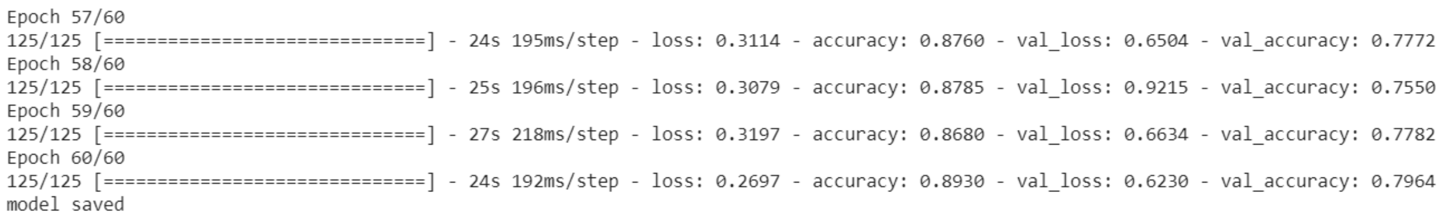


Figure 24: Model Details

3.1.4. Training The model

After Designing the model, the model was trained using the training data generator which can generate new augmented images from the limited training dataset. The history of the model while training is stored and will be later used for viewing the accuracy of the model. The model will be trained according to the provided parameters configured by the user. The epochs, steps for training/testing, batch size, sample size (training/testing), and model name is defined before training. It is recommended to use GPU acceleration for training a CNN model as CNN is embarrassingly parallel and will decrease the training time significantly.



```
Epoch 57/60
125/125 [==============================] - 24s 195ms/step - loss: 0.3114 - accuracy: 0.8760 - val_loss: 0.6504 - val_accuracy: 0.7772
Epoch 58/60
125/125 [==============================] - 25s 196ms/step - loss: 0.3079 - accuracy: 0.8785 - val_loss: 0.9215 - val_accuracy: 0.7550
Epoch 59/60
125/125 [==============================] - 27s 218ms/step - loss: 0.3197 - accuracy: 0.8680 - val_loss: 0.6634 - val_accuracy: 0.7782
Epoch 60/60
125/125 [==============================] - 24s 192ms/step - loss: 0.2697 - accuracy: 0.8930 - val_loss: 0.6230 - val_accuracy: 0.7964
model saved
```

Figure 25: Training the model and saving it

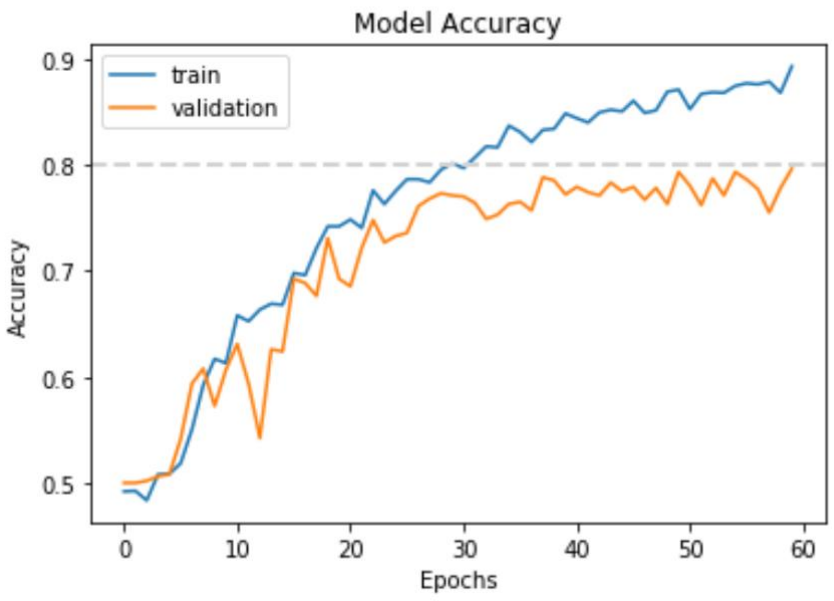


Figure 26: Training and validation accuracy of the model

3.1.5. Using the model to predict images

The Trained model was used to predict new images of cats and dogs. Since the model had validation accuracy less than 80%, so the predictions were a bit off on fuzzy looking images. The following results were obtained in the notebook:

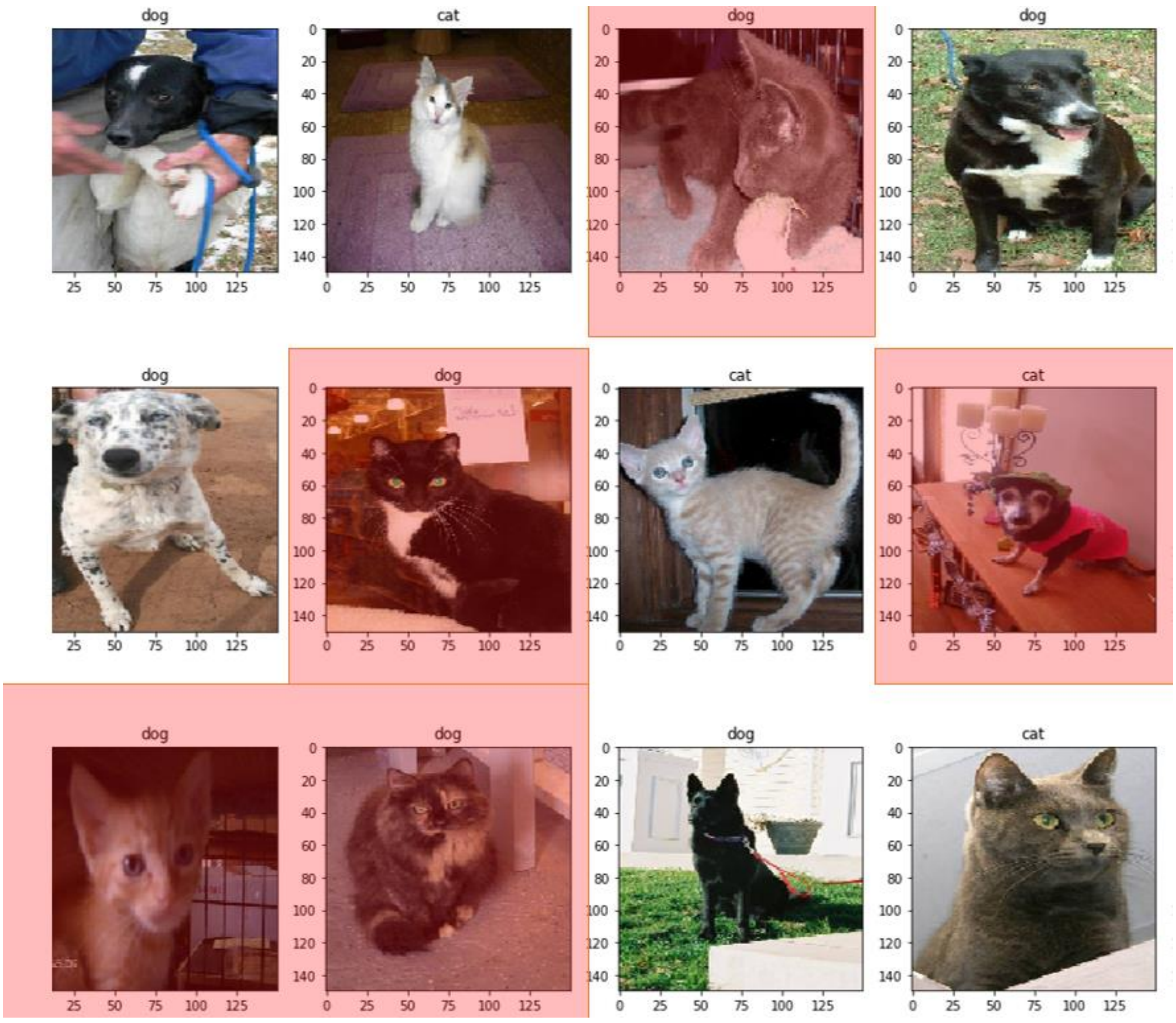


Figure 27: Using the trained model to Predict images of cats and dogs

3.2. Pseudocode

```
Import Modules (Deep learining libraries, Linear algebra, etc)
Import ASIRRA (training and testing data properly allocated)

training_data = ASIRRA.train
validation_data = ASIRRA.validation
testing_data = ASIRRA.testing

Define data_augmentation(input):
    z = zoom(input)
    r = rotate(input)
    f = flip(input)
    return (z,r,f)

Design(model):
    model.add(convolution_layer)
    model.ADD(CONVOLUTIONAL_LAYER(neurons=128, activation = RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=128, activation = RELU))
    model.ADD(POOLING_LAYER(pool_size=(2,2),strides=(2,2)))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=32, activation = RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=64, activation = RELU))
    model.ADD(POOLING_LAYER(pool_size=(2,2),strides=(2,2)))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=64, activation = RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=64, activation = RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=512, activation = RELU))
    model.ADD(POOLING_LAYER(pool_size=(2,2),strides=(2,2)))
    model.ADD(FLATTEN_LAYER())
    model.ADD(DENSE_LAYER(256, activation = RELU))
    model.add(DENSE_LAYER(1, activation = SIGMOID))

Compile(model):

    history = Train(model) using:
                     batch_size = 16
                     data_augmentation(training_data)
                     epochs = 60
                     training_slide=data_augmentation(training_data)//batch_size
                     validation_slide = validation_data//batch_size

    model.save()
    history.plot(accuracy,validation_accuracy)


define Predict(image):
    return model.predict(image) (and plot result)
```

3.3. Flow-chart

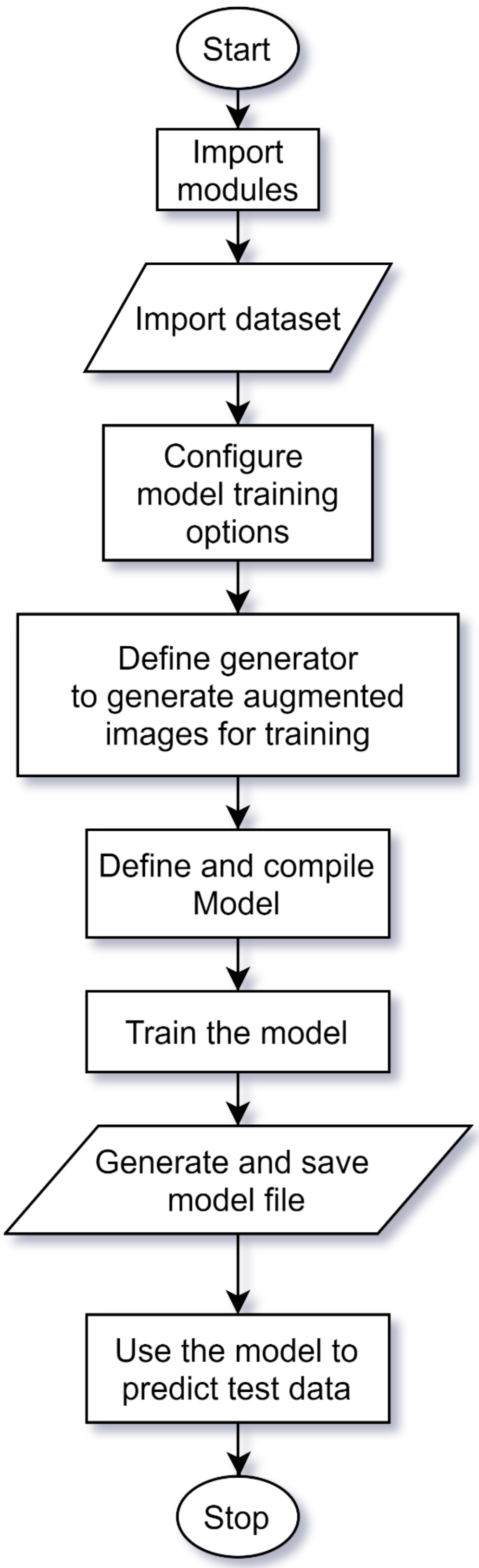

Start
Import modules
Import dataset
Configure model training options
Define generator to generate augmented images for training
Define and compile Model
Train the model
Generate and save model file
Use the model to predict test data
Stop

Figure 28: Flowchart of the solution

3.4. Development Process

The following steps were followed as described in the flowchart and the pseudocode

1. Installing GPU acceleration Nvidia CUDA and TensorFlow GPU
2. Importing Required modules
3. Configure training Configurations
4. Allocate data and check configurations
5. Checking the contents of training dataset
6. Image Augmentation examples
7. Prepare training and validation data
8. Simple model creation and compiling
9. Training the model
10. Plotting the results
11. Using the model to predict test data

3.4.1. Libraries requirements:

Table 1: Table of Essential Libraries

Library	Version	Description
Keras	2.2.4	Keras is a high-level neural networks API, capable of running on top of TensorFlow, CNTK, or Theano.
TensorFlow	2.1.0	TensorFlow is the core open source library to develop and train ML models. (GPU version was installed)
Pandas	0.25.1	Pandas is an open source library providing various data analysis tools for Python.
NumPy	1.16.4	NumPy is the library providing vital tools for matrix and array operations. It makes Linear Algebra tasks exponentially easier.
Matplotlib	3.1.2	Matplotlib is a Python 2D plotting library which produces charts and figures essential for data science projects.

3.4.2. Configuration

Table 2: Required configuration parameters to be specified

Parameter	**Function**
Batch Size	Batch size represents the number of samples processed before the model is updated
Epochs	Number of Iterations to be repeated for the training process
Model Filepath	String of filepath and name in which the model is to be saved
Steps	Training and validation steps per epochs

3.4.3. Allocating Data

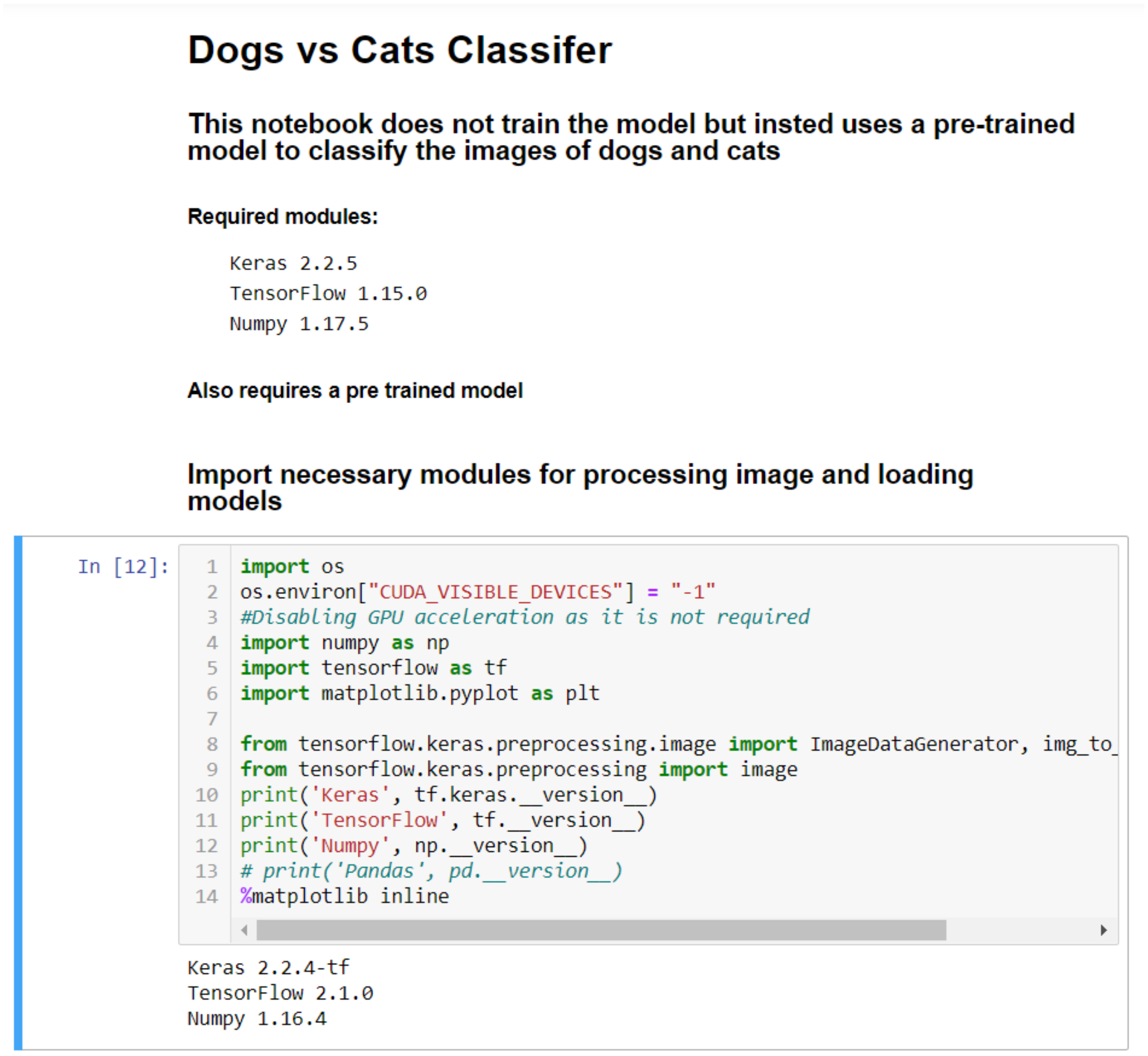
Dividing Labelled data according to the requirements, total 4000 pictures of cats and dogs were divided accordingly:

- 2000 labelled data of cats and dogs each for training
  - 1000 images of cats under ASIRRA/training/cat
  - 1000 images of dogs under ASIRRA/training/dog
- 1000 labelled data of cats and dogs each for validation (500*2 = 1000)
  - 500 images of cats under ASIRRA/validation/cat
  - 500 images of dogs under ASIRRA/validation/dog
- 1000 Unlabelled data for testing

This divided data is hosted in Kaggle (https://www.kaggle.com/kasudy/assira4000divided) for easy implementation of the notebook otherwise the full version of data is available at (https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data) which requires to be divided.

3.5. Achieved Results

Since the model only needs to be trained once another separate notebook was created to use the trained model to classify custom images along with the testing image provided. The results are as follows:



Dogs vs Cats Classifer

This notebook does not train the model but insted uses a pre-trained model to classify the images of dogs and cats

Required modules:

```
Keras 2.2.5
TensorFlow 1.15.0
Numpy 1.17.5
```

Also requires a pre trained model

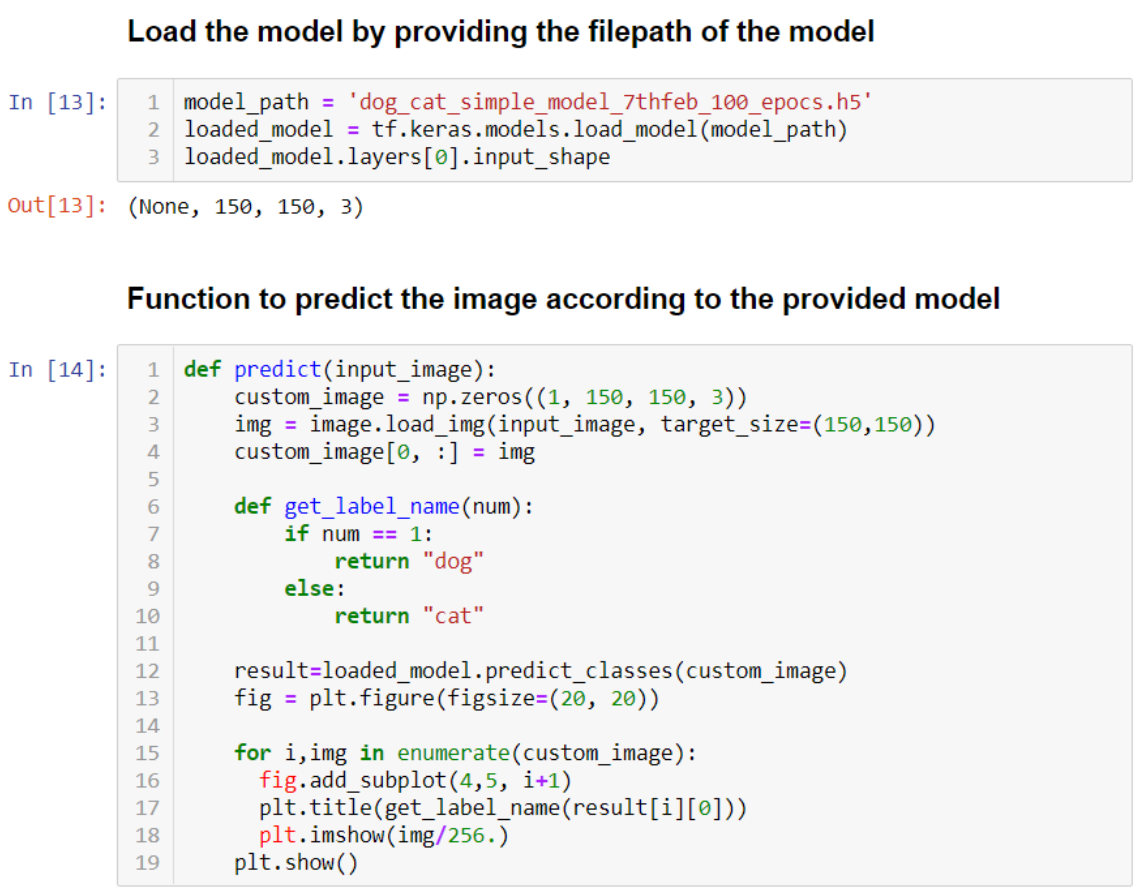
Import necessary modules for processing image and loading models

```
In [12]:
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
#Disabling GPU acceleration as it is not required
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_
from tensorflow.keras.preprocessing import image
print('Keras', tf.keras.__version__)
print('TensorFlow', tf.__version__)
print('Numpy', np.__version__)
# print('Pandas', pd.__version__)
%matplotlib inline
```

```
Keras 2.2.4-tf
TensorFlow 2.1.0
Numpy 1.16.4
```

Figure 29: Importing required libraries for classifying only



Load the model by providing the filepath of the model

In [13]:
```python
model_path = 'dog_cat_simple_model_7thfeb_100_epocs.h5'
loaded_model = tf.keras.models.load_model(model_path)
loaded_model.layers[0].input_shape
```

Out[13]:
```
(None, 150, 150, 3)
```

Function to predict the image according to the provided model

In [14]:
```python
def predict(input_image):
    custom_image = np.zeros((1, 150, 150, 3))
    img = image.load_img(input_image, target_size=(150,150))
    custom_image[0, :] = img

    def get_label_name(num):
        if num == 1:
            return "dog"
        else:
            return "cat"

    result=loaded_model.predict_classes(custom_image)
    fig = plt.figure(figsize=(20, 20))

    for i,img in enumerate(custom_image):
      fig.add_subplot(4,5, i+1)
      plt.title(get_label_name(result[i][0]))
      plt.imshow(img/256.)
    plt.show()
```

Figure 30: Function to predict the image of cat/dog and plot it

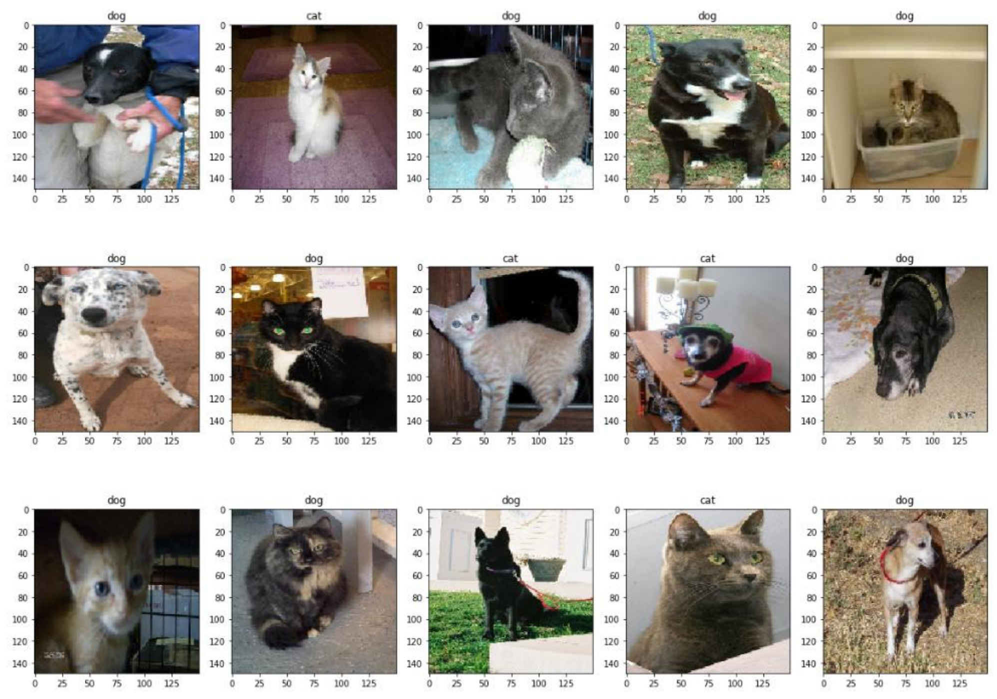
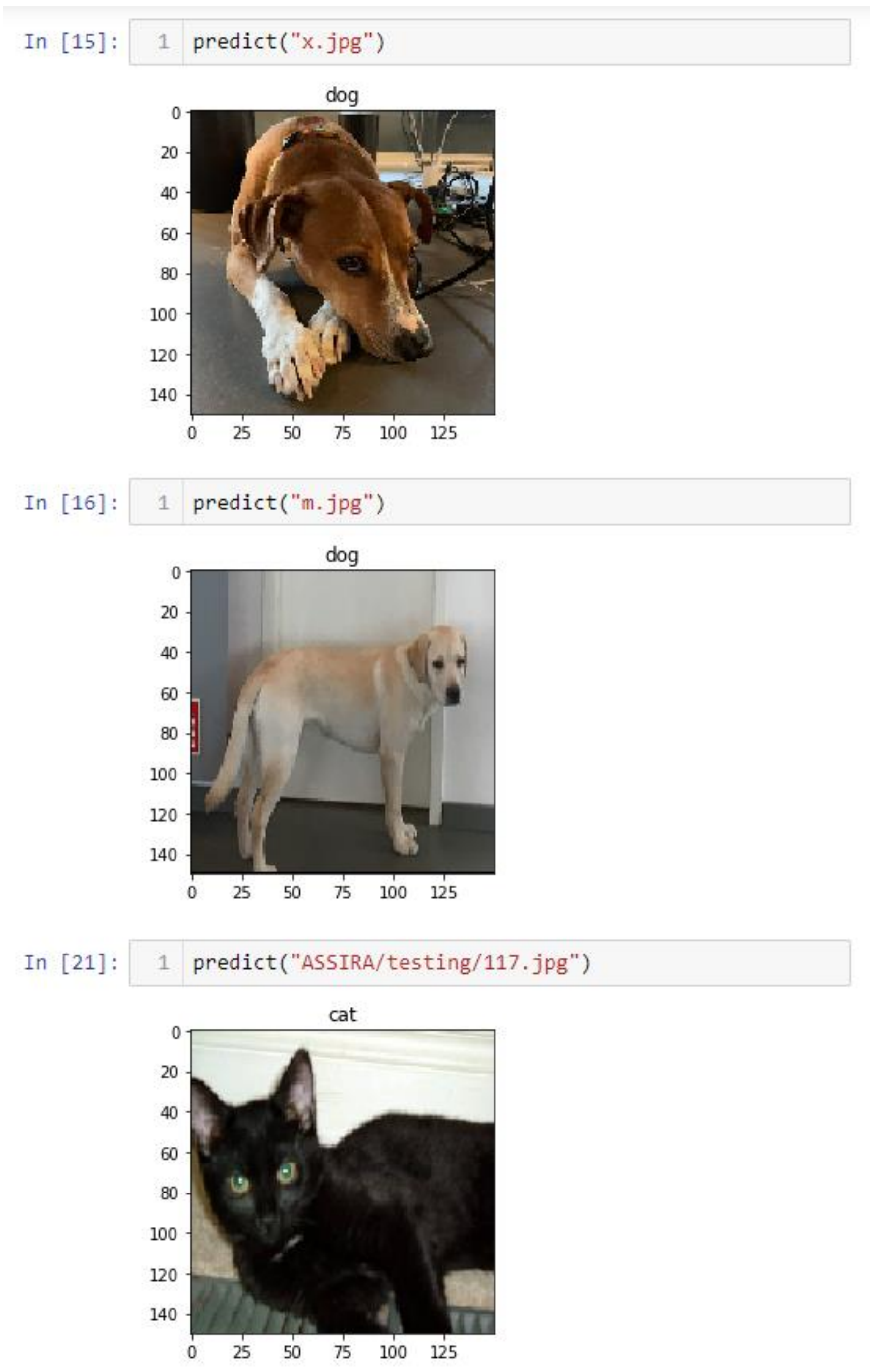


Figure 31: Predicting the categories of images in testing folder



```
In [15]: predict("x.jpg")
```

```
In [16]: predict("m.jpg")
```

```
In [21]: predict("ASSIRA/testing/117.jpg")
```

Figure 32: Classifying Custom images

Therefore, even though most of the predictions are correct the accuracy is still low which could be increased by sing various measures. The full notebook is presented at appendix D. and hosted in google colab.https://colab.research.google.com/drive/1YJp7ch1YGga7GiXqizX4bBvevImG1IDM#scrollTo=fC69LnUIoc0B and https://colab.research.google.com/drive/1bpGEKIUoau246KLvnGxzA8eYTgPpa0wU.

4. Conclusion

This report discusses in detail how Deep learning methods can be used to solve image classification problems and how a system used for differentiating computers and humans are slowly becoming more obsolete. The use of ASIRRA CAPTCHA in 2008 and its failure due to the machine getting features to classify images is discussed. The Kaggle competition is also introduced in this report and the structure of the solution. Dogs vs cats is a classic Deep Neural Network exercise for beginners. This topic is a good example of the Application of Computer Vision to solve real-world problems.

The background portion elaborates more on the basic Deep learning concepts such as neurons, activation function, Artificial Neural Networks, Forward Propagation and convolutional neural networks. Since this project is based on Convolutional Neural Network, the first proposed convolutional neural network architecture LeNet-5, and the modern AlexNet with ReLU activation is also discussed with causing the rapid development in CNN today. After these classic CNN models another simple but deep network, VGG-16 is also discussed which is quite effective in terms of classifying images. The various layers of CNN are discussed along with in-depth analysis of how convolution and max-pooling works. The importance of Convolution operation and max-pooling is also defined. After all the basic concepts are discussed the application of CNN is explored where it was found to be quite essential in the modern days from recognizing digits to detecting disease from digital imagery.

According to the background research and proposed solution a suitable code was created to classify dogs and cats. The model was trained according to the dataset and then used to predict the class of the input images. Even though some of the predictions were wrong it is still usable and has a lot of room for improvements.

The following CNN classifier is not only applicable to classifying cats or dogs but also in vital scenarios such as identifying cancer, skin diseases etc. The image classifier is also used in biometric authentication, face recognition, weather forecasting etc.

Since the model was trained on a simple convolutional network it can also be enhanced by using Pretrained model and adjusting the last layer for better results. This practice of fine tuning pre trained model is known as transfer learning and will drastically improve the accuracy of the model. The modern CNN architectures can also be implemented to have more accuracy on the prediction.

5. References

Bansal, S., 2019. *CNN Architectures : VGG, ResNet, Inception + TL.* [Online]
Available at: https://www.kaggle.com/shivamb/cnn-architectures-vgg-resnet-inception-tl
[Accessed 10 January 2020].

Bianco, S., Cadène, R., Napoletano, P. & Luigi , C., 2018. Benchmark Analysis of Representative Deep Neural Network Architectures. *IEEE Access,* 6(1), pp. 64270-64277.

Brownlee, J., 2019. *How to Classify Photos of Dogs and Cats (with 97% accuracy).* [Online]
Available at: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/
[Accessed 10 january 2020].

Built in, 2019. *Artificial Intelligence, What is Artificial Intelligence? How Does AI Work?.* [Online]
Available at: https://builtin.com/artificial-intelligence
[Accessed 9 January 2020].

Burkov, A., 2019. *The hundred-page machine learning book.* First ed. Quebec: Andriy Burkov.

Campbell, R., 2017. *Demystifying Deep Neural Nets.* [Online]
Available at: https://medium.com/@RosieCampbell/demystifying-deep-neural-nets-efb726eae941
[Accessed 9 January 2020].

Carnegie Mellon University, 2010. *CAPTCHA: Telling Humans and Computers Apart Automatically.* [Online]
Available at: http://www.captcha.net/
[Accessed 10 January 2020].

Chazhoor, A. P., 2019. *ROC curve in machine learning.* [Online]
Available at: https://towardsdatascience.com/roc-curve-in-machine-learning-fea29b14d133
[Accessed 12 January 2020].

Chollet, F., 2016. *Building powerful image classification models using very little data.* [Online]
Available at: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
[Accessed 12 January 2020].

Data School, 2014. *Simple guide to confusion matrix terminology.* [Online]
Available at: https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/
[Accessed 12 January 2020].

Deeplizard, 2017. *Activation Functions in a Neural Network explained.* [Online]
Available at: https://deeplizard.com/learn/video/m0pIlLfpXWE
[Accessed 10 January 2020].

Ding, B., Qian, H. & Zhou, J., 2018. *Activation functions and their characteristics in deep neural networks.* Shenyang, IEEE.

Ferguson, M., ak, R., Yung-Tsun, L. & Kincho, L., 2017. *Automatic localization of casting defects with convolutional neural networks.* s.l., s.n.

Golle, P., 2008. *Machine Learning Attacks Against the Asirra CAPTCHA.* Palo Alto, Palo Alto Research Center.

Goodfellow, I., Courville, A. & Bengio, Y., 2016. *Deep Learning.* First ed. Boston: MIT Press.

Gopalakrishnan, R. & Venkateswarlu, A., 2018. *Types of learning.* [Online]
Available at:
https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788629355/1/ch01lvl1sec12/types-of-learning
[Accessed 10 January 2020].

Haykin, S., 2003. *Neural Networks A comprehensive Foundation.* Second ed. New Delhi: Prentice.

Karim, E., Neehal, N., Foysal, A. F. & Islam, S. M., 2018. *InceptB: A CNN Based Classification Approach for Recognizing Traditional Bengali Games.* Kochi, International Conference on Advances in Computing & Communications.

Krizhevsky, A., Sutskever, I. & Hinton, G. E., 2012. *ImageNet Classification with Deep Convolutional Neural Networks,* Toronto: University of Toronto.

LeCun, Y. & Bengio, Y., 1995. *Convolutional Networks for Images, Speech, and Time-Series.,* Holmdel: AT&T Bell Laboratories.

Lecun, Y., Bottou, L., Yoshua , B. & Haffner, P., 1998. *Gradient-Based Learning Applied to Document Recognition,* New York: Proceedings of the IEEE.

Marco, L. & Farinella, M. G., 2018. *Computer Vision for Assistive Healthcare.* First ed. Amsterdam: Elsevier.

Marshall, S., Nwankpa, C. E., Gachagan, A. & Ijomah, W., 2018. *Activation Functions: Comparison of Trends in Practice and Research for Deep Learning,* Chicago: ArXiv.

Nielsen, M. A., 2015. *Neural Networks and Deep Learning.* First ed. s.l.:Determination Press.

Norvig, P. & Russell, S. J., 2003. *Artificial Intelligence: A Modern Approach.* Second ed. New Delhi: Prentice Hall/Pearson Education.

Prince, S. J., 2012. *Computer Vision: Models, Learning, and Inference.* First ed. London: Cambridge University Press.

Qasim Aziz, Farmer, A. D. & Ruffle, J. K., 2018. Artificial Intelligence-Assisted gastroenterology Promises and Pitfalls. *American Journal of Gastroenterology,* pp. 1-7.

Raschka, S., 2015. *Python Machine Learning.* First ed. Birmingham: Packt Publishing.

Rizwan, M., 2018. *LeNet-5 – A Classic CNN Architecture.* [Online]
Available at: https://engmrk.com/lenet-5-a-classic-cnn-architecture/
[Accessed 12 January 2020].

Sarle, W. S., 1994. *Neural Networks and Statistical Models.* Cary, SAS Institute Inc..

Saul, J., Elson, J., Douceur, J. R. & Howell, J., 2007. *Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization.* San Fransisco, Microsoft Research.

Schuchmann, S., 2019. *History of the first AI Winter.* [Online]
Available at: https://towardsdatascience.com/history-of-the-first-ai-winter-6f8c2186f80b
[Accessed 10 January 2020].

Szeliski, R., 2011. *Computer Vision Algorithms and Applications.* First ed. London: Springer-Verlag London Limited.

Vaezzadeh, M., Saeidi, M., Firouzkhani, A. & Hajnorouzi, A., 2011. Alzheimer Treatment by Applying Ultrasound Waves. *Current Signal Transduction Therapy,* 1(6), pp. 424-427.

Veeranjaneyulu, N. & Bodapati, J. D., 2019. Feature Extraction and Classification Using Deep Convolutional Neural Networks. *Journal of Cyber Security and Mobility,* 8(2), pp. 261-276.

Willems, K., 2019. *Keras Tutorial: Deep Learning in Python.* [Online]
Available at: https://www.datacamp.com/community/tutorials/deep-learning-python
[Accessed 10 January 2020].

Zhang, Q. et al., 2019. Potential for Prediction of Water Saturation Distribution in Reservoirs Utilizing Machine Learning Methods. *Energies,* 12(19), p. 3597.

Zisserman, A. & Simonyan, K., 2014. *Very Deep Convolutional Networks for Large-Scale Image Recognition,* Oxford: arXiv.

Appendix

Appendix A

Kaggle Notebooks:

The notebook created by Harrison Kinsley (Sentdex) features a simple CNN architecture with a well written sets of guides and videos explaining the procedure of creating the cot-dog classifier. This notebook uses low level TensorFlow as the core deep learning and is a good example of the solution.

The notebook created by user Kaggle user Navi demonstrates the performance of Transfer learning VGG-16 in Pytorch. Similarly, user Andrés Larroza does the same using Keras.

The notebook created by Shivan Bansal compared various CNN architectures and plotted the results displaying the effectiveness of the various architectures (Bansal, 2019).

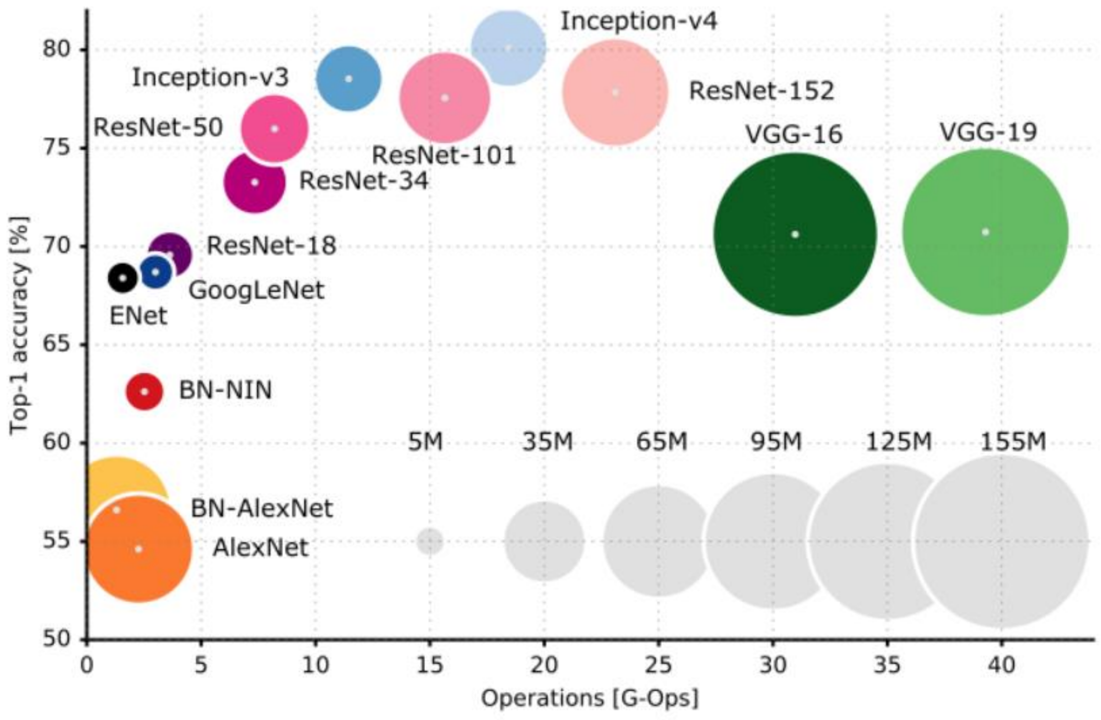


Figure 33: CNN Architectures performance on Dogs vs cats comparison (Bansal, 2019)

Appendix B

Deep learning Frameworks and tools

The following tools are considered to solve the following problem:

Tool	**Description**
TensorFlow	TensorFlow is the core open source library to develop and train ML models
Pytorch	PyTorch is an optimized tensor library for deep learning using GPUs and CPUs like TensorFlow.
Keras	Keras is a high-level neural networks API, capable of running on top of TensorFlow, CNTK, or Theano.
Pandas	Pandas is an open source library providing various data analysis tools for Python.
NumPy	NumPy is the library providing vital tools for matrix and array operations. It makes Linear Algebra tasks exponentially easier.
Nvidia CUDA	NVIDIA CUDA is a parallel computing platform and programming model developed by NVIDIA. It is used for boosting the ML process by utilizing CUDA enable GPUs.
Mat Plot Lib	Matplotlib is a Python 2D plotting library which produces charts and figures essential for data science projects.
Jupyter Notebook	The Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text making machine learning tasks easier.
Google Colab	Google Colab is another web application built on top of Jupiter notebook, provided by google. It provides online kernels (remote virtual computer) which is quite useful when on limited resources and working on teams.
Kaggle Kernel	Kaggle Kernel is another Google Colab like web application which also provides GPU computing and seamless integration with Kaggle services (loading data).
Miniconda	Miniconda is a minimal installer for conda providing most of the essential packages required for Data Science projects. This package is also useful on managing python virtual environments.

Appendix C

Proposed plan to solve the problem: (Previous)

The problem can easily be solved by fine-tuning a pretrained model with the training data of dogs and cats. Then to solve the given problem the following steps will be taken:

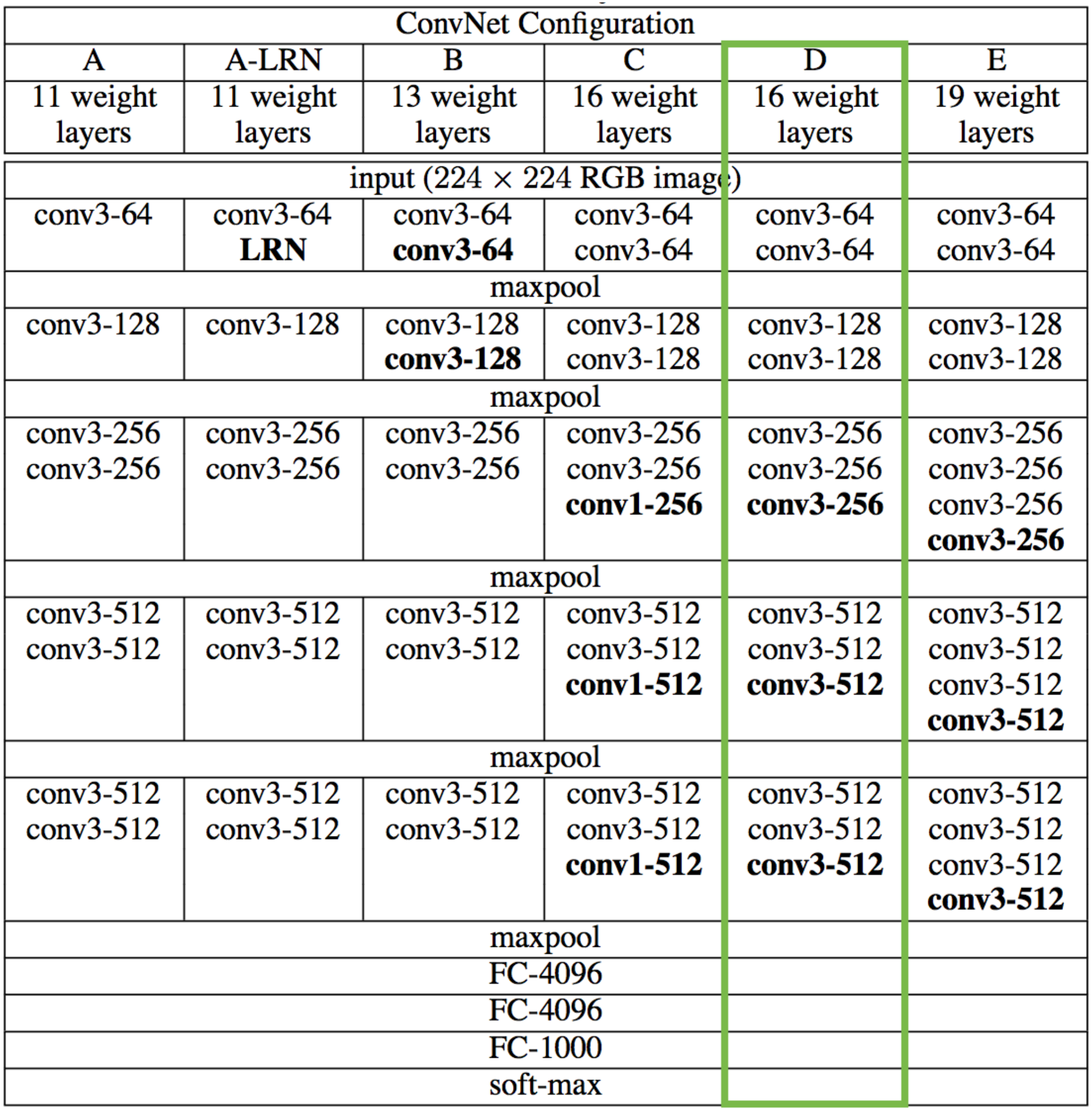
- Data Gathering (From Kaggle)
- Installing required libraries and dependencies
- Analysing datasets
- Preparing Data
- Preparing model using Transfer Learning (Use pre-existing module& fine-tune)
- Dividing the train-datasets into training-dataset and validation-dataset
- Training the model using the training data set
- Observing results on the validation data
- Finally using the trained model to classify testing data
- Recording classified test data details in CSV

Application of CNN to solve this problem

This section elaborates the use of CNN to solve the problem and elaborates all the required details on how it will be modelled. This section also discusses the methods to calculate the accuracy of the model's performance.

Proposed CNN architecture (VGG-16)

The main architecture to be used in this project will be VGG-16 which was discussed in the background section. This will have 13 convolutional layers, 4 pooling layers, and 3 dense layers. This architecture is quite effective for classifying RGB image (Zisserman & Simonyan, 2014).



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 34: VGG-16 architecture details (Zisserman & Simonyan, 2014)

VGG-16 Transfer Learning

Transfer learning is a reusing developed model for another task. This can be quite helpful considering the huge size of VGG-16. Instead of training the whole model we can fine-tune the last 7 layers to make the whole process faster as shown in Figure (Chollet, 2016).

There are various notebooks with VGG-16 used as the core architecture in them. Since VGG16 is a quite bulky architecture, most of the participants had used a transfer learning technique. This allowed them to use pre-existing modules and only train specific parts of the VGG-16 module to get the best results.

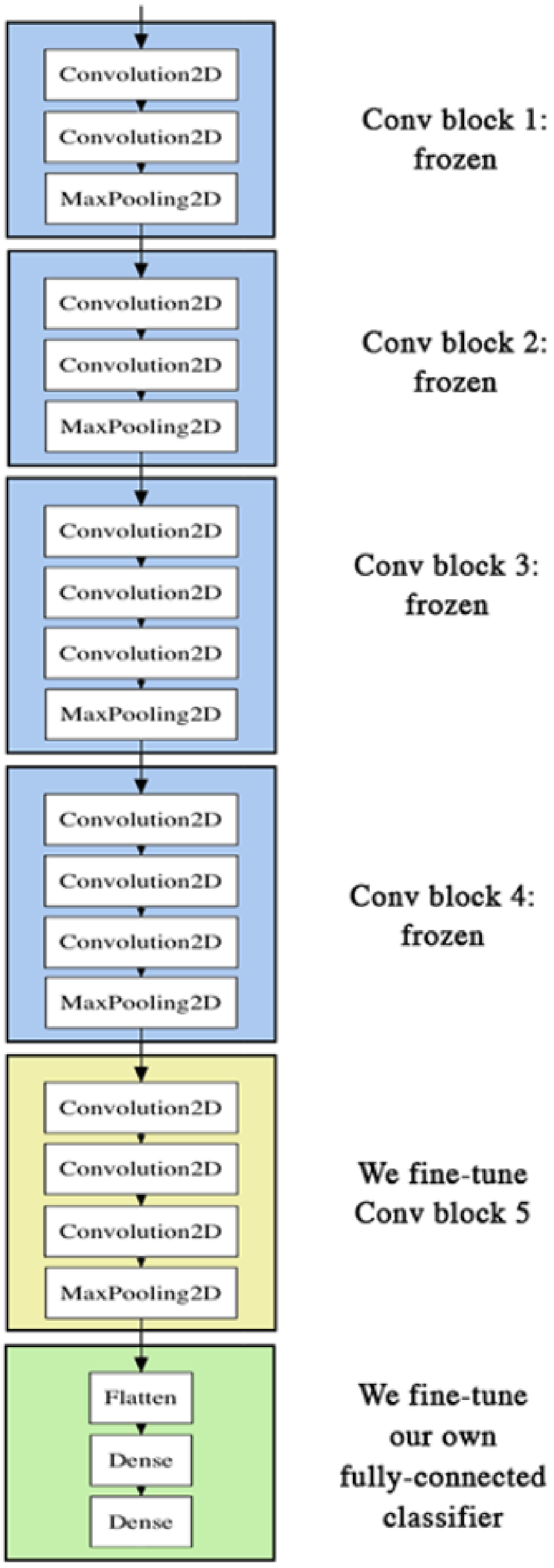
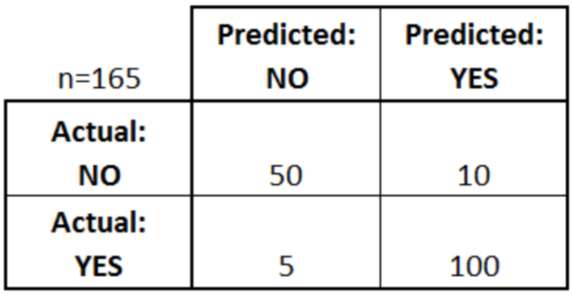


Figure 35: Transfer learning on VGG-16 (Chollet, 2016)

Confusion Matrix to calculate Validation accuracy

A confusion matrix is often used to describe the performance of a classification model on a set of test data for which the true values are known(usually on the validation set) in form of a table. The confusion matrix itself is relatively simple to understand and helps in the calculation of various important metrics such as Accuracy, Misclassification Rate, Precision, Prevalence, True Positive Rate, True Negative Rate, False Positive Rate, and False Negative Rate. A ROC curve can also be plotted using the data from a confusion matrix over several iterations(epochs) during the training process (Data School, 2014).



n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Figure 36: An Example of Confusion Matrix

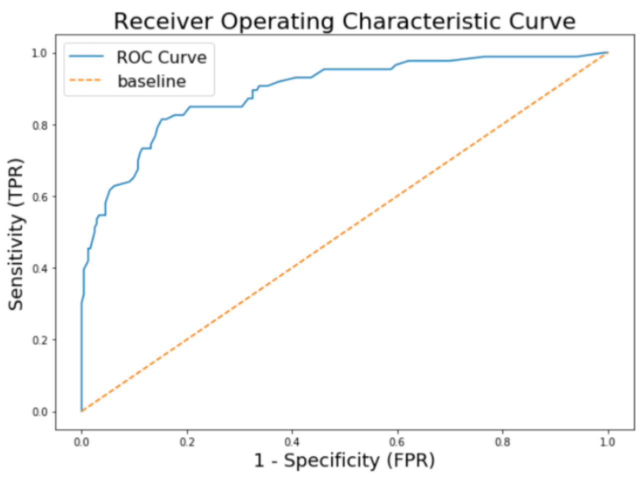


Figure 37: An Example of ROC Curve (Chazhoor, 2019)

Pseudocode

```
IMPORT VGG-16
IMPORT Linear Algebra library
IMPORT CSV handling library
IMPORT 2D Plot
IMPORT training_data = [labelled data]
IMPORT testing_data = [unlabeled data]

DEFINE VGG-16 model:
    model.ADD(CONVOLUTIONAL_LAYER(input_dim=(224,224,3),neurons=64, activation =
    RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=64, activation = RELU))
    model.ADD(POOLING_LAYER(pool_size=(2,2),strides=(2,2)))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=128, activation = RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=128, activation = RELU))
    model.ADD(POOLING_LAYER(pool_size=(2,2),strides=(2,2)))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=256, activation = RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=256, activation = RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=256, activation = RELU))
    model.ADD(POOLING_LAYER(pool_size=(2,2),strides=(2,2)))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=512, activation = RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=512, activation = RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=512, activation = RELU))
    model.ADD(POOLING_LAYER(pool_size=(2,2),strides=(2,2)))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=512, activation = RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=512, activation = RELU))
    model.ADD(CONVOLUTIONAL_LAYER(neurons=512, activation = RELU))
    model.ADD(POOLING_LAYER(pool_size=(2,2),strides=(2,2)))

accuracy = 0
validation_data = [20% of training_data]
while accuracy < 80:
    model TRAIN(VGG-16(training_data))
    FOR each_data IN validaion_data:
        label = each_data.LABEL()
        confusion_matrix.APPEND(label, model.PREDICT(each_data))
    accuracy = ACCURACY(confusion_matrix)


FOR each_data IN testing_data:
    label = each_data.LABEL()
    CSV.Write(label, model.PREDICT(each_data))
```

Flowchart

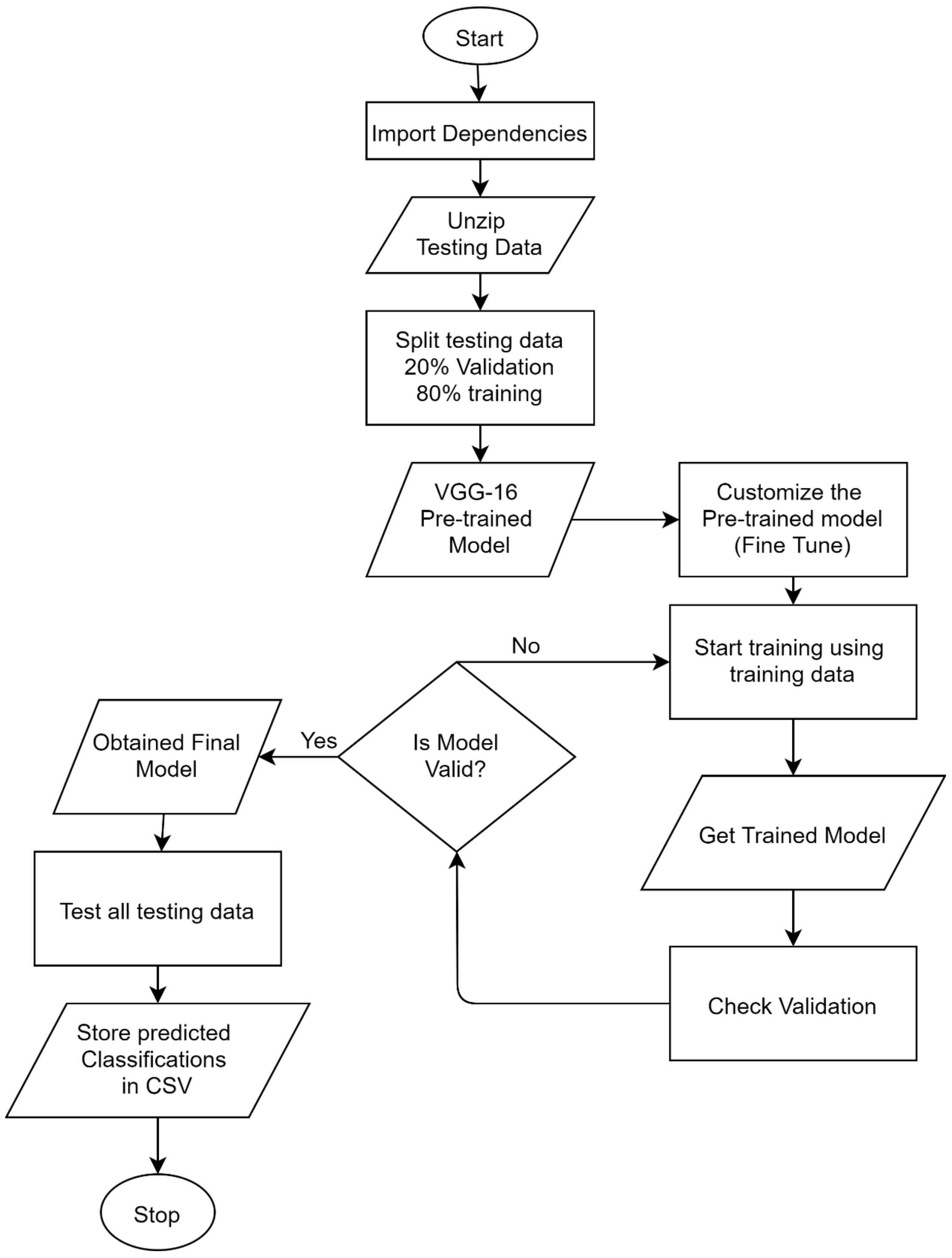


Figure 38: Flow-chart of the solution

Appendix D

Dogs vs Cat classifier

This notebook creates a model using the training data provided by ASSIRA and uses the trained model to classify unlabelled data

The steps followed in this notebook is as follows:

1. Importing Required modules
2. Configure training Configurations
3. Allocate data and check configurations
4. Checking the contents of training dataset
5. Image Augmentation examples
6. Prepare training and validation data
7. Simple model creation and compiling
8. Training the model
9. Plotting the results
10. Using the model to predict test data

1. Importing Required modules:

The essential modules required for running this notebook are:

```
Keras 2.2.4
TensorFlow 2.1.0
Numpy 1.16.4
Pandas 0.25.1
```

In [71]:

```python
import os
import time
import keras
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense
from tensorflow.keras.preprocessing import image

%matplotlib inline
print('Keras', keras.__version__)
print('TensorFlow', tf.__version__)
print('Numpy', np.__version__)
print('Pandas', pd.__version__)
```

```
Keras 2.2.4
TensorFlow 2.1.0
Numpy 1.16.4
Pandas 0.25.1
```

2. Configuration

Configure epocs, batch size and no fo samples to adjust according to the preferrence

In [72]:
```
train_samples      = 2000
validation_samples = 1000

target_size     = (150,150) #size of the image in pixels
batch_size      = 16
epochs          = 60
model_filepath  = 'dog_cat_simple_model_7thfeb_100_epocs.h5'


train_dir       = 'ASSIRA/train'
validation_dir  = 'ASSIRA/validation'
test_dir = "ASSIRA/testing"
```

3. Data Allocation and testing the configurations:

Uses a dataset with following directory structure and files in them

```
/train
    /cat    (1000 images) labelled
    /dog    (1000 images) labelled
/validation
    /cat    (500 images) labelled
    /dog    (500 images) labelled
/testing    (1000 images) unlabelled
```

the dataset used is hosted as ASSIRA4000divided at https://www.kaggle.com/kasudy/assira4000divided This is quite fast to load in google colab

unzip the ASSIRA.zip which will take a while in local machine

In [85]:
```
try:
    print("The model will be saved as : " + model_filepath)
    print("batch-size                  :", batch_size)
    print("epochs                      :", epochs)
    print("train_samples               :", train_samples)
    print("validation_samples          :", validation_samples)
    print("train for                   :", train_samples // batch_size," steps")
    print("validate for                :", validation_samples // batch_size," steps")
    print("\n========================================================\n")
    print("Training directory with images of dogs    : ",len(os.listdir(train_dir+'/dog'))," <<Labelle
    print("Training directory with images of cats    : ",len(os.listdir(train_dir+'/cat'))," <<Labelle
    print("Validation directory with images of dogs  : ",len(os.listdir(validation_dir+'/dog'))," <<La
    print("Validation directory with images of cats  : ",len(os.listdir(validation_dir+'/cat'))," <<La
    print("Test directory with images of cats/dogs   : ",len(os.listdir(test_dir))," <<UNLABELLED>>")
except:
    print("\n========================================================\n")
    print("Error! please make sure you have the data please unzip and re-run this cell")
```

```
The model will be saved as : dog_cat_simple_model_7thfeb_100_epocs.h5
batch-size                 : 16
epochs                     : 60
train_samples              : 2000
validation_samples         : 1000
train for                  : 125  steps
validate for               : 62  steps

=======================================================

Training directory with images of dogs    :  1000  <<Labelled>>
Training directory with images of cats    :  1000  <<Labelled>>
Validation directory with images of dogs  :  500  <<Labelled>>
Validation directory with images of cats  :  500  <<Labelled>>
Test directory with images of cats/dogs   :  1000  <<UNLABELLED>>
```

4. Checking the contents of the training dataset

Checking 6 images of both cats and dogs

```
In [75]: files = [[path+f for f in os.listdir(path)[:6]] for path in [f'ASSIRA/train/{x}/' for x in ['cat', 'do

         fig, axs = plt.subplots(4, 3, figsize=(15,15), subplot_kw={'xticks': [], 'yticks': []})

         for ax, img in zip(axs.flatten(), [item for sublist in files for item in sublist]):
             ax.imshow(load_img(img))
             ax.set_title(img.split('/')[-1])
```

5. Image Augmentation examples

Image augmentation is a useful trick that can increase the data amount by doing various transformation. This is quite helpful when working with less data.

In [81]:
```
# define image transformations
datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')

img_samples = 16
image_dir = 'preview/'
if not os.path.exists(image_dir):
    os.makedirs(image_dir)

img = load_img('ASSIRA/train/dog/dog.101.jpg')
x = img_to_array(img)
x = x.reshape((1,) + x.shape)

from itertools import islice
list(islice(datagen.flow(x,
                         batch_size=1,
                         save_to_dir=image_dir,
                         save_prefix='cat',
                         save_format='jpeg'),
                         img_samples));

rows, cols = 2, img_samples // 2
fig, axs = plt.subplots(rows, cols, figsize=(16,3), subplot_kw={'xticks': [], 'yticks': []})

for ax, img in zip(axs.flatten(), os.listdir(image_dir)[:img_samples]):
    ax.imshow(load_img(image_dir+img))
```

7. Simple model creation and compiling

Using a simple CNN model with 6 Convolution layer and 3 pooling layer. Then Compiling the model

```
In [86]: input_tensor = Input(shape=(150,150,3))

         model = Sequential()
         model.add(Conv2D(128, (3, 3), input_shape=(150,150,3), activation='relu'))
         model.add(Conv2D(128, (3, 3), input_shape=(150,150,3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(32, (3, 3), activation='relu'))
         model.add(Conv2D(64, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(64, (3, 3), activation='relu'))
         model.add(Conv2D(64, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Flatten())
         model.add(Dense(256, activation='relu'))
         model.add(Dropout(0.5))
         model.add(Dense(1, activation='sigmoid'))


         model.compile(loss='binary_crossentropy',
                       optimizer='rmsprop',
                       metrics=['accuracy'])

         print(model.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_18 (Conv2D)           (None, 148, 148, 128)     3584      
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 146, 146, 128)     147584    
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 73, 73, 128)       0         
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 71, 71, 32)        36896     
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 69, 69, 64)        18496     
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 34, 34, 64)        0         
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 32, 32, 64)        36928     
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 30, 30, 64)        36928     
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 15, 15, 64)        0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 14400)             0         
_________________________________________________________________
dense_6 (Dense)              (None, 256)               3686656   
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 257       
=================================================================
Total params: 3,967,329
Trainable params: 3,967,329
Non-trainable params: 0
_________________________________________________________________
None
```

6. Prepare training and validation data

Augment training data but ignore augmentation on validation data as it is not necessary

```
In [79]: # rescale and augment training data
         train_datagen = ImageDataGenerator(
                 rescale=1./255,
                 shear_range=0.2,
                 zoom_range=0.2,
                 horizontal_flip=True)

         # rescale validation data
         validation_datagen = ImageDataGenerator(
             rescale=1./255)

         train_generator = train_datagen.flow_from_directory(
                 train_dir,
                 target_size=target_size,
                 batch_size=batch_size,
                 class_mode='binary')

         validation_generator = validation_datagen.flow_from_directory(
                 validation_dir,
                 target_size=target_size,
                 batch_size=batch_size,
                 class_mode='binary')
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

7. Simple model creation and compiling

Using a simple CNN model with 6 Convolution layer and 3 pooling layer. Then Compiling the model

```
In [86]: input_tensor = Input(shape=(150,150,3))

         model = Sequential()
         model.add(Conv2D(128, (3, 3), input_shape=(150,150,3), activation='relu'))
         model.add(Conv2D(128, (3, 3), input_shape=(150,150,3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(32, (3, 3), activation='relu'))
         model.add(Conv2D(64, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Conv2D(64, (3, 3), activation='relu'))
         model.add(Conv2D(64, (3, 3), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2, 2)))

         model.add(Flatten())
         model.add(Dense(256, activation='relu'))
         model.add(Dropout(0.5))
         model.add(Dense(1, activation='sigmoid'))


         model.compile(loss='binary_crossentropy',
                       optimizer='rmsprop',
                       metrics=['accuracy'])

         print(model.summary())
```

8. Training the model

Training the designed model using the training and validation data. The configurations will be according to the configurations set before

```
In [80]: history = model.fit_generator(
                         train_generator,
                         steps_per_epoch=train_samples // batch_size,
                         epochs=epochs,
                         validation_data=validation_generator,
                         validation_steps=validation_samples // batch_size)

         model.save(model_filepath)
         print("model saved")
```

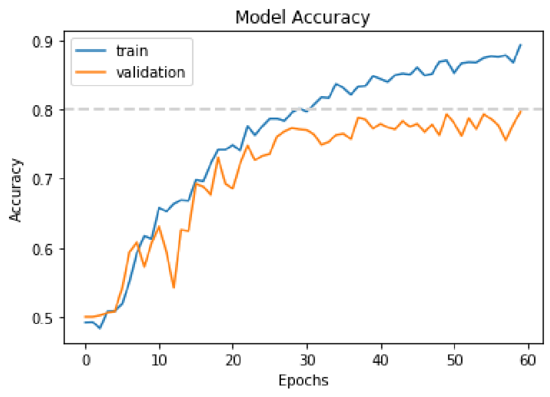
```
Epoch 55/60
125/125 [==============================] - 26s 209ms/step - loss: 0.3121 - accuracy: 0.8745 - val_l
oss: 0.5071 - val_accuracy: 0.7933

Epoch 56/60
125/125 [==============================] - 24s 194ms/step - loss: 0.2973 - accuracy: 0.8770 - val_l
oss: 0.5665 - val_accuracy: 0.7863
Epoch 57/60
125/125 [==============================] - 24s 195ms/step - loss: 0.3114 - accuracy: 0.8760 - val_l
oss: 0.6504 - val_accuracy: 0.7772
Epoch 58/60
125/125 [==============================] - 25s 196ms/step - loss: 0.3079 - accuracy: 0.8785 - val_l
oss: 0.9215 - val_accuracy: 0.7550
Epoch 59/60
125/125 [==============================] - 27s 218ms/step - loss: 0.3197 - accuracy: 0.8680 - val_l
oss: 0.6634 - val_accuracy: 0.7782
Epoch 60/60
125/125 [==============================] - 24s 192ms/step - loss: 0.2697 - accuracy: 0.8930 - val_l
oss: 0.6230 - val_accuracy: 0.7964
model saved
```

9. Plotting the results

plotting the accuracy and validation accuracy of the trained model using matplotlib

```
In [95]: try: #for tensorflow 1
             plt.plot(history.history['acc'])
             plt.plot(history.history['val_acc'])
         except: #for tensorflow 2
             plt.plot(history.history['accuracy'])
             plt.plot(history.history['val_accuracy'])
         plt.title('Model Accuracy')
         plt.ylabel('Accuracy')
         plt.xlabel('Epochs')
         plt.legend(['train', 'validation'])
         plt.axhline(y=0.8, linewidth=2, linestyle='dashed', color='lightgrey')
         plt.show()
```



10. Using the model to predict test data

loading the saved model and then predicting 19 test data using the trained model

```
In [97]: loaded_model = tf.keras.models.load_model(model_filepath)
         loaded_model.layers[0].input_shape

Out[97]: (None, 150, 150, 3)
```

```
In [98]: batch_holder = np.zeros((20, 150, 150, 3))
         img_dir='ASSIRA/testing/'
         for i,img in enumerate(os.listdir(img_dir)):
             if i > 19:
                 break
             img = image.load_img(os.path.join(img_dir,img), target_size=(150,150))
             batch_holder[i, :] = img

         def get_label_name(num):
             if num == 1:
                 return "dog"
             else:
                 return "cat"

         result=loaded_model.predict_classes(batch_holder)
         fig = plt.figure(figsize=(20, 20))

         for i,img in enumerate(batch_holder):
           fig.add_subplot(4,5, i+1)
           plt.title(get_label_name(result[i][0]))
           plt.imshow(img/256.)

         plt.show()
```