A Project Report

ROTATION INVARIANT MULTI-OBJECT DETECTOR

Submitted in partial fulfilment of the requirements for the award of degree

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

by

Sutrave Kanishka (160117733031)

Mallupalli Praneeth Reddy (160117733036)

Department of Computer Science and Engineering,

Chaitanya Bharathi Institute of Technology

(Autonomous),

(Affiliated to Osmania University, Hyderabad)

Hyderabad, TELANGANA (INDIA) –500 075

[2020-2021]

CHAITANYA BHARATHI
INSTITUTE OF TECHNOLOGY (A)
Kokapet (Village), Gandipet, Hyderabad, Telangana-500075. www.cbit.ac.in

ISO Certified 9001:2015

COMMITTED TO RESEARCH, INNOVATION AND EDUCATION 42 years

-

CERTIFICATE

This is to certify that the project titled **Rotation Invariant Multi Object detector** is the bonafide work carried out by **Sutrave Kanishka (160117733031), Mallupalli Praneeth Reddy (160117733036),** students of B.E.(CSE) of Chaitanya Bharathi Institute of Technology(A), Hyderabad, affiliated to Osmania University, Hyderabad, Telangana(India) during the academic year 2020-2021, submitted in partial fulfilment of the requirements for the award of the degree in **Bachelor of Engineering (Computer Science and Engineering)** and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Supervisor
(Dr. E. Padmalatha)

Head, CSE Dept
(Dr. Y. RAMA DEVI)

DECLARATION

We hereby declare that the project entitled "**Rotation Invariant Multi Object detector**" submitted for the B.E (CSE) degree is my original work and the project has not formed the basis for the award of any other degree, diploma, fellowship or any other similar titles.

Kanishka Sutrave (160117733031)
Signature of the Student

Praneeth Reddy (160117733036)
Signature of the Student

Place: Hyderabad
Date: 23-05-2021

ABSTRACT

Object detection has become one of the key computer technologies in the 21st century. From face detection to self-driving cars, humans are using object detection way more and are showing no way back. But the existing technologies have a limitation that they cannot detect properly when the objects are tilted. In real-life situations, we can't expect the images to be perfectly oriented because of positional disturbance of the camera during capture, the image might not be taken by a photographer. Moreover, few scenes obtain significant views while capturing in the landscape while some in portrait. When all those images are passed through the same object detector it might not extract features properly resulting in incorrect predictions.

We have come up with 2 different solutions implementing 2 papers for the above problem.

Our first solution [1] proposes training a pre-trained Resnet50 [10] with perfectly oriented images rotated at random angles by replacing the model's top layer with another dense layer which classifies into 360 classes, each class covering one degree. We solved this as a classification problem, where our model produces a vector of 360 values, each representing the probability at which the image is oriented at that particular angle. Finally, we used the Object detector powered by the Resnet50 model to identify the objects.

In the second solution [2], we have come up with a novel two-step architecture, which efficiently detects multiple objects at any angle in an image efficiently. We utilize eigenvector analysis on the input image based on bright pixel distribution. The vertical and horizontal vectors are used as a reference to detect the deviation of an image from the original orientation. This analysis gives four orientations of the input image which pass through a pre-trained Object Detector using Resnet50 with proposed decision criteria. Our approach, referred to as "Eigen Vectors based Rotation Invariant Multi-Object Deep Detector" (EVRI-MODD), produces rotation invariant detection without any additional training on augmented data and also determines actual image orientation without any prior information. The proposed network achieves high performance on Pascal-VOC 2012 dataset [8]. We evaluate our network performance by rotating input images at random angles between 90° and 270°, and achieve a significant gain in accuracy by 43% over Resnet50.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude and indebtedness to our project guide, **Dr. E. Padmalatha, Asst. Professor** who has supported us throughout our project with patience and knowledge. We are also thankful to Head of the department **Dr.Y.Rama Devi** providing excellent infrastructure and a conducive atmosphere for completing this project successfully. We are also extremely thankful to our Project coordinators **Dr.K.Sagar, Professor** and **Smt G.Vanita**, **Asst. Professor**, and Dept. of CSE, for their valuable suggestions and interest throughout the course of this project. We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed. We sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

Kanishka Sutrave (160117733031)

Praneeth Reddy (160117733036)

LIST OF FIGURES

Table of Contents

1. INTRODUCTION

In Deep Learning, Convolutional Neural Networks (CNNs) are a class of deep feed-forward neural networks used mostly in the analysis of visual imagery.

One of the advantages of CNNs is its translation equivariant property provided by weight sharing. The feature space learned from the mapping function of CNN changes in the same way as the linear transformation in the image. The limitation of the existing system is that the current image classifier such as YOLOv3, RESNET50 cannot accurately identify the image if the image is not properly oriented with that of trained images.

The scope of this project can help in improving the existing object detection system to accurately detect the objects even with much larger orientation and thus makes the classifiers much more accurate.

1.1 Problem Statement including motivation and objective

Detecting an arbitrarily diverted object poses a challenging problem, as features extracted by CNNs are variant to small changes in shift and scale.

They lack performance for images at orientation different from input data.

All the pictures in visual imagery are not in the same orientation. During capture, they differ from each other in terms of alignment due to positional disturbance of the camera. Moreover, few scenes obtain significant view while capturing in landscape, while some in portrait.

Finally, if all images pass through an object detector, then all of them must be in the same orientation, as CNNs are unable to disentangle planar rotation transformations.

When rotated images pass through RESNET50 (Object detector, built on a network called Darknet-53), it makes incorrect detections and even if it is correct, the objectness score is low as compared to that of proper oriented image as shown in figure 1.1. The nearly oriented images have a high score for each object, whereas large deviated images have misclassified objects or correctly classified with lower scores.

Thus, it indicates that RESNET50 is failing in object detection when diverted images pass through it, as it is only trained on images at a specific orientation.

To tackle object detection in rotated scenes, we need to train the CNN on rotations of training images. It is an immense task since it involves rotation and collection of ground truths for every image. The training duration also increases drastically, hence, an efficient method is required, which could make object detectors perform better on rotated images.

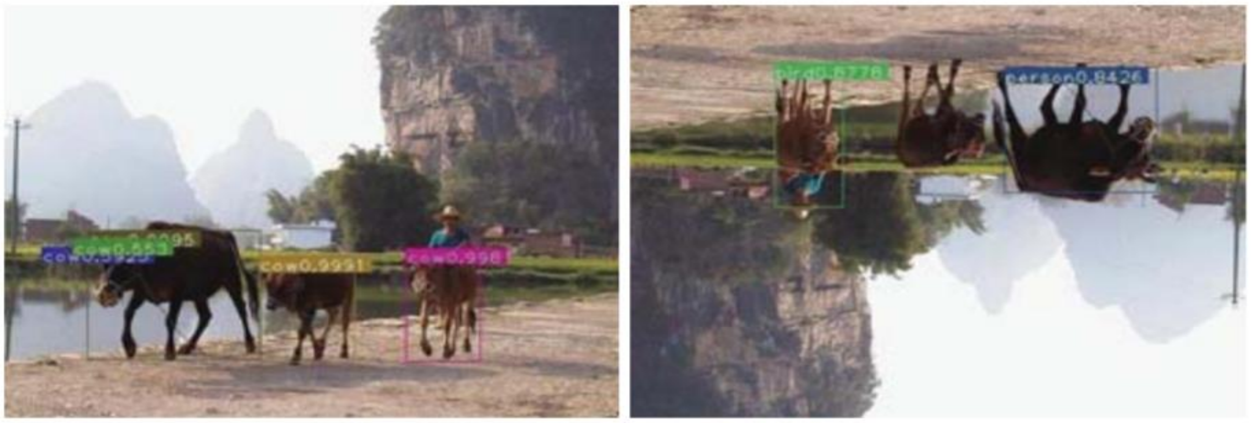


Figure 1.1 Yolo predictions for straight and rotated image

1.2 Methodologies

1.2.1 Deep Learning:

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It's achieving results that were not possible before. In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labelled data and neural network architectures that contain many layers. Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks.

The term "deep" usually refers to the number of hidden layers in the neural network.

Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150.Deep learning models are trained by using large sets of labelled data and neural network architectures that learn features directly from the data without the need for manual feature extraction

1.2.2 Deep Learning Models used:

ResNet50: short for Residual Networks is a classic neural network used as a backbone for many computer vision tasks. This model was the winner of the ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers successfully. Prior to ResNet training, very deep neural networks were difficult due to the problem of vanishing gradients.

1.3 Outline of the results

The input image dataset which is perfectly oriented will be rotated by random angles and is processed through a special function which removes the black borders. Then the images are sent through the orientation correction phase of the implementation which produces the corrected version. Finally, we will predict the classes in both the corrected image as well as input image and compare them.

The accuracy we are calculating is the accuracy at which the corrected class in the images are calculated. Finally, we will calculate the prediction score which says how good our model performed in comparison to the original dataset, i.e., to what extent our correction benefitted the prediction.

1.4 Scope of the project

All the pictures in visual imagery are not in the same orientation. During capture, they differ from each other in terms of alignment due to positional disturbance of the camera. Moreover, few scenes obtain significant view while capturing in ‘landscape’, while some in ‘portrait’. Also, due to various other reasons, the photograph may not be in proper orientation. But they are supposed to get detected correctly, which existing models are not very good at. So, this project solves the issue by correcting the orientation so that the existing object detection algorithms can detect the images accurately.

1.5 Organization of the report

This introduction section is followed by the Literature Survey. The literature survey explains the current existing systems. It also introduces domain specific terminology which forms the background to understand this project. It discusses in depth about some existing solutions' core aspect which also forms the basis for many other solutions. The section also discusses the drawbacks in all the solutions exhaustively. The literature survey section is followed by Design of the Proposed System section. This section discusses the evolution and design of the proposed solution. Next section discusses the implementation of the design discussed in the previous section. The Data Flow Diagrams and Flowcharts are discussed in this section of the project. The algorithm is also discussed in this section. The data set being used, the features of the data set, and their significance are mentioned. The testing process is also included. The next section deals with the result analysis. The system is executed over the test cases and the results are analysed and discussed. The final section deals with conclusions and then references are mentioned.

2. LITERATURE SURVEY

2.1 Introduction to the problem domain Terminology:

Object detection is a computer technology related to computer vision and image processing that detects and defines objects such as humans, buildings and cars from digital images and videos (MATLAB). This technology has the power to classify just one or several objects within a digital image at once. Object detection has been around for years, but is becoming more apparent across a range of industries now more than ever before. Object detection is breaking into a wide range of industries, with use cases ranging from personal security to productivity in the workplace. Facial detection is one form of it, which can be utilized as a security measure to let only certain people into a highly classified area of a government building, for example. It can be used to count the number of people present within a business meeting to automatically adjust other technical tools that will help streamline the time dedicated to that particular meeting. It can also be used within a visual search engine to help consumers find a specific item they're on the hunt for – Pinterest is one example of this, as the entire social and shopping platform is built around this technology.

2.2 Existing Solutions:

For images containing single objects, works such as Spatial Transformer Networks (STN), TI-pooling, Oriented Response Networks (ORN) with Active Rotating Filters (ARF), RIFD-CNN are proposed.

These techniques implement data augmentation in the training phase or architectural modifications, which increases training complexity.

Data augmentation is practically impossible for very large datasets with millions of training images, such as that provided in ILSVRC and also in cases where data privacy is mandatory.

Fast rotation invariant object detection with gradient based detection models introduces the method of training specific models multiple times, each with a different orientation. During testing, a rotation map containing information about orientations at a particular location from dominant orientation is obtained.

They use SURF algorithm on Haar features to calculate the dominant orientation. This method claims high evaluation speed, but training complexity is similar to data augmentation as different orientations are used to train a specific model.

Rotational Invariance using multiple instances of Convolutional Neural Network (RIMCNN) proposes the idea of using the same trained CNN multiple times. It rotates the input image into various orientations and feeds each image to each instance of the architecture. This adds rotational invariance, but the number of instances increases with finer rotations, thus rising in time complexity.

2.3 RELATED WORKS

2.3.1 Fast Rotation Invariant Object Detection with Gradient based Detection Models [3]

- In this paper we follow a two-step approach to improve the processing speed of object detection under rotation.
- As a first step we trained multiple models to cover all needed orientations. This allows to eliminate the need to rotate the image, and allows to reuse the feature pyramid for all models.
- Next to that we use sample points to extract orientation information over the image. This allows us to reduce the number of models that have to be evaluated at each location.
- We used the ACF-detection framework for its fast training and evaluation speed and acquired a speed-up by using a rotation map.
- We compare to the baseline of evaluating multiple rotations of the image, and acquire a speed-up of 8.2 times, while maintaining high accuracy

2.3.2 Oriented Response Networks [4]

- Deep Convolutional Neural Networks (DCNNs) are capable of learning unprecedentedly effective image representations. However, their ability in handling significant local and global image rotations remains limited.
- In this paper, we use Active Rotating Filters (ARFs) that actively rotate during convolution and produce feature maps with location and orientation explicitly encoded.
- An ARF acts as a virtual filter bank containing the filter itself and its multiple unmaterialized rotated versions. During back-propagation, an ARF is collectively updated using errors from all

its rotated versions.

- DCNNs using ARFs, referred to as Oriented Response Networks (ORNs), can produce within-class rotation-invariant deep features while maintaining inter-class discrimination for classification tasks.
- The oriented response produced by ORNs can also be used for image and object orientation estimation tasks.
- Over multiple state-of-the-art DCNN architectures, such as VGG, ResNet, and STN, we consistently observe that replacing regular filters with the proposed ARFs leads to significant reduction in the number of network parameters and improvement in classification performance.

2.4 TOOLS AND TECHNOLOGIES USED:

2.4.1 TensorFlow

TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy.

2.4.2 Keras

Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit R, Theano, or Plaid ML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

2.4.3 Google Colab

Google Colab is a free cloud service provided by Google Inc and provides Machine Learning and Deep Learning developers with free CPU, GPU and TPU runtimes. It is directly synced with Google Drive which provides a lot of convenience to the developers to store their models and datasets and use them directly without much hassle.

2.4.4 Software Requirements

- Pytorch
- VSCode (IDE)

2.3.5 Hardware Requirements

- Nvidia GTX 1080 Ti GPUs.

3. DESIGN OF THE PROPOSED SYSTEM

3.1 Block Diagram

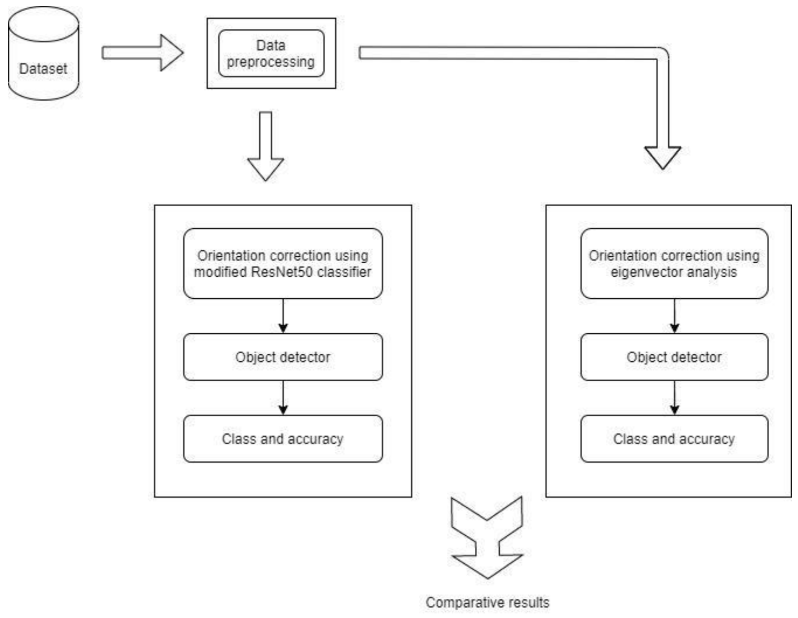


Figure 3.1 Block diagram

Figure 3.1 is a block diagram that gives a brief flow of algorithms in the project. The steps include pre-processing, orientation correction, objects identification and comparative results.

3.2 Module Description

3.2.1 NUMPY

NumPy stands for ‘Numerical Python’ or ‘Numeric Python’. It is an Open Source module of Python which provides fast mathematical computation on arrays and matrices. Since arrays and matrices are an essential part of the Machine Learning ecosystem, NumPy along with Machine Learning modules like Scikit-learn, Pandas, Matplotlib, TensorFlow, etc. complete the Python Machine Learning Ecosystem. NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation.

3.2.2 MATPLOTLIB

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. It is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

3.2.3 OpenCV

OpenCV is the huge open-source library for computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today’s systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When integrated with various libraries, such as Numpy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features.

3.2.4 Image AI

Image AI is a Python library built to empower developers to build applications and systems with self-contained deep learning and Computer Vision capabilities using a few lines of straight forward code. Image AI contains a Python implementation of almost all of the state-of-the-art deep learning algorithms like Retina Net, YOLOv3, and TinyYOLOv3.

Image AI makes use of several APIs that work offline - it has object detection, video detection, and object tracking APIs that can be called without internet access. Image AI makes use of a pre-trained model and can easily be customized.

The Object Detection class of the Image AI library contains functions to perform object detection on any image or set of images, using pre-trained models. With Image AI, you can detect and recognize 80 different kinds of common, everyday objects.

3.2.5 KERAS

Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.

3.3 Theoretical Foundation/ Algorithms

3.3.1 Deep learning

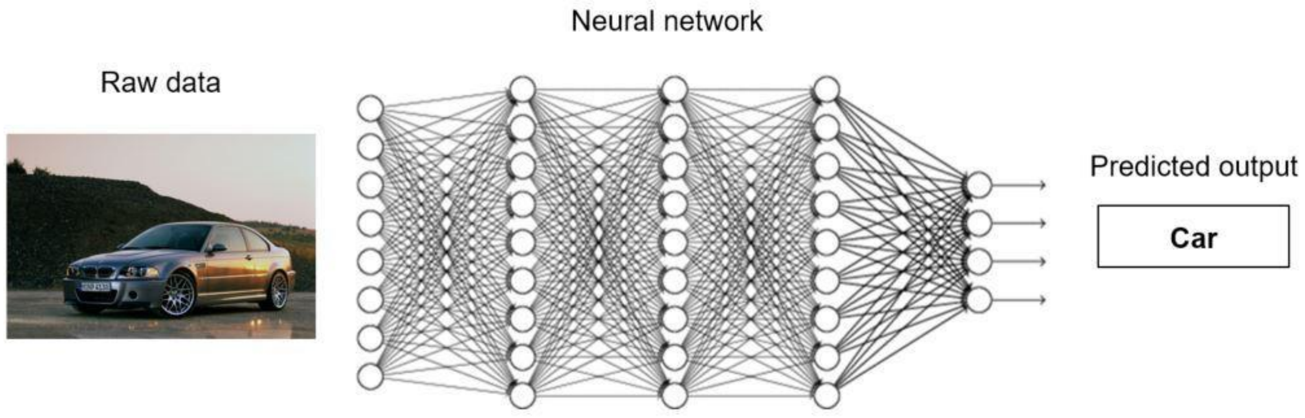


Figure 3.3.1 Overview of deep learning

Figure 3.3.1 shows basic neural network architecture with one input layer, three hidden layers and one output layer**.** Deep learning is an increasingly popular subset of machine learning. Deep learning models are built using neural networks. A neural network takes in inputs, which are then processed in hidden layers using weights that are adjusted during training. Then the model spits out a prediction. The weights are adjusted to find patterns in order to make better predictions. The user does not need to specify what patterns to look for the neural network learns on its own. We use a sequential model of deep learning. Sequential is the easiest way to build a 30 model in Keras. It allows you to build a model layer by layer. Each layer has weights that correspond to the layer that follows it. We use the 'add()' function to add layers to our model. We will add two layers and an output layer.

3.3.2 Convolutional Neural Network

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

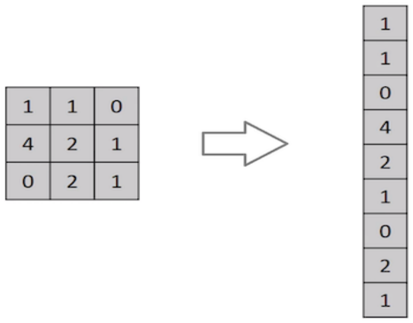


Figure 3.3.2.1 Flattening of a 3x3 image matrix into a 9x1 vector

A ConvNet is able to **successfully capture the Spatial and Temporal dependencies** in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.
Input Image

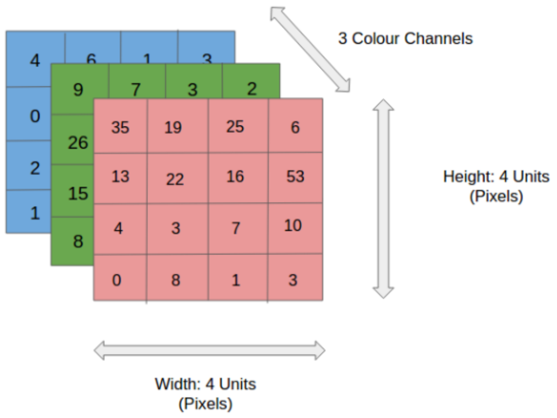


Figure 3.3.2.2 4x4x3 RGB Image

In the *figure 3.3.2.2*, we have an RGB image which has been separated by its three colour planes — Red, Green, and Blue. There are a number of such colour spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc.
The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.

Convolution Layer — The Kernel

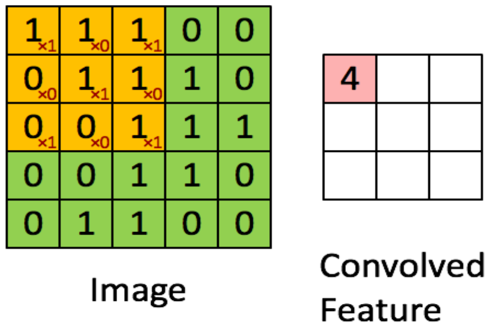


Figure 3.3.2.3 Convoluting a 5x5x1 image with a 3x3x1 kernel to get a 3x3x1 and convolved feature

Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, e.g. RGB)

In the Figure 3.3.2.3, the green section resembles our 5x5x1 input image, I**.** The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the colour yellow. We have selected K as a 3x3x1 matrix.

Kernel/Filter, K =

[[1 0 1]

[0 1 0]

[1 0 1]]

The Kernel shifts 9 times because of Stride Length = 1 (Non-Strided), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering.

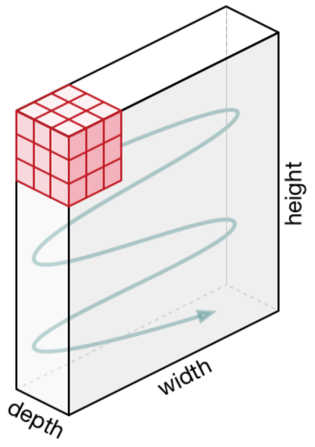


Figure 3.3.2.4 Movement of the Kernel

The filter moves to the right with a certain Stride Value till it parses the complete width as shown in figure 3.3.2.5. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

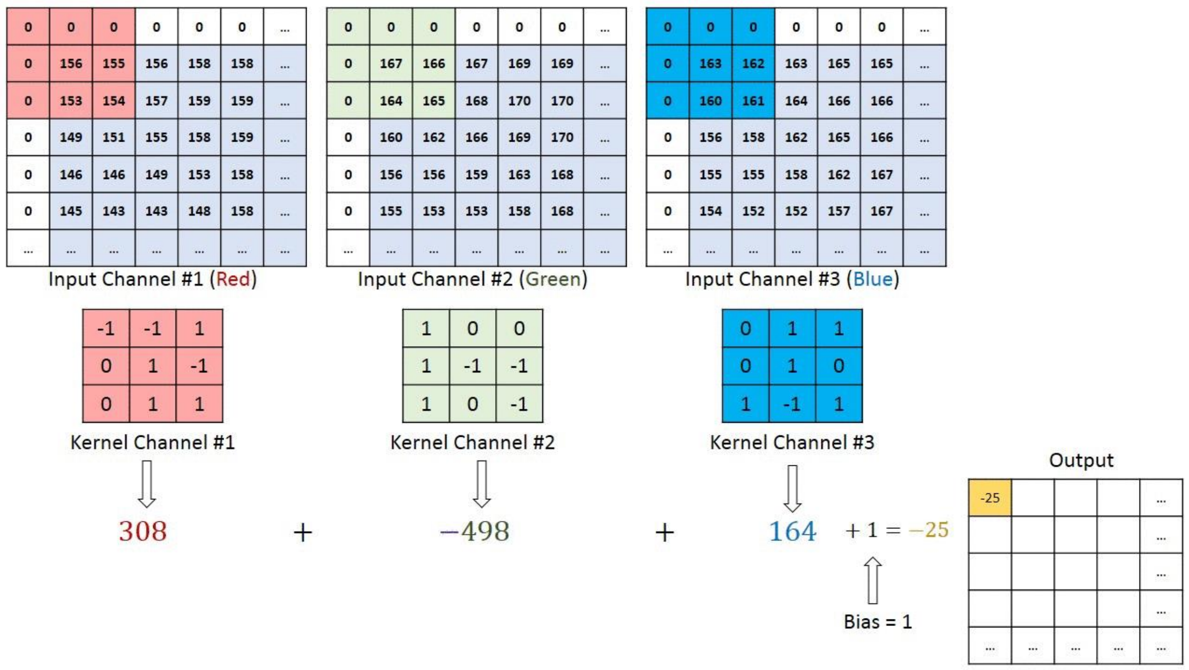


Figure 3.3.2.5 Convolution operation on a MxNx3 image matrix with a 3x3x3 Kernel

The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, colour, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network, which has the wholesome understanding of images in the dataset, similar to how we would our needs.

Pooling Layer:

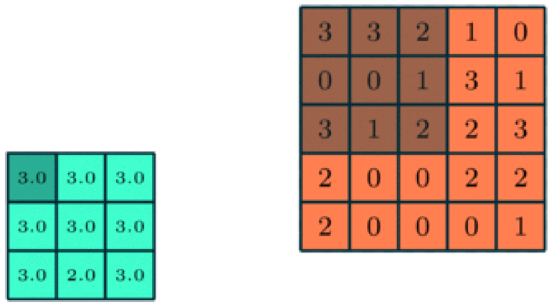


Figure 3.3.2.6 3x3 pooling over 5x5 convolved feature

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature as shown in the *figure 3.3.2.6*. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling returns the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel.

Max Pooling also performs as a Noise Suppressant. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction. On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism. Hence, we can say that Max Pooling performs a lot better than Average Pooling.

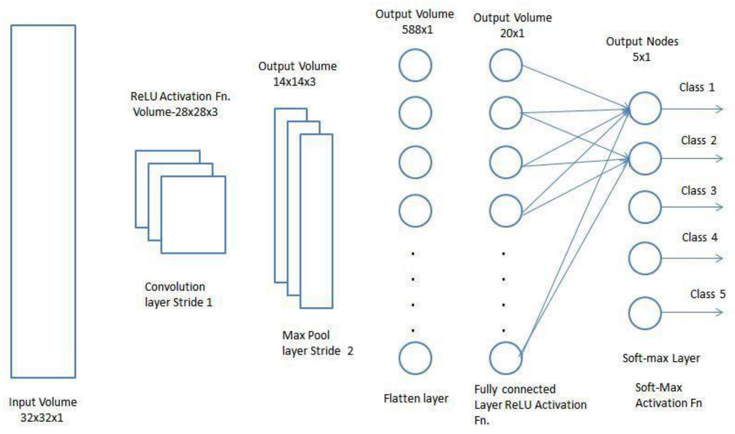


Figure 3.3.2.7 Classification — Fully Connected Layer (FC Layer)

Adding a Fully-Connected layer as shown in *fig. 3.3.2.7* is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.

Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the SoftMax Classification technique.

There are various architectures of CNNs available which have been key in building algorithms which power and shall power AI as a whole in the foreseeable future. Some of them have been listed below:

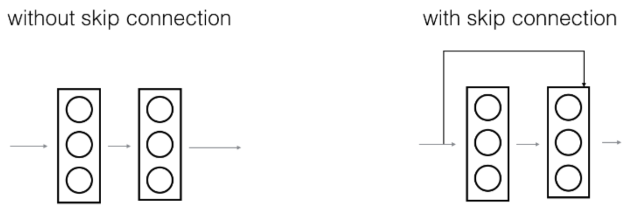
1. LeNet
2. AlexNet
3. VGGNet

3.3.3 ResNet50

ResNet, short for Residual Networks, is a classic neural network used as a backbone for many computer vision tasks. This model was the winner of the ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150+layers successfully. Prior to ResNet training, very deep neural networks were difficult due to the problem of vanishing gradients.

Deep networks are hard to train because of the notorious vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient extremely small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly.

ResNet first introduced the concept of skip connection. The diagram below illustrates skip connection. The figure on the left is stacking convolution layers together one after the other. On the right we still stack convolution layers as before but we now also add the original input to the output of the convolution block. This is called skip connection



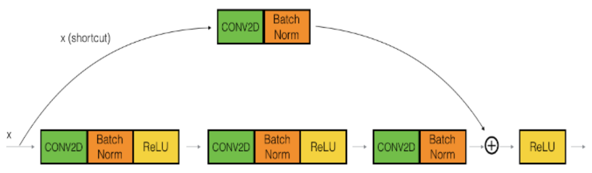


Figure 3.3.3.1 Working of skip connections

One important thing to note here is that the skip connection is applied before the RELU activation as shown in the diagram 3.3.3.1 above. Research has found that this has the best results.

Two reasons why Skip connections work here:

1. They mitigate the problem of vanishing gradient by allowing this alternate shortcut path for gradient to flow through
2. They allow the model to learn an identity function which ensures that the higher layer will perform at least as good as the lower layer, and not worse

In fact, since ResNet skip connections are used in a lot more model architectures like the Fully Convolutional Network (FCN) and U-Net. They are used to flow information from earlier layers in the model to later layers. In these architectures they are used to pass information from the down sampling layers to the up-sampling layers.

The ResNet-50 model consists of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. The ResNet-50 has over 23 million trainable parameters.

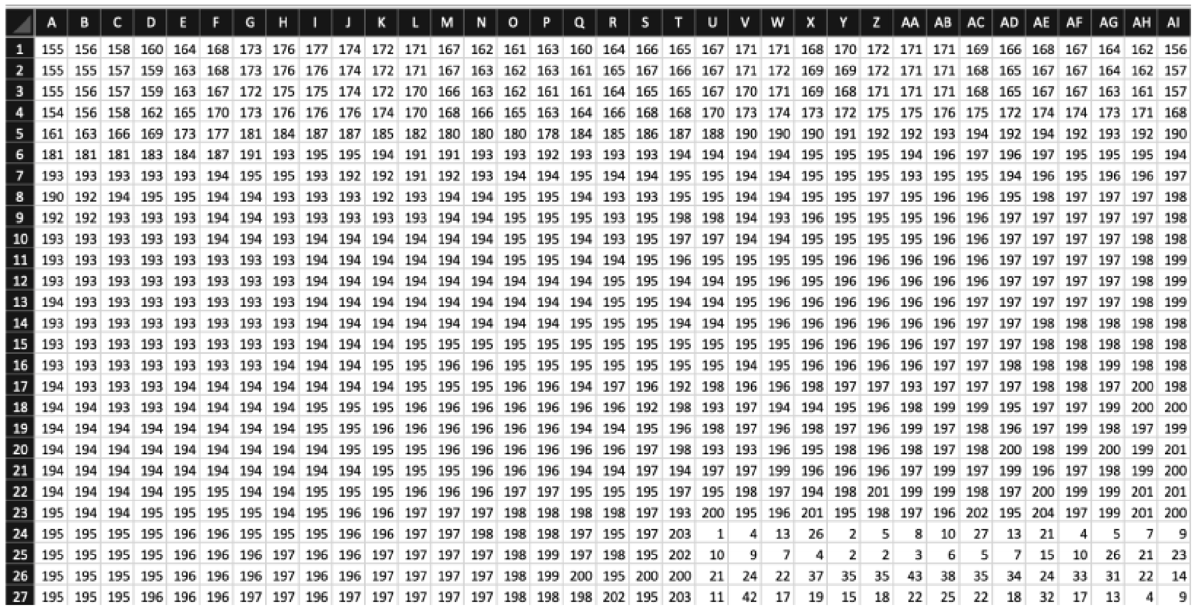
3.3.4 Principal Component Analysis

PCA is a mathematical tool for feature extraction of a dataset. As the number of independent features increases, the dimensionality of the dataset also increases, which makes it harder to draw insights from the data.

PCA accepts all the features of the dataset, but only considers a subset of features that it deems important to establish correlations between the selected features. The result of PCA is reduced dimensionality of the dataset. The process also presents eigenvectors that describe the directions along which the data vary most.

The eigenvector has a unique value, or eigenvalue, associated with it. Using the eigenvalue and eigenvector, the angle of the new XY-axes can be determined.

Once an image is converted to grayscale, its matrix representation is two dimensional, as shown in *Figure 3.3.3.2*. The columns of this matrix can be viewed as features. When PCA is applied to this 100 x 100 matrix, the 100 columns are condensed down to two columns. This would seem to lose information, but this condensed representation makes it possible to find two directions along which the pixel values vary. Two directions are found, because the original representation of 100 columns is slimmed down to two columns. These two directions are perpendicular, and form the new axes.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI
1	155	156	158	160	164	168	173	176	177	174	172	171	167	162	161	163	160	164	166	165	167	171	171	168	170	172	171	171	169	166	168	167	164	162	156
2	155	155	157	159	163	168	173	176	176	174	172	171	167	163	162	163	161	165	167	166	167	171	172	169	169	172	171	171	168	165	167	167	164	162	157
3	155	156	157	159	163	167	172	175	175	174	172	170	166	163	162	161	161	164	165	165	167	170	171	169	168	171	171	171	168	165	167	167	163	161	157
4	154	156	158	162	165	170	173	176	176	176	174	170	168	166	165	163	164	166	168	168	170	173	174	173	172	175	175	176	175	172	174	174	173	171	168
5	161	163	166	169	173	177	181	184	187	187	185	182	180	180	180	178	184	185	186	187	188	190	190	190	191	192	192	193	194	192	194	192	193	192	190
6	181	181	181	183	184	187	191	193	195	195	194	191	191	193	193	192	193	193	193	194	194	194	194	195	195	195	194	196	197	196	197	195	195	195	194
7	193	193	193	193	193	194	195	195	193	192	192	191	192	193	194	194	195	194	194	195	195	194	194	195	195	195	193	195	195	194	196	195	196	196	197
8	190	192	194	195	195	194	194	193	193	193	192	193	194	194	195	195	194	193	193	195	195	194	194	195	195	197	195	196	196	195	198	197	197	197	198
9	192	192	193	193	193	194	194	193	193	193	193	193	194	194	195	195	195	193	195	198	198	194	193	196	195	195	195	196	196	197	197	197	197	197	198
10	193	193	193	193	193	194	194	193	194	194	194	194	194	194	195	195	194	193	195	197	197	194	194	195	195	195	195	196	196	197	197	197	197	198	198
11	193	193	193	193	193	193	193	193	194	194	194	194	194	194	195	195	194	194	195	196	195	195	195	195	196	196	196	196	196	197	197	197	197	198	199
12	193	193	193	193	193	193	193	193	194	194	194	194	194	194	194	194	194	195	195	194	194	195	196	195	196	196	196	196	196	197	197	197	197	198	199
13	194	193	193	193	193	193	193	193	194	194	194	194	194	194	194	194	194	195	195	194	194	195	196	196	196	196	196	196	197	197	197	197	197	198	199
14	193	193	193	193	193	193	193	193	194	194	194	194	194	194	194	194	195	195	195	194	194	195	196	196	196	196	196	196	197	197	198	198	198	198	198
15	193	193	193	193	193	193	193	193	194	194	194	195	195	195	195	195	195	195	195	195	195	195	195	196	196	196	196	197	197	197	198	198	198	198	198
16	193	193	193	193	193	193	193	194	194	194	195	195	196	196	195	195	195	195	195	195	195	194	195	196	196	196	196	197	197	198	198	198	199	198	198
17	194	193	193	193	194	194	194	194	194	194	194	195	195	195	196	196	194	197	196	192	198	196	196	198	197	197	193	197	197	197	198	198	197	200	198
18	194	194	193	193	194	194	194	194	195	195	195	196	196	196	196	196	196	196	192	198	193	197	194	194	195	196	198	199	199	195	197	197	199	200	200
19	194	194	194	194	194	194	194	194	195	195	196	196	196	196	196	196	194	194	195	196	198	197	196	198	197	196	199	197	198	196	197	199	198	197	199
20	194	194	194	194	194	194	194	194	194	195	195	195	196	196	196	196	196	196	197	198	193	193	196	195	198	196	198	197	198	200	198	199	200	199	201
21	194	194	194	194	194	194	194	194	194	194	195	195	195	196	196	196	194	194	197	194	197	197	199	196	196	196	197	199	197	199	196	197	198	199	200
22	194	194	194	194	195	195	194	194	195	195	195	196	196	196	197	197	195	195	195	197	195	198	197	194	198	201	199	199	198	197	200	199	199	201	201
23	195	194	194	195	195	195	195	194	195	196	196	197	197	197	198	198	198	198	197	193	200	195	196	201	195	198	197	196	202	195	204	197	199	201	200
24	195	195	195	195	196	196	195	195	195	196	196	197	197	198	198	198	197	195	197	203	1	4	13	26	2	5	8	10	27	13	21	4	5	7	9
25	195	195	195	195	196	196	196	197	196	196	197	197	197	197	198	199	197	198	195	202	10	9	7	4	2	2	3	6	5	7	15	10	26	21	23
26	195	195	195	195	196	196	196	197	196	196	197	197	197	197	198	199	200	195	200	200	21	24	22	37	35	35	43	38	35	34	24	33	31	22	14
27	195	195	195	196	196	196	197	197	196	197	197	197	197	197	198	198	198	202	195	203	11	42	17	19	15	18	22	25	22	18	32	17	13	4	9

Figure 3.3.3.2 Matrix representation of image

Applications:

- **PCA** is predominantly used as a dimensionality reduction technique in domains like facial recognition, computer vision and image compression. It is also used for finding patterns in data of high dimension in the field of finance, data mining, bioinformatics, psychology
- summarize the information content in large data tables by means of a smaller set of "summary indices" that can be more easily visualized and analyzed
- Eigenfaces for detection, for clustering gene expression data**.**

Advantages:

1. Removes Correlated Features:

In a real-world scenario, this is very common that you get thousands of features in your dataset. You cannot run your algorithm on all the features as it will reduce the performance of your algorithm and it will not be easy to visualize that many features in any kind of graph. So, you MUST reduce the number of features in your dataset.

You need to find out the correlation among the features (correlated variables). Finding correlation manually in thousands of features is nearly impossible, frustrating and time-consuming. PCA does this for you efficiently.

After implementing the PCA on your dataset, all the Principal Components are independent of one another. There is no correlation among them.

2. Improves Algorithm Performance:

With so many features, the performance of your algorithm will drastically degrade. PCA is a very common way to speed up your Machine Learning algorithm by getting rid of correlated variables which don’t contribute to any decision making. The training time of the algorithms reduces significantly with few features.

So, if the input dimensions are too high, then using PCA to speed up the algorithm is a reasonable choice.

3. Reduces Overfitting:

Overfitting mainly occurs when there are too many variables in the dataset. So, PCA helps in overcoming the overfitting issue by reducing the number of features.

4. Improves Visualization:

It is very hard to visualize and understand the data in high dimensions. PCA transforms high dimensional data to low dimensional data (2 dimension) so that it can be visualized easily.

We can use 2D Scree Plot to see which Principal Components result in high variance and have more impact as compared to other Principal Components.

Even the simplest IRIS dataset is 4 dimensional which is hard to visualize. We can use PCA to reduce it to 2-dimensions for better visualization.

Disadvantages:

1. Independent variables become less interpretable:

After implementing PCA on the dataset, your original features will turn into Principal Components. Principal Components are the linear combination of your original features. Principal Components are not as readable and interpretable as original features.

2. Data standardization is must before PCA:

You must standardize your data before implementing PCA, otherwise PCA will not be able to find the optimal Principal Components.

For instance, if a feature set has data expressed in units of Kilograms, Light years, or Millions, the variance scale is huge in the training set. If PCA is applied on such a feature set, the resultant loadings for features with high variance will also be large. Hence, principal components will be biased towards features with high variance, leading to false results.

Also, for standardization, all the categorical features are required to be converted into numerical features before PCA can be applied.

PCA is affected by scale, so you need to scale the features in your data before applying PCA. Use Standard Scaler from Scikit Learn to standardize the dataset features onto unit scale (mean = 0 and standard deviation = 1) which is a requirement for the optimal performance of many Machine Learning algorithms.

3. Information Loss:

Although Principal Components try to cover maximum variance among the features in a dataset, if we don’t select the number of Principal Components with care, it may miss some information as compared to the original list of features.

4. Implementation and proposed System

4.1 Flowchart

Flowcharts are a better way of communicating the logic of a system and act as a guide for blueprint during program designed, helps in debugging process, programs can be easily analysed.

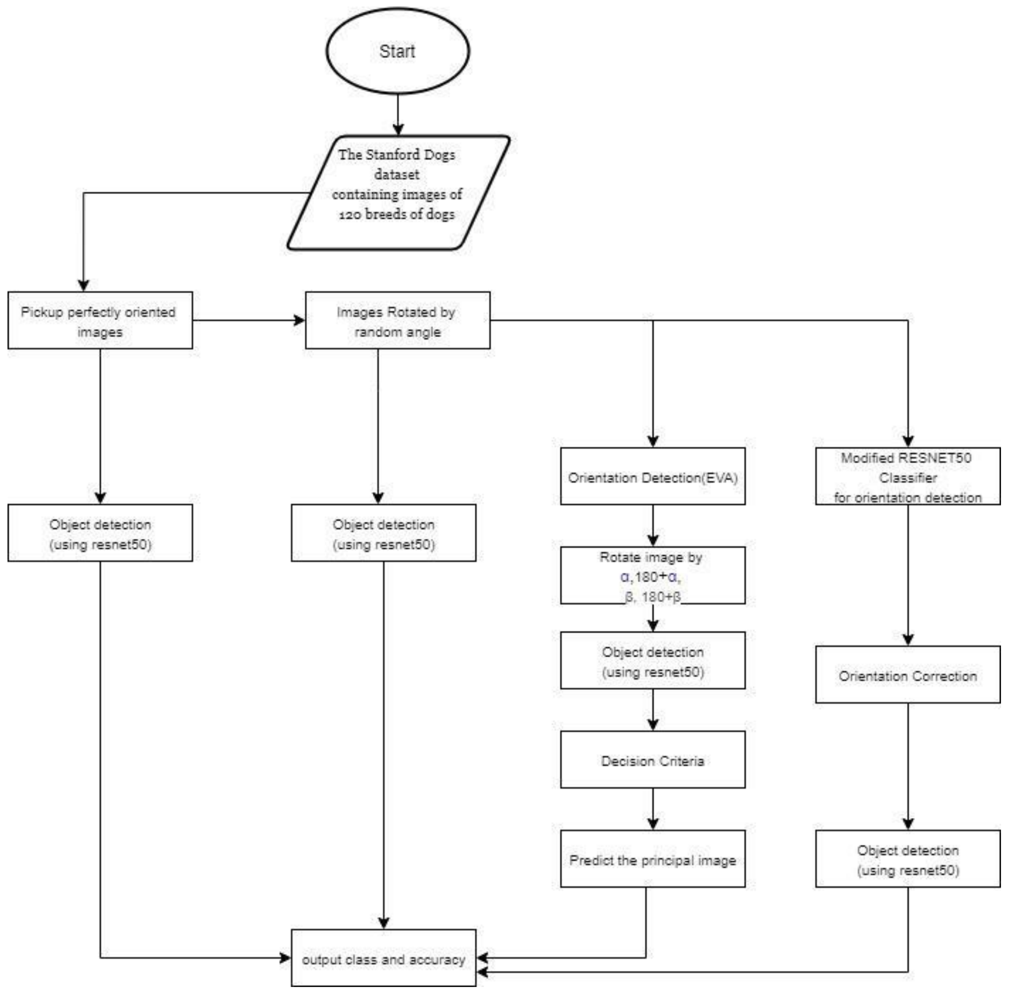

Start
The Stanford Dogs dataset containing images of 120 breeds of dogs
Pickup perfectly oriented images
Images Rotated by random angle
Orientation Detection(EVA)
Modified RESNET50 Classifier for orientation detection
Object detection (using resnet50)
Object detection (using resnet50)
Rotate image by α,180+α, β, 180+β
Object detection (using resnet50)
Orientation Correction
Decision Criteria
Predict the principal image
Object detection (using resnet50)
output class and accuracy

Figure 4.1.1 Flow chart

Figure 4.1.1 is a block diagram that gives a brief flow of algorithms in the project. The steps include pre-processing, Orientation Detection, Correction and predicting the principal image. Finally output of each accuracy of models are displayed.

4.1.1 UML diagrams

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The UML is a very important part of developing object-oriented software and the software development process.

The UML uses mostly graphical notations to express the design of software projects. Building blocks of the UML.

The vocabulary of the UML encompasses three kinds of building blocks.
1. Things.
2. Relationships
3. Diagrams.

4.1.2 Use Case Diagram

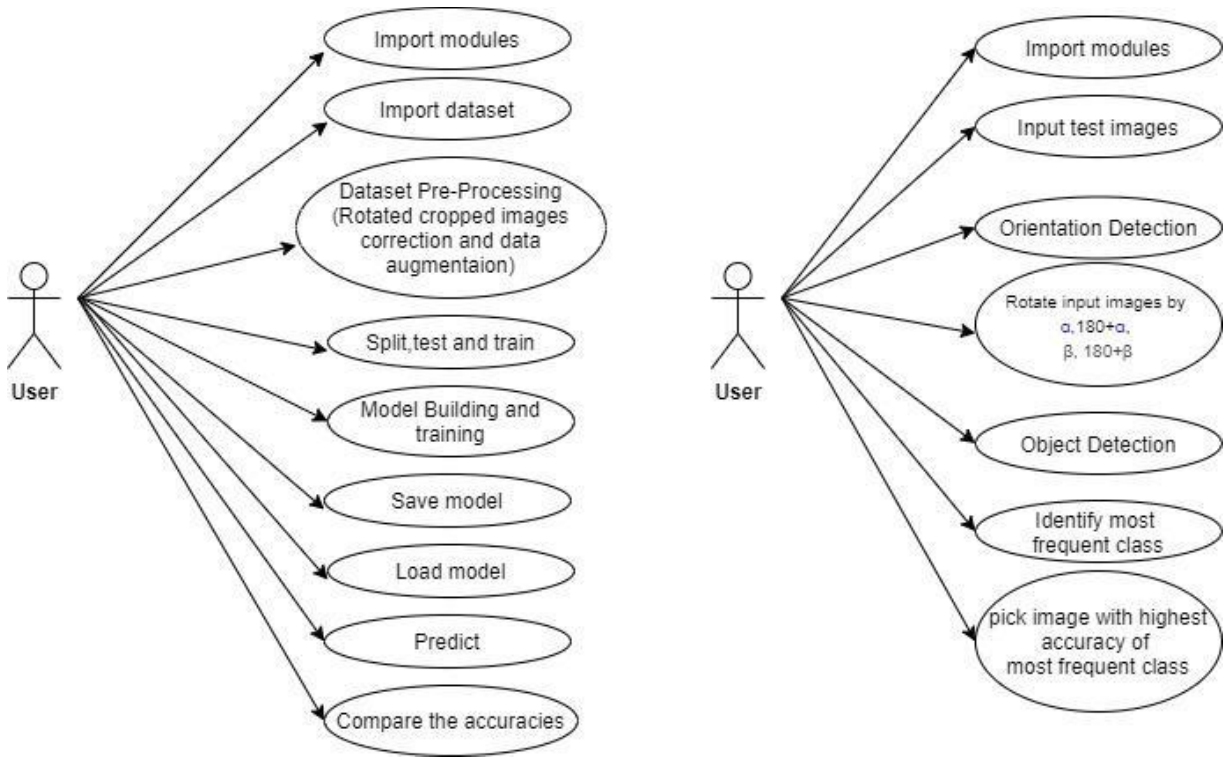

User
Import modules
Import dataset
Dataset Pre-Processing (Rotated cropped images correction and data augmentaion)
Split,test and train
Model Building and training
Save model
Load model
Predict
Compare the accuracies
User
Import modules
Input test images
Orientation Detection
Rotate input images by α,180+α, β, 180+β
Object Detection
Identify most frequent class
pick image with highest accuracy of most frequent class

Figure 4.1.2 Use case diagram

Figure 4.1.2 displays a use case diagram where it shows the functions of a user in this project. A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system.

4.2 Design and Test Steps:

The dataset was taken from Kaggle which consisted of 20,580 images that includes 120 breeds of dogs. We chose 1500 perfectly oriented images out of which 1000 are used for training and 500 are for testing. Data augmentation was performed on 1000 images which resulted in 23,000 images. Later our model is trained on a randomly oriented version of all the images under consideration.

Then we imported the Resnet50 model from Keras library which was already trained on the ImageNet[5] dataset. The architecture was modified by adding a dense layer to it with Softmax activation function and a dropout layer was added to overcome overfitting. Since the dataset has 360 classes i.e., each class representing 1 degree angle and we modified our model such that the output layer consisted of 360 neurons. This layer was added with a softmax activation function which produces probability of the 359 classes being 0 and only 1 class is 1 which is the predicted class angle. We used a Stochastic Gradient descent optimizer used to reduce the loss while training the model.

Then the images are rotated by random angle between 0 – 360 and the angle is stored in a separate numpy array. Then the images are cropped to remove the black border resulting from the rotated images thus ensuring the model is not trained on black borders. Similarly, the images in the testing dataset are processed in the same way. Then we trained the model using the images in the training dataset and labels given to them.

Then the model was saved, this model was loaded and the output for the images in the testing dataset were predicted. Then the test images are fed to our model, which produces prediction angles using which the input images are rotated in opposite directions. The resulting images are fed to the RESNET50 Object Detector which predicts its class and accuracy. This process of Object detection is repeated for both original perfectly oriented images and input rotated images and the accuracies are noted.

Relative performance: Ratio between accuracy of expected class in current image and accuracy of class detected in original image.
68 is the accuracy of the class detected in the original set of images. 14 is the accuracy at which expected classes are detected in rotated images. 60 is the accuracy at which expected classes are detected in corrected images by our model.
So, coming to our metrics, relative performance for our base paper 1 solution is 60/68 i.e., 0.88.

Relative performance without using our solution is 14/68 i.e., 0.20.

In the base paper 2, there is no overhead of training phase explicitly and this solution makes use of Eigenvector analysis of input rotated image which produces two angles α and β. Using those angles, the input image is rotated through α, α+180, β, β+180. All four images are passed through the ResNet50 object detector. The most frequent class among the predicted classes from those four images is picked. The one image among those four images will be chosen which has the highest accuracy for the most frequent class. That image will be the final corrected image.

4.3 Algorithms:

4.3.1 Base Paper-1:

- We imported the Resnet50 model from Keras library which was already trained on Imagenet dataset.
- Then the images are rotated by random angle between 0 – 360 and the angle is stored in separate numpy array.

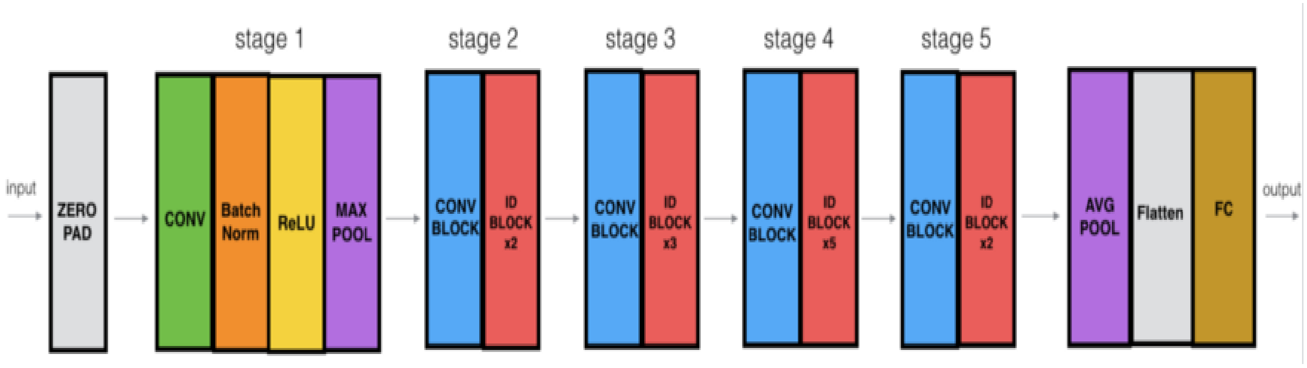


Figure 4.3.1 Resnet50 Architecture

- The Input layer specifies the input shape of the network, which must be equal to the dimensions of the input data. Next, we have two consecutive convolutional layers (Convolution2D) as shown in figure 4.3.1. These layers take the kernel size and the number of different kernels that we want to slide over their input as parameters.
- The next layer is a max pooling layer (MaxPooling2D), which also takes the kernel size (pool_size) as an input parameter. Since the pooling kernel is 2 × 2, the output shape of this layer will be half of the output of the preceding layer.
- The Dropout layer sets a fraction of its inputs (0.25 in the first dropout layer) to zero.

- The Flatten layer simply converts the three-dimensional input to one dimension. It is needed there because there is a dense layer coming next.
- Dense layers are the name that Keras gives to fully-connected layers. They take the number of outputs as a parameter.
- Finally, we have another dropout layer and the final fully-connected layer, which has the same number of outputs as the number of classes we want to predict.
- The last layer uses a softmax activation. This is used to normalize the raw class scores to class probabilities between zero and one.
- During the compilation step, we need to define the loss function, optimizer and metrics that we want to use during the training phase. If we are doing classification, we will typically use **'categorical_crosentropy'** as the loss function.
- The optimizer is the method used to perform the weight updates. Adam[7] is a good default optimizer that usually works well without extra configuration.
- Finally, we are using a function called **angle_error** as a metric. Metrics are used to monitor the accuracy of the model during training. The **angle_error** metric will be in charge of periodically computing the angle difference between predicted angles and true angles. During training, we will monitor the loss value and the angle error so that we can finish the process whenever they stop improving in the validation set.
- The **model**.**fit_generator** method will train the model on batches of data generated by our previously defined Image Data Generator for a number of epochs. In general, an epoch is completed whenever a network has been fed with the whole training set. Since we are generating data on-the-fly, we need to explicitly define the **samples_per_epoch** parameter.
- We also shuffle the data after each epoch, so that batches with different images are generated each time.
- The **validation_data** parameter takes another Image Data Generator object which is in charge of generating the validation images that we will use to periodically evaluate the model during training.
- The **ModelCheckpoint** callback is used to save the model to disk. With the **save_best_only** option, we only save the model whenever the accuracy improves.

- **EarlyStopping** will finish the training process whenever a monitored value has stopped improving. By default, it will monitor the loss value in the validation set.
- The **TensorBoard** callback is used to plot the monitored values. This is how the training is done.

Then the model is saved, this model is loaded and the output for the images in the testing dataset were predicted. Then the test images are fed to our model, which produces prediction angles using which the input images are rotated in opposite directions. The resulting images are fed to the RESNET50 Object Detector which predicts its class and accuracy. This process of Object detection is repeated for both original perfectly oriented images and input rotated images and the accuracies are noted

4.3.2 Base Paper-2:

4.3.2.1 Training Phase:

- Training involves no data augmentation or rotation of images.
- It is trained in the same way as the original, with only one specific orientation. No modifications are made in the architecture, and if a pre-trained model is available, the training can be completely avoided.

4.3.2.2 Testing Phase:

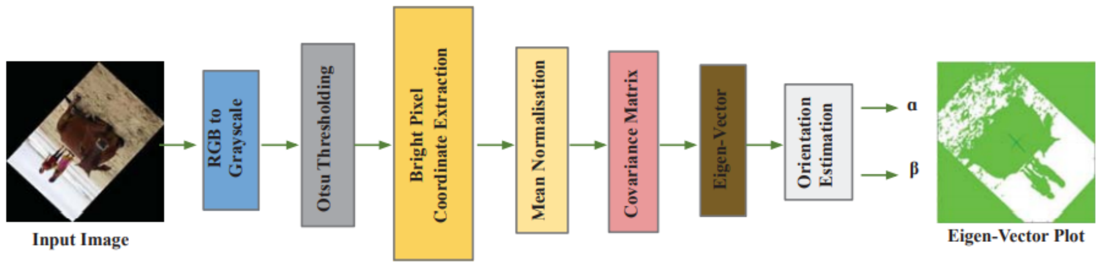


Fig. 3. Pre-processing steps for calculating orientation angles.

Figure 4.3.2 Pre-processing steps for calculating orientation angles

1. Convert the input image to grayscale.
2. Apply Otsu's thresholding to binarize it.
3. Extract the locations of bright pixels into a matrix from the segmented image.
4. Calculate the covariance matrix of these coordinates.
5. Perform singular value decomposition on the obtained covariance matrix.

6. By this, we get two eigenvalues and corresponding eigenvectors, termed as principal components. And the eigenvector corresponding to the largest eigenvalue is the first principal component.

    - The direction of the first principal component of images of a certain class is nearly the same.
    - The first principal component is approximately horizontal for original images with aspect ratio width > height and vertical for images with aspect ratio width < height.
    - The eigenvectors of an image and its 180°rotated version are in the same direction.

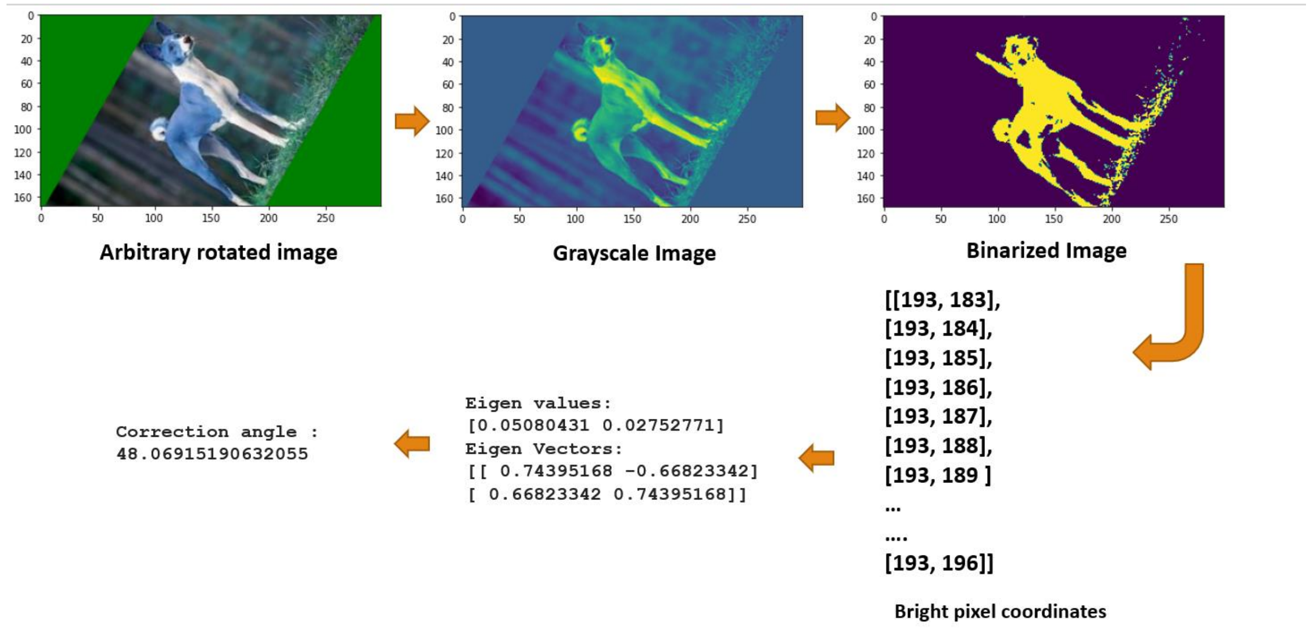
So, the estimated principal components may belong to two versions of an image. Next, we perform the orthogonal transformation [9] of these principal components so that they align in exactly horizontal and vertical positions and consider these as reference vectors.

By calculating the angle between the first principal component of the input image and these reference vectors, we obtain two angles of rotation, say α°and β°.

Then, with consideration of the above mentioned three observations, we rotate the input image by α°, 180 + α°, β°and 180 + β°.

This is the end stage of the pre-processing method and hence, we obtain four images at different orientations.

We feed these four images to the Object detector, to obtain detections and their objectness scores. The principal image among them is selected based on a decision criterion.



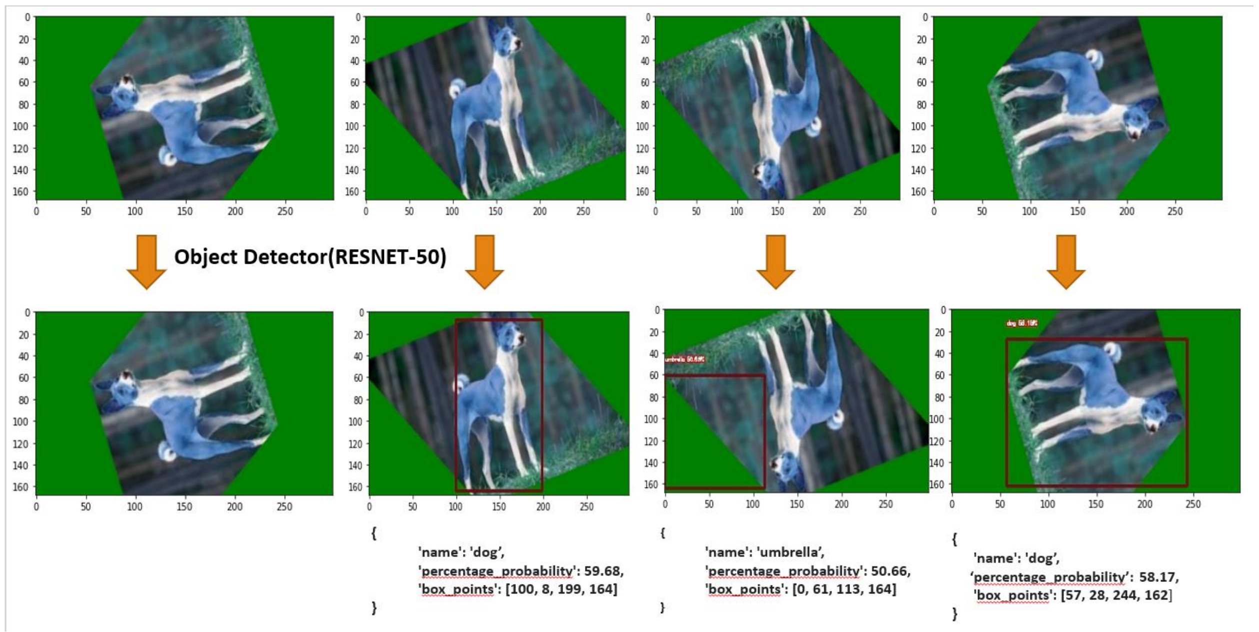
```
[[193, 183],
[193, 184],
[193, 185],
[193, 186],
[193, 187],
[193, 188],
[193, 189 ]
...
....
[193, 196]]
```

```
Eigen values:
[0.05080431 0.02752771]
Eigen Vectors:
[[ 0.74395168 -0.66823342]
[ 0.66823342 0.74395168]]
```

```
Correction angle :
48.06915190632055
```

4.3.3 Output at each stage of pre-processing

As shown in fig 4.3.3, the arbitrarily rotated image of a dog is converted to grayscale image thus reducing the no of channels from 3 to 1 which is then binarized using otsu thresholding where all the pixels with intensity above the threshold are rounded off to 255, rest to 0. Then we extract the locations of bright pixels into a matrix from the segmented image and calculate the covariance matrix of these coordinates. Next, we perform singular value decomposition on the obtained covariance matrix. By this, we get two eigenvalues and corresponding eigenvectors, termed as principal components.



```
{
    'name': 'dog',
    'percentage_probability': 59.68,
    'box_points': [100, 8, 199, 164]
}
```

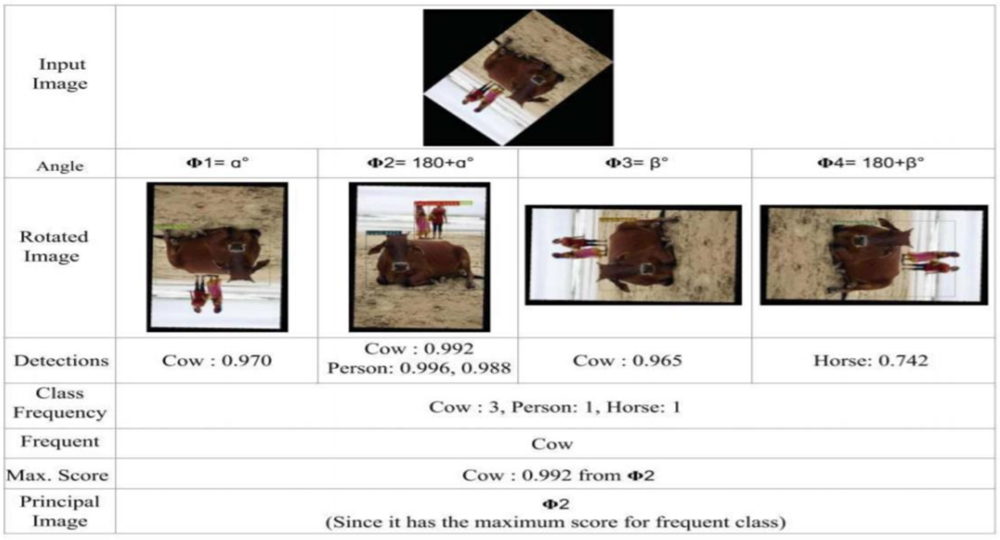
```
{
    'name': 'umbrella',
    'percentage_probability': 50.66,
    'box_points': [0, 61, 113, 164]
}
```

```
{
    'name': 'dog',
    'percentage_probability': 58.17,
    'box_points': [57, 28, 244, 162]
}
```

Fig 4.3.4 Four versions of the rotated images fed to Object Detector

As shown in the figure 4.3.4, by calculating the angle between the first principal component of the input image and the reference vectors, we obtain two angles of rotation, say α°and β°. Then, we rotate the input image by α°, 180 + α°, β°and 180 + β° and hence, we obtain four images at different orientations. We feed these four images to RESNET50, to obtain detections and their objectness scores. The principal image among them is selected based on a decision criterion.

Decision Criteria



Input Image				
Angle	Φ1= α°	Φ2= 180+α°	Φ3= β°	Φ4= 180+β°
Rotated Image				
Detections	Cow : 0.970	Cow : 0.992 Person: 0.996, 0.988	Cow : 0.965	Horse: 0.742
Class Frequency	Cow : 3, Person: 1, Horse: 1			
Frequent	Cow			
Max. Score	Cow : 0.992 from Φ2			
Principal Image	Φ2 (Since it has the maximum score for frequent class)			

Fig. 5. Decision Logic explained for finer rotation invariant image detection based on class frequency and their classified objectness score.

Fig 4.3.5 Decision Criteria

Based on the behavior of Resnet50 for rotated images, a decision criterion based on class frequency and objectiveness score is developed for selecting the principal image among the four, which gives the correct annotation of the input image.

Observations from detections are as follows:

- The detections are accurate in the image having orientation closer to training images. So, the principal image will have good predictions and higher scores.
- The object which is not in the principal image might be detected in any of the other three other orientations. But the score of that detection is less than that of at least one detection in the principal image.

- The objects present in the principal image can also be identified in other orientations with low scores. Thus, the number of times the true detections appear will be more than or equal to false detections.

As shown in the fig 4.3.5, we find the class which occurs the maximum number of times in all four detections. Then, we find the scores of that particular class in every image, if detected.

We find the maximum of those scores and the image having the detection with that score is chosen as the principal image. If there is no repetition, select the detection with the highest score in all four images and the respective image is principal image.

Further, if two or more classes have maximum likelihood, then select the one with maximum objectness score.

4.4 Data set Description:

We are using Stanford Dogs dataset [6] which is available publicly in Kaggle.

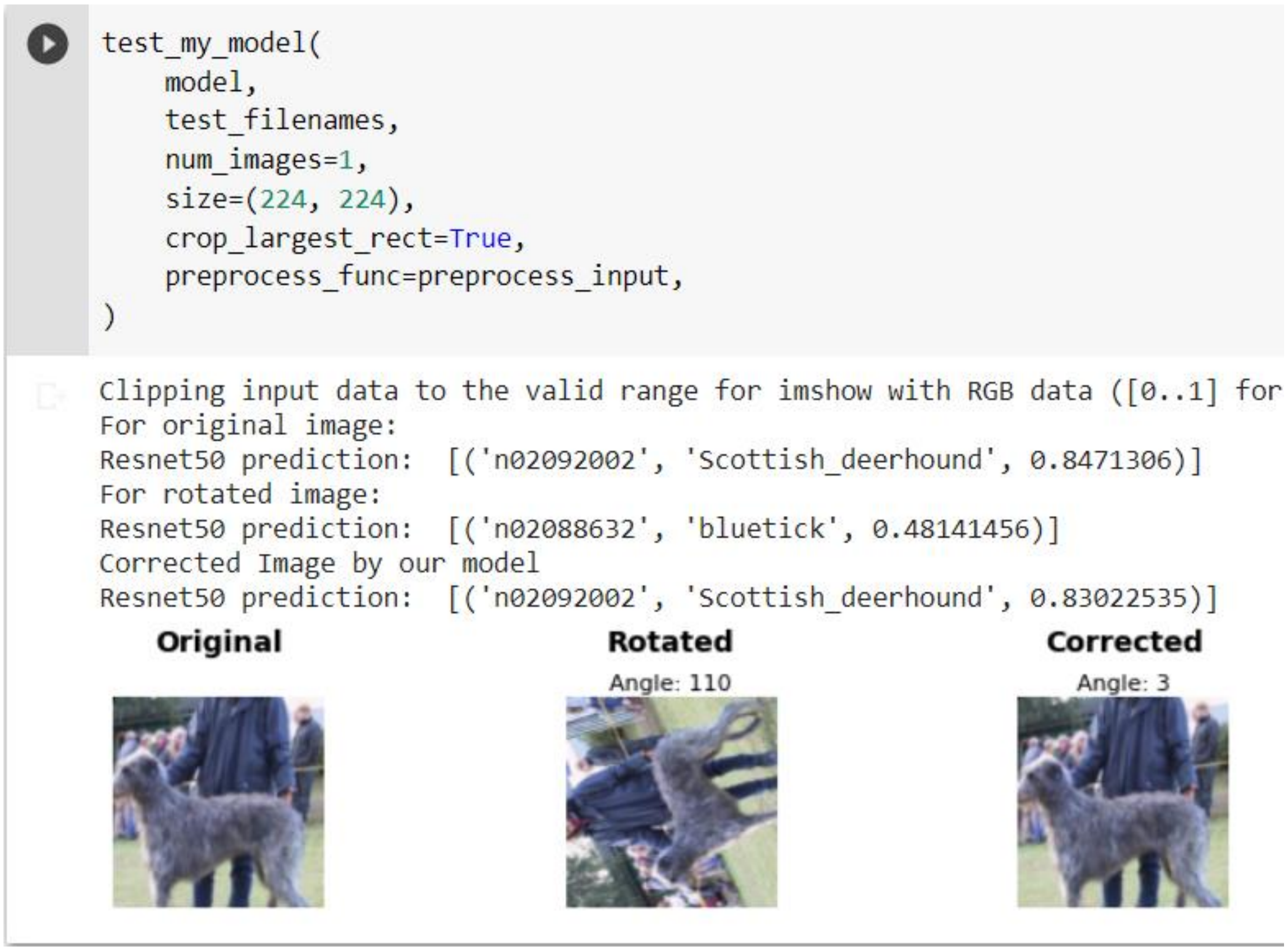
- The Stanford Dogs dataset contains images of 120 breeds of dogs from around the world.
- This dataset has been built using images and annotations from ImageNet for the task of fine-grained image categorization.
- It was originally collected for fine-grain image categorization, a challenging problem as certain dog breeds have near identical features or differ in color and age.
- Number of categories: 120
- Number of images: 20,580
- Annotations: Class labels, Bounding boxes

4.5 Testing process:

In this step our model is given the testing image dataset which has not been used for training the model. The test function rotates each image at random angle, does the preprocessing and feeds to the model. The model predicts the orientation for the images in the test dataset and using the angle predicted the input images are corrected. Then the corrected images are passed through the Resnet50 model which computes their classes and accuracies.

5. RESULTS AND DISCUSSIONS

5.1 Results for modified ResNet50 orientation classifier



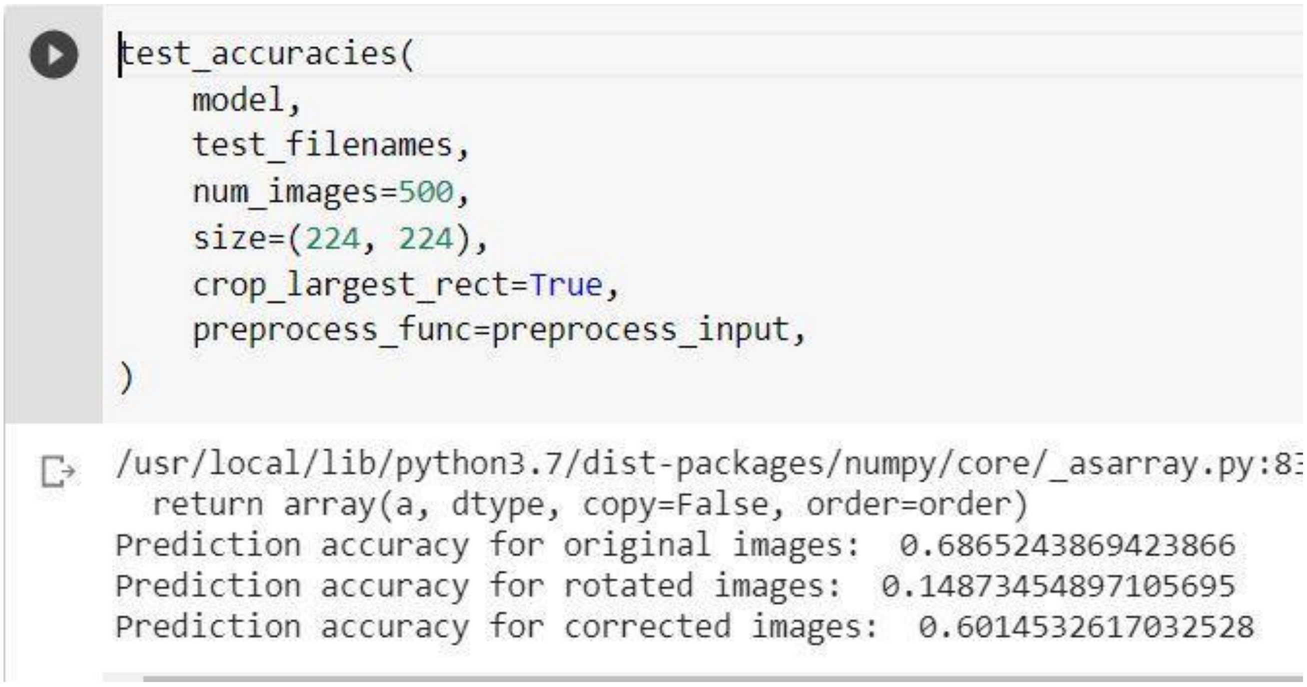
```
test_my_model(
    model,
    test_filenames,
    num_images=1,
    size=(224, 224),
    crop_largest_rect=True,
    preprocess_func=preprocess_input,
)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
For original image:
Resnet50 prediction:  [('n02092002', 'Scottish_deerhound', 0.8471306)]
For rotated image:
Resnet50 prediction:  [('n02088632', 'bluetick', 0.48141456)]
Corrected Image by our model
Resnet50 prediction:  [('n02092002', 'Scottish_deerhound', 0.83022535)]
```

Figure 5.1.1 Results for modified Resnet50 orientation classifier for single image

Figure 5.1.1 shows the output of test_my_model function which takes the trained model, test image filenames, number of images to be tested, size of images, flag for cropping largest rectangle, pre-processing function. The function picks num_images number of random images from test_filenames and converts them to prescribed size, rotates it at random angle, then processes using preprocess_func. The pre-processed image is then fed to modified Resnet50 orientation classifier which detects the orientation. The image is rotated in negative direction at the predicted angle, which is then fed to object detector from imageai library powered by Resnet50 classifier. Finally, the original image, rotated image, corrected image are shown along with their predicted classes and accuracies.

In the above shown instance, the original perfectly oriented image is classified as ‘Scottish_deerhound’ with accuracy of 0.84, while the rotated image is classified as ‘bluetick’, which is the wrong class with an accuracy of 0.48. Whereas, the corrected image is predicted as ‘Scottish_deerhound’ which is the correct class with an accuracy of 0.83.



```
test_accuracies(
    model,
    test_filenames,
    num_images=500,
    size=(224, 224),
    crop_largest_rect=True,
    preprocess_func=preprocess_input,
)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83
  return array(a, dtype, copy=False, order=order)
Prediction accuracy for original images:  0.6865243869423866
Prediction accuracy for rotated images:  0.14873454897105695
Prediction accuracy for corrected images:  0.6014532617032528
```

Figure 5.1.2 Results for modified Resnet50 orientation classifier for 500 test images

Figure 5.1.2 shows the output for test_accuracy function, which takes trained model, test image filenames, number of images to be tested, size of images, flag for cropping largest rectangle, pre-processing function. This function runs the test_my_model function without printing images, and computes the average accuracy.

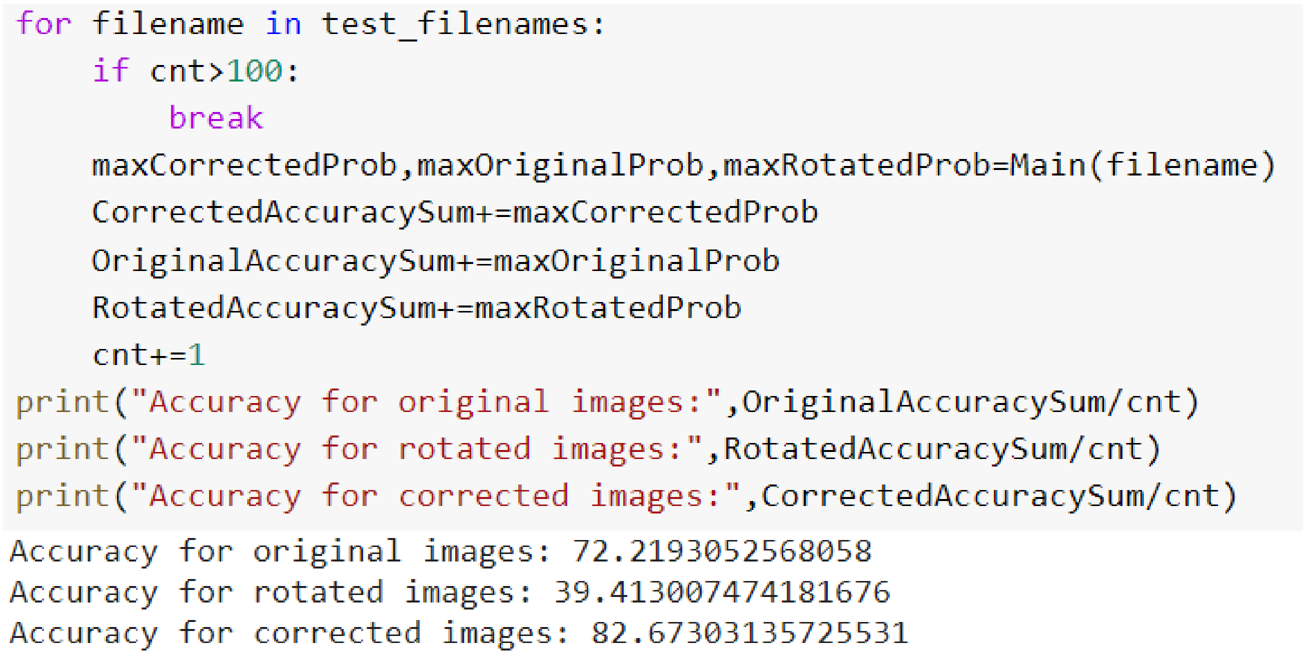
Prediction accuracy for original images is the average of accuracies detected for original images. Prediction accuracy for rotated images and corrected images is the average of accuracies, in which the accuracy for differently classified to that of class detected in original images is considered as zero.

Relative performance for this solution which is the ratio between accuracy of expected class in current image and accuracy of class detected in original image is 0.88

The relative performance shows how good the classes are getting detected in current test images, in comparison with that of original images.

5.2 Results for Eigenvector based solution

5.2.1 Results



```
for filename in test_filenames:
    if cnt>100:
        break
    maxCorrectedProb,maxOriginalProb,maxRotatedProb=Main(filename)
    CorrectedAccuracySum+=maxCorrectedProb
    OriginalAccuracySum+=maxOriginalProb
    RotatedAccuracySum+=maxRotatedProb
    cnt+=1
print("Accuracy for original images:",OriginalAccuracySum/cnt)
print("Accuracy for rotated images:",RotatedAccuracySum/cnt)
print("Accuracy for corrected images:",CorrectedAccuracySum/cnt)
```

```
Accuracy for original images: 72.2193052568058
Accuracy for rotated images: 39.413007474181676
Accuracy for corrected images: 82.67303135725531
```

Figure 5.2.1 Results for Eigenvector based solution for 100 test images

Figure 5.2.1 shows the use of the Main() function, which takes a perfectly oriented image filename as input, rotates it at random angle, and corrects it using our proposed eigenvector based orientation correction. The corrected image is then fed to the Resnet50 object detector which predicts the class and accuracy. The classes and accuracies for original and rotated versions are also detected. Finally, the Main function returns accuracy for corrected image, accuracy for original image and accuracy for rotated image. The above code computes this process for 100 test images, for which the average accuracy for original images, rotated images and corrected images is computed.

Relative performance for this solution is the ratio between accuracy of expected class in current image and accuracy of class detected in original image is 1.13. Which means, our correction model outperformed the original dataset. This is possible because the input dataset is handpicked perfectly oriented images, they had a human error.

6. CONCLUSION AND RECOMMENDATIONS

6.1 Conclusion

Observing the results obtained from our modified Resnet50 orientation classifier and Resnet50 object detector, we can say that the method performs well if the model is well trained. The important feature of the model is that it can correct the orientation and detect the classes with high accuracy without getting affected with symmetricity of object, intensity variation between object and background. The downsides are: this solution doesn't detect orientation if multiple objects are present. Since this solution requires multiple training images, it can't be used in situations such as defense, where confidentiality is important.

The above problems are solved by Eigenvector based solution. Observing the results obtained from our eigenvector orientation correction and Resnet50, we can say that the proposed detection method is elegant and robust, which makes accurate predictions irrespective of the sensitive behavior of CNNs to rotation. The prominent features of proposed method are training simplicity, cost, computational efficiency and no architectural modification. This method also works for finer degrees of rotation in the image acquired by the camera, by passing only four resulting orientations into the network. In recent works like RIMCNNN, the image is fed into the network for N times, where N depends on the degree of rotation. Hence, finer the rotation, more will be the instances, thus the computational complexity. In our work, we overcome this issue by using only four instances for any rotation. Thus, the proposed method is much simplified and time-efficient.

Finally, when we compare the two solutions we proposed, the eigenvector solution has much better relative performance in comparison to the modified Resnet50 model solution. But since both solutions have different use cases based on how they perform orientation detection and need of training images, we can't say which one is a better solution. They both are useful and best in their own use cases.

6.2 Future work

We can train our first solution model with a large dataset for products produced at industry level, so that it can be used for detecting faulty products. Automating product validation can alleviate errors caused during the validation process by human error. However, an impediment to automated product validation is the orientation of the product. It is crucial that a product is correctly oriented before the artificial system can run the quality check. This experiment is designed to address the object orientation problem encountered during the object recognition step of the product validation process

REFERENCES

[1] A. Jain, G. R. S. Subrahmanyam, and D. Mishra, “Rotation invariant digit recognition using convolutional neural network,” in Proceedings of 2nd International Conference on Computer Vision & Image Processing. Springer, 2018, pp. 91–102

[2] B. Giddwani, D. Varma, M. Murali and R. K. Gorthi, "Eigen Vectors based Rotation Invariant Multi-Object Deep Detector," 2020 7th International Conference on Signal Processing and Integrated Networks (SPIN), 2020, pp. 246-251, doi: 10.1109/SPIN48934.2020.9070989.

[3] F. De Smedt and T. Goedeme, “Fast rotation invariant object detection ´ with gradient based detection models,” in Proceedings of the 10th International Conference on Computer Vision Theory and Applications- (Volume 2)(VISIGRAPP 2015), vol. 2. VISIGRAPP; Setubal, 2015, pp. 400–407.

[4] Y. Zhou, Q. Ye, Q. Qiu, and J. Jiao, “Oriented response networks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 519–528

[5] Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L. **Imagenet**: A large-scale hierarchical image **database**. In: 2009 IEEE conference on computer vision and pattern recognition. 2009.

[6] Jessica Li, **Stanford Dogs Dataset:** contains images of 120 breeds of dogs from around the world.

[7] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 2014

[8] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” International journal of computer vision, vol. 88, no. 2, pp. 303–338, 2010.

[9] H. H. Abass and F. M. M. Al-Salbi, “Rotation and scaling image using pca,” Computer and Information Science, vol. 5, no. 1, p. 97, 2012.

[10] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

APPENDICES

#Base paper 1 Implementation

```
from __future__ import division

import math
import cv2
import numpy as np
import matplotlib.pyplot as plt

from keras.preprocessing.image import Iterator
from keras.utils.np_utils import to_categorical
import keras.backend as K
from keras.preprocessing import image as imgtool
from keras.preprocessing.image import img_to_array
from keras.applications import ResNet50
from keras.applications.resnet50 import preprocess_input
from keras.applications.imagenet_utils import decode_predictions
from PIL import Image as Image2



def angle_difference(x, y):
    """
    Calculate minimum difference between two angles.
    """
    return 180 - abs(abs(x - y) - 180)



def angle_error(y_true, y_pred):
    """
    Calculate the mean difference between the true angles
    and the predicted angles. Each angle is represented
    as a binary vector.
    """
    diff = angle_difference(K.argmax(y_true), K.argmax(y_pred))
    return K.mean(K.cast(K.abs(diff), K.floatx()))



def angle_error_regression(y_true, y_pred):
    """
    Calculate the mean difference between the true angles
    and the predicted angles. Each angle is represented
    as a float number between 0 and 1.
    """
    return K.mean(angle_difference(y_true * 360, y_pred * 360))


def binarize_images(x):
    """
    Convert images to range 0-1 and binarize them by making
    0 the values below 0.1 and 1 the values above 0.1.
    """
    x /= 255
    x[x >= 0.1] = 1
    x[x < 0.1] = 0
    return x
def rotate(image, angle):
    """
    Rotates an OpenCV 2 / NumPy image about it's centre by the given angle
    (in degrees). The returned image will be large enough to hold the entire
    new image, with a black background

    """

    image_size = (image.shape[1], image.shape[0])
    image_center = tuple(np.array(image_size) / 2)

    rot_mat = np.vstack(
        [cv2.getRotationMatrix2D(image_center, angle, 1.0), [0, 0, 1]]
    )

    rot_mat_notranslate = np.matrix(rot_mat[0:2, 0:2])


    image_w2 = image_size[0] * 0.5
    image_h2 = image_size[1] * 0.5

    rotated_coords = [
        (np.array([-image_w2,  image_h2]) * rot_mat_notranslate).A[0],
        (np.array([ image_w2,  image_h2]) * rot_mat_notranslate).A[0],
        (np.array([-image_w2, -image_h2]) * rot_mat_notranslate).A[0],
        (np.array([ image_w2, -image_h2]) * rot_mat_notranslate).A[0]
    ]

    x_coords = [pt[0] for pt in rotated_coords]
    x_pos = [x for x in x_coords if x > 0]
    x_neg = [x for x in x_coords if x < 0]

    y_coords = [pt[1] for pt in rotated_coords]
    y_pos = [y for y in y_coords if y > 0]
    y_neg = [y for y in y_coords if y < 0]

    right_bound = max(x_pos)
    left_bound = min(x_neg)
    top_bound = max(y_pos)
    bot_bound = min(y_neg)

    new_w = int(abs(right_bound - left_bound))
    new_h = int(abs(top_bound - bot_bound))

    trans_mat = np.matrix([
        [1, 0, int(new_w * 0.5 - image_w2)],
        [0, 1, int(new_h * 0.5 - image_h2)],
        [0, 0, 1]
    ])

    affine_mat = (np.matrix(trans_mat) * np.matrix(rot_mat))[0:2, :]

    result = cv2.warpAffine(
        image,
        affine_mat,
        (new_w, new_h),
        flags=cv2.INTER_LINEAR
    )

    return result


def largest_rotated_rect(w, h, angle):
    """
    Given a rectangle of size wxh that has been rotated by 'angle' (in
    radians), computes the width and height of the largest possible
    axis-aligned rectangle within the rotated rectangle.

    """

    quadrant = int(math.floor(angle / (math.pi / 2))) & 3
    sign_alpha = angle if ((quadrant & 1) == 0) else math.pi - angle
    alpha = (sign_alpha % math.pi + math.pi) % math.pi

    bb_w = w * math.cos(alpha) + h * math.sin(alpha)
    bb_h = w * math.sin(alpha) + h * math.cos(alpha)

    gamma = math.atan2(bb_w, bb_w) if (w < h) else math.atan2(bb_w, bb_w)

    delta = math.pi - alpha - gamma

    length = h if (w < h) else w

    d = length * math.cos(alpha)
    a = d * math.sin(alpha) / math.sin(delta)

    y = a * math.cos(gamma)
    x = y * math.tan(gamma)

    return (
        bb_w - 2 * x,
        bb_h - 2 * y
    )


def crop_around_center(image, width, height):
    """
    Given a NumPy / OpenCV 2 image, crops it to the given width and height,
    around it's centre point
    """

    image_size = (image.shape[1], image.shape[0])
    image_center = (int(image_size[0] * 0.5), int(image_size[1] * 0.5))

    if(width > image_size[0]):
        width = image_size[0]

    if(height > image_size[1]):
        height = image_size[1]

    x1 = int(image_center[0] - width * 0.5)
    x2 = int(image_center[0] + width * 0.5)
    y1 = int(image_center[1] - height * 0.5)
    y2 = int(image_center[1] + height * 0.5)

    return image[y1:y2, x1:x2]


def crop_largest_rectangle(image, angle, height, width):
    return crop_around_center(image,*largest_rotated_rect(width,height,math.r
adians(angle)))


def generate_rotated_image(image, angle, size=None, crop_center=False,
                           crop_largest_rect=False):
    """
    Generate a valid rotated image for the ImageDataGenerator. If the
    image is rectangular, the crop_center option should be used to make
    it square. To crop out the black borders after rotation, use the
    crop_largest_rect option. To resize the final image, use the size
    option.
    """
    height, width = image.shape[:2]
    if crop_center:
        if width < height:
            height = width
        else:
            width = height

    image = rotate(image, angle)

    if crop_largest_rect:
        image = crop_largest_rectangle(image, angle, height, width)

    if size:
        image = cv2.resize(image, size)

    return image


class ImageDataGenerator(Iterator):
    """
    Given a NumPy array of images or a list of image paths,
    generate batches of rotated images and rotation angles on-the-fly.
    """

    def __init__(self, input, input_shape=None, color_mode='rgb', batch_size=
64,
                 one_hot=True, preprocess_func=None, rotate=True, crop_center
=False,
                 crop_largest_rect=False, shuffle=False, seed=None):
```

```
        self.images = None
        self.filenames = None
        self.input_shape = input_shape
        self.color_mode = color_mode
        self.batch_size = batch_size
        self.one_hot = one_hot
        self.preprocess_func = preprocess_func
        self.rotate = rotate
        self.crop_center = crop_center
        self.crop_largest_rect = crop_largest_rect
        self.shuffle = shuffle

        if self.color_mode not in {'rgb', 'grayscale'}:
            raise ValueError('Invalid color mode:', self.color_mode,
                             '; expected "rgb" or "grayscale".')

        if isinstance(input, (np.ndarray)):
            self.images = input
            N = self.images.shape[0]
            if not self.input_shape:
                self.input_shape = self.images.shape[1:]
                if len(self.input_shape) == 2:
                    self.input_shape = self.input_shape + (1,)
        else:
            self.filenames = input
            N = len(self.filenames)

        super(ImageDataGenerator, self).__init__(N, batch_size, shuffle, seed
)

    def _get_batches_of_transformed_samples(self, index_array):
        batch_x = np.zeros((len(index_array),) + self.input_shape, dtype='flo
at32')
        batch_y = np.zeros(len(index_array), dtype='float32')
        for i, j in enumerate(index_array):
            if self.filenames is None:
                image = self.images[j]
            else:
                is_color = int(self.color_mode == 'rgb')
                image = cv2.imread(self.filenames[j], is_color)
                if is_color:
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            if self.rotate:
                rotation_angle = np.random.randint(360)
            else:
                rotation_angle = 0
            rotated_image = generate_rotated_image(image,rotation_angle,size=
self.input_shape[:2],crop_center=self.crop_center,
                crop_largest_rect=self.crop_largest_rect)
            if rotated_image.ndim == 2:
                rotated_image = np.expand_dims(rotated_image, axis=2)
            batch_x[i] = rotated_image
            batch_y[i] = rotation_angle
        if self.one_hot:
            batch_y = to_categorical(batch_y, 360)
        else:
            batch_y /= 360
        if self.preprocess_func:
            batch_x = self.preprocess_func(batch_x)
        return batch_x, batch_y

    def next(self):
        with self.lock:
            index_array = next(self.index_generator)
        return self._get_batches_of_transformed_samples(index_array)


def display_examples(model, input, num_images=5, size=None, crop_center=False
,
                     crop_largest_rect=False, preprocess_func=None, save_path
=None):
    if isinstance(input, (np.ndarray)):
        images = input
        N, h, w = images.shape[:3]
        if not size:
            size = (h, w)
        indexes = np.random.choice(N, num_images)
        images = images[indexes, ...]
    else:
        images = []
        filenames = input
        N = len(filenames)
        indexes = np.random.choice(N, num_images)
        for i in indexes:
            image = cv2.imread(filenames[i])
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            images.append(image)
        images = np.asarray(images)

    x = []
    y = []
    for image in images:
        rotation_angle = np.random.randint(90,270)
        rotated_image = generate_rotated_image(
            image,
            rotation_angle,
            size=size,
            crop_center=crop_center,
            crop_largest_rect=crop_largest_rect
        )
        x.append(rotated_image)
        y.append(rotation_angle)

    x = np.asarray(x, dtype='float32')
    y = np.asarray(y, dtype='float32')

    if x.ndim == 3:
        x = np.expand_dims(x, axis=3)

    y = to_categorical(y, 360)

    x_rot = np.copy(x)

    if preprocess_func:
        x = preprocess_func(x)
    y = np.argmax(y, axis=1)
    y_pred = np.argmax(model.predict(x), axis=1)

    plt.figure(figsize=(10.0, 2 * num_images))

    title_fontdict = {
        'fontsize': 14,
        'fontweight': 'bold'
    }

    fig_number = 0
    for rotated_image, true_angle, predicted_angle in zip(x_rot, y, y_pred):
        original_image = rotate(rotated_image, -true_angle)
        if crop_largest_rect:
            original_image = crop_largest_rectangle(original_image, -
true_angle, *size)

        corrected_image = rotate(rotated_image, -predicted_angle)
        if crop_largest_rect:
            corrected_image = crop_largest_rectangle(corrected_image, -
predicted_angle, *size)

        if x.shape[3] == 1:
            options = {'cmap': 'gray'}
        else:
            options = {}

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        if fig_number == 1:
            plt.title('Original\n', fontdict=title_fontdict)
        plt.imshow(np.squeeze(original_image).astype('uint8'), **options)
        plt.axis('off')

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        if fig_number == 2:
            plt.title('Rotated\n', fontdict=title_fontdict)
        ax.text(
            0.5, 1.03, 'Angle: {0}'.format(true_angle),
            horizontalalignment='center',
            transform=ax.transAxes,
            fontsize=11
        )
        plt.imshow(np.squeeze(rotated_image).astype('uint8'), **options)
        plt.axis('off')

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        corrected_angle = angle_difference(predicted_angle, true_angle)
        if fig_number == 3:
            plt.title('Corrected\n', fontdict=title_fontdict)
        ax.text(
            0.5, 1.03, 'Angle: {0}'.format(corrected_angle),
            horizontalalignment='center',
            transform=ax.transAxes,
            fontsize=11
        )
        plt.imshow(np.squeeze(corrected_image).astype('uint8'), **options)
        plt.axis('off')

    plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

    if save_path:
        plt.savefig(save_path)



def test_my_model(model, input, num_images=5, size=None, crop_center=False,
                     crop_largest_rect=False, preprocess_func=None, save_path
=None):
    """
    Given a model that predicts the rotation angle of an image,
    and a NumPy array of images or a list of image paths, display
    the specified number of example images in three columns:
    Original, Rotated and Corrected.
    """

    if isinstance(input, (np.ndarray)):
        images = input
        N, h, w = images.shape[:3]
        if not size:
            size = (h, w)
        indexes = np.random.choice(N, num_images)
        images = images[indexes, ...]
    else:
        images = []
        filenames = input
        N = len(filenames)
        indexes = np.random.choice(N, num_images)
        for i in indexes:
            image = cv2.imread(filenames[i])
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            images.append(image)
        images = np.asarray(images)

    x = []
    y = []
    for image in images:
        rotation_angle = np.random.randint(90,270)
        rotated_image = generate_rotated_image(
            image,
            rotation_angle,
            size=size,
            crop_center=crop_center,
            crop_largest_rect=crop_largest_rect
        )
        x.append(rotated_image)
        y.append(rotation_angle)

    x = np.asarray(x, dtype='float32')
    y = np.asarray(y, dtype='float32')

    if x.ndim == 3:
        x = np.expand_dims(x, axis=3)
    y = to_categorical(y, 360)

    x_rot = np.copy(x)

    if preprocess_func:
        x = preprocess_func(x)

    y = np.argmax(y, axis=1)
    y_pred = np.argmax(model.predict(x), axis=1)
    renet50model = ResNet50(weights='imagenet')
    plt.figure(figsize=(10.0, 2 * num_images))

    title_fontdict = {
        'fontsize': 14,
        'fontweight': 'bold'
    }

    fig_number = 0
    for rotated_image, true_angle, predicted_angle in zip(x_rot, y, y_pred):
        rotated_image2=rotated_image.copy()
        plt.imshow(rotated_image)
        original_image = rotate(rotated_image, -true_angle)
        if crop_largest_rect:
            original_image = crop_largest_rectangle(original_image, -
true_angle, *size)

        corrected_image = rotate(rotated_image, -predicted_angle)
        if True:
            corrected_image = crop_largest_rectangle(corrected_image, -
predicted_angle, *size)
        if x.shape[3] == 1:
            options = {'cmap': 'gray'}
        else:
            options = {}
        print('For original image:')
        img = original_image.copy()
        cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/RotNet-
master - Copy/data/temp_test/image2.jpg', img)
        img = imgtool.load_img('/content/drive/MyDrive/MajorProjectBP1/RotNet
-master - Copy/data/temp_test/image2.jpg', target_size = (224, 224))
        # plt.imshow(np.squeeze(rotated_image).astype('uint8'))
        img = imgtool.img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img = preprocess_input(img)
        print('Resnet50 prediction: ',decode_predictions(renet50model.predict
(img), top=1)[0])

        print('For rotated image:')
        img = rotated_image.copy()
        img=img.resize((224, 224))
        # plt.imshow(np.squeeze(rotated_image).astype('uint8'))
        # img = imgtool.img_to_array(img)
        img = np.expand_dims(rotated_image, axis=0)
        img = preprocess_input(img)
        print('Resnet50 prediction: ',decode_predictions(renet50model.predict
(img), top=1)[0])
        print('Corrected Image by our model')
        img = corrected_image.copy()
        cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/RotNet-
master - Copy/data/temp_test/image1.jpg', img)
        img = imgtool.load_img('/content/drive/MyDrive/MajorProjectBP1/RotNet
-master - Copy/data/temp_test/image1.jpg', target_size = (224, 224))
        # # plt.imshow(img)
        img = imgtool.img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img = preprocess_input(img)
        # img=(img.copy()).resize(None,224,224,3)
        print('Resnet50 prediction: ',decode_predictions(renet50model.predict
(img), top=1)[0])

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        if fig_number == 1:
            plt.title('Original\n', fontdict=title_fontdict)
        plt.imshow(np.squeeze(original_image).astype('uint8'), **options)
        plt.axis('off')

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        if fig_number == 2:
            plt.title('Rotated\n', fontdict=title_fontdict)
        ax.text(
            0.5, 1.03, 'Angle: {0}'.format(true_angle),
            horizontalalignment='center',
            transform=ax.transAxes,
            fontsize=11
        )
        plt.imshow(np.squeeze(rotated_image2).astype('uint8'), **options)
        plt.axis('off')

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        corrected_angle = angle_difference(predicted_angle, true_angle)
        if fig_number == 3:
            plt.title('Corrected\n', fontdict=title_fontdict)
        ax.text(
            0.5, 1.03, 'Angle: {0}'.format(corrected_angle),
            horizontalalignment='center',
            transform=ax.transAxes,
            fontsize=11
        )
        plt.imshow(np.squeeze(corrected_image).astype('uint8'), **options)
        plt.axis('off')

    plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

    if save_path:
        plt.savefig(save_path)


def test_accuracies(model, input, num_images=5, size=None, crop_center=False,
                     crop_largest_rect=False, preprocess_func=None, save_path
=None):
    """
    Given a model that predicts the rotation angle of an image,
    and a NumPy array of images or a list of image paths, evaluates the
accuracies for object detected in original image, rotated image, corrected
image.
    """
    if isinstance(input, (np.ndarray)):
        images = input
        N, h, w = images.shape[:3]
        if not size:
            size = (h, w)
        indexes = np.random.choice(N, num_images)
        images = images[indexes, ...]
    else:
        images = []
        filenames = input
        N = len(filenames)
        indexes = np.random.choice(N, num_images)
        for i in indexes:
            image = cv2.imread(filenames[i])
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            images.append(image)
        images = np.asarray(images)

    x = []
    y = []
    for image in images:
        rotation_angle = np.random.randint(90,270)
        rotated_image = generate_rotated_image(
            image,
            rotation_angle,
            size=size,
            crop_center=crop_center,
            crop_largest_rect=crop_largest_rect
        )
        x.append(rotated_image)
        y.append(rotation_angle)

    x = np.asarray(x, dtype='float32')
    y = np.asarray(y, dtype='float32')

    if x.ndim == 3:
        x = np.expand_dims(x, axis=3)

    y = to_categorical(y, 360)

    x_rot = np.copy(x)

    if preprocess_func:
        x = preprocess_func(x)

    y = np.argmax(y, axis=1)
    y_pred = np.argmax(model.predict(x), axis=1)
    resnet50model = ResNet50(weights='imagenet')
    orginal_img_acc_sum=0
    rotated_img_acc_sum=0
    corrected_img_acc_sum=0
    for rotated_image, true_angle, predicted_angle in zip(x_rot, y, y_pred):
        rotated_image2=rotated_image.copy()
        original_image = rotate(rotated_image, -true_angle)
        if crop_largest_rect:
            original_image = crop_largest_rectangle(original_image, -
true_angle, *size)

        corrected_image = rotate(rotated_image, -predicted_angle)
        if True:
            corrected_image = crop_largest_rectangle(corrected_image, -
predicted_angle, *size)
        if x.shape[3] == 1:
            options = {'cmap': 'gray'}
        else:
            options = {}
        img = original_image.copy()
        cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/RotNet-
master - Copy/data/temp_test/image2.jpg', img)
        img = imgtool.load_img('/content/drive/MyDrive/MajorProjectBP1/RotNet
-master - Copy/data/temp_test/image2.jpg', target_size = (224, 224))
        img = imgtool.img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img = preprocess_input(img)
        temp=decode_predictions(renet50model.predict(img), top=1)[0]
        dogClass=temp[0][1]
        orginal_img_acc_sum+=temp[0][2]
        img = rotated_image.copy()
        img=img.resize((224, 224))
        img = np.expand_dims(rotated_image, axis=0)
        img = preprocess_input(img)
        temp=decode_predictions(renet50model.predict(img), top=1)[0]
        if temp[0][1]==dogClass:
            rotated_img_acc_sum+=temp[0][2]
        img = corrected_image.copy()
        cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/RotNet-
master - Copy/data/temp_test/image1.jpg', img)
        img = imgtool.load_img('/content/drive/MyDrive/MajorProjectBP1/RotNet
-master - Copy/data/temp_test/image1.jpg', target_size = (224, 224))
        img = imgtool.img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img = preprocess_input(img)
        temp=decode_predictions(renet50model.predict(img), top=1)[0]
        if temp[0][1]==dogClass:
            corrected_img_acc_sum+=temp[0][2]
    print("Prediction accuracy for original images: ",orginal_img_acc_sum/num
_images)
    print("Prediction accuracy for rotated images: ",rotated_img_acc_sum/num_
images)
    print("Prediction accuracy for corrected images: ",corrected_img_acc_sum/
num_images)

def test_eigenVectors(model, input, num_images=5, size=None, crop_center=Fals
e,
                     crop_largest_rect=False, preprocess_func=None, save_path
=None):

    if isinstance(input, (np.ndarray)):
        images = input
        N, h, w = images.shape[:3]
        if not size:
            size = (h, w)
        indexes = np.random.choice(N, num_images)
        images = images[indexes, ...]
    else:
        images = []
        filenames = input
        N = len(filenames)
        indexes = np.random.choice(N, num_images)
        for i in indexes:
            image = cv2.imread(filenames[i])
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            images.append(image)
        images = np.asarray(images)

    x = []
    y = []
    for image in images:
        rotation_angle = np.random.randint(90,270)
        rotated_image = generate_rotated_image(
            image,
            rotation_angle,
            size=size,
            crop_center=crop_center,
            crop_largest_rect=crop_largest_rect
        )
        x.append(rotated_image)
        y.append(rotation_angle)

    x = np.asarray(x, dtype='float32')
    y = np.asarray(y, dtype='float32')

    if x.ndim == 3:
        x = np.expand_dims(x, axis=3)

    y = to_categorical(y, 360)

    x_rot = np.copy(x)

    if preprocess_func:
        x = preprocess_func(x)

    y = np.argmax(y, axis=1)
    y_pred = np.argmax(model.predict(x), axis=1)
    renet50model = ResNet50(weights='imagenet')
    plt.figure(figsize=(10.0, 2 * num_images))

    title_fontdict = {
        'fontsize': 14,
        'fontweight': 'bold'
    }

    fig_number = 0
    for rotated_image, true_angle, predicted_angle in zip(x_rot, y, y_pred):
        rotated_image2=rotated_image.copy()
        plt.imshow(rotated_image)
        original_image = rotate(rotated_image, -true_angle)
        if crop_largest_rect:
            original_image = crop_largest_rectangle(original_image, -
true_angle, *size)
```

```
        corrected_image = rotate(rotated_image, -predicted_angle)
        if True:
            corrected_image = crop_largest_rectangle(corrected_image, -
predicted_angle, *size)
        if x.shape[3] == 1:
            options = {'cmap': 'gray'}
        else:
            options = {}
        print('For original image:')
        img = original_image.copy()
        cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/RotNet-
master - Copy/data/temp_test/image2.jpg', img)
        img = imgtool.load_img('/content/drive/MyDrive/MajorProjectBP1/RotNet
-master - Copy/data/temp_test/image2.jpg', target_size = (224, 224))
        # plt.imshow(np.squeeze(rotated_image).astype('uint8'))
        img = imgtool.img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img = preprocess_input(img)
        print('Resnet50 prediction: ',decode_predictions(renet50model.predict
(img), top=1)[0])

        print('For rotated image:')
        img = rotated_image.copy()
        img=img.resize((224, 224))
        # plt.imshow(np.squeeze(rotated_image).astype('uint8'))
        # img = imgtool.img_to_array(img)
        img = np.expand_dims(rotated_image, axis=0)
        img = preprocess_input(img)
        print('Resnet50 prediction: ',decode_predictions(renet50model.predict
(img), top=1)[0])
        print('Corrected Image by our model')
        img = corrected_image.copy()
        cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/RotNet-
master - Copy/data/temp_test/image1.jpg', img)
        img = imgtool.load_img('/content/drive/MyDrive/MajorProjectBP1/RotNet
-master - Copy/data/temp_test/image1.jpg', target_size = (224, 224))
        # # plt.imshow(img)
        img = imgtool.img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img = preprocess_input(img)
        # img=(img.copy()).resize(None,224,224,3)
        print('Resnet50 prediction: ',decode_predictions(renet50model.predict
(img), top=1)[0])

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        if fig_number == 1:
            plt.title('Original\n', fontdict=title_fontdict)
        plt.imshow(np.squeeze(original_image).astype('uint8'), **options)
        plt.axis('off')

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        if fig_number == 2:
            plt.title('Rotated\n', fontdict=title_fontdict)
        ax.text(
            0.5, 1.03, 'Angle: {0}'.format(true_angle),
            horizontalalignment='center',
            transform=ax.transAxes,
            fontsize=11
        )
        plt.imshow(np.squeeze(rotated_image2).astype('uint8'), **options)
        plt.axis('off')

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        corrected_angle = angle_difference(predicted_angle, true_angle)
        if fig_number == 3:
            plt.title('Corrected\n', fontdict=title_fontdict)
        ax.text(
            0.5, 1.03, 'Angle: {0}'.format(corrected_angle),
            horizontalalignment='center',
            transform=ax.transAxes,
            fontsize=11
        )
        plt.imshow(np.squeeze(corrected_image).astype('uint8'), **options)
        plt.axis('off')

    plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

    if save_path:
        plt.savefig(save_path)


from __future__ import print_function

import os
import cv2
import numpy as np
import argparse

from keras.applications.imagenet_utils import preprocess_input
from keras.models import load_model

from utils import ImageDataGenerator, crop_largest_rectangle, angle_error, ro
tate


def process_images(model, input_path, output_path,
                   batch_size=64, crop=True):
    extensions = ['.jpg', '.jpeg', '.bmp', '.png']

    output_is_image = False
    if os.path.isfile(input_path):
        image_paths = [input_path]
        if os.path.splitext(output_path)[1].lower() in extensions:
            output_is_image = True
            output_filename = output_path
            output_path = os.path.dirname(output_filename)
    else:
        image_paths = [os.path.join(input_path, f)
                       for f in os.listdir(input_path)
                       if os.path.splitext(f)[1].lower() in extensions]
        if os.path.splitext(output_path)[1].lower() in extensions:
            print('Output must be a directory!')
    predictions = model.predict_generator(
        ImageDataGenerator(
            image_paths,
            input_shape=(224, 224, 3),
            batch_size=64,
            one_hot=True,
            preprocess_func=preprocess_input,
            rotate=False,
            crop_largest_rect=True,
            crop_center=True
        ),
        val_samples=len(image_paths)
    )

    predicted_angles = np.argmax(predictions, axis=1)

    if output_path == '':
        output_path = '.'

    if not os.path.exists(output_path):
        os.makedirs(output_path)

    for path, predicted_angle in zip(image_paths, predicted_angles):
        image = cv2.imread(path)
        rotated_image = rotate(image, -predicted_angle)
        if crop:
            size = (image.shape[0], image.shape[1])
            rotated_image = crop_largest_rectangle(rotated_image, -
predicted_angle, *size)
        if not output_is_image:
            output_filename = os.path.join(output_path, os.path.basename(path
))
        cv2.imwrite(output_filename, rotated_image)


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('model', help='Path to model')
    parser.add_argument('input_path', help='Path to image or directory')
    parser.add_argument('-o', '--output_path', help='Output directory')
    parser.add_argument('-b', '--
batch_size', help='Batch size for running the network')
    parser.add_argument('-c', '--
crop', dest='crop', default=False, action='store_true',
                        help='Crop out black borders after rotating')
    args = parser.parse_args()

    print('Loading model...')
    model_location = load_model(args.model, custom_objects={'angle_error': an
gle_error})
    output_path = args.output_path if args.output_path else args.input_path

    print('Processsing input image(s)...')
    process_images(model_location, args.input_path, output_path,
                   args.batch_size, args.crop)
```

#Training

```
from __future__ import print_function

import os
import sys

from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard, Redu
ceLROnPlateau
from keras.applications.resnet50 import ResNet50
from keras.applications.imagenet_utils import preprocess_input
from keras.models import Model
from keras.layers import Dense, Flatten
from keras.optimizers import SGD

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
from utils import angle_error, ImageDataGenerator
from data.street_view import get_filenames as get_street_view_filenames


data_path = os.path.join('data', 'street_view')
train_filenames, test_filenames = get_street_view_filenames(data_path)

print(len(train_filenames), 'train samples')
print(len(test_filenames), 'test samples')

model_name = 'resnet50_newDS'

# number of classes
nb_classes = 360
# input image shape
input_shape = (224, 224, 3)

# load base model
base_model = ResNet50(weights='imagenet', include_top=False,
                      input_shape=input_shape)

# append classification layer
x = base_model.output
x = Flatten()(x)
final_output = Dense(nb_classes, activation='softmax', name='fc360')(x)

# create the new model
model = Model(inputs=base_model.input, outputs=final_output)

model.summary()
output_folder = '/content/drive/MyDrive/MajorProjectBP1/RotNet-
master - Copy/'
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

model.compile(loss='categorical_crossentropy',optimizer=SGD(lr=0.01, momentum
=0.9),metrics=[angle_error])
batch_size = 64
nb_epoch = 50
monitor = 'val_angle_error'
checkpointer = ModelCheckpoint(
    filepath=os.path.join(output_folder, model_name + '.hdf5'),
    monitor=monitor,
    save_best_only=True
)
reduce_lr = ReduceLROnPlateau(monitor=monitor, patience=3)
early_stopping = EarlyStopping(monitor=monitor, patience=5)
tensorboard = TensorBoard()
model.fit_generator(
    ImageDataGenerator(train_filenames,input_shape=input_shape,batch_size=bat
ch_size,
        preprocess_func=preprocess_input,crop_center=True,crop_largest_rect=T
rue,shuffle=True),
    steps_per_epoch=len(train_filenames) / batch_size,
    epochs=nb_epoch,
    validation_data=ImageDataGenerator(test_filenames,input_shape=input_shape
,batch_size=batch_size,
        preprocess_func=preprocess_input,crop_center=True,crop_largest_rect=T
rue),
    validation_steps=len(test_filenames) / batch_size,
    callbacks=[checkpointer, reduce_lr, early_stopping, tensorboard],
    workers=10
)
```

#Base paper 2 Implementation

```
# importing opencv
import cv2
import numpy as np
from numpy import linalg as LA
from sympy import Matrix, pretty
import math
from matplotlib import pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.inception_v3 import preprocess_input
from keras.utils.data_utils import GeneratorEnqueuer
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import math, os
from imageai.Detection import ObjectDetection
from keras.applications.resnet50 import ResNet50
from keras.applications.resnet50 import preprocess_input
from keras.applications.imagenet_utils import decode_predictions


def rotate(image, angle, center = None, scale = 1.0):
      """
    Rotates the image by the specified angle
    """
     (h, w) = image.shape[:2]

    if center is None:
        center = (w / 2, h / 2)

    # Perform the rotation
    M = cv2.getRotationMatrix2D(center, angle, scale)
    rotated = cv2.warpAffine(image, M, (w, h),borderValue=(0,128,0))

    return rotated

def covarince(x,y):
    """
    Calculation of covariance for the given two lists
    """

    ag=0
    for i in range(len(x)):
        ag+=x[i]*y[i]
    ag/=(len(x)-1)
    return ag

def CorrectionMain(inputImg):
    """
    Reads the test image from the specified path,rotates it at random angle
and produces the correction angle.
    """

    image = cv2.imread('/content/drive/MyDrive/MajorProjectBP1/Dataset/Good D
og Images/DogImageForEigenVector.jpg')
    image=inputImg
    image=rotate(image,60)
    gi = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    ret, bi = cv2.threshold(gi, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    mat=[]
    x=[]
    y=[]
    for i in range(len(bi)):
        for j in range(len(bi[0])):
            if(bi[i][j]==255):
                x.append(i)
                y.append(j)
                mat.append([i,j])
    xmean=0
    ymean=0
    xmax=-1
    ymax=-1
    xmin=9999999999999
    ymin=9999999999999

    for row in mat:
        xmean+=row[0]
        xmax=max(xmax,row[0])
        xmin=min(xmin,row[0])
        ymean+=row[1]
        ymax=max(ymax,row[1])
        ymin=min(ymin,row[1])


    xmean=xmean/len(mat)
    ymean=ymean/len(mat)

    mat2=[]
    xut=[]
    yut=[]
    for row in mat:
        mat2.append([(row[0]-xmean)/(xmax-xmin),(row[1]-ymean)/(ymax-ymin)])
        xut.append((row[0]-xmean)/(xmax-xmin))
        yut.append((row[1]-ymean)/(ymax-ymin))

    covxy=covarince(xut,yut)
    covxx=covarince(xut,xut)
    covyy=covarince(yut,yut)

    covar=[[covxx,covxy],[covxy,covyy]]
    w, v = LA.eig(covar)

    CorrectionAngle=math.atan(v[0][0]/v[1][0])*57.2958

    alpha = math.atan(v[0][0]/v[1][0])*57.295
    beta = abs(90 + alpha)
    image1=rotate(image,alpha)
    image2=rotate(image,180+alpha)
    image3=rotate(image,beta)
    image4=rotate(image,180+beta)
    return CorrectionAngle

def Main(InputImage):
    image = cv2.imread('/content/drive/MyDrive/MajorProjectBP1/Dataset/Good D
og Images/DogImageForEigenVector.jpg')
    image=cv2.imread(InputImage)
    originalImg=image.copy()
    image=rotate(image, np.random.randint(90,270))
    rotatedImage=image.copy()
    CorrectionAngle=CorrectionMain(image)
    alpha=CorrectionAngle
    beta=abs(90+alpha)
    image1=rotate(image,alpha)
    image2=rotate(image,180+alpha)
    image3=rotate(image,beta)
    image4=rotate(image,180+beta)

    cv2.imwrite("/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image1.jpeg", image1)
    cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image2.jpeg', image2)
    cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image3.jpeg', image3)
    cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image4.jpeg', image4)

    detector = ObjectDetection()
    detector.setModelTypeAsRetinaNet()
    detector.setModelPath('/content/drive/MyDrive/MajorProjectBP1/resnet50_co
co_best_v2.1.0.h5')
    detector.loadModel()
    min_prob=10
    detections=[]
    detections.append(detector.detectObjectsFromImage(input_image="/content/d
rive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image1.jpeg", output_i
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage5.jpg", minimum_percentage_probability=min_prob))
    detections.append(detector.detectObjectsFromImage(input_image="/content/d
rive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image2.jpeg", output_i
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage6.jpg", minimum_percentage_probability=min_prob))
```

```
    detections.append(detector.detectObjectsFromImage(input_image="/content/d
rive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image3.jpeg", output_i
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage7.jpg", minimum_percentage_probability=min_prob))
    detections.append(detector.detectObjectsFromImage(input_image="/content/d
rive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image4.jpeg", output_i
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage8.jpg", minimum_percentage_probability=min_prob))

    image5 = cv2.imread('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampl
etestoutputs/image5.jpg')
    image6 = cv2.imread('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampl
etestoutputs/image6.jpg')
    image7 = cv2.imread('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampl
etestoutputs/image7.jpg')
    image8 = cv2.imread('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampl
etestoutputs/image8.jpg')
    countDict=dict()
    for detection in detections:
        for object in detection:
            if object.get('name') not in countDict.keys():
                countDict[object.get('name')]=1
            else:
                countDict[object.get('name')]+=1
    # print(countDict)
    MostFreqClass=max(countDict,key=countDict.get)
    # print(MostFreqClass)
    maxFreqClassImg=0
    maxProb=0
    i=1
    # print(MostFreqClass)
    for detection in detections:
        for object in detection:
            if object.get('name')==MostFreqClass:
                temp=object.get('percentage_probability')
                if temp!=None and temp>maxProb:
                    maxProb=temp
                    maxFreqClassImg=i
        i+=1

    finalCorrectedImage= cv2.imread('/content/drive/MyDrive/MajorProjectBP1/D
ataset/sampletestoutputs/image'+str(maxFreqClassImg)+'.jpeg')
    cv2.imwrite("/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image9.jpeg",originalImg)
    OriginalDetection=detector.detectObjectsFromImage(input_image="/content/d
rive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image9.jpeg", output_i
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage11.jpg", minimum_percentage_probability=min_prob)
    maxOriginalProb=0
    for object in OriginalDetection:
        if object.get('name')==MostFreqClass:
            temp=object.get('percentage_probability')
            if temp!=None and temp>maxOriginalProb:
                maxOriginalProb=temp
    cv2.imwrite("/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image10.jpeg",rotatedImage)
    RotatedDetection=detector.detectObjectsFromImage(input_image="/content/dr
ive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image10.jpeg", output_i
```

```
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage12.jpg", minimum_percentage_probability=min_prob)
    maxRotatedProb=0
    for object in RotatedDetection:
        if object.get('name')==MostFreqClass:
            temp=object.get('percentage_probability')
            if temp!=None and temp>maxRotatedProb:
                maxRotatedProb=temp
    return maxProb,maxOriginalProb,maxRotatedProb
    plt.imshow(finalCorrectedImage)




def test_eigenvectorsmodel(InputImage):
    """
    Given an input image,rotate it by random angle followed by orientation
correction, identify the most frequent class and output the accuracy of the
model, along with accuracies for original image and rotated image
    """

    image=cv2.imread(InputImage)
    originalImg=image.copy()
    rangle=np.random.randint(90,270)
    image=rotate(image, rangle)
    rotatedImage=image.copy()
    CorrectionAngle=CorrectionMain(image)
    alpha=CorrectionAngle
    beta=abs(90+alpha)
    image1=rotate(image,alpha)
    image2=rotate(image,180+alpha)
    image3=rotate(image,beta)
    image4=rotate(image,180+beta)

    cv2.imwrite("/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image1.jpeg", image1)
    cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image2.jpeg', image2)
    cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image3.jpeg', image3)
    cv2.imwrite('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image4.jpeg', image4)

    detector = ObjectDetection()
    detector.setModelTypeAsRetinaNet()
    detector.setModelPath('/content/drive/MyDrive/MajorProjectBP1/resnet50_co
co_best_v2.1.0.h5')
    detector.loadModel()
    min_prob=30
    detections=[]
    detections.append(detector.detectObjectsFromImage(input_image="/content/d
rive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image1.jpeg", output_i
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage5.jpg", minimum_percentage_probability=min_prob))
    detections.append(detector.detectObjectsFromImage(input_image="/content/d
rive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image2.jpeg", output_i
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage6.jpg", minimum_percentage_probability=min_prob))
    detections.append(detector.detectObjectsFromImage(input_image="/content/d
rive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image3.jpeg", output_i
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage7.jpg", minimum_percentage_probability=min_prob))
    detections.append(detector.detectObjectsFromImage(input_image="/content/d
rive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image4.jpeg", output_i
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage8.jpg", minimum_percentage_probability=min_prob))
    image5 = cv2.imread('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampl
etestoutputs/image5.jpg')
    image6 = cv2.imread('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampl
etestoutputs/image6.jpg')
    image7 = cv2.imread('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampl
etestoutputs/image7.jpg')
    image8 = cv2.imread('/content/drive/MyDrive/MajorProjectBP1/Dataset/sampl
etestoutputs/image8.jpg')
    countDict=dict()
    for detection in detections:
        for object in detection:
            if object.get('name') not in countDict.keys():
                countDict[object.get('name')]=1
            else:
                countDict[object.get('name')]+=1
    MostFreqClass=max(countDict,key=countDict.get)
    maxFreqClassImg=0
    maxProb=0
    i=1
    for detection in detections:
        for object in detection:
            if object.get('name')==MostFreqClass:
                temp=object.get('percentage_probability')
                if temp!=None and temp>maxProb:
                    maxProb=temp
                    maxFreqClassImg=i
        i+=1
    finalCorrectedImage= cv2.imread('/content/drive/MyDrive/MajorProjectBP1/D
ataset/sampletestoutputs/image'+str(maxFreqClassImg)+'.jpeg')
    cv2.imwrite("/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image9.jpeg",originalImg)
    OriginalDetection=detector.detectObjectsFromImage(input_image="/content/d
rive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image9.jpeg", output_i
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage11.jpg", minimum_percentage_probability=min_prob)
    maxOriginalProb=0
    for object in OriginalDetection:
        if object.get('name')==MostFreqClass:
            temp=object.get('percentage_probability')
            if temp!=None and temp>maxOriginalProb:
                maxOriginalProb=temp
    cv2.imwrite("/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestout
puts/image10.jpeg",rotatedImage)
    RotatedDetection=detector.detectObjectsFromImage(input_image="/content/dr
ive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/image10.jpeg", output_i
mage_path="/content/drive/MyDrive/MajorProjectBP1/Dataset/sampletestoutputs/i
mage12.jpg", minimum_percentage_probability=min_prob)
    maxRotatedProb=0
```

```
for object in RotatedDetection:
    if object.get('name')==MostFreqClass:
        temp=object.get('percentage_probability')
        if temp!=None and temp>maxRotatedProb:
            maxRotatedProb=temp

num_images=1
plt.figure(figsize=(10.0, 2 * num_images))

title_fontdict = {
    'fontsize': 14,
    'fontweight': 'bold'
}
options = {}
fig_number = 1
ax = plt.subplot(num_images, 3, fig_number)
if fig_number == 1:
    plt.title('Original\n', fontdict=title_fontdict)
plt.imshow(np.squeeze(originalImg).astype('uint8'), **options)
plt.axis('off')

fig_number += 1
ax = plt.subplot(num_images, 3, fig_number)
if fig_number == 2:
    plt.title('Rotated\n', fontdict=title_fontdict)
ax.text(
    0.5, 1.03, 'Angle: {0}'.format(rangle),
    horizontalalignment='center',
    transform=ax.transAxes,
    fontsize=11
)
plt.imshow(np.squeeze(rotatedImage).astype('uint8'), **options)
plt.axis('off')

fig_number += 1
ax = plt.subplot(num_images, 3, fig_number)
if fig_number == 3:
    plt.title('Corrected\n', fontdict=title_fontdict)
ax.text(
    0.5, 1.03, '',
    horizontalalignment='center',
    transform=ax.transAxes,
    fontsize=11
)
plt.imshow(np.squeeze(finalCorrectedImage).astype('uint8'), **options)
plt.axis('off')

plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

return maxProb,maxOriginalProb,maxRotatedProb
```