Implementation and Comparison of various variants of CNN on Diabetic Retinopathy Dataset

Report submitted in fulfillment of the requirements
for the Undergraduate Project of

Third Year B.Tech.

by

Ankan Bohara and Siddharth Sahay

Under the guidance of

Prof. K.K. Shukla

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI
Varanasi 221005, India
May 2019

Dedicated to

Our parents, teachers and the Almighty

Declaration

We certify that

1. The work contained in this report is original and has been done by myself and the general supervision of my supervisor.

2. The work has not been submitted for any project.

3. Whenever we have used materials (data, theoretical analysis, results) from other sources, we have given due credit to them by citing them in the text of the thesis and giving their details in the references.

4. Whenever we have quoted written materials from other sources, we have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT (BHU) Varanasi
Date: May,2019

Ankan Bohara and Siddharth Sahay
Junior Undergraduate
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Certificate

*This is to certify that the work contained in this report entitled "**Implementation and Comparison of various variants of CNN on Diabetic Retinopathy Dataset**" being submitted by **Ankan Bohara and Siddharth Sahay (Roll No. 16075060 and 16074016)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, is a bona fide work of our supervision.*

Place: IIT (BHU) Varanasi
Date: May, 2019

Prof. K.K Shukla
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Acknowledgments

We would like to express our sincere gratitude to our supervisor, Prof. K.K Shukla, for guiding us throughout the project.

Place: IIT (BHU) Varanasi

Date: May,2019 **Ankan Bohara and Siddharth Sahay**

Abstract

Diabetic retinopathy is a complication of diabetes, caused by high blood sugar levels damaging the back of the eye (retina). It is the leading cause of blindness in the working-age population of the developed world, estimated to affect over 93 million people.

In this paper we aim to design a computer-based system for detection and classification of non-profilerative diabetic retinopathy. The main goal is to automatically classify the grade of diabetic retinopathy given any retina image. We have used a convolutional neural network(CNN) for this task. To reduce sensitivity of the model to noise and increase robustness, we have implemented custom loss functions such as the correntropy loss and homotopy loss.

Keywords: Diabetic retinopathy, Convolutional neural networks(CNN), Loss functions, correntropy loss, homotopy loss.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Diabetic retinopathy is extreme and broadly spread eye disease. It is the commonest reason for legitimate eye disease in the working-age population of developed nations.

Diabetic retinopathy happens when diabetes harms the blood vessels inside the retina, spilling blood and liquids into the encompassing tissue. This fluid leakage produces microaneurysms, hemorrhages, hard exudates, and soft exudates. Diabetic retinopathy is difficult to detect and may only be recognized when the retina is destroyed to a level where its ailment becomes even incomprehensible.

1.2 Motivation of the Research Work

The expanding number of diabetic retinopathy cases world-wide requires to heighten endeavors in creating tools to help in the determination of diabetic retinopathy. Thus requiring an automated system to identify the risk earliest.

Current state of the art techniques are not robust to noises so the main motivation of this project is to come up with a robust loss functions that are susceptible to noises.

1.3 Our Work

In the due course of this project we first implemented loss functions like homotopy loss and correntropy loss and tested these on MNIST dataset.

Several variations include addition of noises in the original dataset as well as flipping of original labels are done to verify the robustness of the classifiers on the MNIST dataset.

Then we have tried various CNN architectures :

- VGG16
- ResNet50
- AlexNet
- InceptionResNetV2
- Xception

On top of them we have built custom layers. We have tested these architectures on the original diabetic retinopathy dataset consisting of 35,126 images. We have used homotopy loss function and categorical cross entropy loss function on these architectures.

Chapter 2

Convolutional Neural Networks

2.1 Introduction

CNNs are a category of Neural Networks that have been proven to be fairly successful in various Computer Vision tasks such as Object Recognition, Face Recognition, Robotic Vision, Self-Driving Cars etc.. These networks in-fact trace their inspiration from the Biological Visual Cortex which has certain cell-regions that are sensitive to specific regions of the visual field.

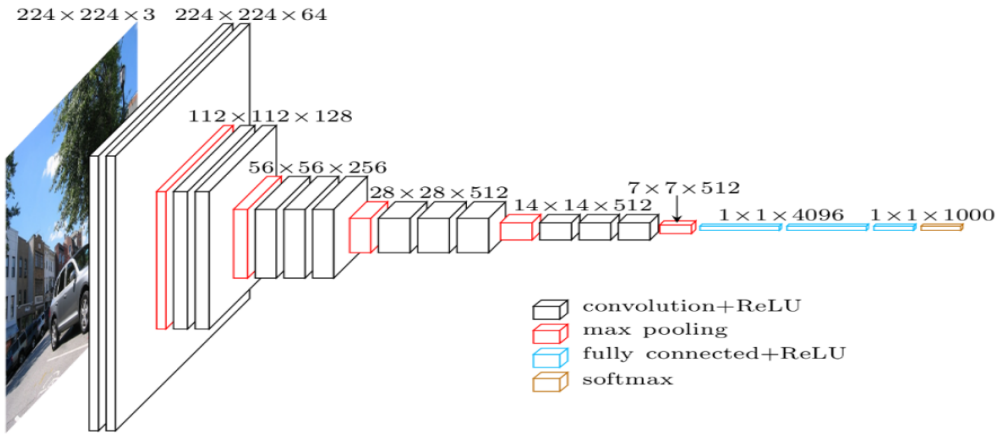

Figure 2.1: CNN Architecture[1]

There have been researches in the effective use of ConvNets in other areas as well(such as NLP sentence classification), but well focus our attention on CNNs used

on images, or on pixels to be precise. A typical Convnet architecture has a series of Convolutional, Non-Linear, Pooling, and FC layers through which an image is passed and output is obtained.

2.2 CNN Models

In this project, we have implemented several state-of-the-art CNN models and compared their metrics, training times and parameters. In each of the following models, we have added **two dense layers** and **two dropout layers** followed by the **softmax layer**. The dense layers help in reducing the dimensions of features closer to the number of classes for prediction, while the dropout layers help in reducing the problem of over-fitting.

The various architectures are described in the following sections.

2.2.1 AlexNet

AlexNet[10] contains eight layers with weights; the first five are convolutional and the remaining three are fully-connected. Output is a softmax layer on number of classes. The input to the net is a 227 * 227 * 3 (RGB image). Total number of training parameters is 62378344.

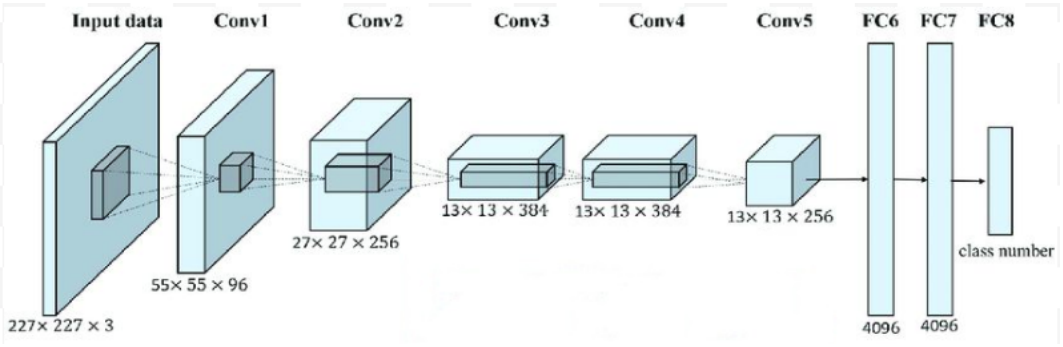

Figure 2.2: AlexNet Architecture[2]

2.2.2 VGG-16

VGG-16[11] is a CNN model proposed by the Visual Geometry Group consisting of a total of 16 Convolution layers+maxpooling layers. It takes input image of size 224 * 224 * 3 (RGB image), and outputs one of the classes as prediction. The filters used in Convolutional layers are of 3 * 3 size, while pooling filters are 2 * 2 size.

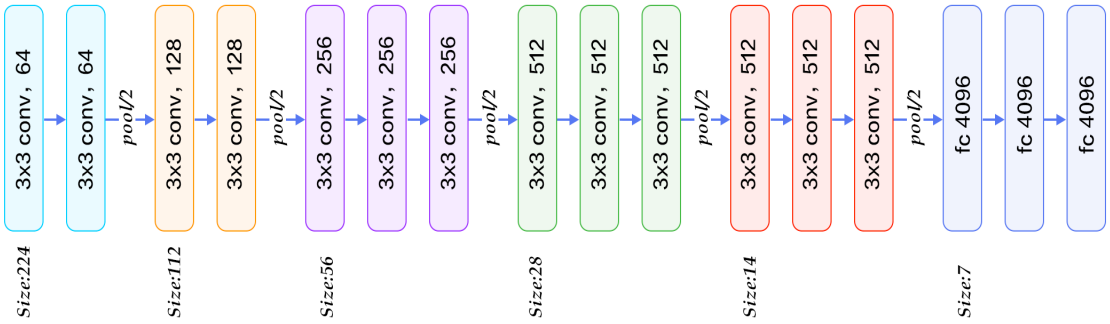

Figure 2.3: VGG16 Architecture[3]

2.2.3 ResNet50

ResNet50[12] is a 50 layer Residual Convoltional Network. Residual can be simply understood as subtraction of feature learned from input of that layer. It also takes input image of default size 224 * 224 * 3 (RGB image). It is a very large scale network consisting of around 25.6 million trainable parameters.

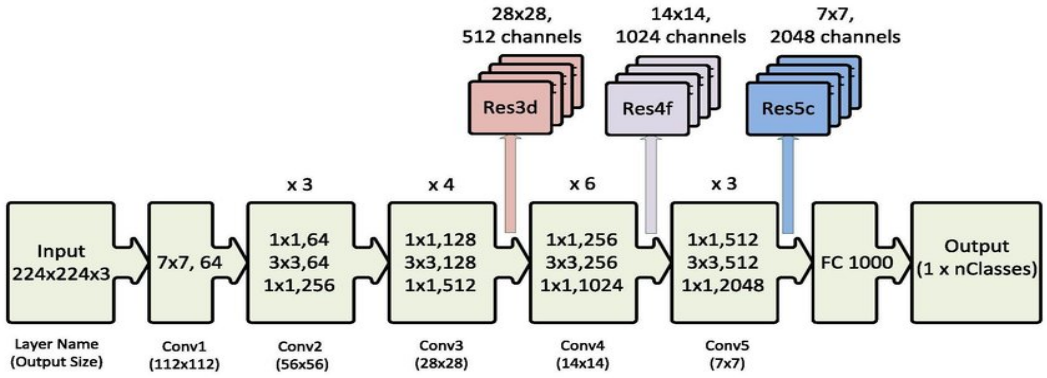

Figure 2.4: ResNet50 Architecture[4]

2.2.4 Inceptionv3

Inceptionv3[13] model is made up of both symmetric and asymmetric building blocks consisting of Convolutions, pooling, dropout, dense layers etc. Batchnorm is used extensively throughout the model and applied to activation inputs. Loss is computed via Softmax. Batch normalization reduces the amount by what the hidden unit values shift around (covariance shift). The default input size is 299 * 299 * 3 (RGB image).

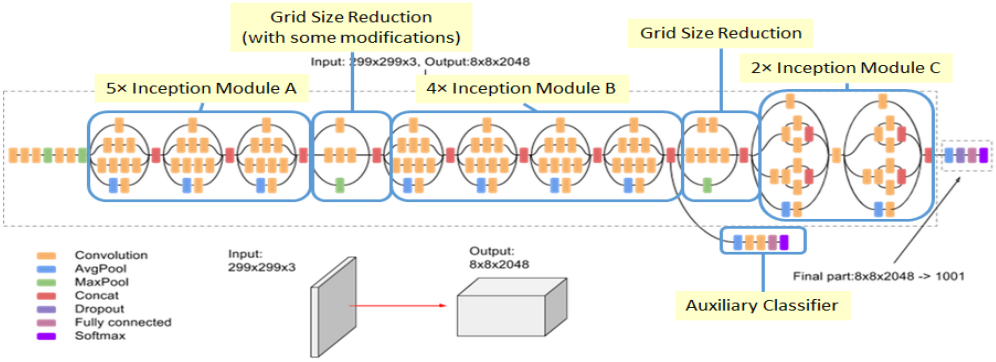

Figure 2.5: Inception Architecture[5]

2.2.5 Xception

Xception[14] is an extension of the Inception architecture which replaces the standard Inception modules with depthwise separable convolutions, consisting of

Depthwise convolution, i.e. a spatial convolution performed independently over each channel of an input, and

Pointwise convolution, i.e. a 1x1 convolution, projecting the channels output by the depthwise convolution onto a new channel space.

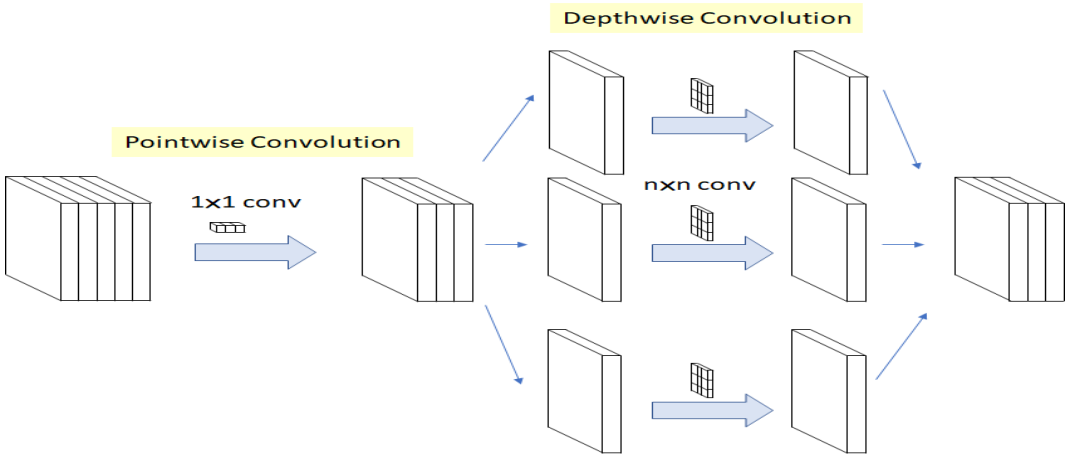

Figure 2.6: Xception Architecture[6]

2.2.6 InceptionResnetv2

In the InceptionResnetv2[15] model, multiple inception layers/modules are combined via residual connections. This reduces degradation due to deep structures and improves training time.

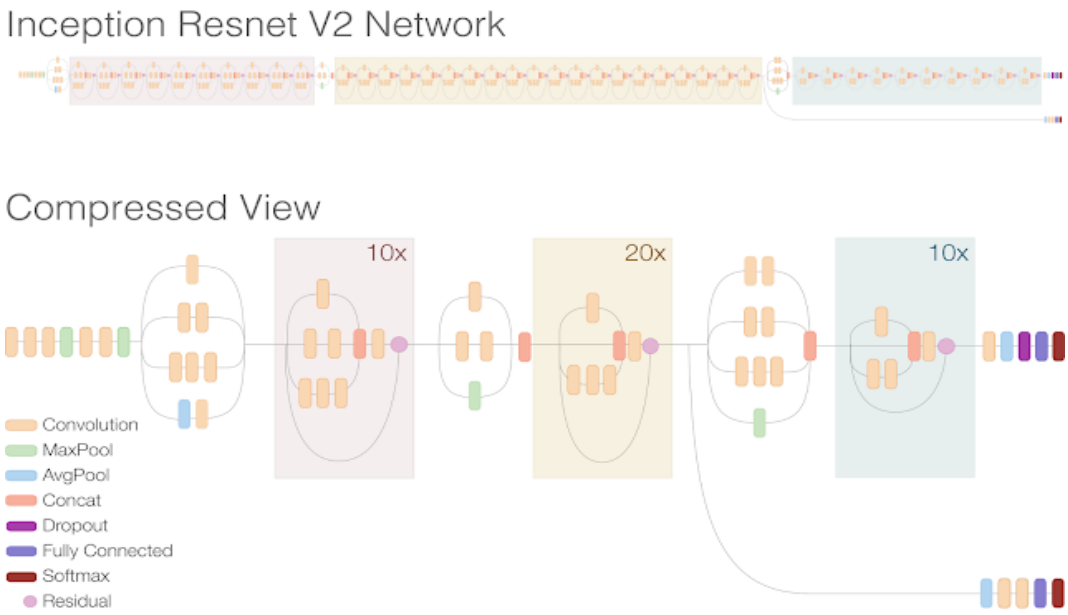

Figure 2.7: InceptionResNetv2 Architecture[7]

Chapter 3

Loss Functions

3.1 Introduction

In machine learning systems machine learns using a loss function. It is a measurement of how well a specific algorithm models the data. Gradually, with the help of some optimization function, loss function learns to reduce the error in prediction.

There are various factors involved in choosing a loss function for a specific problem such as type of machine learning algorithm chosen, ease of calculating the derivatives and to some degree the percentage of outliers in the data set

Broadly loss functions fall in two categories:

- Regression losses
- Classification losses

We need classification loss in our problem. There are various classification losses available. We will discuss:

- Categorical Crossentropy Loss
- Homotopy Loss
- Correntropy Loss

3.2 Homotopy Loss

A function $\theta(x)$ is called proper as a loss function if $\theta(x)$ satisfies:

- $\theta(x)$ is a non decreasing function of $|x|$
- $\theta(0) = 0$
- $\theta(x)$ is increasing for $x>0$ and $\theta(x)<\theta(\infty)$

Homotopy loss function[8] satisfying all the above conditions:

$$\rho(x;\alpha,c) = \frac{1}{\alpha}((\frac{|x|}{c}+1)^{\alpha}-1)$$

where c is a non homogeneous parameter and α is a homotopy parameter. In classification settings x represents :

$$x = y - \hat{y}$$

where y and $\hat{y}$ represents actual label and predicted label by the classifier respectively.

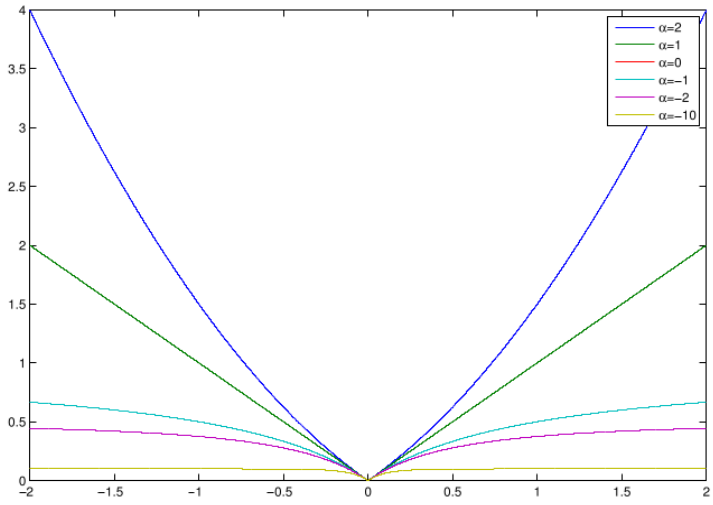

Figure 3.1: Homotopy loss $\rho(x;\alpha,c)$ with $\alpha = 2, 1, 0, -1, -2, -10$ where horizontal ordinate is x[8]

3.3 Correntropy Loss

The most popular kernel in Correntropy is the Gaussian kernel, given by

$$\kappa_\sigma(s_i, t_i) = \exp(-\frac{||s_i - t_i||^2}{2\sigma^2})$$

where $||\ .\ ||$ is Euclidean norm and s_i *and* t_i are samples drawn from the population. For classification task whose goal is to maximize similarity between the classifier output $\mathbf{S}$ and the label $\mathbf{T}$, CLF(Crossentropy Induced Loss Function) was defined as follows[9]:

CLF($\mathbf{S}$,$\mathbf{T}$) $= \beta[1 - E(\kappa_\sigma(\mathbf{S}, \mathbf{T}))]$ where $\beta = [1 - \exp(-\frac{1}{2\sigma^2})]^{-1}$. Similarly the empirical CLF between the classifier output samples $\mathbf{S}$ and the label samples $\mathbf{T}$ can be calculated as:

$$CLF(S, T) = \beta[1 - \frac{1}{N}\sum_{i=1}^{N}\kappa_\sigma(s_i, t_i)]$$

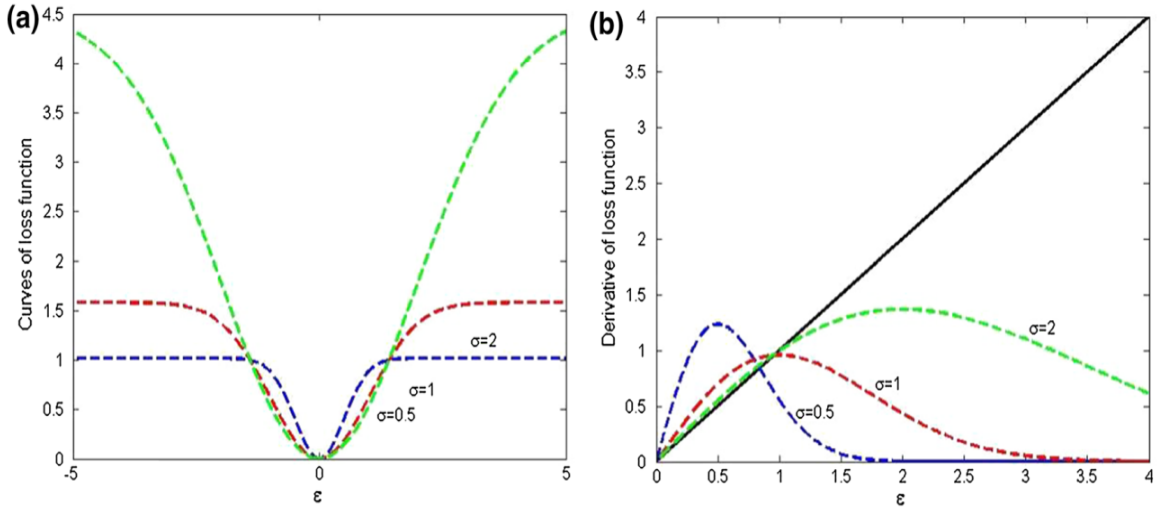

Figure 3.2: (a) The loss function values of CLF with different kernel size σ as the error ϵ increasing[9] (b) The derivatives of CLF with different kernel size σ[9]

3.4 Categorical Crossentropy

Cross entropy indicates the distance between the predicted distribution by the model and the actual distribution.

$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{N} y_i \log_e \hat{y}_i$$

where y_i, $\hat{y}_i$ and N represents the actual label, the predicted value of the current model and the number of training examples respectively. Here y_i is the corresponding class label not one hot encoded vector.

Chapter 4

Dataset

The dataset used in the project is Kaggle's Diabetic Retinopathy Dataset[16] which consists of a large set of high-resolution retina images taken under a variety of imaging conditions. Both left and right eye images are provided for each subject along with the grade of diabetic retinopathy in a scale of 0-4 as follows:

Table 4.1: Dataset Description

Class	**Description**	**Distribution of images**
0	No DR	25810
1	Mild	2443
2	Moderate	5292
3	Severe	873
4	Proliferative DR	708
Total		35126

Each image is of extremely high resolution (on average 2000 * 3000 * 3 pixels). The dataset comprises of 35,126 training samples. As a result, the total size of the Retinopathy Dataset is 33GB.

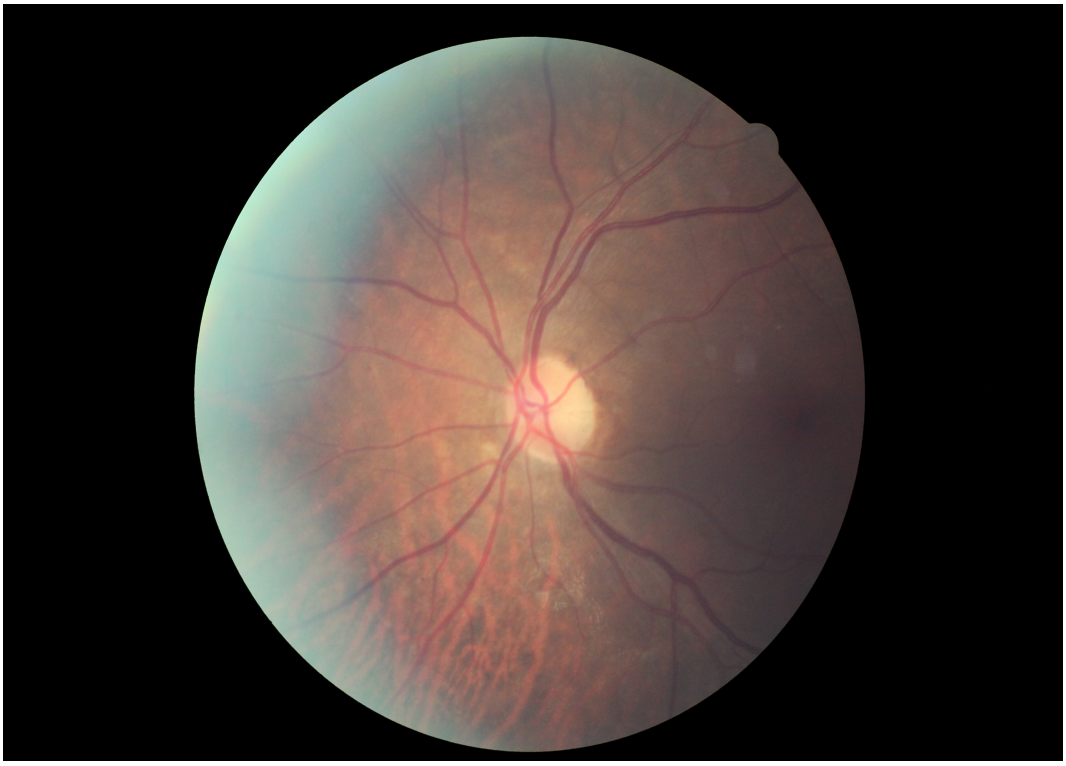

Figure 4.1: Grade 0 Diabetic Retinopathy (Non-Diabetic)

Figure 4.2: Grade 4 Diabetic Retinopathy (Highly Diabetic)

Chapter 5

Implementation Details and Our Work

This section will highlight the libraries used and the implementation details. The code is written in Python language.

5.1 Libraries Used

These are some of the major libraries used in the project

- keras : for training the model
- numpy : for storing the train and test data
- pandas : for reading the train and test data
- matplotlib : for making plots
- pickle : for saving the model
- opencv : for image loading and preprocessing

5.2 Data Structures

This section will describe the data structures used throughout the project.

- **ImageNameDataHash** :

  Dictionary to store image file names with corresponding image arrays. All images are preprocessed and added to the data structure.

- **uniquePatientIDList** :

  List of all unique patients with both retinas having same level of Diabetic Retinopathy according to training labels.

5.3 Description of Functions

5.3.1 Homotopy Loss Function

```
def homotopy_coeff(y_true, y_pred,alpha=2,c=1):
    loss=(K.pow((1+abs(y_true-y_pred)/c),alpha)-1)
    loss=loss/alpha
    return loss

def homotopy_loss(alpha=2,c=1):
    def homotopy(y_true,y_pred):
        return homotopy_coeff(y_true,y_pred,alpha,c)
    return homotopy
```

This is the implementation of homotopy loss function defined by :

$$\rho(x;\alpha,c) = \frac{1}{\alpha}((\frac{|x|}{c}+1)^{\alpha}-1)$$

and values of default parameters are chosen as mentioned in [8]

5.3.2 Correntropy Loss Function

```
def correntropy_coeff(y_true, y_pred,sigma=2.2):
    beta=1-K.exp(-1/(2*sigma*sigma))
    beta=1/beta
    loss=-K.sum(K.square(y_true - y_pred))/(2*sigma*sigma)
    loss= K.exp(loss)
    loss=(1-K.mean(loss))
    loss = beta*(loss)
    return loss
```

```
def correntropy_loss(sigma=2.2):
    def correntropy(y_true,y_pred):
        return correntropy_coeff(y_true,y_pred,sigma)
    return correntropy
```

This is the implementation of correntropy loss function defined by : CLF($\mathbf{S}$,$\mathbf{T}$) = $\beta[1 - E(\kappa_\sigma(\mathbf{S}, \mathbf{T}))]$ where $\beta = [1 - \exp(-\frac{1}{2\sigma^2})]^{-1}$. and values of default parameters are chosen as mentioned in [9]

5.3.3 Reading Training Data

This function will read the training data from the directory given as the argument in the function and also checking whether images are of correct dimension or not. Plus it is storing the global hash of image names with the image in **ImageNameDataHash** dictionary.

```
def readTrainData(trainDir):
    global ImageNameDataHash
    images = os.listdir(trainDir)
    for imageFileName in images:
        # load the image, pre-process it, and store it in the data list
        imageFullPath = os.path.join(trainDir, imageFileName)
        img = load_img(imageFullPath)
        arr = img_to_array(img)
        arr = cv2.resize(arr, (HEIGHT,WIDTH)) #downsample image
        arr = np.array(arr, dtype="float") / 255.0
        imageFileName = imageFileName.replace('.jpeg','')
        ImageNameDataHash[str(imageFileName)] = np.array(arr)
    return
```

5.3.4 Training Model

Various CNN models are combined with above described loss functions and trained on the dataset. Test Accuracies and plots are saved. Adam optimizer is used for learning. Evaluation metric is accuracy.

```
def createModel():
    # returns our fully constructed deep learning + Keras image classifier
    base_model = InceptionResNetV2(weights='imagenet', input_shape=inputShape)
    # Add new top layers
```

```
    x = base_model.output
    x = Flatten()(x)
    x = Dropout(0.2)(x)
    x = Dense(32, activation='relu')(x)
    x = Dense(16, activation='relu')(x)
    predictions = Dense(NUM_CLASSES, activation='softmax')(x)

    # This is the model we will train
    model = Model(inputs=base_model.input, outputs=predictions)
    opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
    model.compile(loss=homotopy_loss(), optimizer=opt, metrics=["accuracy"])
    return model

#train, test and save model
model = createModel()
fit = model.fit(aug.flow(Xtrain, trainY), validation_data=(Xval, valY), epochs=EPOCHS)
```

5.3.5 Results and Plots

The models implemented in the above section are saved to disk along with their training histories. These histories are then used to plot training curves for the model using matplotlib.

```
# save the model to disk
print("Saving model to disk")
with open(model_dir+"trained_"+used_model+"cc"+".json", "w") as json_file:
    json_file.write(model.to_json())
model.save_weights(model_dir+"trained_"+used_model+"cc"+".h5")
with open(model_dir+"trained_"+used_model+"cc"+"_history", 'wb') as file_pi:
    pickle.dump(H.history, file_pi)

# plot the training loss and accuracy
print("Generating plots...")
plt.figure()
N = EPOCHS
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["acc"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy on diabetic retinopathy detection")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.savefig(model_dir+"curves_plot_"+used_model+".png")
```

Chapter 6

Experiments and Results

The Homotopy and Correntropy loss functions were implemented and tested first on the MNIST dataset, and then on the Diabetic Retinopathy Dataset with several CNN model variants.

6.1 On MNIST Dataset

The MNIST dataset consists of 60,000 handwritten training sample images of 28 * 28 * 1 pixels each. The aim is to predict each digit image of 10,000 test samples into one of the 10 classes(0-9).

6.1.1 Experiments

By varying hyperparameters and adding Gaussian Noise in the input images, we have made a comparison between Homotopy and Correntropy loss functions against the Categorical Crossentropy Loss.

The CNN model used consists of 2 Convolutional Layers with Max Pooling and 2 Dense Layers. The final layer is a Softmax layer for class prediction. Each model was trained on 5 epochs with batch size of 32. The training was done on Google Colab with GPU enabled, and each epoch took around 10s.

The optimizer used in training is Adam and the evaluation metric is accuracy. The noise function used is:

$$G(x) = (1 - A) * g(0, \sigma_1^2) + A * g(0, \sigma_2^2))$$

where A is mixture coefficient and $g(0, \sigma^2)$ represents Gaussian distribution with mean 0 and variance σ^2.

6.1.2 Results

Table 6.1 contains the accuracies obtained on different loss functions. Fig 6.1 and Fig 6.2 show the learning curves of the three models with and without Gaussian noise.

It can be inferred that noise has a lesser impact while using categorical crossentropy and homotopy loss functions.

This shows the robustness of these two loss functions.

Table 6.1: Accuracies obtained on different loss functions

Loss Function	**Without Gaussian Noise**	**With Gaussian Noise**
Categorical Crossentropy	98.81	97.88
Homotopy	98.58	97.71
Correntropy	97.16	96.44

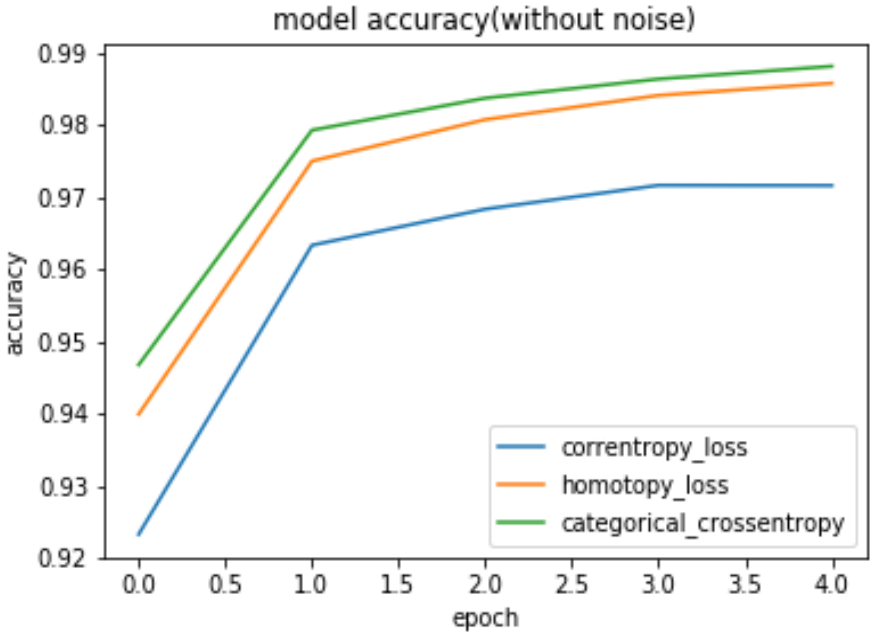

Figure 6.1: Accuracies obtained without adding noise

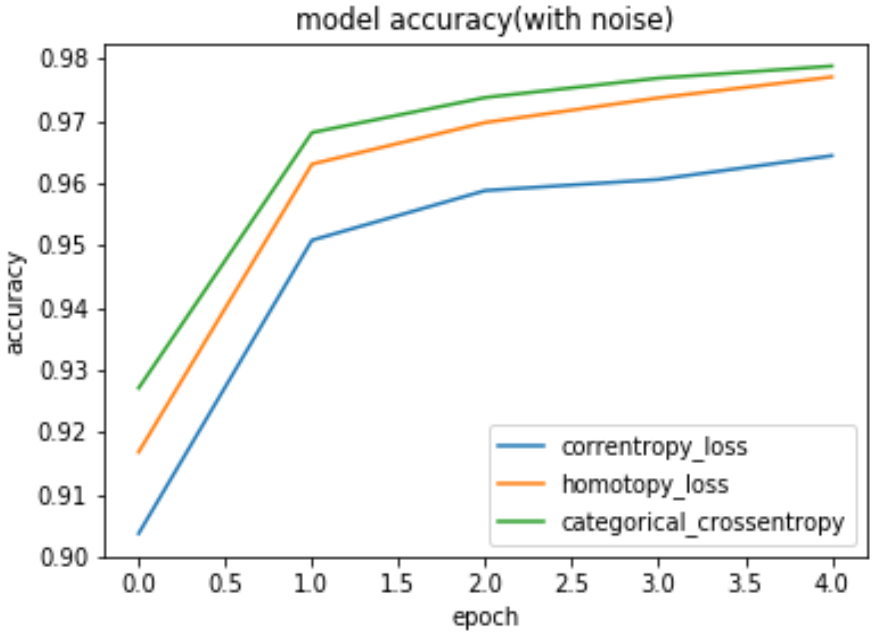

Figure 6.2: Accuracies obtained with added Gaussian Noise

6.2 On Diabetic Retinopathy Dataset

6.2.1 Experiments

Homotopy and Categorical Crossentropy Loss functions were implemented along with several state-of-the-art CNN variants, and the accuracies were visualized and compared. Data augmentation was done. Each CNN variant was trained on 3 epochs(due to time and hardware constraints) with a batch size of 32. The 35,126 dataset images were partitioned in a 75:25 Train-Test split. Each epoch took around 45 minutes on the PARAM SHIVAY supercomputer. A softmax layer is used to classify the input into one of the five classes. For homotopy loss we have chosen $\alpha = 2.0 and c = 1.0$ was chosen. Optimizer used in the training is Adam with learning rate equal to 10^{-3} and metric used for evaluation is accuracy.

6.2.2 Results

Table 6.2 shows the results obtained on various variants of CNN architecture vs. Loss Function.

Fig 6.3 shows the relative comparison of the various variants of CNN while using categorical crossentropy loss function. It can be inferred that ResNet50 has performed the best among all of these variants.

Similarly while using homotopy loss function, ResNet50 has still performed the best as compared to others, shown in Fig 6.5

Fig 6.4 and 6.6 shows the learning curves of various variants of CNN on different loss functions.

Variants	Categorical Crossentropy	Homotopy
ResNet50	73.26	73.19
VGG16	73.05	73.05
AlexNet	73.05	70.39
InceptionResNetV2	73.05	73.05
Xception	73.05	73.05

Table 6.2: Accuracies obtained on various CNN architectures using Categorical Crossentropy and Homotopy Loss

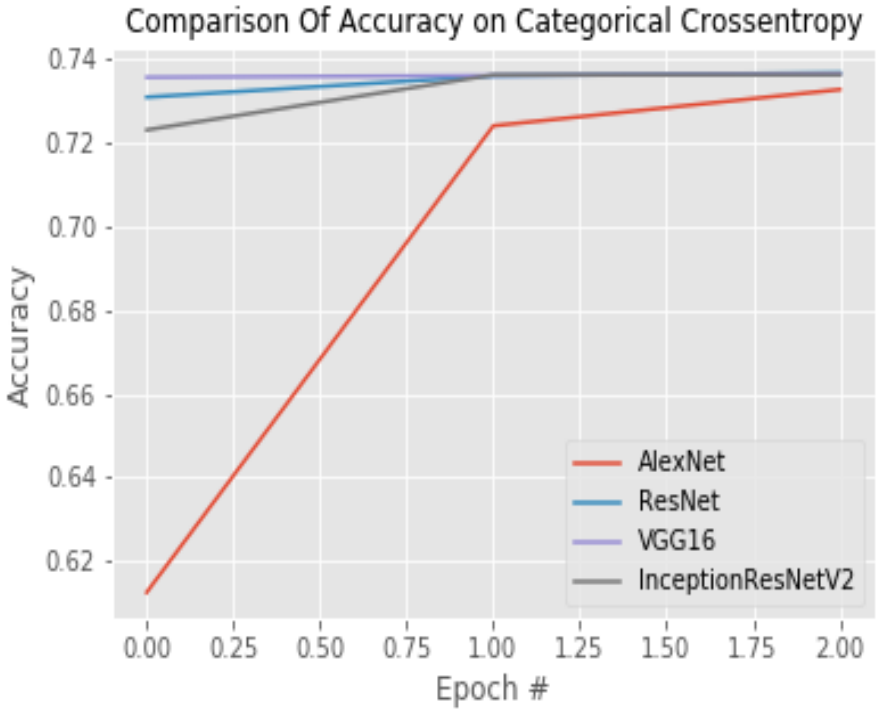

Figure 6.3: Accuracies - Categorical Crossentropy

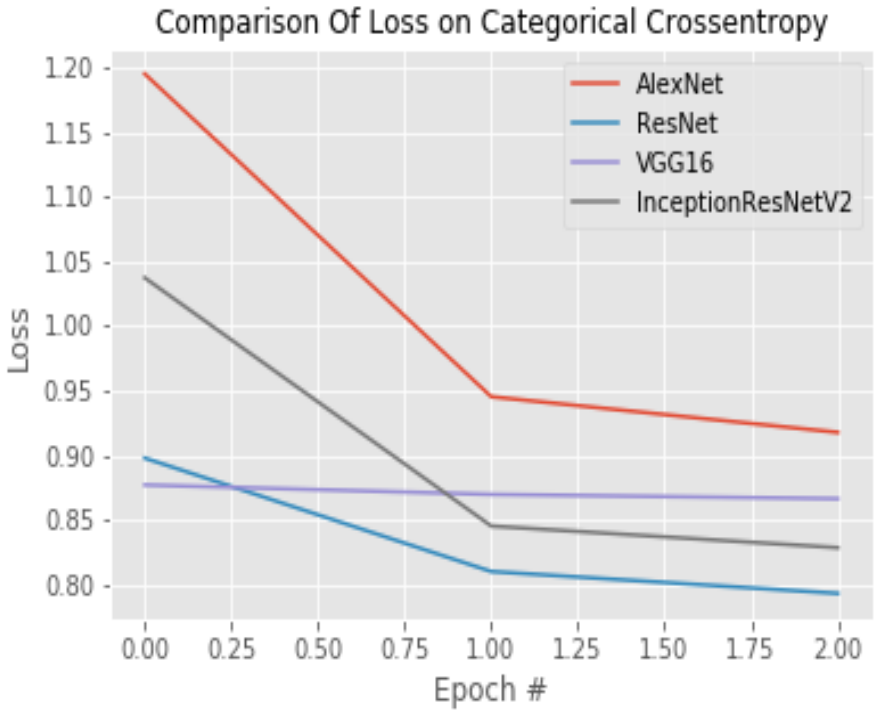

Figure 6.4: Loss - Categorical Crossentropy

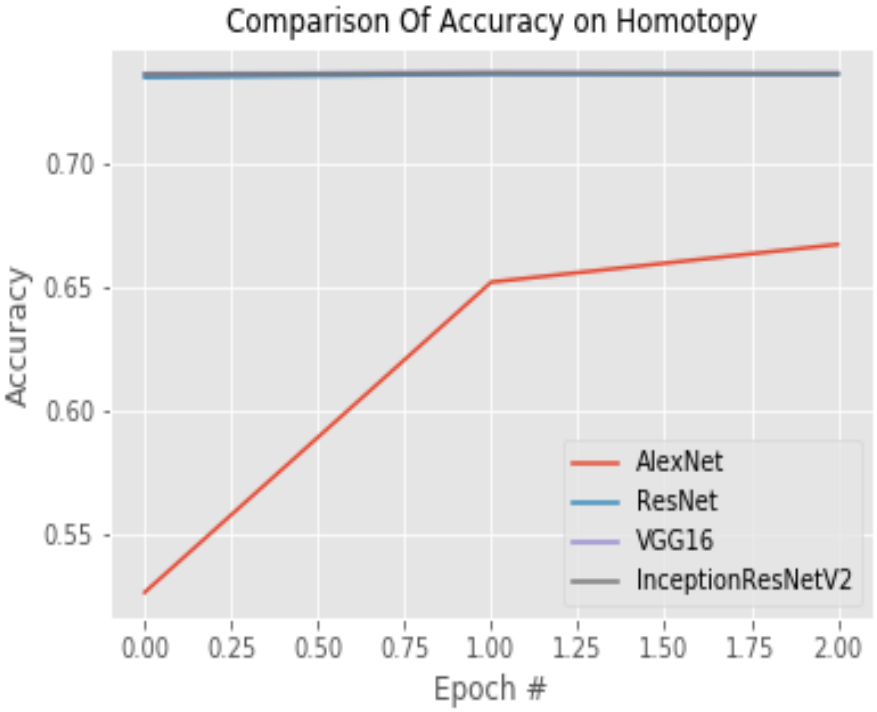

Figure 6.5: Accuracies - Homotopy Loss

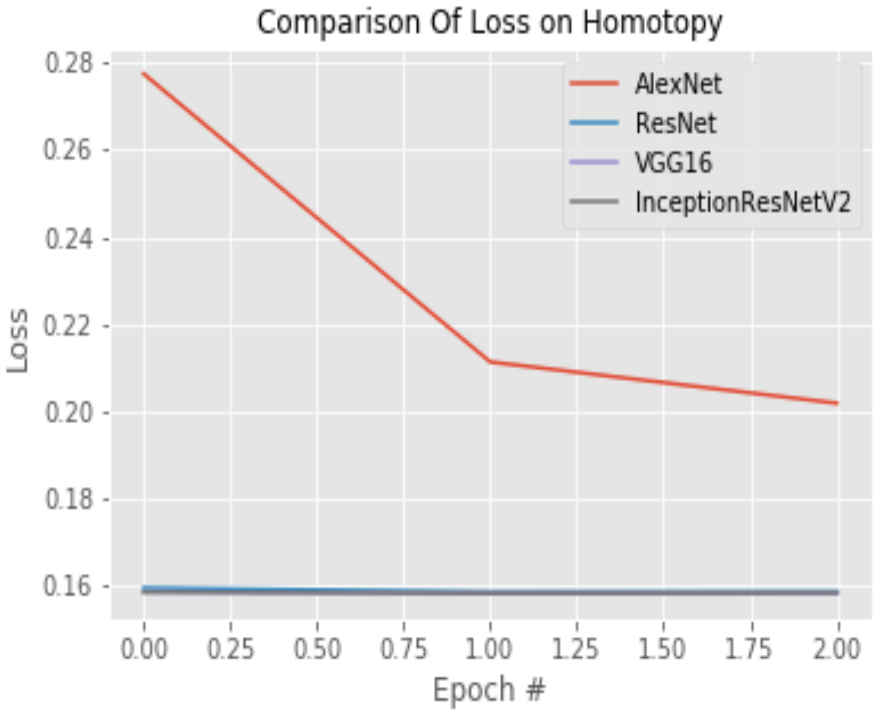

Figure 6.6: Loss - Homotopy Loss

Chapter 7

Conclusions and Discussions

7.1 Conclusions

From the results, it can be observed that homotopy performs almost similar to categorical crossentropy in both the presence and absence of noise in the dataset.

However, in the experiment with MNIST data, it is seen that correntropy is not as robust to noise as homotopy or categorical crossentropy.

The maximum accuracy obtained on MNIST dataset is 98.58% using the homotopy loss function and a simple CNN model in 5 epochs. In the Diabetic Retinopathy dataset, the maximum accuracy obtained is 73.26% by the ResNet50 model, which is quite good considering only three epochs and no data processing.

The real retinopathy images obtained are not noise-free. So the presence of noise should not affect the robustness of the model. One has to choose loss function accordingly, and in our case homotopy loss, a new loss function performs well with and without noise. Thus choice of loss function also affects the classification accuracy.

In modern-day application programming interfaces, the responsiveness of the system is also a key factor. Complex architectures take more time to load and predict as compare to simple architectures. Thus the choice of architecture is equally important while choosing the CNN architecture for the task.

7.2 Challenges and Future Work

The major challenge in using deep learning technique with this dataset is the high-resolution images which can't be fitted in the RAM. For this, we have resized it to 299*299*3 pixels so that it can fit the RAM, but it opens a wide area of work in handling such high-resolution images. Also training time for such a large scale dataset is very high.

Complex architecture performs as good as simple architecture on this dataset data augmentation and preprocessing maybe crucial steps to improve accuracy.

This opens the discussion on how to handle such high-resolution image data.

Bibliography

[1] D. Frossard, *VGG in TensorFlow*, 2016. [Online]. Available: https://www.cs.toronto.edu/ frossard/post/vgg16

[2] X. Han, Y. Zhong, L. Cao, and L. Zhang, "Pre-trained alexnet architecture with pyramid pooling and supervision for high spatial resolution remote sensing image scene classification," *Remote Sensing*, vol. 9, p. 848, 2017.

[3] Y. Nanda, *VGG neural network*, 2018. [Online]. Available: https://www.quora.com/What-is-the-VGG-neural-network

[4] A. Singh and D. Ranjan Kisku, "Detection of rare genetic diseases using facial 2d images with transfer learning," 12 2018.

[5] S.-H. Tsang, *Review: Inception-v3-1st Runner Up (Image Classification) in ILSVRC 2015*, 2018. [Online]. Available: https://medium.com/@sh.tsang/review-inception-v3-1st-runner-up-image-classification-in-ilsvrc-2015-17915421f77c

[6] ——, *Review: Xception, With Depthwise Separable Convolution, Better Than Inception-v3 (Image Classification)*, 2018. [Online]. Available: https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568

[7] A. Alemi, *Improving Inception and Image Classification in TensorFlow*, 2016. [Online]. Available: https://ai.googleblog.com/2016/08/improving-inception-and-image.html

[8] Y. Wang, L. Yang, and C. Yuan, "A robust outlier control framework for classification designed with family of homotopy loss function," *Neural Networks*, vol. 112, pp. 41 – 53, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608019300243

[9] L. Chen, H. Qu, J. Zhao, B. Chen, and J. C. Principe, "Efficient and robust deep learning with correntropy-induced loss function," *Neural Computing and Applications*, vol. 27, no. 4, pp. 1019–1031, May 2016. [Online]. Available: https://doi.org/10.1007/s00521-015-1916-x

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[11] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating Very Deep Convolutional Networks for Classification and Detection," *arXiv e-prints*, p. arXiv:1505.06798, May 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv e-prints*, p. arXiv:1512.03385, Dec 2015.

[13] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *arXiv e-prints*, p. arXiv:1512.00567, Dec 2015.

[14] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *arXiv e-prints*, p. arXiv:1610.02357, Oct 2016.

[15] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *arXiv e-prints*, p. arXiv:1602.07261, Feb 2016.

[16] kaggle, "diabetic-retinopathy-detection," https://www.kaggle.com/c/diabetic-retinopathy-detection/data, april 2015.

Appendix A

Code

```
#Model is mentioned here
used_model="InceptionV2homo"
model_dir="./InceptionV3/"
print('-'*10,"Using",used_model,'-'*10)

#Importing Libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
import random
import sys
import cv2
import pickle
import matplotlib
import time
from datetime import datetime
from matplotlib import pyplot as plt
import keras.backend as K
from subprocess import check_output
from keras.models import Model,Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Dropout, Flatten
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from keras.applications.vgg16 import VGG16
from keras.applications.vgg19 import VGG19
from keras.applications.resnet50 import ResNet50
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_resnet_v2 import InceptionResNetV2
from keras.utils import to_categorical
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras.utils import print_summary
from keras.utils import plot_model
```

```
36
37 #Setting parameters for the training
38
39 NUM_CLASSES = 5
40
41 # we need images of same size so we convert them into the size
42 WIDTH = 224
43 HEIGHT = 224
44 DEPTH = 3
45 inputShape = (HEIGHT, WIDTH, DEPTH)
46
47 # initialize number of epochs to train for, initial learning rate and batch size
48 EPOCHS = 3
49 INIT_LR = 1e-3
50 BS = 32
51 matplotlib.use('agg')
52
53 #global variables
54 ImageNameDataHash = {}
55 uniquePatientIDList = []
56
57 def readTrainData(trainDir):
58     global ImageNameDataHash
59     # loop over the input images
60     images = os.listdir(trainDir)
61     print("Number of files in " + trainDir + " is " + str(len(images)))
62     cnt=0 #count the number of images that have been loaded
63     t1=time.time()
64     for imageFileName in images:
65         if (imageFileName == "trainLabels.csv"):
66             continue
67         # load the image, pre-process it, and store it in the data list
68         imageFullPath = os.path.join(trainDir, imageFileName)
69         #print(imageFullPath)
70         img = load_img(imageFullPath)
71         arr = img_to_array(img)  # Numpy array with shape (233,233,3)
72         dim1 = arr.shape[0]
73         dim2 = arr.shape[1]
74         dim3 = arr.shape[2]
75         if (dim1 < HEIGHT or dim2 < WIDTH or dim3 < DEPTH):
76             print("Error image dimensions are less than expected "+str(arr.shape))
77         arr = cv2.resize(arr, (HEIGHT,WIDTH)) #Numpy array with shape (HEIGHT, WIDTH,3)
78         #print(arr.shape) # 128,128,3
79         dim1 = arr.shape[0]
80         dim2 = arr.shape[1]
81         dim3 = arr.shape[2]
82         if (dim1 != HEIGHT or dim2 != WIDTH or dim3 != DEPTH):
83             print("Error after resize, image dimensions are not equal to expected "+str(arr.shape))
84         #print(type(arr))
85         # scale the raw pixel intensities to the range [0, 1] - TBD TEST
86         arr = np.array(arr, dtype="float") / 255.0
87         imageFileName = imageFileName.replace('.jpeg','')
88         ImageNameDataHash[str(imageFileName)] = np.array(arr)
89         cnt+=1
```

```
        if cnt%1000==0:
            t2=time.time()
            print(cnt,"Images Loaded")
            print("Time taken for loading 1000 images is",(t2-t1))
            t1=t2
    return

print("Loading images at..."+ str(datetime.now()))
sys.stdout.flush()
t1=time.time()
readTrainData("./dataset/train_299/")
t2=time.time()
time_taken = (t2 - t1)//60
print("Time for data loading is",time_taken,"minutes")
print("Loaded " + str(len(ImageNameDataHash)) + " images at..."+ str(datetime.now())) # 1000

#csv contains image level
#10_left 0
#10_right 0
import csv
def readTrainCsv():
    raw_df = pd.read_csv('./dataset/trainLabels.csv', sep=',')
    print(type(raw_df)) #<class 'pandas.core.frame.DataFrame'>
    row_count=raw_df.shape[0] #gives number of row count row_count=35126
    col_count=raw_df.shape[1] #gives number of col count col count=2
    print("row_count="+str(row_count)+" col count="+str(col_count))
    raw_df["PatientID"] = ''
    header_list = list(raw_df.columns)
    print(header_list) # ['image', 'level', 'PatientID']
    # double check if level of left and right are same or not
    ImageLevelHash = {}
    patientIDList = []
    for index, row in raw_df.iterrows():
        # 0 is image, 1 is level, 2 is PatientID, 3 is data
        key = row[0] + ''
        patientID = row[0] + ''
        patientID = patientID.replace('_right','')
        patientID = patientID.replace('_left','')
        #print("Adding patient ID"+ patientID)
        raw_df.at[index, 'PatientID'] = patientID
        patientIDList.append(patientID)
        ImageLevelHash[key] = str(row[1]) # level

    global uniquePatientIDList
    uniquePatientIDList = sorted(set(patientIDList))
    count=0;
    for patientID in uniquePatientIDList:
        left_level = ImageLevelHash[str(patientID+'_left')]
        right_level = ImageLevelHash[str(patientID+'_right')]
        #right_exists = str(patientID+'_right') in raw_df.values
        if (left_level != right_level):
            count = count+1
    print("count of images with both left and right eye level not matching="+str(count))#2240
    print("number of unique patients="+str(len(uniquePatientIDList)))#17563
```

```
144     return raw_df
145 
146 random.seed(10)
147 print("Reading trainLabels.csv...")
148 df = readTrainCsv()
149 
150 for i in range(0,10):
151     s = df.loc[df.index[i], 'PatientID'] # get patient id of patients
152     print(str(i) + " patient's patientID="+str(s))
153 
154 # df has 3 columns ['image', 'level', 'PatientID']
155 keepImages =  list(ImageNameDataHash.keys())
156 df = df[df['image'].isin(keepImages)]
157 print(len(df)) # 1000
158 
159 #convert hash to dataframe
160 imageNameArr = []
161 dataArr = []
162 for index, row in df.iterrows():
163     key = str(row[0])
164     if key in ImageNameDataHash:
165         imageNameArr.append(key)
166         dataArr.append(np.array(ImageNameDataHash[key])) # np.array
167 
168 df2 = pd.DataFrame({'image': imageNameArr, 'data': dataArr})
169 df2_header_list = list(df2.columns)
170 
171 if len(df) != len(df2):
172     print("Error length of df != df2")
173 
174 for idx in range(0,len(df)):
175     #comparing with original data frame to remove inconsistencies
176     if (df.loc[df.index[idx], 'image'] != df2.loc[df2.index[idx], 'image']):
177         print("Error " + df.loc[df.index[idx], 'image'] +"==" + df2.loc[df2.index[idx], 'image'])
178 
179 df = pd.merge(df2, df, left_on='image', right_on='image', how='outer')
180 df_header_list = list(df.columns)
181 
182 #Training data
183 X = df['data']
184 Y = df['level']
185 
186 Y = np.array(Y)
187 # convert the labels from integers to vectors
188 Y =  to_categorical(Y, num_classes=NUM_CLASSES)
189 
190 # partition the data into training and testing splits using 75% training and 25% for validation
191 print("Parttition data into 75:25...")
192 sys.stdout.flush()
193 print("Unique patients in dataframe df=" + str(df.PatientID.nunique())) # 500
194 unique_ids = df.PatientID.unique()
195 print('unique_ids shape='+ str(len(unique_ids))) #500
196 
197 # Refer https://www.kaggle.com/kmader/tf-data-tutorial-with-retina-and-keras
```

```
#stratify = rr_df['level'])
train_ids, valid_ids = train_test_split(unique_ids, test_size = 0.25, random_state = 12)
trainid_list = train_ids.tolist()
print('trainid_list shape=', str(len(trainid_list))) # 375

traindf = df[df.PatientID.isin(trainid_list)]
valSet = df[~df.PatientID.isin(trainid_list)]


traindf = traindf.reset_index(drop=True)
valSet = valSet.reset_index(drop=True)

print(traindf.head())
print(valSet.head())

trainX = traindf['data']
trainY = traindf['level']

valX = valSet['data']
valY = valSet['level']

#(trainX, valX, trainY, valY) = train_test_split(X,Y,test_size=0.25, random_state=10)
print('trainX shape=', trainX.shape[0], 'valX shape=', valX.shape[0]) # 750, 250

trainY =  to_categorical(trainY, num_classes=NUM_CLASSES)
valY =  to_categorical(valY, num_classes=NUM_CLASSES)

#construct the image generator for data augmentation
print("Generating images...")
sys.stdout.flush()
aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
height_shift_range=0.1, shear_range=0.2,
zoom_range=0.2,horizontal_flip=True,fill_mode="nearest")

#Correntropy Loss Function
def correntropy_coeff(y_true, y_pred,sigma=2.2):
    beta=1-K.exp(-1/(2*sigma*sigma))
    beta=1/beta
    alpha=2.2
    c=1
    #loss=K.exp(-K.sum(K.square(y_true - y_pred))/(2*sigma*sigma))
    loss=(K.pow((1+abs(y_true-y_pred)/c),alpha)-1)/alpha
    loss=loss*(-1/(2*sigma*sigma))
    loss=K.exp(loss)
    loss=1-loss
    loss = beta*(loss)
    return loss

def correntropy_loss(sigma=2.2):
    def correntropy(y_true,y_pred):
        return correntropy_coeff(y_true,y_pred,sigma)
    return correntropy

#Homotopy Loss Function
```

```
252 def homotopy_coeff(y_true, y_pred,alpha=2,c=1):
253     loss=(K.pow((1+abs(y_true-y_pred)/c),alpha)-1)/alpha
254     return loss
255 
256 def homotopy_loss(alpha=2,c=1):
257     def homotopy(y_true,y_pred):
258         return homotopy_coeff(y_true,y_pred,alpha,c)
259     return homotopy
260 
261 def createModel():
262     # returns our fully constructed deep learning + Keras image classifier
263     base_model = InceptionResNetV2(include_top=False, weights='imagenet', input_shape=inputShape)
264     # Add new top layers
265     x = base_model.output
266     x = Flatten()(x)
267     x = Dropout(0.2)(x)
268     x = Dense(32, activation='relu')(x)
269     x = Dense(16, activation='relu')(x)
270     predictions = Dense(NUM_CLASSES, activation='softmax')(x)
271 
272     # This is the model we will train
273     model = Model(inputs=base_model.input, outputs=predictions)
274 
275     # First: train only the top layers (which were randomly initialized)
276     for layer in base_model.layers:
277         layer.trainable = False
278     opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
279     # use binary_crossentropy if there are two classes
280     model.compile(loss=homotopy_loss(), optimizer=opt, metrics=["accuracy"])
281     return model
282 
283 print("Reshaping trainX at..."+ str(datetime.now()))
284 #print(trainX.sample())
285 print(type(trainX)) # <class 'pandas.core.series.Series'>
286 print(trainX.shape) # (750,)
287 
288 Xtrain = np.zeros([trainX.shape[0],HEIGHT, WIDTH, DEPTH])
289 for i in range(trainX.shape[0]): # 0 to traindf Size -1
290     Xtrain[i] = trainX[i]
291 print(Xtrain.shape) # (750,128,128,3)
292 print("Reshaped trainX at..."+ str(datetime.now()))
293 
294 print("Reshaping valX at..."+ str(datetime.now()))
295 print(type(valX)) # <class 'pandas.core.series.Series'>
296 print(valX.shape) # (250,)
297 Xval = np.zeros([valX.shape[0],HEIGHT, WIDTH, DEPTH])
298 for i in range(valX.shape[0]): # 0 to traindf Size -1
299     Xval[i] = valX[i]
300 print(Xval.shape) # (250,128,128,3)
301 print("Reshaped valX at..."+ str(datetime.now()))
302 
303 # initialize the model
304 print("compiling model...")
305 sys.stdout.flush()
```

```
306 model = createModel()
307
308 #print the summary of model
309
310 print_summary(model, line_length=None, positions=None, print_fn=None)
311
312 # model png file
313
314 plot_model(model, to_file=model_dir+'model_'+used_model+'.png')
315
316 # train the network
317 print("training network...")
318 sys.stdout.flush()
319 #class_mode ='categorical', # 2D one-hot encoded labels
320 t1=time.time()
321 H = model.fit_generator(aug.flow(Xtrain, trainY, batch_size=BS),validation_data=(Xval, valY),
322 steps_per_epoch=len(trainX) // BS,epochs=EPOCHS, verbose=1)
323
324 t2=time.time()
325 print("Training time for the model is",(t2-t1)//60,"minutes")
326 # save the model to disk
327 print("Saving model to disk")
328 sys.stdout.flush()
329 with open(model_dir+"trained_"+used_model+"cc"+".json", "w") as json_file:
330     json_file.write(model.to_json())
331 model.save_weights(model_dir+"trained_"+used_model+"cc"+".h5")
332 #model.save("trained_"+used_model+".h5")
333 with open(model_dir+"trained_"+used_model+"cc"+"_history", 'wb') as file_pi:
334     pickle.dump(H.history, file_pi)
335
336 # set the matplotlib backend so figures can be saved in the background
337 # plot the training loss and accuracy
338 print("Generating plots...")
339 sys.stdout.flush()
340 matplotlib.use("Agg")
341 matplotlib.pyplot.style.use("ggplot")
342 matplotlib.pyplot.figure()
343 N = EPOCHS
344 matplotlib.pyplot.plot(np.arange(0, N), H.history["loss"], label="train_loss")
345 matplotlib.pyplot.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
346 matplotlib.pyplot.plot(np.arange(0, N), H.history["acc"], label="train_acc")
347 matplotlib.pyplot.plot(np.arange(0, N), H.history["val_acc"], label="val_acc")
348 matplotlib.pyplot.title("Training Loss and Accuracy on diabetic retinopathy detection")
349 matplotlib.pyplot.xlabel("Epoch #")
350 matplotlib.pyplot.ylabel("Loss/Accuracy")
351 matplotlib.pyplot.legend(loc="lower left")
352 matplotlib.pyplot.savefig(model_dir+"curves_plot_"+used_model+".png")
```