A

Major Project

On

SELF- SUPERVISED LEARNING FOR MEDICAL IMAGING

(Submitted in partial fulfillment of the requirements for the award of Degree)

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

Lekkala Harika Chowdary (187R1A05F3)

Koneru Pragna(187R1A05G1)

Nalla Bhavani (187R1A05G7)

Under the Guidance of

Dr. B.LAXMAIAH
(Associate Professor)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

(Accredited by NAAC, NBA, Permanently Affiliated to JNTUH, Approved by AICTE, New Delhi) Recognized Under Section 2(f) & 12(B) of the UGCAct.1956, Kandlakoya (V),Medchal Road, Hyderabad-501401.

2018-22

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled "SELF-SUPERVISED LEARNING FOR MEDICAL IMAGING" is being submitted by **L. Harika Chowdary(187R1A05F3), K. Pragna (187R1A05G1)** & **N. Bhavani(187R1A05G7)** in partial fulfillment ofthe requirements for the award of the degree of B.Tech in Computer Science and Engineering to the Jawaharlal Nehru Technological University Hyderabad, is a record of bonafide work carried out by him/her under our guidance and supervision during the year 2021-22.

The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma.

Dr. B. Laxmaiah
(Associate Professor)
INTERNAL GUIDE

Dr. A. Raji Reddy
DIRECTOR

Dr. K. Srujan Raju
HOD

EXTERNAL EXAMINER

Submitted for viva voice Examination held on ________________________

ACKNOWLEDGMENT

Apart from the efforts of us, the success of any project depends largely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.

We take this opportunity to express my profound gratitude and deep regard to my guide **Dr. B.LAXMAIAH,** Associate Professor for his exemplary guidance, monitoring and constant encouragement throughout the project work. The blessing, help and guidance given byhim shall carry us a long way in the journey of life on which we are about to embark. We also take this opportunity to express a deep sense of gratitude to Project Review Committee (PRC) **Mr. J. Narasimha Rao, Dr. T. S. MastanRao, Dr. Suwarna Gothane, Mr. A. Uday Kiran, Mr. A. Kiran Kumar, Mrs. G. Latha** for their cordial support, valuable information and guidance, which helped us in completing this task through various stages.

We are also thankful to **Dr. K. Srujan Raju,** Head, Department of Computer Science and Engineering for providing encouragement and support for completing this project successfully.

We are obliged to **Dr. A. Raji Reddy,** Director for being cooperative throughout the course of this project. We also express our sincere gratitude to Sri. **Ch. Gopal Reddy,** Chairman for providing excellent infrastructure and a nice atmosphere throughout the course of this project.

The guidance and support received from all the members of **CMR Technical Campus** who contributed to the completion of the project. We are grateful for their constant support and help.

Finally, we would like to take this opportunity to thank our family for their constant encouragement, without which this assignment would not be completed. We sincerely acknowledge and thank all those who gave support directly and indirectly in the completion of this project.

Lekkala Harika Chowdary (187R1A05F3)

Koneru Pragna (187R1A05G1)

Nalla Bhavani (187R1A05G7)

`

ABSTRACT

Deep learning role in medical imaging is increasing quite effectively. As the models are providing promising accuracy, the early detection and risk mitigation of several diseases is becoming easy. Diabetic Retinopathy is one such disease, where early detection plays a severe role as it can lead to vision loss.

To implement a model in supervised manner, we need huge amount of labeled data set which can be very costly. So as to overcome these problems, in this paper we have implemented a self-supervised learning model for detection of diabetic retinopathy, using very limited dataset. This model is implemented using one of the pretext/proxy task image rotations developed on Dense NET architecture. The model is fine-tuned with the various quantities of subsets of the original dataset and compared internally.

`

LIST OF FIGURES/TABLES

TABLE OF CONTENTS:

`

1. INTRODUCTION

1. INTRODUCTION

1.1 PROJECT SCOPE

Recognition is identifying or distinguishing a thing or an individual from the past experiences or learning. Similarly, Disease recognition is nothing but recognizing or identifying the disease. Disease recognition framework is simply the working of a machine to prepare itself or interpret the disease. There is an enormous amount of data that is being produced on a daily basis from different areas using different imaging modalities such as MRI, CT, microscopy, etc., leading to an unprecedented potential for machine learning algorithms.

The ophthalmology field has benefited from recent advances in deep learning, particularly in the case of deep convolutional neural networks (CNNs) when applied to large data sets, such as two-dimensional (2D) fundus photography, a low-key imaging technology that captures the back of the eye. These images can be taken using a smartphone and are available in a standardized fashion, often in very large quantities. Using data from diabetes screening programs and biobanks, cardiovascular risk factors, presence of diabetic retinopathy, and even gender, can be predicted with a high degree of accuracy Using Convolutional Neural Networks, we can predict the diabetic retinopathy and detect the different stages of it. The model also finds the accuracy of the model and finds the value by using “Kappa-kaggle score”.

1.2 PROJECT PURPOSE

A practitioner using Convolutional Neural Networks (CNN) for the task of medical imaging is faced with a plethora of options when it comes to the training methodology for the CNN. Several factors can influence the decision making process including, but not limited to the size, noise level and quality of the dataset at hand, computational resources available and robustness of the trained CNN. The task of Diabetic Retinopathy detection, using a Convolutional neural Networks and Self-Supervised Learning, has great importance and use.

1.3 PROJECT FEATURES

Diabetic Retinopathy disease recognition has been one of the active and challenging research areas in the field of image processing. Deep learning technique as well as hinders to work with disease recognition and find the accuracy of the model. Graphical representation is showed using the accuracy of the model. The model is compared with other models and the accuracy of the model is determined using a graph.

2. SYSTEM ANALYSIS

2. SYSTEM ANALYSIS

2.1 SYSTEM ANALYSIS

System Analysis is the important phase in the system development process. The System is studied to the minute details and analyzed. The system analyst plays an important role of an interrogator and dwells deep into the working of the present system. In analysis, a detailed study of these operations performed by the system and their relationships within and outside the system is done. A key question considered here is, “what must be done to solve the problem?” The system is viewed as a whole and the inputs to the system are identified. Once analysis is completed the analyst has a firm understanding of what is to be done.

2.2 PROBLEM DEFINITION

The goal of this project is to create a model that will be able to recognize and determine diabetic retinopathy from its image by using the concepts of Convolution Neural Network. Though the goal is to create a model which can recognize the diabetic retinopathy, it can be extended to detect the different stages of the disease. The major goal of the proposed system is understanding Convolutional Neural Network, and applying it to the medical imaging of diabetic retinopathy.

2.3 EXISTING SYSTEM

Unsupervised learning in general can be formulated as learning an embedding space, where the data that is similar semantically are closer and vice versa. The self-supervised learning does the same by constructing such representation space with the help of proxy task from the data itself. The learning’s of model at the time of proxy task can also be used in various other downstream tasks. Recently, several methods in the line of research have been developedand found applications in numerous fields.

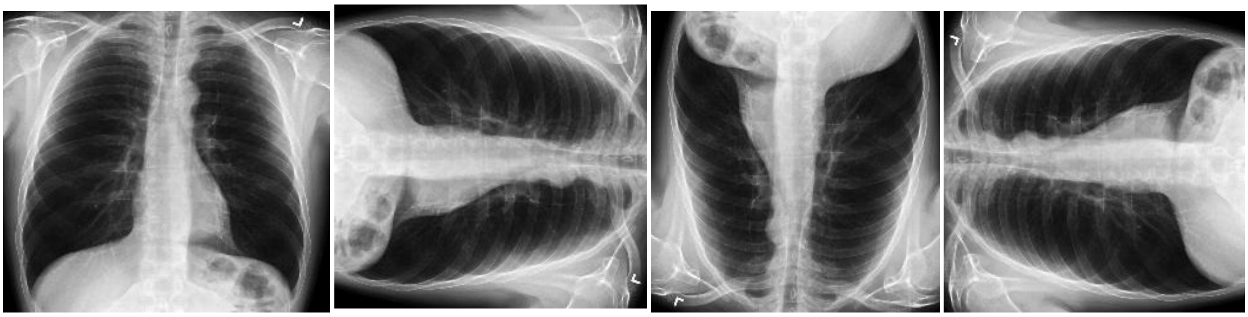
Self-supervised learning consists of two major parts of processing first is the proxy task and second is fine-tuning. There are different types of self-supervised learning methods that differ in their first building block i.e., proxy task. There are various types of proxy tasks developed in this line of research.

2.3.1 LIMITATIONS OF EXISTING SYSTEM

- You cannot get precise information regarding data sorting, and the output as data used in supervised learning is labelled and not known.
- Less accuracy of the results is because the input data is not known and not labelled by people in advance.

2.4 PROPOSED SYSTEM

The model follows a self-supervised paradigm and proposes to learn image representations by training to recognize the geometric transformation that is applied to the image that it gets as input. More specifically, we first define a small set of discrete geometric transformations, then each image on the dataset and the produced transformed images are fed to the model that is trained to recognize the transformation of each image



0 degrees 90 degrees 180 degrees 270 degrees

2.4.1 ADVANTAGES OF THE PROPOSED SYSTEM

- It requires much less labelled data
- It provides a significant improvement in accuracy
- We can reduce the need for expensive annotated data to build image classification models

2.4.2 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. Three key considerations involved in the feasibility analysis are:

1. Economic Feasibility
2. Technical Feasibility
3. Social Feasibility

2.4.3 ECONOMIC FEASIBILITY

The developing system must be justified by cost and benefit. Criteria to ensure that effort is concentrated on project, which will give best, return at the earliest. One of the factors, which affect the development of a new system, is the cost it would require. The following are some of the important financial questions asked during preliminary investigation:

- The costs conduct a full system investigation.
- The cost of the hardware and software.
- The benefits in the formof reduced costs or fewer costly errors. Since the system is developed as part of project work, there is no manual cost to spend for the proposed system. Also all the resources are already available, it give an indication of the system is economically possible for development.

2.4.4 TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

2.4.5 BEHAVIORAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system. This includes the following question:

- Will the proposed system cause harm?

The project would be beneficial because it satisfies the objectives when developed and installed. All behavioral aspects are considered carefully and conclude that the project is behaviorally feasible.

2.5 HARDWARE & SOFTWARE REQUIREMENTS

2.5.1 HARDWARE REQUIREMENTS:

Hardware interfaces specifies the logical characteristics of each interface between the software product and the hardware components of the system. The following are some hardware requirements.

- Processor : INTEL CORE i5Processor
- Hard Disk : 50 GB and Above.
- Input Devices : Keyboard, Mouse.
- RAM : 8GB and Above.

2.5.2 SOFTWARE REQUIREMENTS:

Software Requirements specifies the logical characteristics of each interface and software components of the system. The following are some software requirements.

- Operating System : WINDOWS XP
- Programming Language : PYTHON
- Tools : Anaconda, Google Colab.

3. METHODOLOGY

3. METHODOLOGY

3.1 CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks are deep artificial neural networks. We can use it to classify images (e.g., name what they see), cluster them by similarity (photo search) and perform object recognition within scenes. It can be used to identify faces, individuals, street signs, tumors, platypuses and many other aspects of visual data. The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels) which have a small receptive field but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product, and producing a 2- dimensional activation map of that filter.

As a result, the network learns when they see some specific type of feature at some spatial position in the input. Then the activation maps are fed into a down sampling layer, and like convolutions, this method is applied one patch at a time. CNN has also fully connected layer that classifies output with one label per node. The CNN architecture consists of two main parts: feature extraction and classification. In the feature extraction layers, each layer of the network receives the output from its immediate previous layer as its input, and passes the current output as input to the next layer. The CNN architecture is composed with the combination of three types of layers: convolution, max-pooling, and classification.

A. CONVOLUTION NETWORKS

The convolutional layer is the first layer which can extract features from the images. Because pixels are only related to the adjacent and close pixels, convolution allows us to preserve the relationship between different parts of an image. Convolution is filtering the image with a smaller pixel filter to decrease the size of the image without losing the relationship between pixels. When we apply convolution to the 5x5 image by using a 3x3 filter with 1x1 stride (1- pixel shift at each step), we will end up having a 3x3 output (64% decrease in complexity.

B. POOLING LAYER

When constructing CNN, it is common to insert pooling layers after each convolution layer to reduce the spatial size of the features maps. Pooling layers also help with the over fitting problem. We select a pooling size to reduce the amount of the parameters by selecting the maximum, average, or sum values inside these pixels. Max Pooling, one of the most common pooling techniques.

C. FULLY CONNECTED

A fully connected network is in any architecture where each parameter is linked to one another to determine the relation and effect of each parameter on the labels. Since convolution and pooling layers reduce time-space complexity, we can construct a fully connected network in the end to classifythe images. Now we think, it is time to share an overview look of our proposed convolutional neural network. It has similarity with other handwritten digit recognition architectures [1,6,8,10,11] but has changed in a number of filters, neurons and activation functions for better performance. It has seven layers.

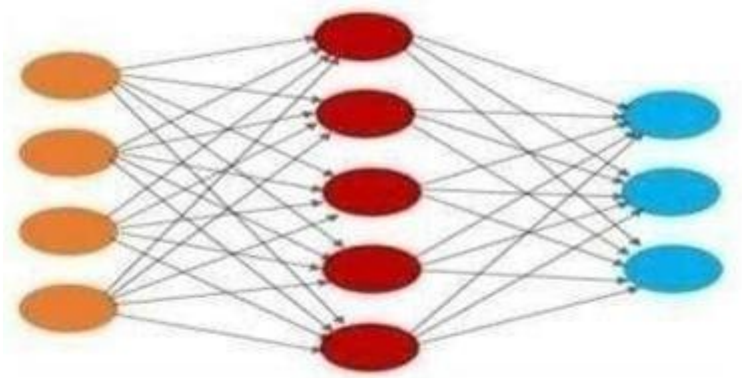


Fig 3.1 Neural networks

3.2 DENSENET 121

In short, Densenet -121 architecture of CNN is differed from the standards of CNN. The basic Operations and layers:

1 7*7 Convolution

58 3*3 Convolution

61 1*1 Convolution

4 Average Pool

1 Fully Connected layer

In a traditional feed-forward Convolutional Neural Network (CNN), each convolutional layer except the first one (which takes in the input), receives the output of the previous convolutional layer and produces an output feature map that is then passed on to the next convolutional layer. Therefore, for 'L' layers, there are 'L' direct connections; one between each layer and the next layer.

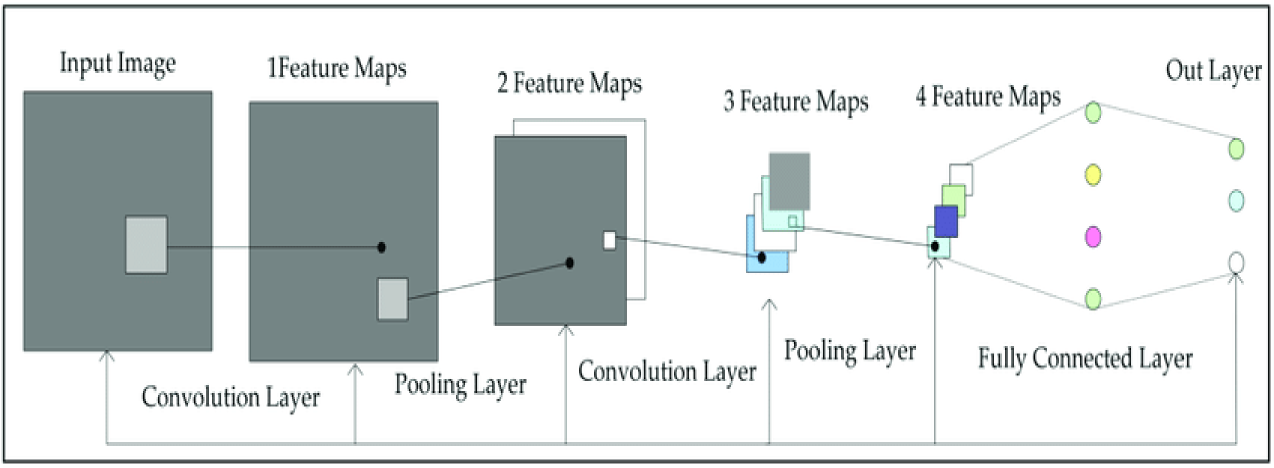


Fig 3.2 DenseNet 121

However, as the number of layers in the CNN increase, i.e. as they get deeper, the 'vanishing gradient' problem arises. This means that as the path for information from the input to the output layers increases, it can cause certain information to 'vanish' or get lost which reduces the ability of the network to train effectively.

DenseNets resolve this problem by modifying the standard CNN architecture and simplifying the connectivity pattern between layers. In a DenseNet architecture, each layer is connected directly with every other layer, hence the name Densely Connected Convolutional Network. For 'L' layers, there are L(L+1)/2 direct connections.

3.3 DENSENET COMPONENTS:

Components of DenseNet include:

- Connectivity
- DenseBlocks
- Growth Rate
- Bottleneck layers

A. Connectivity

In each layer, the feature maps of all the previous layers are not summed, but concatenated and used as inputs. Consequently, DenseNets require fewer parameters than an equivalent traditional CNN, and this allows for feature reuse as redundant feature maps are discarded. So, the l^{th} layer receives the feature-maps of all preceding layers, $x_0,...,x_{l-1}$, as input:

$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}]).$$

where $[x_0,x_1,...,x_{l-1}]$ is the concatenation of the feature-maps, i.e. the output produced in all the layers preceding l (0,...,l-1). The multiple inputs of H_l are concatenated into a single tensor to ease implementation.

B. Dense Blocks

The use of the concatenation operation is not feasible when the size of feature maps changes. However, an essential part of CNNs is the down-sampling of layers which reduces the size of feature- maps through dimensionality reduction to gain higher computation speeds.

To enable this, DenseNets are divided into DenseBlocks, where the dimensions of the feature maps remains constant within a block, but the number of filters between them is changed. The layers between the blocks are called Transition Layers which reduce the the number of channels to half of that of the existing channels.

For each layer, from the equation above, H_l is defined as a composite function which applies three consecutive operations: batch normalization (BN), a rectified linear unit (ReLU) and a convolution (Conv). A deep DenseNet with three dense blocks is shown. The layers between two adjacent blocks are the transition layers which perform downsampling (i.e. change the size of the feature-maps) via convolution and pooling operations, whilst within the dense block the size of the feature maps is the same to enable feature concatenation.

C. Growth Rate

One can think of the features as a global state of the network. The size of the feature map grows after a pass through each dense layer with each layer adding 'K' features on top of the global state (existing features). This parameter 'K' is referred to as the growth rate of the network, which regulates the amount of information added in each layer of the network. If each function H_l produces k feature maps, then the l^{th} layer has input feature-maps, where k_0 is the number of channels in the input layer. Unlike existing network architectures, DenseNets can have very narrow layers.

$$k_l = k_0 + k * (l - 1)$$

D. Bottleneck layers

Although each layer only produces k output feature-maps, the number of inputs can be quite high, especially for further layers. Thus, a 1x1 convolution layer can be introduced as a bottleneck layer before each 3x3 convolution to improve the efficiency and speed of computations.

As DenseNets require fewer parameters and allow feature reuse, they result in more compact models and have achieved state-of-the-art performances and better results across competitive datasets, as compared to their standard CNN or ResNet counterparts.

3.4 PROJECT ARCHITECTURE

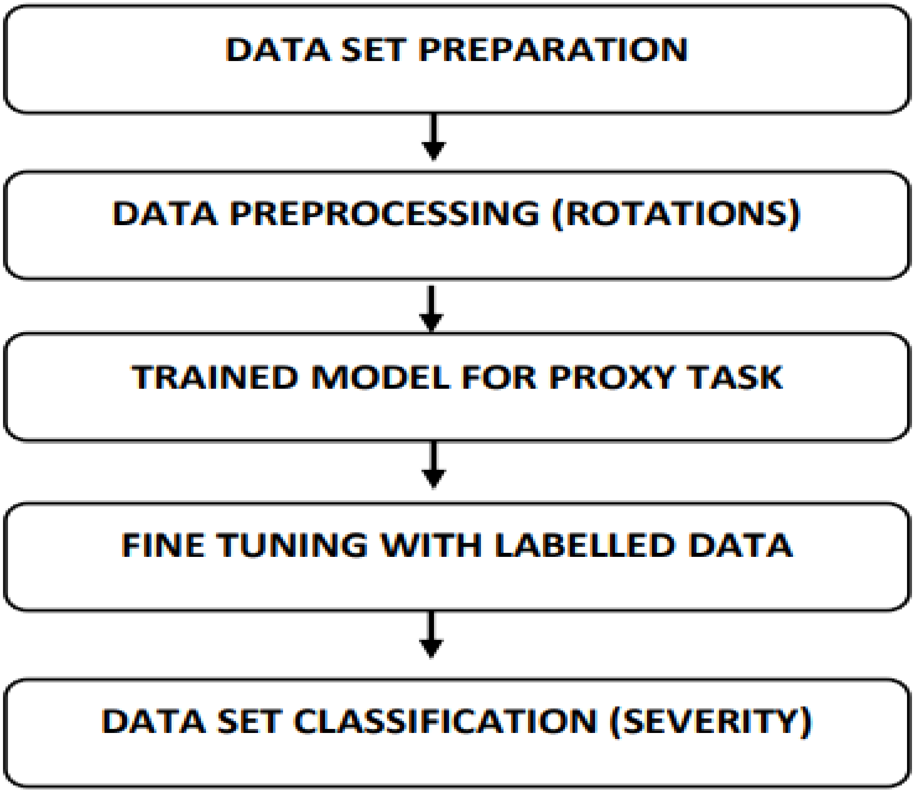


Fig.3.4 Project Architecture

To create self-supervised learning model for medical imaging, precisely for detection of Diabetic Retinopathy. We are using Rotation technique as proxy task, and use the unlabelled data with different geometric progressions and make Dense ConvNet model predict the probability of geometric progression. Then use some labelled data to fine-tune the model to detect the severity of the DR as follows

i. NO DR

ii. MILD DR

iii. MODERATE DR

iv. SEVERE DR

v. PROLIFERATE DR

The Figure basic deals with the architecture diagram of the proposed system. The proposed model is divided into four different stages in order to classify and detect the digits:

A. Dataset Preparation

B. Data Preprocessing

C. Fine-Tuning with Labelled Data

D. Classification

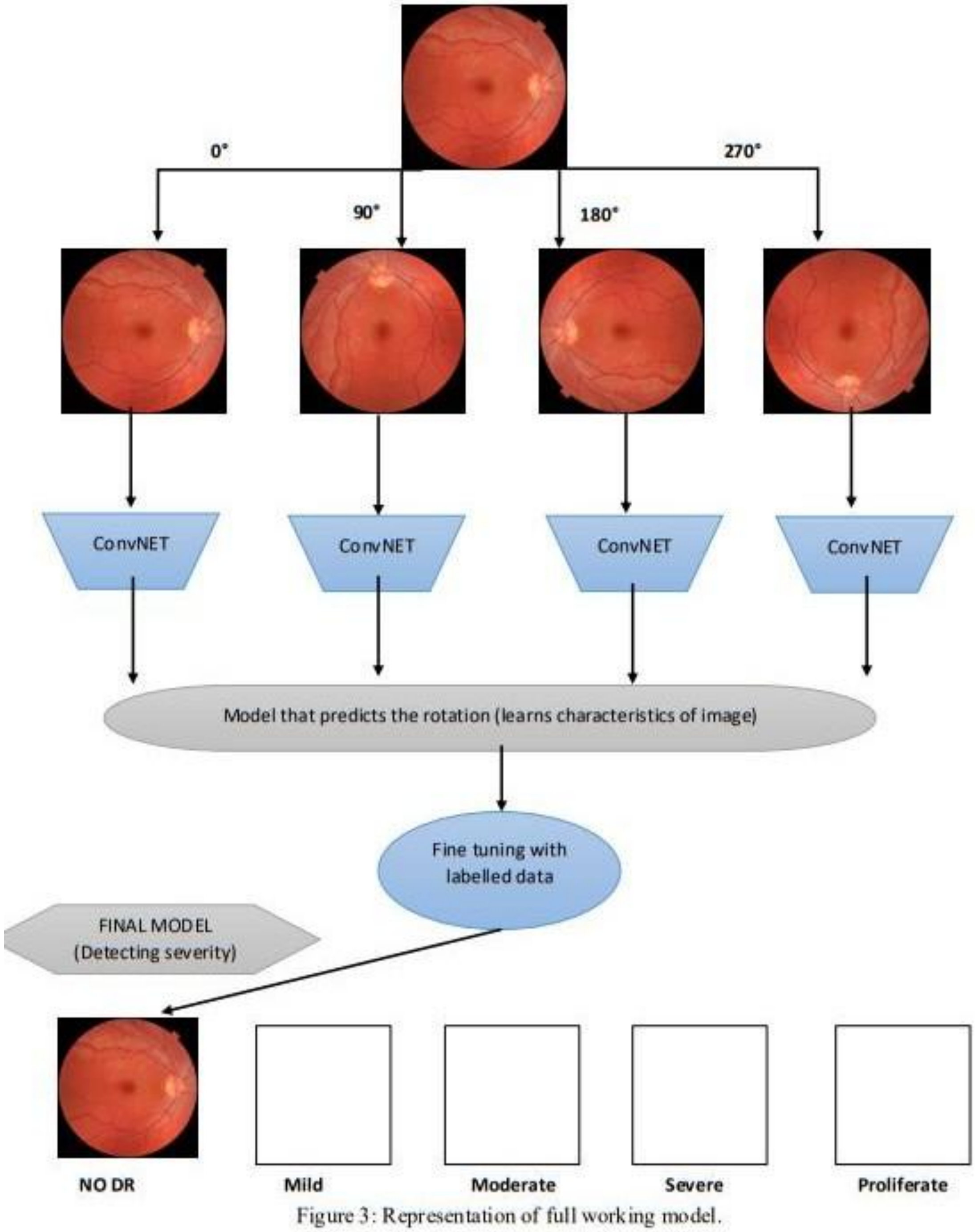


Fig 3.4.1 Working model of Self-supervised learning for Medical Imaging

3.5 MODULE DESCRIPTION

A. DATASET PREPERATION

The data is obtained from Kaggle mentioned in Diabetic Retinopathy 2019 Kaggle challenge. X It contains images of retinal fundus resized into 224 x 224, categorized into five types, NO DR, mild, moderate, severe and proliferate. For the proxy task we combine all the types as it does not require any labelling. For finetuning, we use 5%, 10%, 25% and 50% of data check the efficiency at each specific size of data.

Data Preparation process is an important part of Data Science. It includes two concepts such as data Cleaning and Feature Engineering. These two are compulsory for achieving better accuracy andperformance in the Machine Learning and Deep Learning projects.

B. DATA PREPROCESSING

Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis. Therefore, certain steps are executed to convert the data into a small clean data set. This technique is performed before the execution of the Iterative Analysis. The set of steps is known as Data Preprocessing. It includes –

- Data Cleaning
- Data Integration
- Data Transformation
- Data Reduction

The data is resized into 244 x 244 and performed various geometric progressions i.e., rotations into multiples of 90 degrees, (0, 90, 180, 270 degrees). This pre-processed data is thenfed to the ConvNET model with DenseNET121 encoder architecture. It was handled withlearning rate of 1e-5, for 200 epochs with batch size of 32.

C. FINE-TUNING WITH LABELLED DATA

The data is resized into 244 x 244 and performed various geometric progressions i.e., rotations into multiples of 90 degrees, (0, 90, 180, 270 degrees). This pre-processed data is thenfed to the ConvNET model with DenseNET121 encoder architecture. It was handled with learning rate of 1e-5, for 200 epochs with batch size of 32.

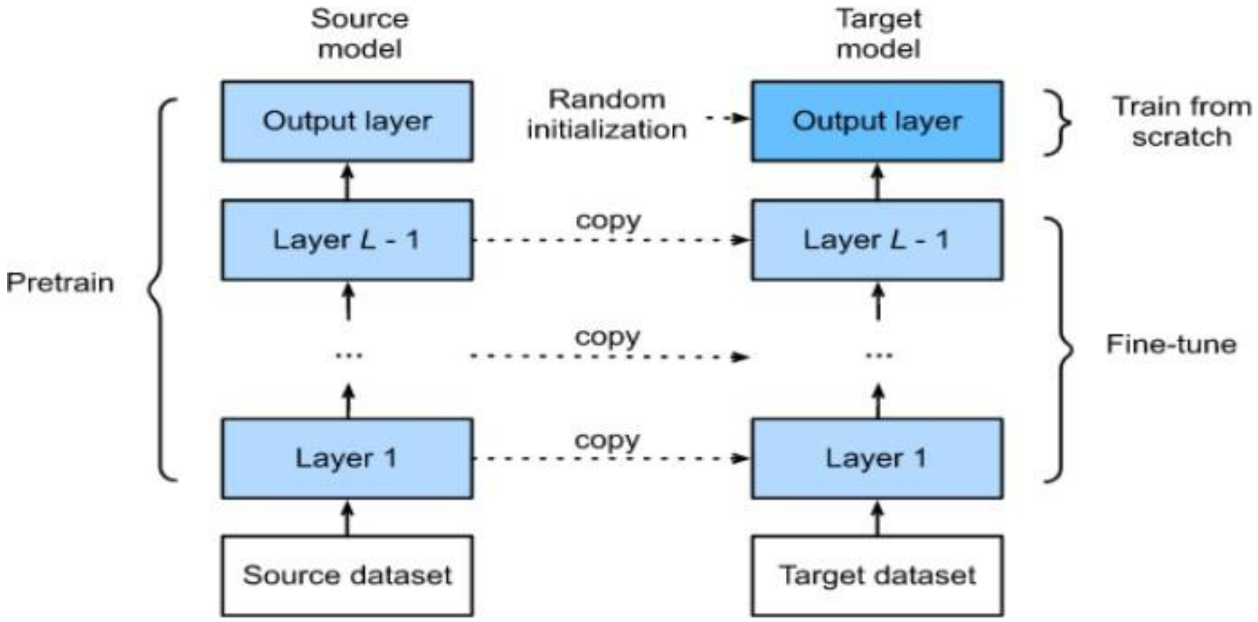


Fig 3.5 Fine-tuning with labeled data

D. CLASSIFICATION

The final model is generated after the fine tuning which can classify or detect the Diabetic retinopathy stages. It is tested with “qw_kappa_kaggle” scores based on the accuracy and obtained very promising results in comparison to dataset size.

Submissions are scored based on the quadratic weighted kappa, which measures the agreement between two ratings. This metric typically varies from 0 (random agreement between raters) to 1 (complete agreement between raters). In the event that there is less agreement between the raters thanexpected by chance, the metric may go below 0. The quadratic weighted kappa is calculated between the scores which are expected/known and the predicted scores. Results have 5 possible ratings, 0,1,2,3,4. The quadratic weighted kappa is calculated as follows.

First, an N x N histogram matrix O is constructed, such that Oi,j corresponds to the number of adoption records that have a rating of i (actual) and received a predicted rating j. An N-by-N matrix of weights, w, is calculated based on the difference between actual and predicted rating scores. An N-by-N histogram matrix of expected ratings, E, is calculated, assuming that there is no correlation between rating scores. This is calculated as the outer product between the actual rating's histogram vector of ratings and the predicted rating's histogram vector of ratings, normalized such that E and O have the same sum.

4 steps for Weighted Kappa Metric:

- First, create a multi class confusion matrix O between predicted and actual ratings.
- Second, construct a weight matrix w which calculates the weight between the actual and predicted ratings.

- Third, calculate value_counts() for each rating in preds and actuals.
- Fourth, calculate E, which is the outer product of two value_count vectors.

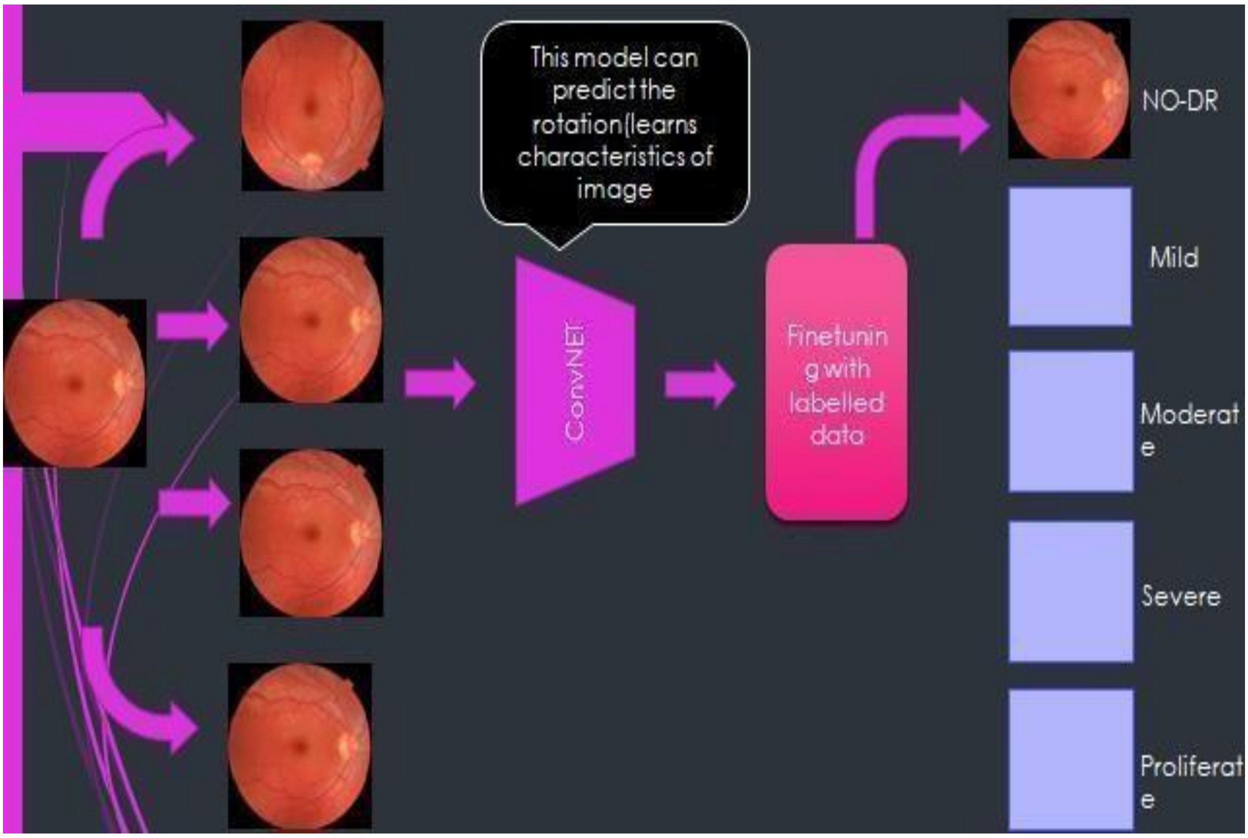


Fig 3.5.2 Classification and Working model

3.6 DIABETIC RETINOPATHY

People with diabetes can have an eye disease called diabetic retinopathy. This is when high blood sugar levels cause damage to blood vessels in the retina. These blood vessels can swell and leak. Or they can close, stopping blood from passing through. Sometimes abnormal new blood vessels grow on the retina. All of these changes can steal your vision.

It is expected that by 2040 around 600 million people suffer from diabetics in the world. Diabetic retinopathy- the leading cause of vision loss in working-age adults worldwide is expected in around one third of them. I Mild DR is the early stage of diabetic retinopathy, which can be detected by presence of microaneurysms. The more advance stages can cause in severe vision loss. So the early detection is quite necessary in huge scale and can be treated to reduce the vision loss. But the ratio of ophthalmologists to the patients is quite high and regular check is quite hard.

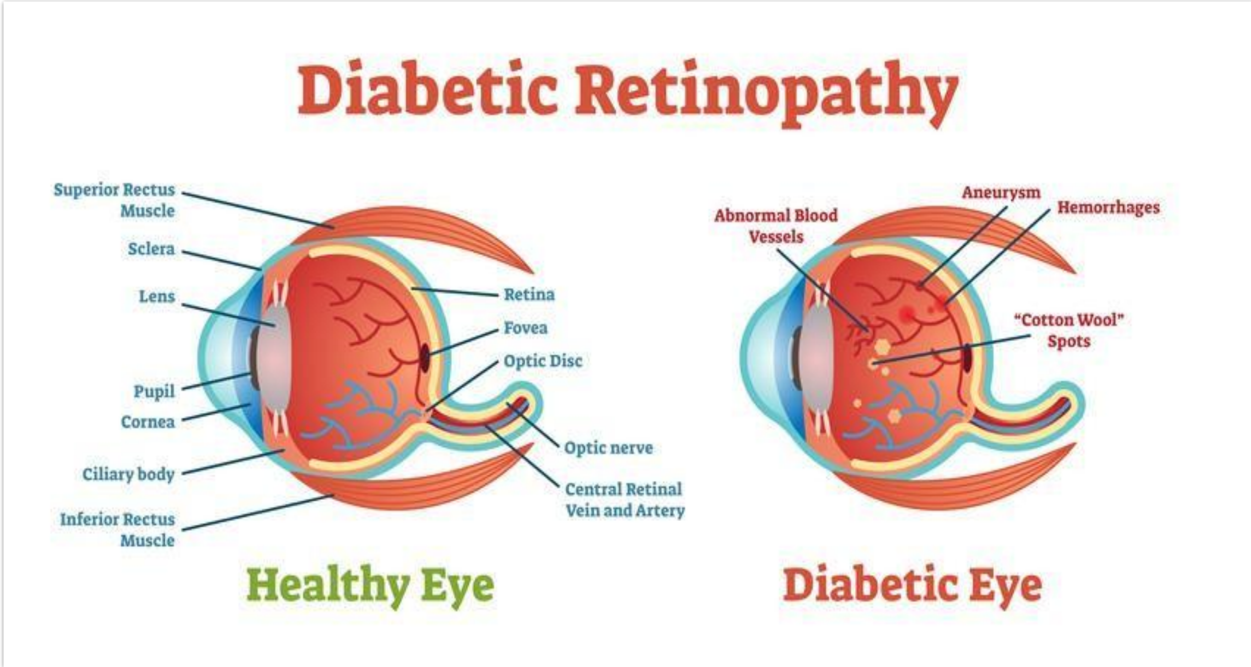


Fig 3.6 Diabetic Retinopathy

3.7 SELF SUPERVISED LEARNING

Supervised learning is the type of machine learning in which the models are trained with well labelled data and the model predicts the output. Practically providing huge amounts of labelled data isn't cost or time efficient all the time and supervised learning is in need of lot of labeled data. In contrast, the unlabelled data is available in abundance. This thrives the motivation for unsupervised learning and also self-supervised learning. The self-supervised model learns useful representations of the data from unlabelled pool of data using proxy tasks and then fine-tunes the model with few labels for the supervised downstream task. It can perform tasks as simple as image classification or complextask such as semantic segmentation etc.

- The self-supervised model learns useful representations of the data from unlabeled pool of data using proxy tasks and then fine-tunes the model with few labels for the supervised downstream task.
- It can perform tasks as simple as image classification or complex task such as semantic segmentation etc.

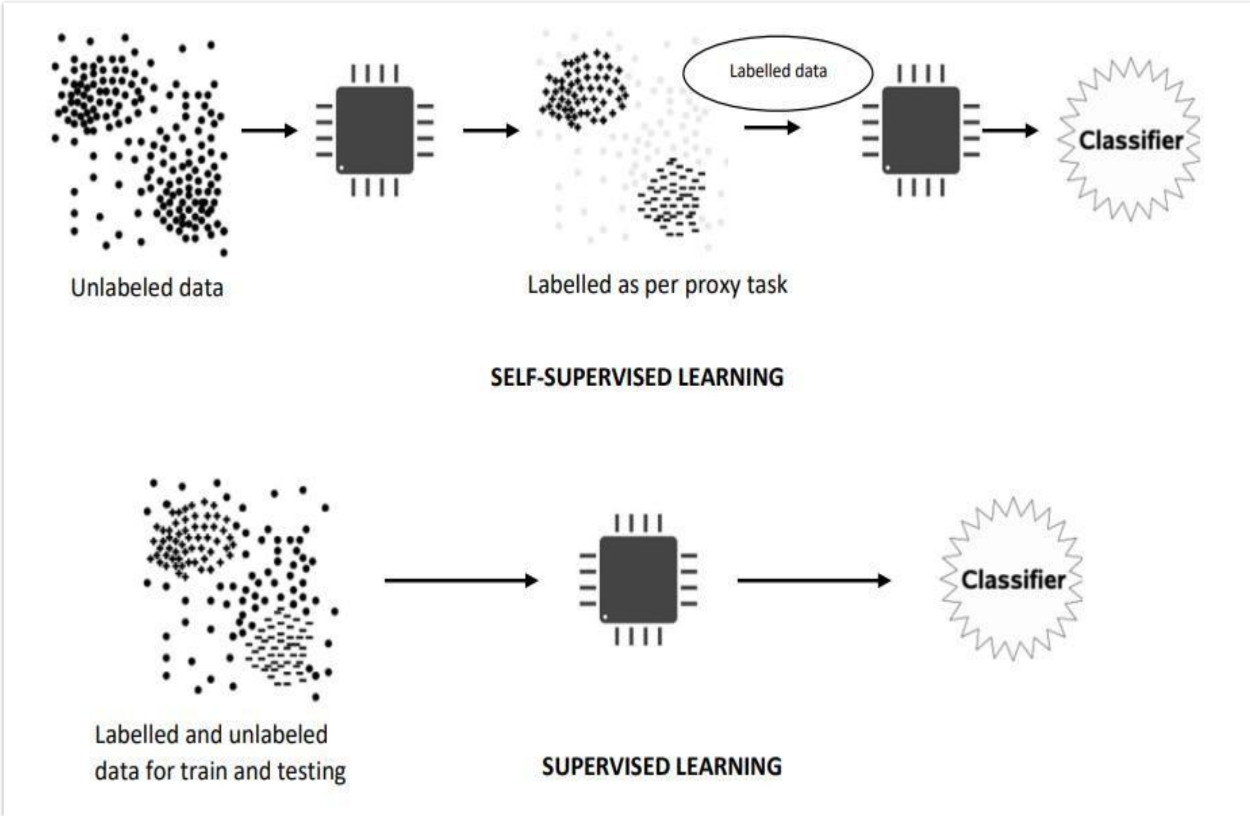


Fig 3.7 Self-Supervised Learning vs Supervised Learning

3.8 USECASE DIAGRAM

In the use case diagram we have basically two actors who are the user and the system. The user has the rights to login, access to resources and to view the details. Whereas the system has the login, access to resources of the users and also the right to update and remove the details, and he can also view the user files.

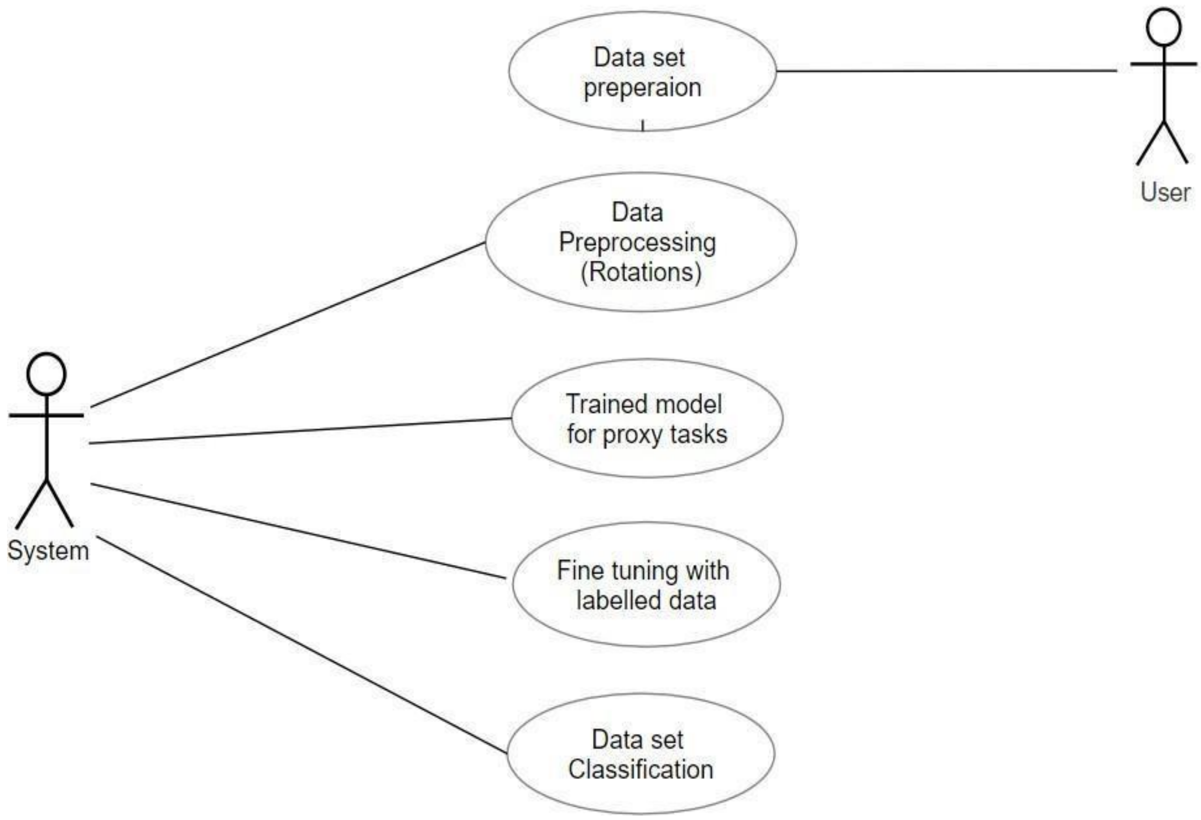


Fig.3.8 Usecase Diagram for Self supervised learning for Medical Imaging

3.9 CLASS DIAGRAM

Class Diagram is a collection of classes and objects.

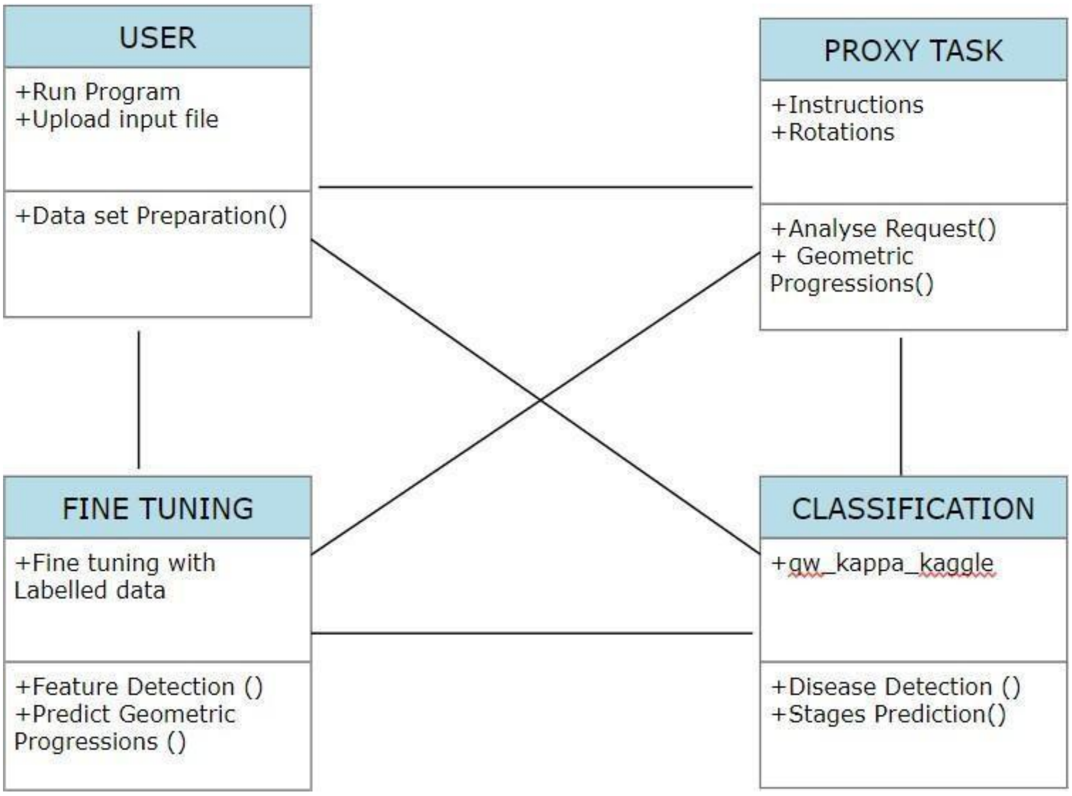


Fig.3.9 Class Diagram for Self-supervised learning for medical Imaging

3.10 SEQUENCE DIAGRAM

A sequence diagram shows object interactions arranged in time sequence.

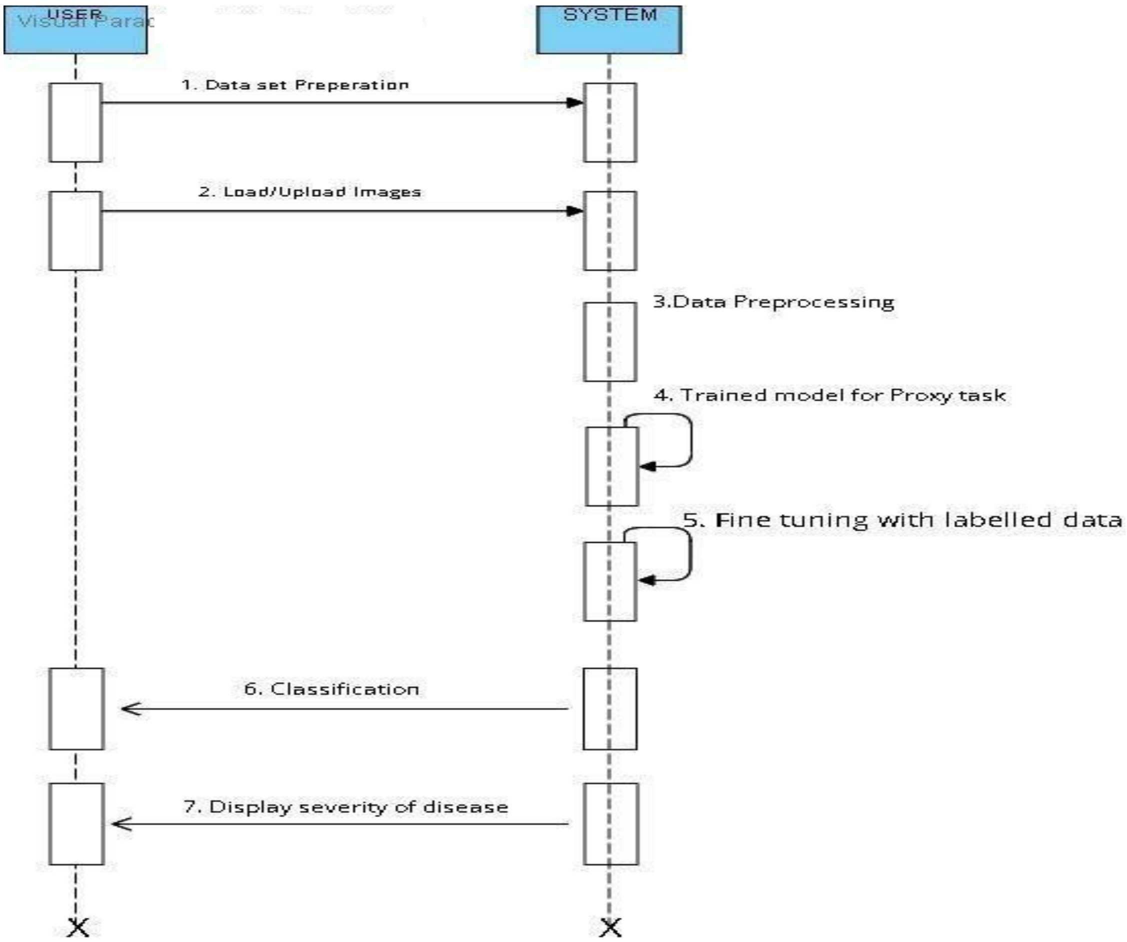


Fig.3.10 Sequence Diagram for Self-supervised learning for medical Imaging

3.11 ACTIVITY DIAGRAM

It describes about flow of activity states.

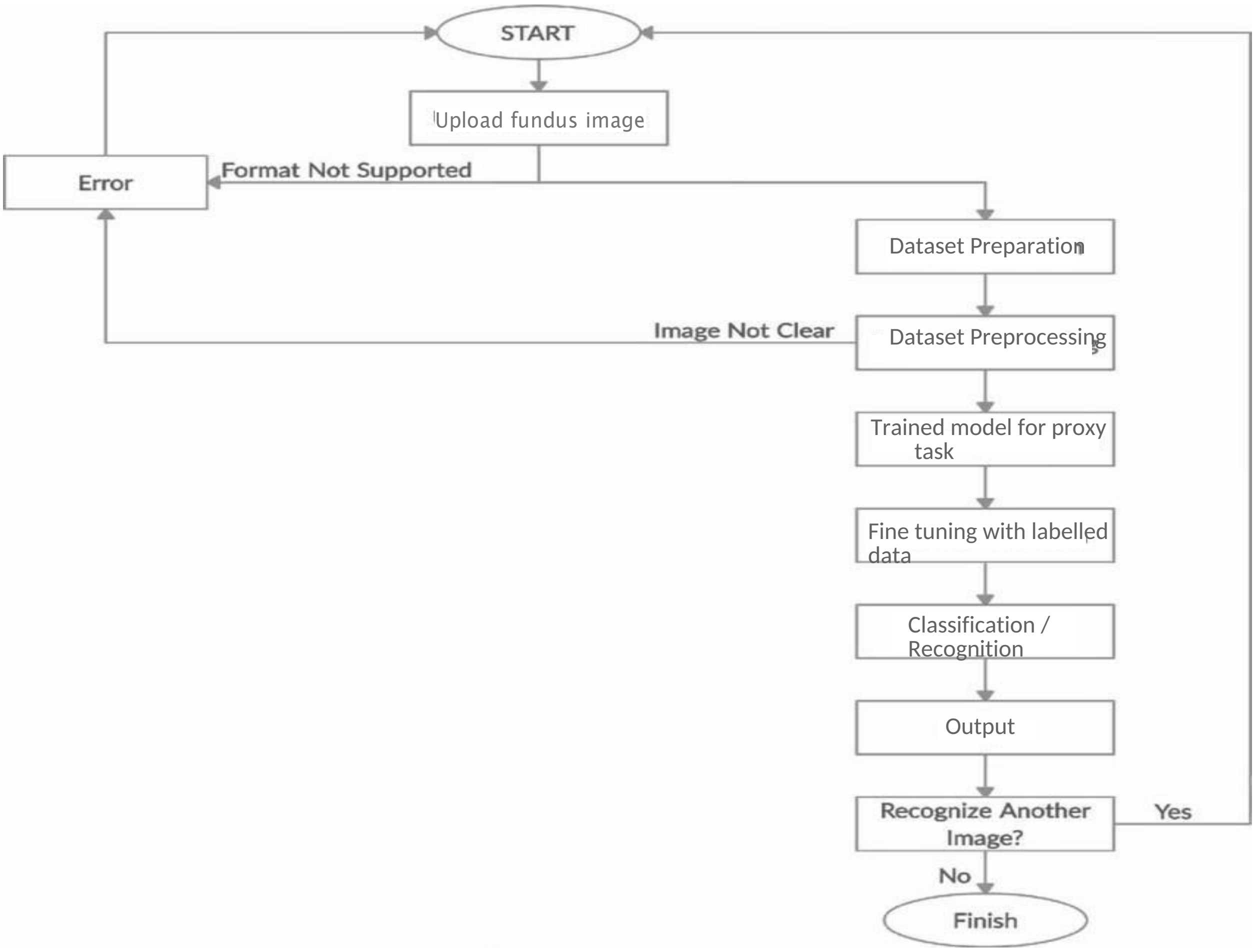


Fig.3.11 Activity Diagram User for Self-supervised learning for medical imaging

3. IMPLEMENTATION

4.1 IMPORT THE LIBRARIES AND LOAD THE DATASET

```
name: conda-env
dependencies:
  - tensorflow-gpu=2.1
  - absl-py
  - h5py
  - hdf5
  - pandas
  - python
  - scikit-image
  - scikit-learn
  - scipy
  - tensorboard
  - requests
  - cudatoolkit
  - cudnn
  - pip
  - networkx==2.2
  - seaborn
  - tqdm
  - pydot
  - pip:
    - tensorflow_hub
    - nibabel
    - hyperopt
  - pydot
  - pip:
    - tensorflow_hub
    - nibabel
    - hyperopt
    - albumentations
    - tensorflow_addons
    - joblib
    - opencv-python
  - pydot
  - pip:
    - tensorflow_hub
    - nibabel
    - hyperopt
    - albumentations
    - tensorflow_addons
    - joblib
    - opencv-python
```

```
[ ] !pip install PyJWT==1.7.1

    Collecting PyJWT==1.7.1
      Downloading https://files.pythonhosted.org/packages/87/8b/6a9f14b5f781697e51259d81657e6048fd31a113229cf346880bb7545565/
    Installing collected packages: PyJWT
    Successfully installed PyJWT-1.7.1

[ ] %cd self-supervised-3d-tasks-master/

    /content/drive/MyDrive/self-supervised-3d-tasks-master
```

4.2 PERFORMING ROTATIONS TO THE DATASET

```
import numpy as np
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.python.keras import Sequential
from self_supervised_3d_tasks.algorithms.algorithm_base import AlgorithmBuilderBase
from self_supervised_3d_tasks.utils.model_utils import (
    apply_encoder_model,
    apply_encoder_model_3d,
    apply_prediction_model)
from self_supervised_3d_tasks.preprocessing.preprocess_rotation import (
    rotate_batch,
    rotate_batch_3d,
)


class RotationBuilder(AlgorithmBuilderBase):
    def __init__(
            self,
            data_dim=384,
            number_channels=3,
            lr=1e-4,
            data_is_3D=False,
            data_is_3D=False,
            top_architecture="big_fully",
            **kwargs
    ):
        super(RotationBuilder, self).__init__(data_dim, number_channels, lr, data_is_3D, **kwargs)

        self.image_size = data_dim
        self.img_shape = (self.image_size, self.image_size, number_channels)
        self.img_shape_3d = (
            self.image_size,
            self.image_size,
            self.image_size,
            number_channels,
        )
        self.top_architecture = top_architecture

    def apply_model(self):
        if self.data_is_3D:
            self.enc_model, self.layer_data = apply_encoder_model_3d(self.img_shape_3d, **self.kwargs)
        else:
            self.enc_model, self.layer_data = apply_encoder_model(self.img_shape, **self.kwargs)

        return self.apply_prediction_model_to_encoder(self.enc_model)
```

```
    def apply_prediction_model_to_encoder(self, encoder_model):
        if self.data_is_3D:
            x = Dense(10, activation="softmax")
        else:
            x = Dense(4, activation="softmax")
        units = np.prod(encoder_model.outputs[0].shape[1:])
        sub_model = apply_prediction_model((units,), prediction_architecture=self.top_architecture, include_top=False)
        return Sequential([encoder_model, Flatten(), sub_model, x])

    def get_training_model(self):
        model = self.apply_model()
        model.compile(
            optimizer=Adam(lr=self.lr),
            loss="categorical_crossentropy",
            metrics=["accuracy"],
        )

        return model

    def get_training_preprocessing(self):
        def f(x, y):  # not using y here, as it gets generated
            return rotate_batch(x, y)

        def f_3d(x, y):
            return rotate_batch_3d(x, y)

        if self.data_is_3D:
            return f_3d, f_3d
        else:
            return f, f

    def purge(self):
        for i in reversed(range(len(self.cleanup_models))):
            del self.cleanup_models[i]
        del self.cleanup_models
        self.cleanup_models = []


def create_instance(*params, **kwargs):
    return RotationBuilder(*params, **kwargs)
```

4.3 TRAINING THE MODEL

```
from self_supervised_3d_tasks.data.numpy_2d_loader import Numpy2DLoader
from self_supervised_3d_tasks.utils.model_utils import init, print_flat_summary
from pathlib import Path

import tensorflow.keras as keras
from self_supervised_3d_tasks.data.numpy_3d_loader import DataGeneratorUnlabeled3D, PatchDataGeneratorUnlabeled3D

from self_supervised_3d_tasks.data.make_data_generator import get_data_generators
from self_supervised_3d_tasks.data.image_2d_loader import DataGeneratorUnlabeled2D
from self_supervised_3d_tasks.algorithms import cpc, jigsaw, relative_patch_location, rotation, exemplar
from self_supervised_3d_tasks.utils.model_utils import get_writing_path

keras_algorithm_list = {
    "cpc": cpc,
    "jigsaw": jigsaw,
    "rpl": relative_patch_location,
    "rotation": rotation,
    "exemplar": exemplar
}

```

```python
data_gen_list = {
    "kaggle_retina": DataGeneratorUnlabeled2D,
    "pancreas3d": DataGeneratorUnlabeled3D,
    "pancreas2d": Numpy2DLoader,
    "brats": DataGeneratorUnlabeled3D,
    "ukb2d": DataGeneratorUnlabeled2D,
    "ukb3d": PatchDataGeneratorUnlabeled3D
}


def get_dataset(data_dir, batch_size, f_train, f_val, train_val_split, dataset_name,
                train_data_generator_args={}, val_data_generator_args={}, **kwargs):
    data_gen_type = data_gen_list[dataset_name]

    train_data, validation_data = get_data_generators(data_dir, train_split=train_val_split,
                                                      train_data_generator_args={**{"batch_size": batch_size,
                                                                                    "pre_proc_func": f_train},
                                                                                 **train_data_generator_args},
                                                      val_data_generator_args={**{"batch_size": batch_size,
                                                                                  "pre_proc_func": f_val},
                                                                               **val_data_generator_args},
                                                      data_generator=data_gen_type)

    return train_data, validation_data


def train_model(algorithm, data_dir, dataset_name, root_config_file, epochs=250, batch_size=2, train_val_split=0.9,
                base_workspace="~/netstore/workspace/", save_checkpoint_every_n_epochs=5, **kwargs):
    kwargs["root_config_file"] = root_config_file

    working_dir = get_writing_path(Path(base_workspace).expanduser() / (algorithm + "_" + dataset_name),
                                   root_config_file)
    algorithm_def = keras_algorithm_list[algorithm].create_instance(**kwargs)

    f_train, f_val = algorithm_def.get_training_preprocessing()
    train_data, validation_data = get_dataset(data_dir, batch_size, f_train, f_val, train_val_split, dataset_name, **kwargs)
    model = algorithm_def.get_training_model()
    print_flat_summary(model)

    tb_c = keras.callbacks.TensorBoard(log_dir=str(working_dir))
    mc_c = keras.callbacks.ModelCheckpoint(str(working_dir / "weights-improvement-{epoch:03d}.hdf5"), monitor="val_loss",
                                           mode="min", save_best_only=True)  # reduce storage space
    mc_c_epochs = keras.callbacks.ModelCheckpoint(str(working_dir / "weights-{epoch:03d}.hdf5"), period=save_checkpoint_every_n_epochs)  #
    callbacks = [tb_c, mc_c, mc_c_epochs]

    # Trains the model
    model.fit_generator(
            generator=train_data,
            steps_per_epoch=len(train_data),
            validation_data=validation_data,
            validation_steps=len(validation_data),
            epochs=epochs,
            callbacks=callbacks
        )


def main():
    init(train_model)


if __name__ == "__main__":
    main()
```

```
!python train.py

2021-06-21 04:54:59.308846: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic li
Traceback (most recent call last):
  File "train.py", line 1, in <module>
    from self_supervised_3d_tasks.train import main
  File "/content/self-supervised-3d-tasks/self_supervised_3d_tasks/train.py", line 2, in <module>
    from self_supervised_3d_tasks.utils.model_utils import init, print_flat_summary
  File "/content/self-supervised-3d-tasks/self_supervised_3d_tasks/utils/model_utils.py", line 16, in <module>
    from tensorflow_core.python.keras.layers import Wrapper, UpSampling2D
ModuleNotFoundError: No module named 'tensorflow_core'
```

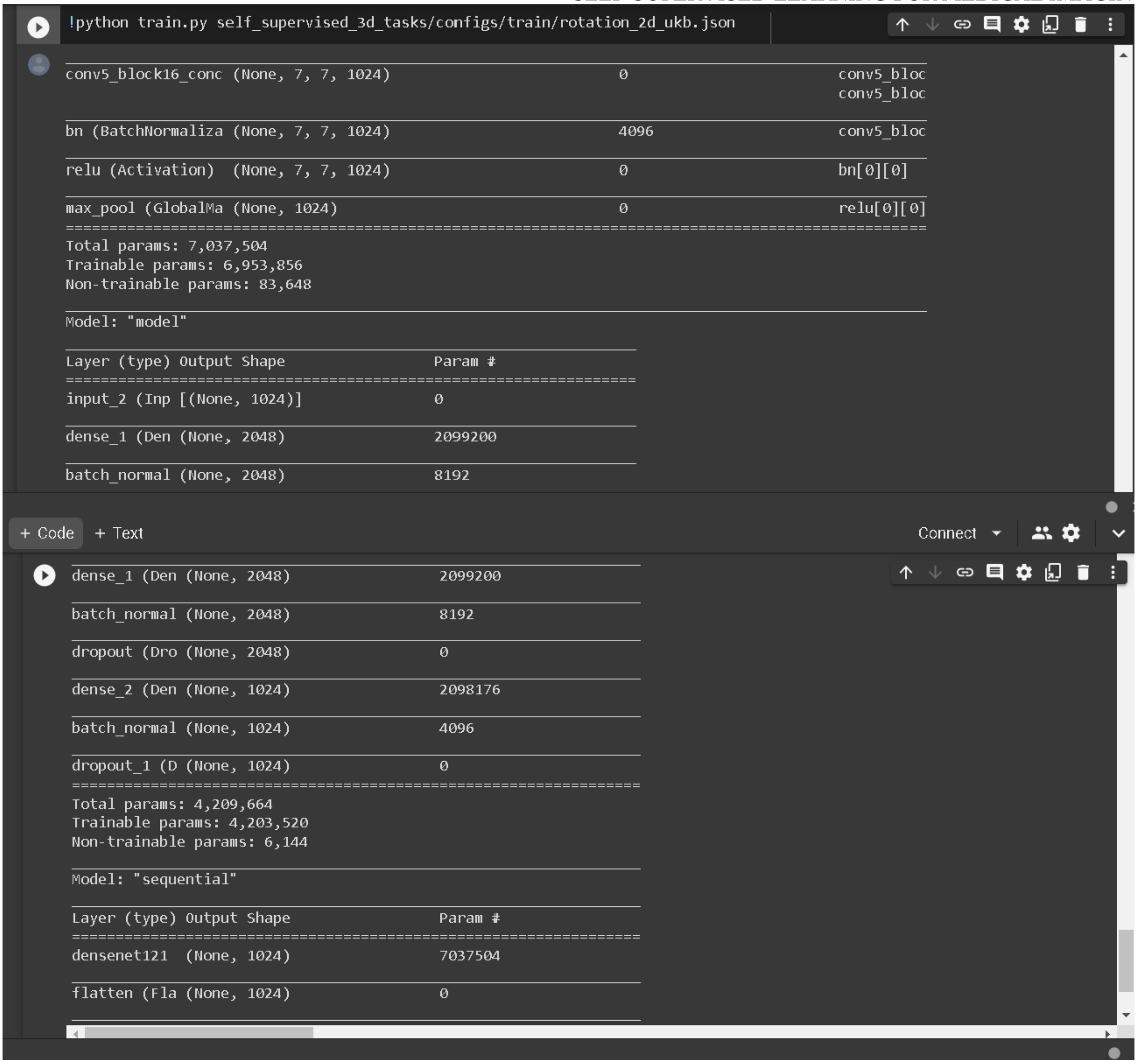
```
[ ] !python train.py self_supervised_3d_tasks/configs/train/rotation_2d_ukb.json

2021-07-04 13:17:54.648632: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic libra
2021-07-04 13:17:54.648765: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic libra
2021-07-04 13:17:54.648824: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:30] Cannot dlopen some TensorRT librarie
###########################################
training {'algorithm': 'rotation', 'data_dir': '/content/self-supervised-3d-tasks/fundus_train', 'train_data_generator_a
###########################################
GPU usage:
  memory.used memory.free
0       0 MiB   11441 MiB
USING GPU:
0
writing to: /root/netstore/workspace/rotation_ukb2d_1
Model: "densenet121"
```

Layer (type)	Output Shape	Param #	Connected
input_1 (InputLaye	[(None, 224, 224, 3)]	0	
zero_padding2d (Ze	(None, 230, 230, 3)	0	input_1[0]
conv1/conv (Conv2D	(None, 112, 112, 64)	9408	zero_paddi
conv1/bn (BatchNor	(None, 112, 112, 64)	256	conv1/conv

```
!python train.py self_supervised_3d_tasks/configs/train/rotation_2d_ukb.json
```

conv1/bn (BatchNor	(None, 112, 112, 64)	256	conv1/conv
conv1/relu (Activa	(None, 112, 112, 64)	0	conv1/bn[0
zero_padding2d_1 (	(None, 114, 114, 64)	0	conv1/relu
pool1 (MaxPooling2	(None, 56, 56, 64)	0	zero_paddi
conv2_block1_0_bn	(None, 56, 56, 64)	256	pool1[0][0
conv2_block1_0_rel	(None, 56, 56, 64)	0	conv2_bloc
conv2_block1_1_con	(None, 56, 56, 128)	8192	conv2_bloc
conv2_block1_1_bn	(None, 56, 56, 128)	512	conv2_bloc
conv2_block1_1_rel	(None, 56, 56, 128)	0	conv2_bloc
conv2_block1_2_con	(None, 56, 56, 32)	36864	conv2_bloc
conv2_block1_conca	(None, 56, 56, 96)	0	pool1[0][0 conv2_bloc
conv2_block2_0_bn	(None, 56, 56, 96)	384	conv2_bloc



```
!python train.py self_supervised_3d_tasks/configs/train/rotation_2d_ukb.json

conv5_block16_conc (None, 7, 7, 1024)          0          conv5_bloc
                                                          conv5_bloc

bn (BatchNormaliza (None, 7, 7, 1024)          4096       conv5_bloc

relu (Activation)  (None, 7, 7, 1024)          0          bn[0][0]

max_pool (GlobalMa (None, 1024)                0          relu[0][0]
==================================================================================
Total params: 7,037,504
Trainable params: 6,953,856
Non-trainable params: 83,648
__________________________________________________________________________________
Model: "model"
_________________________________________________________________
Layer (type) Output Shape                 Param #
=================================================================
input_2 (Inp [(None, 1024)]               0
_________________________________________________________________
dense_1 (Den (None, 2048)                 2099200
_________________________________________________________________
batch_normal (None, 2048)                 8192
```

```
dense_1 (Den (None, 2048)                 2099200
_________________________________________________________________
batch_normal (None, 2048)                 8192
_________________________________________________________________
dropout (Dro (None, 2048)                 0
_________________________________________________________________
dense_2 (Den (None, 1024)                 2098176
_________________________________________________________________
batch_normal (None, 1024)                 4096
_________________________________________________________________
dropout_1 (D (None, 1024)                 0
=================================================================
Total params: 4,209,664
Trainable params: 4,203,520
Non-trainable params: 6,144
_________________________________________________________________
Model: "sequential"
_________________________________________________________________
Layer (type) Output Shape                 Param #
=================================================================
densenet121  (None, 1024)                 7037504
_________________________________________________________________
flatten (Fla (None, 1024)                 0
_________________________________________________________________
```

4.4 FINETUNING THE MODEL

```
import csv
import functools
import gc
import os
import random
from os.path import expanduser
from pathlib import Path

import numpy as np
import tensorflow as tf
from tensorflow.keras import backend as K
from tensorflow.keras.optimizers import Adam
from tensorflow.python.keras import Model
from tensorflow.python.keras.callbacks import CSVLogger

import self_supervised_3d_tasks.utils.metrics as metrics
from self_supervised_3d_tasks.utils.callbacks import TerminateOnNaN, NaNLossError, LogCSVWithStart
from self_supervised_3d_tasks.utils.metrics import weighted_sum_loss, jaccard_distance, \
    weighted_categorical_crossentropy, weighted_dice_coefficient, weighted_dice_coefficient_loss, \
    weighted_dice_coefficient_per_class, brats_wt_metric, brats_et_metric, brats_tc_metric
from self_supervised_3d_tasks.test_data_backend import CvDataKaggle, StandardDataLoader
from self_supervised_3d_tasks.train import (
    keras_algorithm_list,
)
from self_supervised_3d_tasks.utils.model_utils import (
    apply_prediction_model,
    get_writing_path,
    print_flat_summary)
from self_supervised_3d_tasks.utils.model_utils import init


def get_score(score_name):
    if score_name == "qw_kappa":
        return metrics.score_kappa
    elif score_name == "bin_accuracy":
        return metrics.score_bin_acc
    elif score_name == "cat_accuracy":
        return metrics.score_cat_acc
    elif score_name == "dice":
        return metrics.score_dice
    elif score_name == "dice_pancreas_0":
        return functools.partial(metrics.score_dice_class, class_to_predict=0)
    elif score_name == "dice_pancreas_1":
        return functools.partial(metrics.score_dice_class, class_to_predict=1)
    elif score_name == "dice_pancreas_2":
        return functools.partial(metrics.score_dice_class, class_to_predict=2)
    elif score_name == "jaccard":
        return metrics.score_jaccard
```

```python
    elif score_name == "qw_kappa_kaggle":
        return metrics.score_kappa_kaggle
    elif score_name == "cat_acc_kaggle":
        return metrics.score_cat_acc_kaggle
    elif score_name == "brats_wt":
        return metrics.brats_wt
    elif score_name == "brats_tc":
        return metrics.brats_tc
    elif score_name == "brats_et":
        return metrics.brats_et
    else:
        raise ValueError(f"score {score_name} not found")


def make_custom_metrics(metrics):
    metrics = list(metrics)

    if "weighted_dice_coefficient" in metrics:
        metrics.remove("weighted_dice_coefficient")
        metrics.append(weighted_dice_coefficient)
    if "brats_metrics" in metrics:
        metrics.remove("brats_metrics")
        metrics.append(brats_wt_metric)
        metrics.append(brats_tc_metric)
        metrics.append(brats_et_metric)
    if "weighted_dice_coefficient_per_class_pancreas" in metrics:
        metrics.remove("weighted_dice_coefficient_per_class_pancreas")

        def dice_class_0(y_true, y_pred):
            return weighted_dice_coefficient_per_class(y_true, y_pred, class_to_predict=0)

        def dice_class_1(y_true, y_pred):
            return weighted_dice_coefficient_per_class(y_true, y_pred, class_to_predict=1)

        def dice_class_2(y_true, y_pred):
            return weighted_dice_coefficient_per_class(y_true, y_pred, class_to_predict=2)

        metrics.append(dice_class_0)
        metrics.append(dice_class_1)
        metrics.append(dice_class_2)

    return metrics


def make_custom_loss(loss):
    if loss == "weighted_sum_loss":
        loss = weighted_sum_loss()
    elif loss == "jaccard_distance":
        loss = jaccard_distance
    elif loss == "weighted_dice_loss":
```

```
        loss = weighted_dice_coefficient_loss
    elif loss == "weighted_categorical_crossentropy":
        loss = weighted_categorical_crossentropy()

    return loss


def get_optimizer(clipnorm, clipvalue, lr):
    if clipnorm is None and clipvalue is None:
        return Adam(lr=lr)
    elif clipnorm is None:
        return Adam(lr=lr, clipvalue=clipvalue)
    else:
        return Adam(lr=lr, clipnorm=clipnorm, clipvalue=clipvalue)


def make_scores(y, y_pred, scores):
    scores_f = [(x, get_score(x)(y, y_pred)) for x in scores]
    return scores_f


def run_single_test(algorithm_def, gen_train, gen_val, load_weights, freeze_weights, x_test, y_test, lr,
                    batch_size, epochs, epochs_warmup, model_checkpoint, scores, loss, metrics, logging_path, kwargs,
                    clipnorm=None, clipvalue=None, model_callback=None, working_dir=None):
    print(metrics)
    print(loss)

    metrics = make_custom_metrics(metrics)
    loss = make_custom_loss(loss)

    if load_weights:
        enc_model = algorithm_def.get_finetuning_model(model_checkpoint)
    else:
        enc_model = algorithm_def.get_finetuning_model()

    pred_model = apply_prediction_model(input_shape=enc_model.outputs[0].shape[1:], algorithm_instance=algorithm_def,
                                        **kwargs)

    outputs = pred_model(enc_model.outputs)
    model = Model(inputs=enc_model.inputs[0], outputs=outputs)
    print_flat_summary(model)

    if epochs > 0:
        callbacks = [TerminateOnNaN()]

        logging_csv = False
        if logging_path is not None:
            logging_csv = True
            logging_path.parent.mkdir(exist_ok=True, parents=True)
            logger_normal = CSVLogger(str(logging_path), append=False)
```

```
            logger_after_warmup = LogCSVWithStart(str(logging_path), start_from_epoch=epochs_warmup, append=True)
        if freeze_weights or load_weights:
            enc_model.trainable = False

        if freeze_weights:
            print(("-" * 10) + "LOADING weights, encoder model is completely frozen")
            if logging_csv:
                callbacks.append(logger_normal)
        elif load_weights:
            assert epochs_warmup < epochs, "warmup epochs must be smaller than epochs"

            print(
                ("-" * 10) + "LOADING weights, encoder model is trainable after warm-up"
            )
            print(("-" * 5) + " encoder model is frozen")

            w_callbacks = list(callbacks)
            if logging_csv:
                w_callbacks.append(logger_normal)

            model.compile(optimizer=get_optimizer(clipnorm, clipvalue, lr), loss=loss, metrics=metrics)
            model.fit(
                x=gen_train,
                validation_data=gen_val,
                epochs=epochs_warmup,
                callbacks=w_callbacks,
            )
            epochs = epochs - epochs_warmup

            enc_model.trainable = True
            print(("-" * 5) + " encoder model unfrozen")

            if logging_csv:
                callbacks.append(logger_after_warmup)
        else:
            print(("-" * 10) + "RANDOM weights, encoder model is fully trainable")
            if logging_csv:
                callbacks.append(logger_normal)

        if working_dir is not None:
            save_checkpoint_every_n_epochs = 5
            mc_c = tf.keras.callbacks.ModelCheckpoint(str(working_dir / "weights-improvement-{epoch:03d}.hdf5"),
                                                      monitor="val_loss",
                                                      mode="min", save_best_only=True)  # reduce storage space
            mc_c_epochs = tf.keras.callbacks.ModelCheckpoint(str(working_dir / "weights-{epoch:03d}.hdf5"),
                                                             period=save_checkpoint_every_n_epochs)  # reduce storage space
            callbacks.append(mc_c)
            callbacks.append(mc_c_epochs)

        # recompile model
```

```
        model.compile(optimizer=get_optimizer(clipnorm, clipvalue, lr), loss=loss, metrics=metrics)
        model.fit(
            x=gen_train, validation_data=gen_val, epochs=epochs, callbacks=callbacks
        )

    model.compile(optimizer=get_optimizer(clipnorm, clipvalue, lr), loss=loss, metrics=metrics)
    y_pred = model.predict(x_test, batch_size=batch_size)
    scores_f = make_scores(y_test, y_pred, scores)

    if model_callback:
        model_callback(model)

    # cleanup
    del pred_model
    del enc_model
    del model

    algorithm_def.purge()
    K.clear_session()

    for i in range(15):
        gc.collect()

    for s in scores_f:
        print("{} score: {}".format(s[0], s[1]))
```

```
    return scores_f


def write_result(base_path, row):
    with open(base_path / "results.csv", "a") as csvfile:
        result_writer = csv.writer(csvfile, delimiter=",")
        result_writer.writerow(row)


class MaxTriesExceeded(Exception):
    def __init__(self, func, *args):
        self.func = func
        if args:
            self.max_tries = args[0]

    def __str__(self):
        return f'Maximum amount of tries ({self.max_tries}) exceeded for {self.func}.'


def try_until_no_nan(func, max_tries=4):
    for _ in range(max_tries):
        try:
            return func()
        except NaNLossError:
            print(f"Encountered NaN-Loss in {func}")
```

```
        raise MaxTriesExceeded(func, max_tries)


def run_complex_test(
        algorithm,
        dataset_name,
        root_config_file,
        model_checkpoint,
        epochs_initialized=5,
        epochs_random=5,
        epochs_frozen=5,
        repetitions=2,
        batch_size=8,
        exp_splits=(100, 10, 1),
        lr=1e-3,
        epochs_warmup=2,
        scores=("qw_kappa",),
        loss="mse",
        metrics=("mse",),
        clipnorm=None,
        clipvalue=None,
        do_cross_val=False,
        **kwargs,
):
    model_checkpoint = expanduser(model_checkpoint)
    if os.path.isdir(model_checkpoint):
        weight_files = list(Path(model_checkpoint).glob("weights-improvement*.hdf5"))

        if epochs_initialized > 0 or epochs_frozen > 0:
            assert len(weight_files) > 0, "empty directory!"

        weight_files.sort()
        model_checkpoint = str(weight_files[-1])

    kwargs["model_checkpoint"] = model_checkpoint
    kwargs["root_config_file"] = root_config_file
    metrics = list(metrics)

    working_dir = get_writing_path(
        Path(model_checkpoint).expanduser().parent
        / (Path(model_checkpoint).expanduser().stem + "_test"),
        root_config_file,
    )

    algorithm_def = keras_algorithm_list[algorithm].create_instance(**kwargs)

    results = []
    header = ["Train Split"]

    exp_types = []
```

```
    if epochs_frozen > 0:
        exp_types.append("Weights_frozen_")

    if epochs_initialized > 0:
        exp_types.append("Weights_initialized_")

    if epochs_random > 0:
        exp_types.append("Weights_random_")

    for exp_type in exp_types:
        for sc in scores:
            for min_avg_max in ["_min", "_avg", "_max"]:
                header.append(exp_type + sc + min_avg_max)

    write_result(working_dir, header)

    if do_cross_val:
        data_loader = CvDataKaggle(dataset_name, batch_size, algorithm_def, n_repetitions=repetitions, **kwargs)
    else:
        data_loader = StandardDataLoader(dataset_name, batch_size, algorithm_def, **kwargs)

    for train_split in exp_splits:
        percentage = 0.01 * train_split
        print("\n--------------------")
        print("running test for: {}%".format(train_split))
        print("--------------------\n")

        a_s = []
        b_s = []
        c_s = []

        for i in range(repetitions):
            logging_base_path = working_dir / "logs"

            # Use the same seed for all experiments in one repetition
            tf.random.set_seed(i)
            np.random.seed(i)
            random.seed(i)

            gen_train, gen_val, x_test, y_test = data_loader.get_dataset(i, percentage)

            if epochs_frozen > 0:
                logging_a_path = logging_base_path / f"split{train_split}frozen_rep{i}.log"
                a = try_until_no_nan(
                    lambda: run_single_test(algorithm_def, gen_train, gen_val, True, True, x_test, y_test, lr,
                                            batch_size, epochs_frozen, epochs_warmup, model_checkpoint, scores, loss,
                                            metrics,
                                            logging_a_path,
                                            kwargs, clipnorm=clipnorm, clipvalue=clipvalue))  # frozen
```

```
            a_s.append(a)
        if epochs_initialized > 0:
            logging_b_path = logging_base_path / f"split{train_split}initialized_rep{i}.log"
            b = try_until_no_nan(
                lambda: run_single_test(algorithm_def, gen_train, gen_val, True, False, x_test, y_test, lr,
                                        batch_size, epochs_initialized, epochs_warmup, model_checkpoint, scores,
                                        loss, metrics,
                                        logging_b_path, kwargs, clipnorm=clipnorm, clipvalue=clipvalue))
            b_s.append(b)
        if epochs_random > 0:
            logging_c_path = logging_base_path / f"split{train_split}random_rep{i}.log"
            c = try_until_no_nan(
                lambda: run_single_test(algorithm_def, gen_train, gen_val, False, False, x_test, y_test, lr,
                                        batch_size, epochs_random, epochs_warmup, model_checkpoint, scores, loss,
                                        metrics,
                                        logging_c_path,
                                        kwargs, clipnorm=clipnorm, clipvalue=clipvalue,
                                        working_dir=working_dir))  # random
            c_s.append(c)

    def get_avg_score(list_abc, index):
        sc = [x[index][1] for x in list_abc]
        return np.mean(np.array(sc))

    def get_min_score(list_abc, index):
```

```
    def get_min_score(list_abc, index):
        sc = [x[index][1] for x in list_abc]
        return np.min(np.array(sc))

    def get_max_score(list_abc, index):
        sc = [x[index][1] for x in list_abc]
        return np.max(np.array(sc))

    scores_a = []
    scores_b = []
    scores_c = []

    for i in range(len(scores)):
        if epochs_frozen > 0:
            scores_a.append(get_min_score(a_s, i))
            scores_a.append(get_avg_score(a_s, i))
            scores_a.append(get_max_score(a_s, i))

        if epochs_initialized > 0:
            scores_b.append(get_min_score(b_s, i))
            scores_b.append(get_avg_score(b_s, i))
            scores_b.append(get_max_score(b_s, i))

        if epochs_random > 0:
```

```
                scores_c.append(get_avg_score(c_s, i))
                scores_c.append(get_max_score(c_s, i))

        data = [str(train_split) + "%"]

        if epochs_frozen > 0:
            data += scores_a

        if epochs_initialized > 0:
            data += scores_b

        if epochs_random > 0:
            data += scores_c

        results.append(data)
        write_result(working_dir, data)


def main():
    init(run_complex_test, "test")


if __name__ == "__main__":
    main()
```

```
[ ] !python finetune.py self_supervised_3d_tasks/configs/finetune/rotation_2d.json
Epoch 10/17
44/44 [==============================] - 31s 714ms/step - loss: 0.4451 - accuracy: 0.8069 - val_loss: 0.3405 - val_accu
Epoch 11/17
44/44 [==============================] - 31s 714ms/step - loss: 0.4347 - accuracy: 0.8078 - val_loss: 0.3247 - val_accu
Epoch 12/17
44/44 [==============================] - 31s 713ms/step - loss: 0.4081 - accuracy: 0.8282 - val_loss: 0.3318 - val_accu
Epoch 13/17
44/44 [==============================] - 31s 713ms/step - loss: 0.3888 - accuracy: 0.8462 - val_loss: 0.3375 - val_accu
Epoch 14/17
44/44 [==============================] - 32s 716ms/step - loss: 0.3682 - accuracy: 0.8562 - val_loss: 0.2840 - val_accu
Epoch 15/17
44/44 [==============================] - 31s 715ms/step - loss: 0.3624 - accuracy: 0.8590 - val_loss: 0.2936 - val_accu
Epoch 16/17
44/44 [==============================] - 31s 715ms/step - loss: 0.3447 - accuracy: 0.8683 - val_loss: 0.2968 - val_accu
Epoch 17/17
44/44 [==============================] - 31s 713ms/step - loss: 0.3369 - accuracy: 0.8687 - val_loss: 0.3283 - val_accu
qw_kappa_kaggle score: 0.7365178391296926
cat_acc_kaggle score: 0.639344262295082

--------------------
running test for: 25%
```

```
!python finetune.py self_supervised_3d_tasks/configs/finetune/rotation_2d.json
running test for: 25%
--------------------

Loading Test data
95.65%
['accuracy']
binary_crossentropy
Model: "model_1"
_________________________________________________________________
Layer (type) Output Shape                  Param #
=================================================================
input_3 (Inp [(None, 1024)]                0
_________________________________________________________________
dense_3 (Den (None, 1024)                  1049600
_________________________________________________________________
batch_normal (None, 1024)                  4096
_________________________________________________________________
dropout_2 (D (None, 1024)                  0
_________________________________________________________________
dense_4 (Den (None, 5)                     5125
=================================================================
Total params: 1,058,821
Trainable params: 1,056,773
```

```
!python finetune.py self_supervised_3d_tasks/configs/finetune/rotation_2d.json

Total params: 1,058,821
Trainable params: 1,056,773
Non-trainable params: 2,048
_________________________________________________________________
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)          Output Shape                    Param #        Connected
==================================================================================================
input_1 (InputLaye [(None, 224, 224, 3)]              0

zero_padding2d (Ze (None, 230, 230, 3)                0              input_1[0]

conv1/conv (Conv2D (None, 112, 112, 64)               9408           zero_paddi

conv1/bn (BatchNor (None, 112, 112, 64)               256            conv1/conv

conv1/relu (Activa (None, 112, 112, 64)               0              conv1/bn[0

zero_padding2d_1 ( (None, 114, 114, 64)               0              conv1/relu

pool1 (MaxPooling2 (None, 56, 56, 64)                 0              zero_paddi
```

```
!python finetune.py self_supervised_3d_tasks/configs/finetune/rotation_2d.json
22/22 [============================] - 16s 749ms/step - loss: 0.4609 - accuracy: 0.7790 - val_loss: 0.3896 - val_accu
Epoch 16/17
22/22 [============================] - 16s 748ms/step - loss: 0.4709 - accuracy: 0.7796 - val_loss: 0.3773 - val_accu
Epoch 17/17
22/22 [============================] - 16s 748ms/step - loss: 0.4360 - accuracy: 0.8000 - val_loss: 0.4739 - val_accu
qw_kappa_kaggle score: 0.5708044299692636
cat_acc_kaggle score: 0.33560709413369716
Loading Test data
95.65%
['accuracy']
binary_crossentropy
Model: "model_1"
_________________________________________________________________
Layer (type) Output Shape              Param #
=================================================================
input_3 (Inp [(None, 1024)]            0
_________________________________________________________________
dense_3 (Den (None, 1024)              1049600
_________________________________________________________________
batch_normal (None, 1024)              4096
_________________________________________________________________
dropout_2 (D (None, 1024)              0
```

```
bn (BatchNormaliza (None, 7, 7, 1024)                 4096           conv5_

relu (Activation)  (None, 7, 7, 1024)                 0              bn[0][0]

avg_pool (GlobalAv (None, 1024)                       0              relu[0][0]

model_1 (Model)    (None, 5)                          1058821        avg_pool[0
==================================================================================================
Total params: 8,096,325
Trainable params: 8,010,629
Non-trainable params: 85,696
__________________________________________________________________________________________________
----------LOADING weights, encoder model is trainable after warm-up
----- encoder model is frozen
Train for 22 steps, validate for 5 steps
Epoch 1/3
22/22 [============================] - 21s 933ms/step - loss: 0.8392 - accuracy: 0.5381 - val_loss: 0.7164 - val_accu
Epoch 2/3
22/22 [============================] - 14s 615ms/step - loss: 0.7577 - accuracy: 0.5922 - val_loss: 0.6908 - val_accu
Epoch 3/3
22/22 [============================] - 14s 617ms/step - loss: 0.7374 - accuracy: 0.6009 - val_loss: 0.6680 - val_accu
----- encoder model unfrozen
Train for 22 steps, validate for 5 steps
Epoch 1/17
22/22 [============================] - 30s 1s/step - loss: 0.6991 - accuracy: 0.6363 - val_loss: 0.6887 - val_accurac
```

```
22/22 [==============================] - 14s 617ms/step - loss: 0.7374 - accuracy: 0.6009 - val_loss: 0.6680 - val_accu
----- encoder model unfrozen
Train for 22 steps, validate for 5 steps
Epoch 1/17
22/22 [==============================] - 30s 1s/step - loss: 0.6991 - accuracy: 0.6363 - val_loss: 0.6887 - val_accurac
Epoch 2/17
22/22 [==============================] - 16s 748ms/step - loss: 0.6484 - accuracy: 0.6529 - val_loss: 0.6420 - val_accu
Epoch 3/17
22/22 [==============================] - 16s 750ms/step - loss: 0.6413 - accuracy: 0.6665 - val_loss: 0.6595 - val_accu
Epoch 4/17
22/22 [==============================] - 16s 748ms/step - loss: 0.6052 - accuracy: 0.6863 - val_loss: 0.6780 - val_accu
Epoch 5/17
22/22 [==============================] - 16s 750ms/step - loss: 0.5910 - accuracy: 0.6961 - val_loss: 0.5854 - val_accu
Epoch 6/17
22/22 [==============================] - 16s 746ms/step - loss: 0.5829 - accuracy: 0.7047 - val_loss: 0.6049 - val_accu
Epoch 7/17
22/22 [==============================] - 16s 747ms/step - loss: 0.5758 - accuracy: 0.7168 - val_loss: 0.5786 - val_accu
Epoch 8/17
22/22 [==============================] - 16s 748ms/step - loss: 0.5514 - accuracy: 0.7160 - val_loss: 0.5349 - val_accu
Epoch 9/17
22/22 [==============================] - 16s 747ms/step - loss: 0.5446 - accuracy: 0.7206 - val_loss: 0.5201 - val_accu
Epoch 10/17
22/22 [==============================] - 17s 750ms/step - loss: 0.5342 - accuracy: 0.7309 - val_loss: 0.5357 - val_accu
Epoch 11/17
22/22 [==============================] - 16s 747ms/step - loss: 0.5165 - accuracy: 0.7505 - val_loss: 0.5573 - val_accu
Epoch 12/17
```

```
22/22 [==============================] - 16s 747ms/step - loss: 0.5165 - accuracy: 0.7505 - val_loss: 0.5573 - val_accu
Epoch 12/17
22/22 [==============================] - 16s 750ms/step - loss: 0.5072 - accuracy: 0.7462 - val_loss: 0.6714 - val_accu
Epoch 13/17
22/22 [==============================] - 16s 746ms/step - loss: 0.5018 - accuracy: 0.7603 - val_loss: 0.6110 - val_accu
Epoch 14/17
22/22 [==============================] - 16s 748ms/step - loss: 0.4689 - accuracy: 0.7704 - val_loss: 0.5838 - val_accu
Epoch 15/17
22/22 [==============================] - 16s 748ms/step - loss: 0.4601 - accuracy: 0.7908 - val_loss: 0.4933 - val_accu
Epoch 16/17
22/22 [==============================] - 16s 743ms/step - loss: 0.4666 - accuracy: 0.7822 - val_loss: 0.4722 - val_accu
Epoch 17/17
22/22 [==============================] - 16s 748ms/step - loss: 0.4494 - accuracy: 0.7865 - val_loss: 0.4693 - val_accu
qw_kappa_kaggle score: 0.568897900348861
cat_acc_kaggle score: 0.5423497267759563
```

4.5 TESTING THE DATA

```
from self_supervised_3d_tasks.data.kaggle_retina_data import get_kaggle_generator, get_kaggle_cross_validation
from self_supervised_3d_tasks.data.make_data_generator import get_data_generators
from self_supervised_3d_tasks.data.numpy_2d_loader import Numpy2DLoader
from self_supervised_3d_tasks.data.segmentation_task_loader import SegmentationGenerator3D, PatchSegmentationGenerator3D
import numpy as np


def get_dataset_regular_train(
        batch_size,
        f_train,
        f_val,
        train_split,
        data_generator,
        data_dir_train,
        val_split=0.1,
        train_data_generator_args={},
        val_data_generator_args={},
        **kwargs,
):
    train_split = train_split * (1 - val_split)  # normalize train split

    train_data_generator, val_data_generator, _ = get_data_generators(
        data_generator=data_generator,
        data_path=data_dir_train,
        train_split=train_split,
```

```
        val_split=val_split,  # we are eventually not using the full dataset here
        train_data_generator_args={
            **{"batch_size": batch_size, "pre_proc_func": f_train},
            **train_data_generator_args,
        },
        val_data_generator_args={
            **{"batch_size": batch_size, "pre_proc_func": f_val},
            **val_data_generator_args,
        },
        **kwargs,
    )
    return train_data_generator, val_data_generator


def get_dataset_regular_test(
        batch_size,
        f_test,
        data_generator,
        data_dir_test,
        train_data_generator_args={},
        test_data_generator_args={},
        **kwargs,
):
    if "val_split" in kwargs:
        del kwargs["val_split"]
    return get_data_generators(
        data_generator=data_generator,
        data_path=data_dir_test,
        train_data_generator_args={
            **{"batch_size": batch_size, "pre_proc_func": f_test},
            **test_data_generator_args,
        },
        **kwargs,
    )


def get_dataset_kaggle_train_original(
        batch_size,
        f_train,
        f_val,
        train_split,
        csv_file_train,
        data_dir,
        val_split=0.1,
        train_data_generator_args={},
        val_data_generator_args={},
        **kwargs,
):
    train_split = train_split * (1 - val_split)  # normalize train split
    train_data_generator, val_data_generator, _ = get_kaggle_generator(
```

```
        data_path=data_dir,
        csv_file=csv_file_train,
        train_split=train_split,
        val_split=val_split,  # we are eventually not using the full dataset here
        train_data_generator_args={
            **{"batch_size": batch_size, "pre_proc_func": f_train},
            **train_data_generator_args,
        },
        val_data_generator_args={
            **{"batch_size": batch_size, "pre_proc_func": f_val},
            **val_data_generator_args,
        },
        **kwargs,
    )
    return train_data_generator, val_data_generator


def get_dataset_kaggle_test(
        batch_size,
        f_test,
        csv_file_test,
        data_dir,
        train_data_generator_args={},  # DO NOT remove
        test_data_generator_args={},
        **kwargs,
):
    if "val_split" in kwargs:
        del kwargs["val_split"]

    return get_kaggle_generator(
        data_path=data_dir,
        csv_file=csv_file_test,
        train_data_generator_args={
            **{"batch_size": batch_size, "pre_proc_func": f_test},
            **test_data_generator_args,
        },
        **kwargs,
    )


def get_data_from_gen(gen):
    print("Loading Test data")

    data = None
    labels = None
    max_iter = len(gen)
    i = 0
    for d, l in gen:
        if data is None:
            data = d
```

```
                labels = l
            else:
                data = np.concatenate((data, d), axis=0)
                labels = np.concatenate((labels, l), axis=0)

            print(f"\r{(i * 100.0) / max_iter:.2f}%", end="")
            i += 1
            if i == max_iter:
                break

        print("")

        return data, labels


    def get_dataset_train(dataset_name, batch_size, f_train, f_val, train_split, kwargs):
        if dataset_name == "kaggle_retina":
            return get_dataset_kaggle_train_original(
                batch_size, f_train, f_val, train_split, **kwargs
            )
        elif dataset_name == "pancreas3d":
            return get_dataset_regular_train(
                batch_size, f_train, f_val, train_split, data_generator=SegmentationGenerator3D, **kwargs,
            )
        elif dataset_name == 'brats' or dataset_name == 'ukb3d':
            return get_dataset_regular_train(
                batch_size, f_train, f_val, train_split, data_generator=PatchSegmentationGenerator3D, **kwargs,
            )
        elif dataset_name == "pancreas2d":
            return get_dataset_regular_train(
                batch_size, f_train, f_val, train_split, data_generator=Numpy2DLoader, **kwargs,
            )
        else:
            raise ValueError("not implemented")


    def get_dataset_test(dataset_name, batch_size, f_test, kwargs):
        if dataset_name == "kaggle_retina":
            gen_test = get_dataset_kaggle_test(batch_size, f_test, **kwargs)
        elif dataset_name == "pancreas3d":
            gen_test = get_dataset_regular_test(
                batch_size, f_test, data_generator=SegmentationGenerator3D, **kwargs
            )
        elif dataset_name == 'brats' or dataset_name == 'ukb3d':
            gen_test = get_dataset_regular_test(
                batch_size, f_test, data_generator=PatchSegmentationGenerator3D, **kwargs
            )
        elif dataset_name == "pancreas2d":
            gen_test = get_dataset_regular_test(
                batch_size, f_test, data_generator=Numpy2DLoader, **kwargs,
```

```
        )
    else:
        raise ValueError("not implemented")

    return get_data_from_gen(gen_test)


class StandardDataLoader:
    def __init__(self, dataset_name, batch_size, algorithm_def,
                 **kwargs):
        self.algorithm_def = algorithm_def
        self.batch_size = batch_size
        self.dataset_name = dataset_name
        self.kwargs = kwargs

    def get_dataset(self, repetition, train_split):
        f_train, f_val = self.algorithm_def.get_finetuning_preprocessing()

        gen_train, gen_val = get_dataset_train(
            self.dataset_name, self.batch_size, f_train, f_val, train_split, self.kwargs
        )

        x_test, y_test = get_dataset_test(self.dataset_name, self.batch_size, f_val, self.kwargs)
        return gen_train, gen_val, x_test, y_test

class CvDataKaggle:
    def __init__(self, dataset_name, batch_size, algorithm_def,
                 n_repetitions,
                 csv_file,
                 data_dir,
                 val_split=0.1,
                 test_data_generator_args={},
                 val_data_generator_args={},
                 train_data_generator_args={},
                 **kwargs):
        assert dataset_name == "kaggle_retina", "CV only implemented for kaggle so far"

        f_train, f_val = algorithm_def.get_finetuning_preprocessing()
        self.cv = get_kaggle_cross_validation(data_path=data_dir, csv_file=csv_file,
                                              k_fold=n_repetitions,
                                              train_data_generator_args={
                                                  **{"batch_size": batch_size, "pre_proc_func": f_train},
                                                  **train_data_generator_args,
                                              },
                                              val_data_generator_args={
                                                  **{"batch_size": batch_size, "pre_proc_func": f_val},
                                                  **val_data_generator_args,
                                              },
                                              test_data_generator_args={
                                                  **{"batch_size": batch_size, "pre_proc_func": f_val},
                                                  **test_data_generator_args,
                                              }, **kwargs)
        self.val_split = val_split

    def get_dataset(self, repetition, train_split):
        train_split = train_split * (1 - self.val_split)  # normalize train split

        gen_train, gen_val, gen_test = self.cv.make_generators(test_chunk=repetition, train_split=train_split,
                                                               val_split=self.val_split)

        x_test, y_test = get_data_from_gen(gen_test)
        return gen_train, gen_val, x_test, y_test
```

5. RESULTS

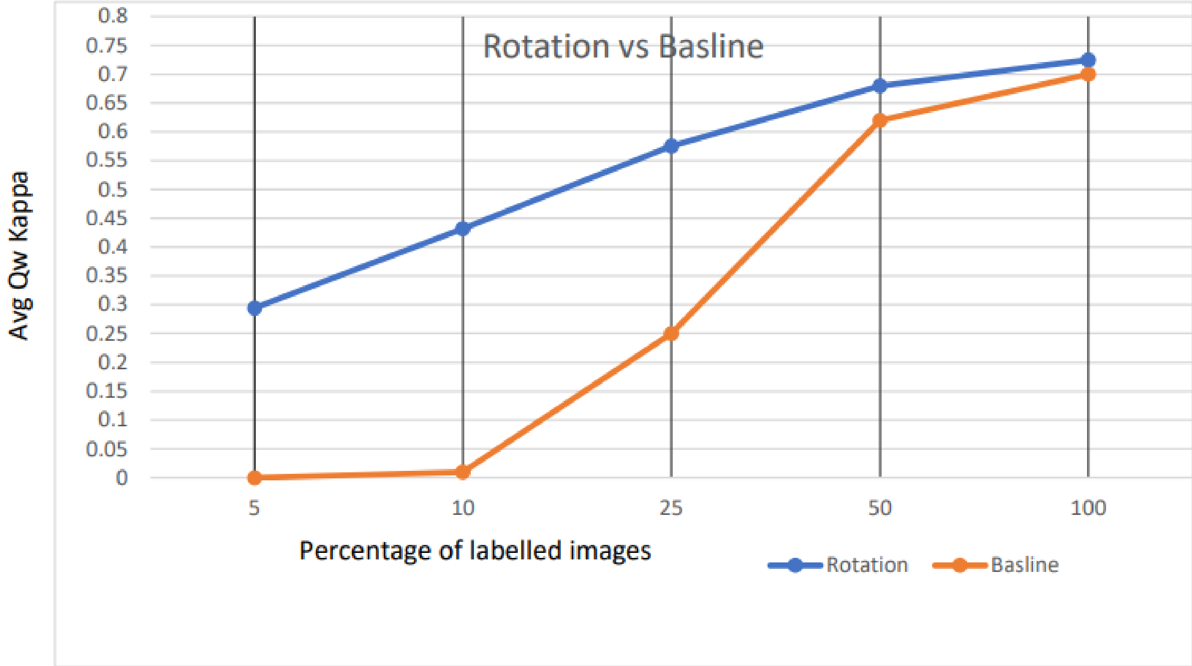
5.RESULTS

The final model detects and classifies the provided retinal fundus data into one of the five types mentioned earlier. The Kaggle dataset contains roughly 3600 images each of them rated by clinician on the scale of 0 to 4 (NO DR, mild, moderate, severe, proliferate). To evaluate our tasks on this benchmark we pretrained the model with all the images of the dataset. And then finetuned them on the same Kaggle data but with different sizes of subsets, which can be considered as a data efficient evaluation. The results due to data efficient evaluation are not up to the mark when compared to other transfer learning using large corpus. We are evaluating using 5-fold cross validation for the dataset. The metric used in the task is Quadratic weighted kappa, which measures the agreement between two ratings. Its values vary from random (0) to complete (1) agreement, and if there is less agreement than chance it may become negative.

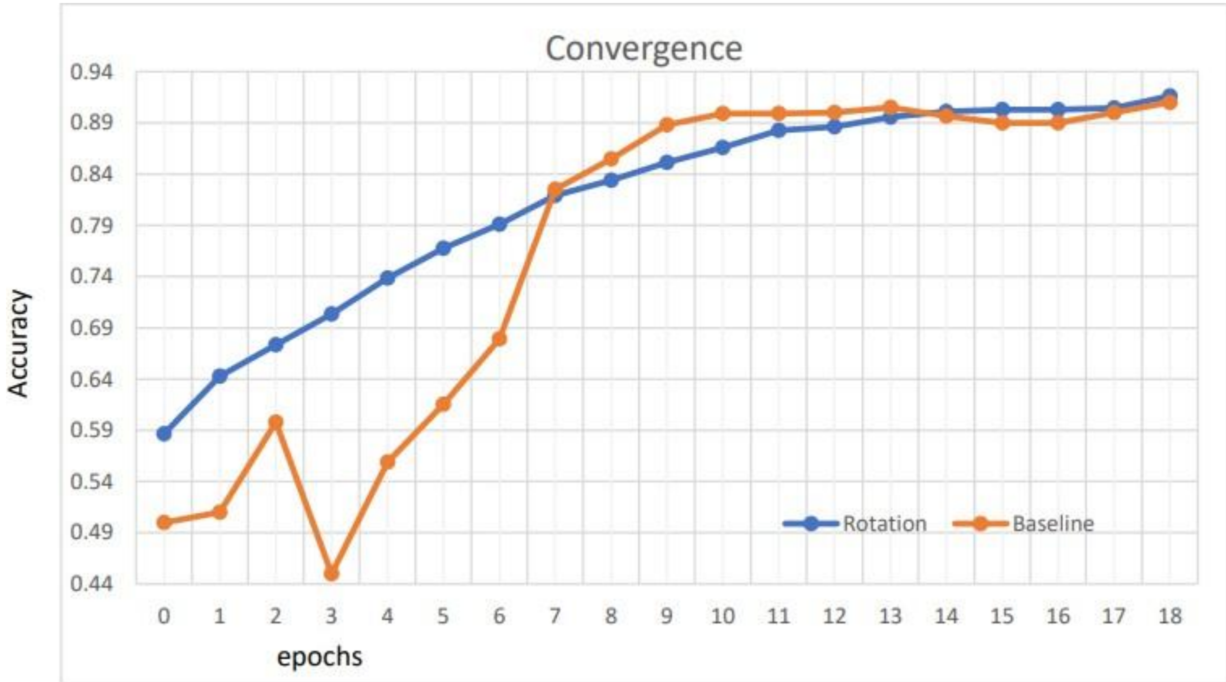
Train Split	**Qw_kappa_kaggle MIN**	**Qw_kappa_kaggle AVG**	**Qw_kappa_kaggle MAX**
10%	0.2888881102	0.4321430821	0.5084661884
5%	0.1751079345	0.2944362057	0.4669641719
50%	0.6116147969	0.6798179485	0.7365178391
25%	0.4738074393	0.5753686169	0.6937023326
100%	0.6955872731	0.7247486635	0.7555889924

Results obtained from the model, showing minimum, average and maximum quadratic weighted kappa scores for different subsets of data used in fine-tuning.

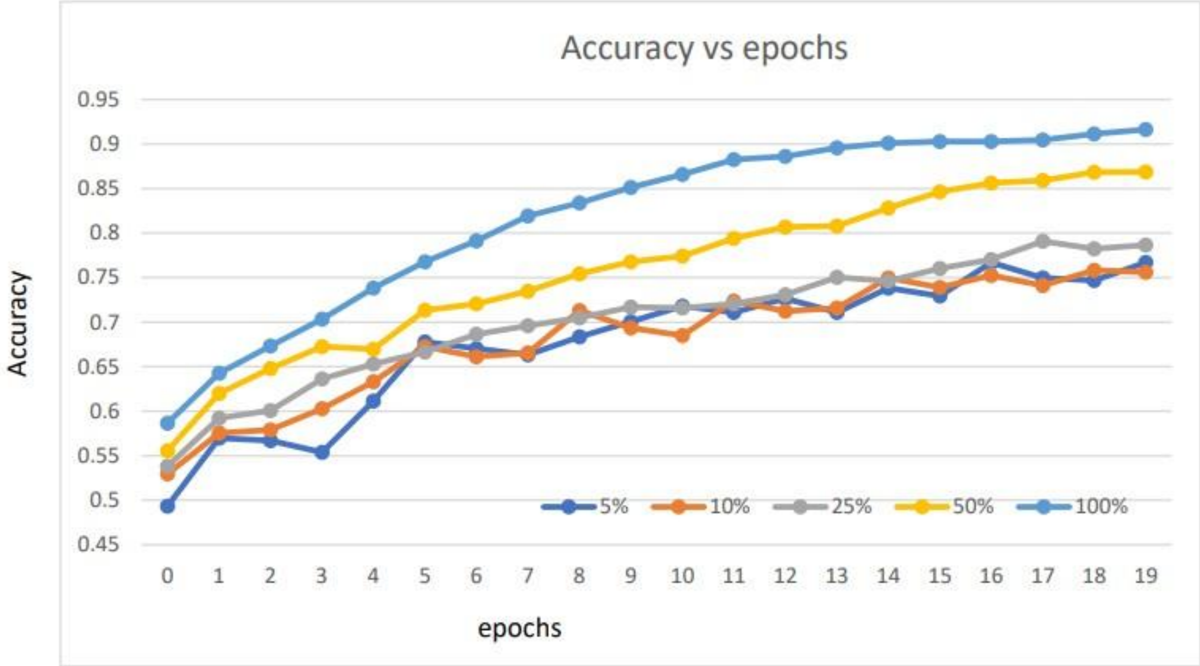
Avg QW kappa scores vs percentage of labelled images comparing Rotation technique and baseline values:



Convergence Rates comparision:



Accuracy vs epochs for various percentages of labelled data, showing convergence rates and differences in ranges of accuracy. (fifth repetition is used for every percentage.



6. TESTING

6. TESTING

6.1 INTRODUCTION TO TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, subassemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

6.2 TYPES OFTESTING

6.2.1 UNITTESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unitbefore integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

6.2.2 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

6.2.3 FUNCTIONAL TESTING

Functional tests provide systematic demonstrations that functions tested are available as specified by the businessand technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised. Systems/Procedures interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases.

In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes.

7. CONCLUSIONS AND FUTURE SCOPE

7. CONCLUSION

7.1 CONCLUSION

In this work we have implemented a ConvNet model which can outperform the supervised base line results. Moreover, we have implemented with very less data and used data efficient evaluation. Our results, particularly in the low data regime demonstrate the possibility to reduce the manual annotation effort required in the medical imaging domain, where data and annotation scarcity is an obstacle. Furthermore, we observe performance which is comparable to pretraining our methods on a large unlabelled corpus, and fine-tuning them on a different smaller downstream-specific dataset. We believe there is room for improvement along this line, such as designing new proxy tasks, evaluating different architectural options, and including other data modalities (e.g., text) in conjunction with images/scans. We believe the field of detection of Diabetic Retinography using deep learning improves and mitigates the risk of vision loss for many people, and make if cost efficient for regular check-up.

7.2 FUTURE SCOPE

Although these results are not very promising for real life use. We believe there is room for improvement along this line, such as designing new proxy tasks, evaluating different architectural options, and including other data modalities (e.g., text) in conjunction with images/scans.

We believe the field of detection of Diabetic Retinography using deep learning improves and mitigates the risk of vision loss for many people, and make if cost efficient for regular check-up**.**

8. BIBILOGRAPHY

8. BIBILOGRAPHY

I International Diabetes Federation. IDF Diabetes Atlas 7th edn (International Diabetes Federation, Brussels, Belgium, 2015).

II https://youtu.be/SaJL4SLfrcY

III C. Doersch, A. Gupta and A. A. Efros, "Unsupervised Visual Representation Learning by Context Prediction," 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1422-1430, doi: 10.1109/ICCV.2015.167.

IV Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles [1603.09246v3] Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles (arxiv.org)

V Zhang R., Isola P., Efros A.A. (2016) Colorful Image Colorization. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol 9907. Springer, Cham. https://doi.org/10.1007/978-3-319-46487-9_40

VI Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. CoRR, abs/1803.07728, 2018. URL http://arxiv.org/abs/1803.07728.

VII Jiawei Wang, Shuai Zhu, Jiao Xu, and Da Cao. The retrieval of the beautiful: Selfsupervised salient object detection for beauty product retrieval. In Proceedings of the 27th ACM International Conference on Multimedia, MM '19, page 2548–2552, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368896. doi: 10.1145/3343031.3356059. URL https://doi.org/10.1145/3343031. 3356059.

VIII Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei Efros. Context encoders: Feature learning by inpainting. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

IX Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. CoRR, abs/1807.03748, 2018. URL http://arxiv.org/abs/1807.03748.

X https://www.kaggle.com/sovitrath/diabetic-retinopathy-224x224-2019-data XI 3D self-supervised formedical imaging. https://arxiv.org/abs/2006.03829