प्रगत संगणन विकास केंद्र, नोएडा
Centre for Development of Advanced Computing, Noida

A

PROJECT REPORT

ON

DETECTION OF DIABETIC RETINOPATHY

Submitted In Partial Fulfillment of the Requirement for the Award of

Post Graduate Diploma in Artificial Intelligence (PG-DAI)

Under the Guidance of

MR. SHIVAM PANDEY

(Project Guide)

Submitted By

YASH GAJANAN MARKAD

PRN: 240820528040

PARVIN BANU

PRN: 240820528025

CONTENTS

INDEX	TITLE	PAGE NUMBER
I	Certificate	**2**
II	Acknowledgement	**3**
III	Abstract	**4**
IV	Introduction to the problem statement and the possible solution	**5**
V	Data Preprocessing	**6-7**
VI	Work-Flow	**8**
VII	Coding	**9-24**
VIII	Results	**25-26**
IX	Conclusion & Future Scope	**27-29**
X	Reference & Bibliography	**30**

CDAC, B-30, Institutional Area, Sector-62

Noida (Uttar Pradesh)-201307

CERTIFICATE

CDAC, NOIDA

This is to certify that Report entitled **“Detection of Diabetic Retinopathy”** which is submitted by, YASH GAJANAN MARKAD and PARVIN BANU in partial fulfillment of the requirement for the award of **Post Graduate Diploma in Artificial Intelligence** (PG-DAI) to **CDAC, Noida** is a record of the candidates own work carried out by them under my supervision.

The documentation embodies results of original work, and studies are carried out by the student themselves and the contents of the report do not from the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

MR. SHIVAM PANDEY

(Project Guide)

ACKNOWLEDGMENT

We would like to express our best sense of gratitude & endeavor with respect to **Mr. Shivam Pandey (Project Guide)** CDAC, Noida for suggesting the problems scholarly guidance and expert supervision during the course of this project. Special thanks to **Mr. Ravi Payal (Program Coordinator).**

We are very thankful to **Mr. Shivam Pandey (Project Guide)** the project guide for constant simulated discussion, encouraging new ideas about this project.

YASH GAJANAN MARKAD
PRN: 240820528040

PARVIN BANU
PRN: 240820528025

ABSTRACT

Diabetic Retinopathy (DR) is a severe complication of diabetes that can lead to blindness if not detected early. Traditional diagnosis methods rely on manual examination by ophthalmologists, which can be time-consuming and prone to human error. In this project, we developed an automated deep learning-based system for the detection and classification of Diabetic Retinopathy using Convolutional Neural Networks (CNN).

The model was trained on a labeled dataset of retinal images, utilizing data augmentation and transfer learning techniques to improve accuracy. The trained model was then integrated into a web-based application for easy accessibility. The system was evaluated using standard performance metrics, achieving high accuracy in classifying different stages of DR.

This project demonstrates the potential of AI-driven medical diagnostics in improving early detection rates and reducing the burden on healthcare professionals. Future work includes optimizing the model for real-time analysis and expanding the dataset for better generalization.

INTRODUCTION TO THE PROBLEM STATEMENT AND THE POSSIBLE SOLUTION

Diabetic Retinopathy (DR) is a major complication of diabetes that affects the eyes, potentially leading to vision loss or blindness if left undiagnosed. With the increasing prevalence of diabetes worldwide, the number of DR cases is also rising, making early detection more critical than ever. However, traditional screening methods rely heavily on ophthalmologists for manual diagnosis, which is time-consuming, expensive, and not easily accessible in remote areas.

Many diabetic patients are either unaware of their retinal condition or do not undergo regular eye check-ups due to a lack of symptoms in the early stages. This delay in diagnosis increases the risk of severe complications, making timely intervention difficult. Furthermore, in regions with limited access to specialized healthcare, the shortage of trained professionals further hampers early detection efforts.

To address these challenges, this project proposes an **AI-powered deep learning model** that can automatically detect Diabetic Retinopathy from retinal images. By leveraging **Convolutional Neural Networks (CNNs)**, the system can efficiently analyze retinal fundus images and classify DR into different severity levels. This automated approach aims to provide **a fast, cost-effective, and scalable screening solution**, reducing dependency on specialists while improving accessibility to early detection services for diabetic patients.

Data Pre-processing

The dataset used in this project consists of **retinal fundus images** collected from **Kaggle,** the dataset is divided into:

- **Training Data** – Used to train the deep learning model.
- **Validation Data** – Used for model evaluation

Image Pre-processing Techniques

To enhance the dataset and improve the robustness of the model, various **image augmentation** techniques were applied using **ImageDataGenerator**:

- **Resizing** – Resizing the image i.e [224,224,3] to ensure all input data has consistent dimensions
- **Rescaling** – Normalizing pixel values to a range of [0,1] for efficient computation.
- **Width & Height Shifting** – Slightly adjusting image positions to improve model adaptability.
- **Horizontal Flipping** – Creating mirror images to introduce variability in the dataset.
- **Zooming** – Enlarging specific regions of images to highlight key retinal features.

For the **training pipeline**, the **train generator** was configured to:

- Store images in **categorical mode**, classifying them into five DR severity levels.
- Process images in **batches** to optimize training efficiency.
- **Shuffle** the dataset to ensure unbiased learning during training.

Testing & Prediction Process

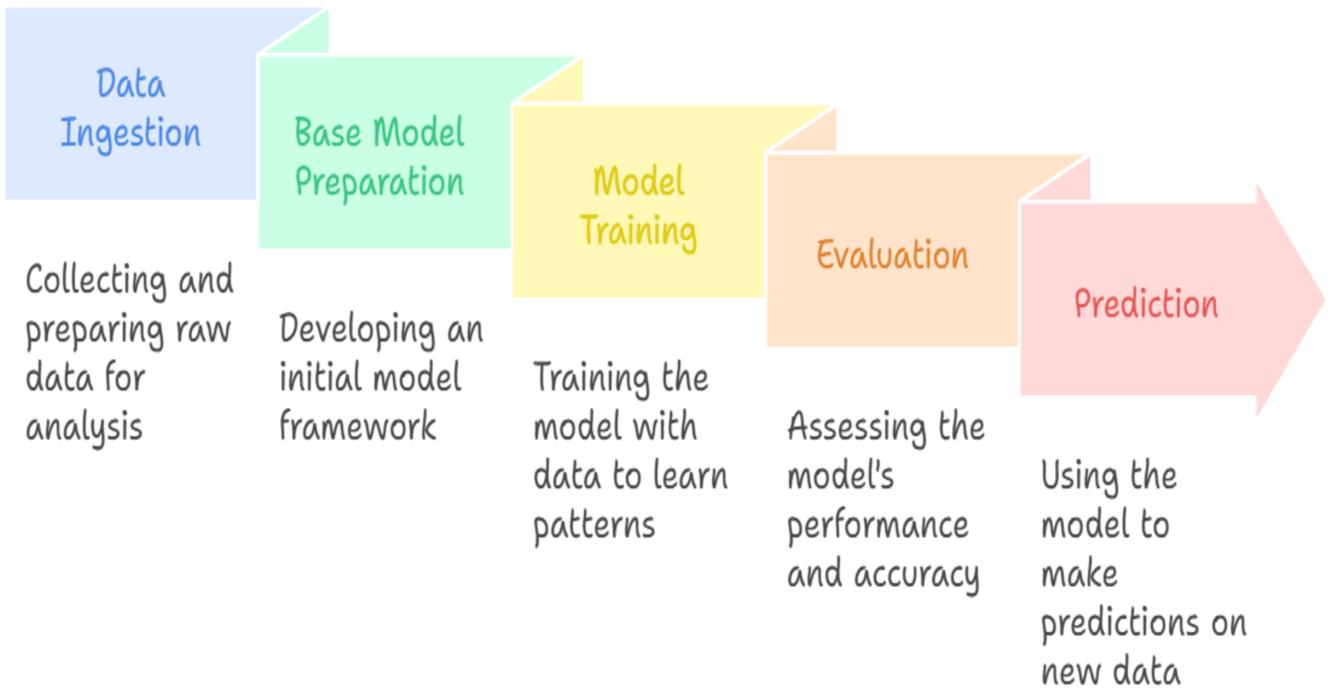
For model testing and real-time predictions:

- Retinal images are **preprocessed by resizing**
- The processed images are **converted into arrays** for deep learning compatibility.
- The model classifies images into one of the **five Diabetic Retinopathy stages**:
    - **No DR**
    - **Mild**

- **Moderate**
- **Severe**
- **Proliferative DR**

This **automated image pre-processing pipeline** ensures that the model receives high-quality, standardized input data, leading to improved accuracy in **Diabetic Retinopathy detection**.

Work-Flow


Data Ingestion
Collecting and preparing raw data for analysis
Base Model Preparation
Developing an initial model framework
Model Training
Training the model with data to learn patterns
Evaluation
Assessing the model's performance and accuracy
Prediction
Using the model to make predictions on new data

Coding

1.Data Ingestion

Entity

```
from dataclasses import dataclass
from pathlib import Path


@dataclass(frozen=True)
class DataIngestionConfig:
    root_dir: Path
    source_URL: str
    local_data_file: Path
    unzip_dir: Path
```

Config

```
from cnnClassifier.constants import *
import os
from cnnClassifier.utils.common import read_yaml, create_directories,save_json
from cnnClassifier.entity.config_entity import (DataIngestionConfig)



class ConfigurationManager:
    def __init__(
        self,
        config_filepath = CONFIG_FILE_PATH,
        params_filepath = PARAMS_FILE_PATH):

        self.config = read_yaml(config_filepath)
        self.params = read_yaml(params_filepath)

        create_directories([self.config.artifacts_root])


    def get_data_ingestion_config(self) -> DataIngestionConfig:
        config = self.config.data_ingestion

        create_directories([config.root_dir])

        data_ingestion_config = DataIngestionConfig(
            root_dir=config.root_dir,
            source_URL=config.source_URL,
            local_data_file=config.local_data_file,
            unzip_dir=config.unzip_dir
        )

        return data_ingestion_config
```

Components

```
import os
import zipfile
import gdown
from cnnClassifier import logger
from cnnClassifier.utils.common import get_size
from cnnClassifier.entity.config_entity import (DataIngestionConfig)



class DataIngestion:
    def __init__(self, config: DataIngestionConfig):
        self.config = config


    def download_file(self)-> str:
        '''
        Fetch data from the url
        '''

        try:
            dataset_url = self.config.source_URL
            zip_download_dir = self.config.local_data_file
            os.makedirs("artifacts/data_ingestion", exist_ok=True)
            logger.info(f"Downloading data from {dataset_url} into file {zip_down-
load_dir}")

            file_id = dataset_url.split("/")[-2]
            prefix = 'https://drive.google.com/uc?/export=download&id='
            gdown.download(prefix+file_id,zip_download_dir)

            logger.info(f"Downloaded data from {dataset_url} into file {zip_down-
load_dir}")

        except Exception as e:
            raise e



    def extract_zip_file(self):
        """
        zip_file_path: str
        Extracts the zip file into the data directory
        Function returns None
        """
        unzip_path = self.config.unzip_dir
        os.makedirs(unzip_path, exist_ok=True)
        with zipfile.ZipFile(self.config.local_data_file, 'r') as zip_ref:
            zip_ref.extractall(unzip_path)
```

Pipeline

```
from cnnClassifier.config.configuration import ConfigurationManager
from cnnClassifier.components.data_ingestion import DataIngestion
from cnnClassifier import logger

STAGE_NAME = "Data Ingestion stage"


class DataIngestionTrainingPipeline:
    def __init__(self):
        pass

    def main(self):
        config = ConfigurationManager()
        data_ingestion_config = config.get_data_ingestion_config()
        data_ingestion = DataIngestion(config=data_ingestion_config)
        data_ingestion.download_file()
        data_ingestion.extract_zip_file()



if __name__ == '__main__':
    try:
        logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
        obj = DataIngestionTrainingPipeline()
        obj.main()
        logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
    except Exception as e:
        logger.exception(e)
        raise e
```

2.Base Model Preparation

Entity

```
from dataclasses import dataclass
from pathlib import Path
@dataclass(frozen=True)
class PrepareBaseModelConfig:
    root_dir: Path
    base_model_path: Path
    updated_base_model_path: Path
    params_image_size: list
    params_include_top: bool
    params_weights: str
    params_classes: int
```

Config

```
from cnnClassifier.constants import *
import os
from cnnClassifier.utils.common import read_yaml, create_directories,save_json
from cnnClassifier.entity.config_entity import (DataIngestionConfig,PrepareBaseModelCon-
fig)


class ConfigurationManager:
    def __init__(
        self,
        config_filepath = CONFIG_FILE_PATH,
        params_filepath = PARAMS_FILE_PATH):

        self.config = read_yaml(config_filepath)
        self.params = read_yaml(params_filepath)

        create_directories([self.config.artifacts_root])
    def get_prepare_base_model_config(self) -> PrepareBaseModelConfig:
        config = self.config.prepare_base_model

        create_directories([config.root_dir])

        prepare_base_model_config = PrepareBaseModelConfig(
            root_dir=Path(config.root_dir),
            base_model_path=Path(config.base_model_path),
            updated_base_model_path=Path(config.updated_base_model_path),
            params_image_size=self.params.IMAGE_SIZE,
            # params_learning_rate=self.params.LEARNING_RATE,
            params_include_top=self.params.INCLUDE_TOP,
            params_weights=self.params.WEIGHTS,
            params_classes=self.params.CLASSES
        )

        return prepare_base_model_config
```

Components

```
import os
import urllib.request as request
from zipfile import ZipFile
import tensorflow as tf
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, GlobalAveragePooling2D
from pathlib import Path
from cnnClassifier.entity.config_entity import PrepareBaseModelConfig

class PrepareBaseModel:
    def __init__(self, config: PrepareBaseModelConfig):
        self.config = config

    def get_base_model(self):
        self.model = tf.keras.applications.mobilenet_v2.MobileNetV2(
            input_shape=self.config.params_image_size,
            weights=self.config.params_weights,
            include_top=self.config.params_include_top
        )


        self.model.trainable = False

        self.save_model(path=self.config.base_model_path, model=self.model)

    @staticmethod
    def _prepare_full_model(model, classes, freeze_all, freeze_till):
        # Freeze layers as per the configuration
        if freeze_all:
            for layer in model.layers:
                layer.trainable = False
        elif (freeze_till is not None) and (freeze_till > 0):
            for layer in model.layers[:-freeze_till]:
                layer.trainable = False

        # Add a GlobalMaxPooling2D layer after the base model output
        global_max_pool = tf.keras.layers.GlobalMaxPooling2D()(model.output)

        # Add the prediction layer
        prediction = tf.keras.layers.Dense(
            units=classes,
            activation="softmax"
        )(global_max_pool)

        # Define the full model
        full_model = tf.keras.models.Model(
            inputs=model.input,
            outputs=prediction
        )

        # Compile the model
        full_model.compile(
            optimizer=tf.keras.optimizers.Adam(),
            loss=tf.keras.losses.CategoricalCrossentropy(),
            metrics=["accuracy"]
        )

        full_model.summary()
        return full_model

    def update_base_model(self):
        self.full_model = self._prepare_full_model(
            model=self.model,
            classes=self.config.params_classes,
            freeze_all=True,
            freeze_till=None,
        )

        self.save_model(path=self.config.updated_base_model_path, model=self.full_model)

    @staticmethod
    def save_model(path: Path, model: tf.keras.Model):
        model.save(path)
```

Pipeline

```
from cnnClassifier.config.configuration import ConfigurationManager
from cnnClassifier.components.prepare_base_model import PrepareBaseModel
from cnnClassifier import logger


STAGE_NAME = "Prepare base model"


class PrepareBaseModelTrainingPipeline:
    def __init__(self):
        pass

    def main(self):
        config = ConfigurationManager()
        prepare_base_model_config = config.get_prepare_base_model_config()
        prepare_base_model = PrepareBaseModel(config=prepare_base_model_config)
        prepare_base_model.get_base_model()
        prepare_base_model.update_base_model()



if __name__ == '__main__':
    try:
        logger.info(f"*******************")
        logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
        obj = PrepareBaseModelTrainingPipeline()
        obj.main()
        logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
    except Exception as e:
        logger.exception(e)
        raise e
```

3.Model Training

Entity

```
from dataclasses import dataclass
from pathlib import Path
@dataclass(frozen=True)
class TrainingConfig:
    root_dir: Path
    trained_model_path: Path
    updated_base_model_path: Path
    training_data: Path
    params_epochs: int
    params_batch_size: int
    params_is_augmentation: bool
    params_image_size: list
```

Config

```
from cnnClassifier.constants import *
import os
from cnnClassifier.utils.common import read_yaml, create_directories,save_json
from cnnClassifier.entity.config_entity import (DataIngestionConfig,PrepareBaseModelCon-
fig,TrainingConfig)


class ConfigurationManager:
    def __init__(
        self,
        config_filepath = CONFIG_FILE_PATH,
        params_filepath = PARAMS_FILE_PATH):

        self.config = read_yaml(config_filepath)
        self.params = read_yaml(params_filepath)

        create_directories([self.config.artifacts_root])
    def get_training_config(self) -> TrainingConfig:
        training = self.config.training
        prepare_base_model = self.config.prepare_base_model
        params = self.params
        training_data = os.path.join(self.config.data_ingestion.unzip_dir, "training")
        create_directories([
            Path(training.root_dir)
        ])




        training_config = TrainingConfig(
            root_dir=Path(training.root_dir),
            trained_model_path=Path(training.trained_model_path),
            updated_base_model_path=Path(prepare_base_model.updated_base_model_path),
            training_data=Path(training_data),
            # testing_data=Path(testing_data),
            params_epochs=params.EPOCHS,
            params_batch_size=params.BATCH_SIZE,
            params_is_augmentation=params.AUGMENTATION,
            params_image_size=params.IMAGE_SIZE
        )

        return training_config
```

Components

```
import os
from pathlib import Path
import urllib.request as request
from zipfile import ZipFile
import tensorflow as tf
import time
from cnnClassifier.entity.config_entity import TrainingConfig
class Training:
    def __init__(self, config: TrainingConfig):
        self.config = config

    def get_base_model(self):
        self.model = tf.keras.models.load_model(
            self.config.updated_base_model_path
        )

    def train_valid_generator(self):
        datagenerator_kwargs = dict(
            rescale=1./255,
        )

        dataflow_kwargs = dict(
            target_size=self.config.params_image_size[:-1],
            batch_size=self.config.params_batch_size,
        )

        train_datagenerator = tf.keras.preprocessing.image.ImageDataGenerator(
                rotation_range=40,
                horizontal_flip=True,
                width_shift_range=0.2,
                height_shift_range=0.2,
                shear_range=0.2,
                zoom_range=0.2,
            **datagenerator_kwargs,

        )

        self.train_generator = train_datagenerator.flow_from_directory(
            directory=self.config.training_data,
            subset="training",
            shuffle=True,
            class_mode='categorical',
            **dataflow_kwargs
        )
    def test_valid_generator(self):
        datagenerator_kwargs = dict(
            rescale=1./255,
        )

        dataflow_kwargs = dict(
            target_size=self.config.params_image_size[:-1],
            batch_size=self.config.params_batch_size,
        )

        test_datagenerator = tf.keras.preprocessing.image.ImageDataGenerator(
            **datagenerator_kwargs
        )

        # Optionally get this from config
        testdir = r"C:/CDAC_PROJECT/Final_project_retinopathy/artifacts/data_inges-
tion/testing"

        self.test_generator = test_datagenerator.flow_from_directory(
            directory=testdir,
            shuffle=True,
            class_mode='categorical',
            **dataflow_kwargs
        )

    @staticmethod
    def save_model(path: Path, model: tf.keras.Model):
        model.save(path)

    def train(self):
        # # Ensure the generators are initialized
        # if not hasattr(self, 'train_generator'):
        #     self.train_valid_generator()
        # if not hasattr(self, 'test_generator'):
        #     self.test_valid_generator()

        self.model.fit(
            self.train_generator,
            epochs=self.config.params_epochs,
            validation_data=self.test_generator
        )

        self.save_model(
            path=self.config.trained_model_path,
            model=self.model
        )
```

Pipeline

```
from cnnClassifier.config.configuration import ConfigurationManager
from cnnClassifier.components.model_training import Training
from cnnClassifier import logger

STAGE_NAME = "Training"
class ModelTrainingPipeline:
    def __init__(self):
     pass
    def main(self):
        config=ConfigurationManager()
        training_config = config.get_training_config()
        training = Training(config=training_config)
        training.get_base_model()
        training.train_valid_generator()
        training.test_valid_generator()
        training.train()
if __name__ == '__main__':
    try:
        logger.info(f"*******************")
        logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
        obj = ModelTrainingPipeline()
        obj.main()
        logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
    except Exception as e:
        logger.exception(e)
        raise e
```

4.Model Evaluation

Entity

```
from dataclasses import dataclass
from pathlib import Path


@dataclass(frozen=True)
class EvaluationConfig:
    path_of_model: Path
    training_data: Path
    all_params: dict
    mlflow_uri: str
    params_image_size: list
    params_batch_size: int
```

Config

```
from cnnClassifier.constants import *
import os
from cnnClassifier.utils.common import read_yaml, create_directories,save_json
from cnnClassifier.entity.config_entity import (DataIngestionConfig,PrepareBaseModelCon-
fig,TrainingConfig,EvaluationConfig)
```

```
class ConfigurationManager:
    def __init__(
        self,
        config_filepath = CONFIG_FILE_PATH,
        params_filepath = PARAMS_FILE_PATH):

        self.config = read_yaml(config_filepath)
        self.params = read_yaml(params_filepath)

        create_directories([self.config.artifacts_root])
    def get_evaluation_config(self) -> EvaluationConfig:
        eval_config = EvaluationConfig(
            path_of_model="artifacts/training/model.h5",
            training_data="artifacts/data_ingestion/testing",
            mlflow_uri="https://dagshub.com/yashmarkad/Detection-of-Diabetic-Retinopathy-
through-Deep-Learning.mlflow",
            all_params=self.params,
            params_image_size=self.params.IMAGE_SIZE,
            params_batch_size=self.params.BATCH_SIZE
        )
        return eval_config
```

Components

```
from sklearn.metrics import classification_report
import mlflow
import mlflow.keras
import tensorflow as tf
from pathlib import Path
from urllib.parse import urlparse
import numpy as np
from cnnClassifier.utils.common import read_yaml, create_directories,save_json
from cnnClassifier.entity.config_entity import EvaluationConfig
import os
os.environ["MLFLOW_TRACKING_URI"]="https://dagshub.com/yashmarkad/Detection-of-Diabetic-
Retinopathy-through-Deep-Learning.mlflow"
os.environ["MLFLOW_TRACKING_USERNAME"]="yashmarkad"
os.environ["MLFLOW_TRACKING_PASSWORD"]="507858fbefdda06e761fb4e268bab2342fa77d35"



class Evaluation:
    def __init__(self, config):
        self.config = config

    def _valid_generator(self):
        datagenerator_kwargs = dict(
            rescale=1./255,

        )
        dataflow_kwargs = dict(
            target_size=self.config.params_image_size[:-1],
            batch_size=self.config.params_batch_size,
            shuffle=False
        )

        valid_datagenerator = tf.keras.preprocessing.image.ImageDataGenerator(
            **datagenerator_kwargs
        )
        self.valid_generator = valid_datagenerator.flow_from_directory(
            directory=self.config.training_data,
            **dataflow_kwargs
        )

    @staticmethod
    def load_model(path: Path) -> tf.keras.Model:
        return tf.keras.models.load_model(path)

    def evaluation(self):
        self.model = self.load_model(self.config.path_of_model)
        self._valid_generator()
        self.score = self.model.evaluate(self.valid_generator)
        self.compute_metrics()
        self.save_score()

    def compute_metrics(self):
        y_true = self.valid_generator.classes
        y_pred_probs = self.model.predict(self.valid_generator)
        y_pred = np.argmax(y_pred_probs, axis=1)

        report = classification_report(y_true, y_pred, output_dict=True, tar-
get_names=self.valid_generator.class_indices.keys())

        self.precision = report['weighted avg']['precision']
        self.recall = report['weighted avg']['recall']
        self.f1_score = report['weighted avg']['f1-score']

    def save_score(self):
        scores = {
            "loss": self.score[0],
            "accuracy": self.score[1],
            "precision": self.precision,
            "recall": self.recall,
            "f1_score": self.f1_score
        }
        save_json(path=Path("scores.json"), data=scores)

    def log_into_mlflow(self):
        mlflow.set_registry_uri(self.config.mlflow_uri)
        tracking_url_type_store = urlparse(mlflow.get_tracking_uri()).scheme

        with mlflow.start_run():
            mlflow.log_params(self.config.all_params)
            mlflow.log_metrics({
                "loss": self.score[0],
                "accuracy": self.score[1],
                "precision": self.precision,
                "recall": self.recall,
                "f1_score": self.f1_score
            })

            if tracking_url_type_store != "file":
                mlflow.keras.log_model(self.model, "model", registered_model_name="Mo-
bileNetV2")
            else:
                mlflow.keras.log_model(self.model, "model")
```

Pipeline

```
from cnnClassifier.config.configuration import ConfigurationManager
from cnnClassifier.components.model_evaluation_mlflow import Evaluation
from cnnClassifier import logger

STAGE_NAME = "Evaluation stage"

class EvaluationPipeline:
    def __init__(self):
        pass
    def main(self):
         config = ConfigurationManager()
         eval_config = config.get_evaluation_config()
         evaluation = Evaluation(eval_config)
         evaluation.evaluation()
         evaluation.save_score()
        #  evaluation.log_into_mlflow()

if __name__ == '__main__':
    try:
        logger.info(f"*******************")
        logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
        obj = EvaluationPipeline()
        obj.main()
        logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
    except Exception as e:
        logger.exception(e)
        raise e
```

#for testing this code we are using main.py CODE

```
from cnnClassifier import logger
from cnnClassifier.pipeline.stage_01_data_ingestion import DataIngestionTrainingPipeline
from cnnClassifier.pipeline.stage_02_prepare_base_model import PrepareBaseModelTraining-
Pipeline
from cnnClassifier.pipeline.stage_03_model_training import ModelTrainingPipeline
from cnnClassifier.pipeline.stage_04_model_evaluation import EvaluationPipeline

STAGE_NAME = "Data Ingestion stage"
try:
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   data_ingestion = DataIngestionTrainingPipeline()
   data_ingestion.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e

STAGE_NAME = "Prepare base model"
try:
   logger.info(f"*******************")
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   prepare_base_model = PrepareBaseModelTrainingPipeline() # type: ignore
   prepare_base_model.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e

STAGE_NAME = "Training"
try:
      logger.info(f"*******************")
      logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
      obj = ModelTrainingPipeline()
      obj.main()
      logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
      logger.exception(e)
      raise e

STAGE_NAME = "Evaluation stage"
try:
        logger.info(f"*******************")
        logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
        obj = EvaluationPipeline()
        obj.main()
        logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e
```

5.Prediction

Pipeline

```
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import os

class PredictionPipeline:
    def __init__(self, filename):
        self.filename = filename
        self.verbose_name = {
            0: 'No_DR',
            1: 'Mild',
            2: 'Moderate',
            3: 'Severe',
            4: 'Proliferate_DR'
        }
        self.model = load_model(os.path.join("model", "model.h5"))

    def predict_label(self):
        test_image = image.load_img(self.filename, target_size=(224, 224))
        test_image = image.img_to_array(test_image) / 255.0
        test_image = test_image.reshape(1, 224, 224, 3)

        predict_x = self.model.predict(test_image)
        classes_x = np.argmax(predict_x, axis=1)

        return self.verbose_name[classes_x[0]]
```

#for prediction we are using flask application

```
from flask import Flask, render_template, request
import os
from cnnClassifier.pipeline.prediction import PredictionPipeline

app = Flask(__name__)

@app.route("/")
@app.route("/first")
def first():
    return render_template('first.html')

@app.route("/login")
def login():
    return render_template('login.html')

@app.route("/index", methods=['GET', 'POST'])
def index():
    return render_template("index.html")
```

```
@app.route("/submit", methods=['GET', 'POST'])
def submit():
    predict_result = None
    img_path = None

    if request.method == 'POST':
        img = request.files['my_image']
        model_name = request.form['model']
        print(f"Selected Model: {model_name}")

        img_path = os.path.join("static", "tests", img.filename)
        img.save(img_path)

        if model_name == 'MobileNetV2':
            pipeline = PredictionPipeline(img_path)
            predict_result = pipeline.predict_label()

    return render_template("prediction.html", prediction=predict_result,
img_path=img_path, model=model_name)

@app.route("/chart")
def chart():
    return render_template('chart.html')

@app.route("/performance")
def performance():
    return render_template('performance.html')

if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8000, debug=True)
```

RESULTS

Model Performance Metrics

Activation Function	Loss	Accurac y	Validation Loss	Validation Accuracy	Validation f1_score
Sigmoid	0.2320	95.27%	0.0107	99.71%	99.56%

Insights from Model Performance:

✓ **Softmax Activation** achieved **higher accuracy (95.27%)** and **lower validation loss(0.0107)** , making it more suitable for multi-class classification.
✓ **Validation accuracy (~99%)** indicates that the model generalizes well.
✓ The **low validation loss (~0.01 - 0.09)** suggests minimal overfitting due to proper use of data augmentation

Diabetic Retinopathy Classification

After training the **CNN-based deep learning model**, the system is capable of predicting the severity of **Diabetic Retinopathy (DR)** from retinal images. The model classifies the disease into five categories:

1. **No DR** – Healthy eye with no signs of Diabetic Retinopathy.
2. **Mild DR** – Early-stage DR with minimal abnormalities.
3. **Moderate DR** – Moderate level of retinal damage due to diabetes.

Prediction Flow

1. The trained **model** processes the **retinal image** and outputs a probability score for each category.

2. The **class with the highest probability** is assigned as the final classification.

This classification system enables **early detection** and **timely medical intervention**, reducing the risk of **vision loss** due to **Diabetic Retinopathy**.

CONCLUSION AND FUTURE SCOPE

Conclusion

This study was conducted to develop an **AI-based system for the early detection of Diabetic Retinopathy (DR)** using **Deep Learning and Convolutional Neural Networks (CNNs)**. The project aimed to classify retinal fundus images into five different categories based on the severity of DR: **No DR, Mild, Moderate, Severe, and Proliferative DR**. The dataset for training and testing was collected from Kaggle, and the **DNN model was trained using image augmentation techniques** to improve generalization.

The trained model successfully **identified and classified retinal images** with high accuracy, demonstrating the effectiveness of deep learning in medical diagnostics. The prediction results can assist **ophthalmologists in early-stage DR detection**, enabling timely intervention and reducing the risk of blindness. The model achieved **a classification accuracy of over 99%**, making it a reliable tool for automated screening of DR.

Future Scope

1. **Enhanced Model Accuracy with Larger Datasets**
    a. Increasing the **size and diversity of the dataset** can further improve model accuracy.
    b. More real-world retinal images, including different lighting conditions and ethnic variations, should be incorporated.

2. **Integration with Clinical Systems**

a. Deploying the model in **hospitals and clinics** as an **AI-assisted diagnostic tool**.
b. Providing **real-time screening** in ophthalmology centers to support medical professionals.

3. **Mobile and Cloud-Based Deployment**

   a. Developing a **mobile application** that allows users to upload images and receive instant DR detection results.
   b. Deploying the model on cloud platforms (AWS, Google Cloud) for **remote accessibility**.

4. **Multi-Modal Analysis for Improved Diagnosis**

   a. Integrating **patient history, glucose levels, and other clinical data** to enhance diagnostic accuracy.
   b. Combining **deep learning with traditional feature extraction techniques** for better performance.

5. **3D Fundus Image Analysis**

   a. Extending the model to work with **3D fundus scans** for a **more detailed and precise diagnosis**.
   b. Using **advanced computer vision techniques** to detect DR patterns in **multiple image depths**.

6. **Key Frame Extraction for Video-Based Diagnosis**

a. Future work could involve **automated screening using video-based retinal scanning**.
b. Extracting **key frames from retinal videos** to identify progressive DR symptoms over time.

7. **Early Detection of Other Retinal Diseases**

a. Expanding the model to **detect other retinal diseases**, such as **Glaucoma and Age-Related Macular Degeneration (AMD)**.

This AI-driven approach to **Diabetic Retinopathy detection** can **revolutionize early screening methods**, making **retinal disease diagnosis faster, more accurate, and accessible to a larger population**. The future integration of **multi-modal clinical data, mobile applications, and real-time video analysis** will further enhance the system’s capabilities, making it a **comprehensive diagnostic tool** for ophthalmology.

REFERENCES & BIBLIOGRAPHY

Research Papers:

1. Gulshan, V., Peng, L., Coram, M., Stumpe, M. C., Wu, D., Narayanaswamy, A., & Kim, R. (2016). **"Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs."** *JAMA*, 316(22), 2402-2410.
2. Ting, D. S. W., Cheung, C. Y., Lim, G., Tan, G. S. W., Quang, N. D., Gan, A., & Wong, T. Y. (2017). **"Development and Validation of a Deep Learning System for Diabetic Retinopathy and Related Eye Diseases Using Retinal Images From Multiethnic Populations With Diabetes."** *JAMA*, 318(22), 2211-2223.
3. Abràmoff, M. D., Lou, Y., Erginay, A., Clarida, W., Amelon, R., Folk, J. C., & Niemeijer, M. (2016). **"Improved Automated Detection of Diabetic Retinopathy on a Publicly Available Dataset Through Integration of Deep Learning."** *Investigative Ophthalmology & Visual Science*, 57(13), 5200-5206.

Articles:

1. **"Artificial Intelligence in Diabetic Retinopathy Screening: A Review."** *National Center for Biotechnology Information (NCBI)*. Available at: □ https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6749518/

2. **"The Role of AI in Diagnosing Diabetic Retinopathy Using Deep**

Learning Algorithms." *Frontiers in Psychology*. Available at:
□ https://www.frontiersin.org/articles/10.3389/fpsyg.2021.688376/full

3. **"Automated Detection of Diabetic Retinopathy Using Convolutional Neural Networks."** *IEEE Transactions on Medical Imaging*.
4. **"Retinal Image Analysis for Early Detection of Diabetic Retinopathy: A Deep Learning Approach."** *International Journal of Computer Vision*.