Diabetic Retinopathy Classification

03.07.2024

zTEAM MEMBERS

PRAGADESHWARAN S

CNN Model and architecture

RASHEEN FAHMI M

Model performance and Improvement

ITHIKASH R

Grafana Integration

ARUN KUMAR S

Dataset collection and Preparation

PROBLEM STATEMENT

The existing methods for detecting and grading Diabetic Retinopathy often rely on subjective assessments and extensive manual labor, leading to inefficiencies and potential inconsistencies in diagnosis. Moreover, the increasing prevalence of diabetes and the limited availability of ophthalmologists further exacerbate the challenges in timely screening and diagnosis. Therefore, there is a need to develop a robust and reliable automated system that can accurately detect and grade diabetic retinopathy, enabling early intervention and personalized treatment plans.

DATASET AND SOURCE

1.Dataset Overview

The dataset consists of retinal scanned images that are labeled as either showing signs of Diabetic Retinopathy (DR) or not (No DR). The images are taken from Kaggle's public dataset, which is commonly used for medical image analysis and machine learning tasks related to DR classification.

2.Source

- ➔ **Dataset Name:** Diagnosis of Diabetic Retinopathy
- ➔ **Source**: Kaggle
- ➔ **Dataset download :** Diabetic Retinopathy
- ➔ **Description**: This dataset contains high-resolution retinal images taken using fundus photography. Each image is labeled with a classification indicating whether Diabetic Retinopathy is present.

3.Dataset Composition

- ➔ **Training Data**: Consists of labeled retinal images that will be used to train the classification model.
- ➔ **Test Data**: A separate set of images for evaluating the performance of the trained model.

4.Data Characteristics

- ➔ **Image Format**: JPG
- ➔ **Image Resolution**: High-resolution images (may need resizing for model training).
- ➔ **Label Distribution**:
  - ◆ 0: Presence of Diabetic Retinopathy
  - ◆ 1: Absence of Diabetic Retinopathy
- ➔ **File Format**: The dataset is available as a compressed file containing images and labels in CSV format.

DATASET PREPARATION

Modules used:

```
import os, sys
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```
from tensorflow.keras.models import Sequential
from pathlib import Path
import random
import cv2
import warnings
```

Dataset loading

```
train_root_path =r'C:\Users\praga\Downloads\EDGE MATRIX\retina\train'
test_root_path =r'C:\Users\praga\Downloads\EDGE MATRIX\retina\test'
validation_path=r'C:\Users\praga\Downloads\EDGE MATRIX\retina\valid'
```

Dimensions

```
batch_size = 32
img_height = 224
img_width = 224
```

Augmentation and scaling of images

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
shear_range = 0.2,
zoom_range = 0.2,
horizontal_flip = True,
rescale=1./255
)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

Training set

```
train_ds = tf.keras.utils.image_dataset_from_directory(
    images_path,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size = (img_height, img_width),
    batch_size=batch_size)
```

Testing set

```
val_ds = tf.keras.utils.image_dataset_from_directory(
    images_path,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size = (img_height, img_width),
    batch_size=batch_size)
```

MODEL ARCHITECTURE

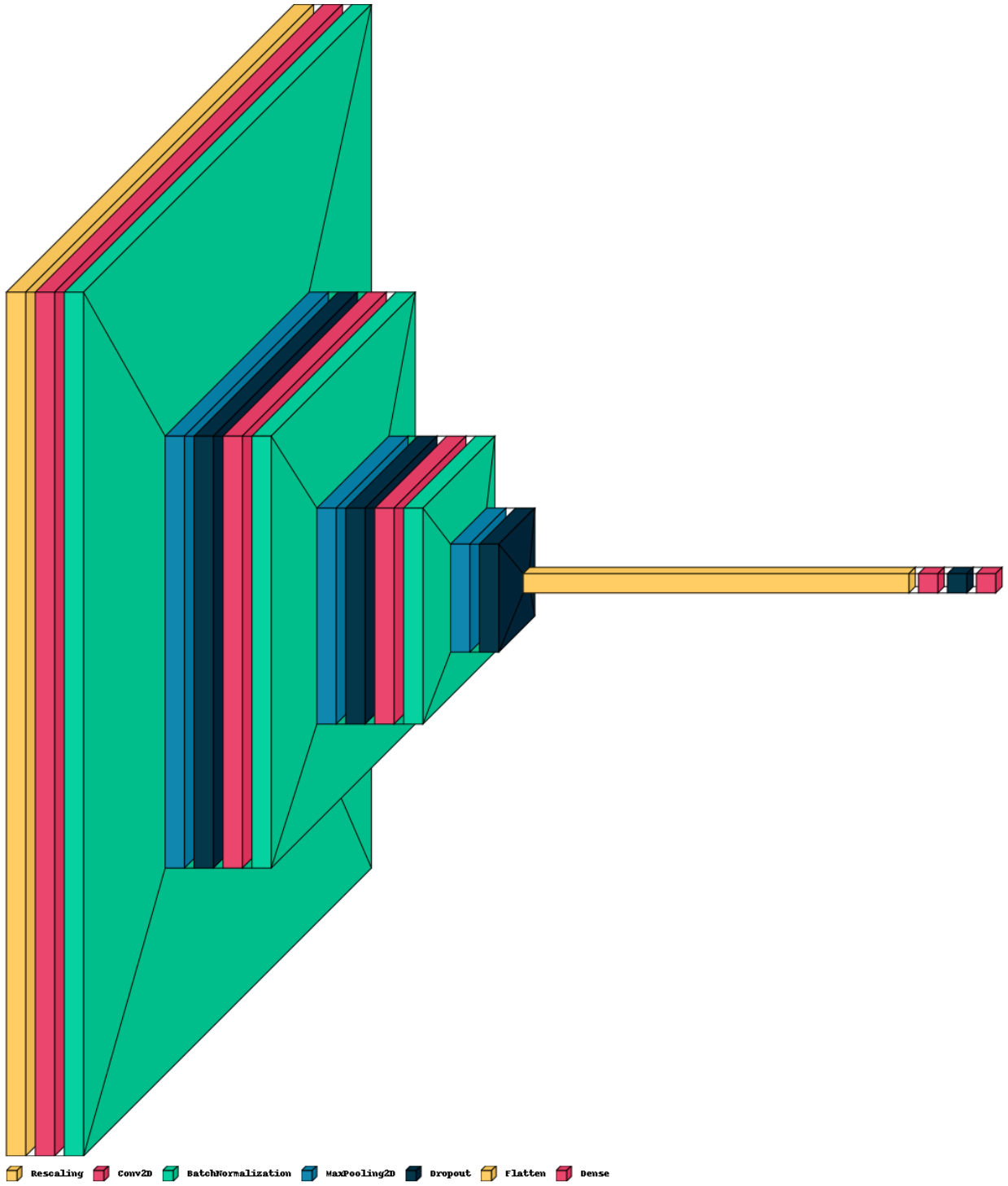
```
model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
    layers.Conv2D(16, 3, padding="same", activation="relu"),
    layers.BatchNormalization(),
```

```
    layers.MaxPooling2D(),
    layers.Dropout(0.25),

    layers.Conv2D(32, 3, padding="same", activation="relu"),
    layers.BatchNormalization(),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),

    layers.Conv2D(64, 3, padding="same", activation="relu"),
    layers.BatchNormalization(),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),

    layers.Flatten(),
    layers.Dense(128, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation="softmax")
])
```


Rescaling
Conv2D
BatchNormalization
MaxPooling2D
Dropout
Flatten
Dense

Architectural Flow

- The image depicts a sequential flow starting with image rescaling and passing through multiple convolutional layers, interspersed with max-pooling and dropout layers.
- It then flattens the processed feature maps and passes them through dense layers to produce the final output, which is typically a classification label.

Building the CNN model: -

The model is built as a Sequential model, meaning the layers are added sequentially one after the other.

Model Layers:

1. Rescaling(1./255, input_shape=(img_height, img_width, 3)): -

- Rescales the pixel values of the input images from [0, 255] to [0, 1].
- `input_shape` specifies the shape of the input images (height, width, and color channels).

2. Conv2D(16, 3, padding="same", activation="relu"): -

- 2D convolutional layer with 16 filters and a 3x3 kernel.
- "same" padding maintains the input dimensions.
- ReLU activation function introduces non-linearity.

3. MaxPooling2D():

- Max-pooling layer that reduces spatial dimensions by taking the maximum value over a 2x2 window.

4. Conv2D(32, 3, padding="same", activation="relu"):

- 2D convolutional layer with 32 filters and a 3x3 kernel.
- "same" padding and ReLU activation.

5. MaxPooling2D():

- Max-pooling layer to reduce spatial dimensions.

6. Conv2D(64, 3, padding="same", activation="relu"):

- 2D convolutional layer with 64 filters and a 3x3 kernel.
- "same" padding and ReLU activation.

7. MaxPooling2D():

- Max-pooling layer to reduce spatial dimensions further.

8. Flatten():

- Flattens the 3D output of the convolutional layers into a 1D vector, preparing it for the dense layers.

9. Dense(128, activation="relu"):

- Fully connected (dense) layer with 128 units and ReLU activation.

10. Dense(num_classes):

- Final fully connected layer with units equal to the number of classes (`num_classes`).
- No activation function is specified here as it will be applied later during loss calculation.

CNN MODEL PERFORMANCE :

Training and Validation Metrics:

The following plots depict the training and validation accuracy and loss over the course of the model's training epochs.

Plot Descriptions:

1. Training and Validation Accuracy

- X-axis: Represents the number of epochs.
- Y-axis: Represents the accuracy metric, which ranges from 0 to 1.
- Blue Line: Indicates the training accuracy.
- Orange Line: Indicates the validation accuracy.

Observation:

- The training accuracy shows a steady increase over the epochs, indicating that the model is learning and improving its performance on the training data.
- The validation accuracy, however, shows fluctuations and does not follow the training accuracy closely, suggesting potential overfitting after the initial epochs.

2. Training and Validation Loss

- X-axis: Represents the number of epochs.
- Y-axis: Represents the loss metric.
- Blue Line: Indicates the training loss.
- Orange Line: Indicates the validation loss.

Observation:

- The training loss shows a consistent decrease, indicating that the model is minimizing the error on the training data.
- The validation loss, similar to validation accuracy, shows fluctuations and begins to increase after the initial epochs, further suggesting overfitting.

Conclusion

The plots indicate that while the model is performing well on the training data, it may not generalize as effectively to unseen validation data. This is evidenced by the increasing validation loss and fluctuating validation accuracy.

FUTURE WORKS:

1.Integration with Office Retinal Scanners:

- **Real-time Detection:** Integrate the trained model with office retinal scanners to perform real-time detection of eye conditions.
- **Automatic Alerts:** Develop a system that automatically alerts the medical staff or the user if any eye disease is detected.
- **Record Keeping:** Implement a database system to keep records of scans and detected conditions for future reference and trend analysis.

2.Expansion of Disease Detection:

- **Inclusion of More Diseases:** Extend the model to detect additional eye diseases such as diabetic retinopathy, glaucoma, cataracts, etc.
- **Comprehensive Dataset:** Collect and incorporate a more diverse and comprehensive dataset that includes images of various eye diseases.

3.Disease Prevention and Spread Mitigation:

- **Madras Eye Detection:** Specifically enhance the model to detect Madras eye (conjunctivitis) and implement measures to prevent its spread among coworkers.
- **Quarantine and Sanitization Alerts:** Develop features that alert the office administration to perform immediate sanitization and enforce quarantine measures to prevent the spread of infectious diseases.

4.User Interface and Usability Enhancements:

- **Mobile Application:** Develop a mobile application that can work with mobile phone cameras to detect eye conditions.
- **User-friendly Interface:** Create an intuitive and user-friendly interface for non-technical users to interact with the system easily.