1. **Objectives:**

* Using Python programming, build and save the binary image classification (ML or machine learning) models that can detect if a person has eyeglasses or not in a facial image.
* Compare the performances of the saved or pre-trained models.
* Build a web application using a hosting platform (example: Streamlit) that allows a user to upload a facial image and the app uses the pre-trained models to classify if a person has eyeglasses or not.

1. **Methodology:**

This section discusses the data source and the technical approaches used to fulfill the objectives.

1. **Image Data Source:** 6789 (35% validation/65% training split) images were obtained from a [Kaggle repository](https://www.kaggle.com/datasets/janwidziski/face-obstructions/data) as the image data source.

1. **ML Models in Python:** Jupyter Notebook installed in a PC (CPU powered) was used to code and execute different models that trained on image data using a convolutional neural network (CNN) implemented in Keras. These programs follow this typical sequence - train a binary classification model on image data (i.e., human face images with and without glasses), save the trained model, and classify new/test images (0 – no eyeglasses present, 1 – eyeglasses present) using the model.

Two categories of models were built –

1) custom model from scratch----------Category 1

2) custom model on top of the pre-trained MobileNetV2 model-------Category 2

Three models and two models were built based on Category 1 and 2 models respectively where the sole difference in the models within a category is the number of epochs. Adjusting epochs is a hyperparameter tuning technique.

|  |  |  |  |
| --- | --- | --- | --- |
| **Model #** | **Model Build**  **Type** | **Batch Size** | **Epoch #** |
| Model 1 | Category 1 - custom model from scratch | 100 | 3 |
| Model 2 | 100 | 50 |
| Model 3 | 100 | 100 |
| Model 4 | Category 2 - custom model on top of the pre-trained MobileNetV2 model | 100 | 10 |
| Model 5 | 100 | 50 |

Here's a summary of the functionalities of the two sets of models:

1. Import libraries:
   * Import necessary libraries as per program requirement.
2. Define directories:
   * Specify directories for training and validation/testing data.
3. Set image dimensions:
   * Set the width and height for input images.
4. Training parameters:
   * Set the number of training epochs and batch size.
5. Model architecture:
6. Category 1:

* Follows common CNN architecture including convolutional layers followed by max-pooling layers, and then fully connected layers. Add dropout layers.

1. Category 2:
2. Create base model:
   * Utilize MobileNetV2 as the base model with pre-trained weights on ImageNet.
   * Freeze the convolutional layers of the base model to retain pre-trained knowledge.
3. Build custom model:
   * Construct a custom model on top of the pre-trained base model employing transfer learning.
   * Add a Global Average Pooling layer, a dense layer with ReLU activation, a dropout layer, and a final dense layer with a sigmoid activation function.
4. Compile model:
5. Category 1: Compile the model using the *rmsprop* optimizer, binary cross-entropy loss function, and accuracy as the evaluation metric.
6. Category 2: Compile the model using the *Adam* optimizer, binary cross-entropy loss function, and accuracy as the evaluation metric.
7. Data augmentation:
   * Define data augmentation for the training set, which includes rescaling, shearing, zooming, and horizontal flipping.
8. Data generators:
   * Create data generators for both training and validation sets using the defined data augmentation and normalization.
9. Train the Model:
   * Train the model using the fit\_generator method, specifying the training and validation generators, as well as other parameters like steps per epoch and validation steps.
10. Plot training history:

* Plot loss and accuracy on training and testing data over epochs.

1. Save & load the model:
   * Save the trained model to a user specified file name with *‘.h5'* file extension and load it for making predictions on new images.
2. Prediction/Classification loop & export the results:

* Load images from a specified folder, preprocess them, and use the trained model to predict the class for each image in batch.
* Save prediction or classification results to an Excel file and download it.

1. **Aspects of the Two Model Categories:**

Category 1: Having complete control over the architecture allows designing a model specifically tailored to the intended task. Also, since there are no pre-training limitations, the model is not constrained by pre-trained weights making it easier to adapt the model to the provided dataset.

Category 2: This model category performs fine-tuning of pre-trained models by utilizing pre-trained models (e.g., transfer learning) on large datasets and fine-tuning them on the provided dataset. Models pre-trained on large image datasets, such as ImageNet, may have learned useful features that can be beneficial to the intended task.

1. **Using Streamlit as hosting platform:** Using Streamlit cloud hosting platform, a public website is created that allows and instructs a user to upload a facial image and the app uses 5 pre-trained models to display binary image classification results (0 – no eyeglasses present, 1 – eyeglasses present). The [Streamlit app](https://app-ml-nikesh-dscapstone.streamlit.app/) is linked to a [Github repository](https://github.com/nikAcharya1/streamlit-ML-Nikesh). The repository contains the saved models and a python program to execute the Streamlit web application.
2. **Comparison of performances of the models:** 
   * 1. [**35 New Test Images**](https://drive.google.com/drive/folders/1AbTbpHTA1oUGhWuUlvjDISBFHXhlTqHz?usp=drive_link)*(clickable link to view the images)***:**



|  |  |
| --- | --- |
| **Model #** | **Accuracy (%)** |
| Model 1 | 11 |
| Model 2 | 31 |
| Model 3 | 91 |
| Model 4 | 74 |
| Model 5 | 14 |

[*>> Link to loss and accuracy learning curves for all five models.*](https://docs.google.com/spreadsheets/d/1XKiHaasarMuvMecLOyYyZpUjQUsgSES4/edit?usp=drive_link&ouid=105718179794056849833&rtpof=true&sd=true)

Among the three models based on Category 1, Model 3 is the most accurate. The possible reason behind this – model 3 was trained for a larger number of epochs and is expected to have more refined weights, potentially capturing finer details in the source image data. It's common for models with larger number of epochs, such as Model 3 with 100 epochs, to exhibit signs of overfitting. Despite the overfitting observed in the loss and accuracy learning curves, Model 3 still performs well on test data, resulting in higher accuracy.

Among the two models based on Category 2, Model 4 is the more accurate than Model 5 even with a lower number of epochs.

* + 1. [**44 Test Images from Yale Database**](https://drive.google.com/drive/folders/1QlV0r0mqkiacMHBhbIY2I8Z1MFxqUyZd?usp=drive_link)*(clickable link to view the images)***:** Out of 165 face images of 15 persons (or 11 images per person) contained in the Yale database, 44 images (or image set for 4 persons) were manually cropped to get each image with human face covering majority of the image. Model 3 was observed to be the most accurate among all five models when testing the models with this pre-processed image dataset. Model 3 failed to correctly classify only 4 images (out of 44) and they were non-eyeglass faces.

|  |  |
| --- | --- |
| **Model #** | **Accuracy (%)** |
| Model 1 | 30 |
| Model 2 | 61 |
| Model 3 | 91 |
| Model 4 | 50 |
| Model 5 | 36 |

* + 1. **Conclusion:** Based on the two sets of test image data, Model Category 1 is observed to be more accurate than Model Category 2. The possible reasons are : (i) Category 2 or the pre-trained model might have learned features that are more generic and less task-specific, (ii) Category 2 may not work well if the training and validation datasets are significantly different from the datasets used to pre-train MobileNetV2, and (iii) Category 1 is not constrained by pre-trained weights making it easier to adapt the model to the training and validation datasets. It is also observed that all five models have higher accuracies when classifying face image with eyeglasses.

1. **Model Limitations and Challenges:**

* The models seem to perform more accurately on test images when following conditions are met:
  + 1. Only one person is in the image.
    2. Face should cover most of the image.
* Computational resources – training the models on CPU power is time consuming and can take hours.

**References:**

* + 1. Textbook - Practical Machine Learning and Image Processing: For Facial Recognition, Object Detection, and Pattern Recognition Using Python (Publisher: Apress, Publication year: 2019, ISBN: 9781484241486) by Himanshu Singh
    2. Kaggle Dataset - [Dataset for training Face obstruction type Classifier](https://www.kaggle.com/datasets/janwidziski/face-obstructions/data)
    3. [Python | Image Classification](https://www.geeksforgeeks.org/python-image-classification-using-keras/) using Keras by Geeksforgeeks website.
    4. [Training and Validation Loss in Deep Learning](https://www.baeldung.com/cs/training-validation-loss-deep-learning) by Baeldung website.
    5. Yale Database: http://cvc.yale.edu/projects/yalefaces/yalefaces.html