**Overview**

* **Project Name**: Celebrity Classification through Computer Vision
* **Problem Statement:** Recognize celebrity in images using Artificial Intelligence
* **Objective**: The aim of this project is to classify pictures of celebrity as one of 17 celebrities. The 17 celebrities are famous personalities from different fields including but not limited to Movie industry, Tech Industry, Chess, Boxing and Tennis
* **Outcome**: A computer vision model classifying the celebrities with 90% accuracy

**Table of Contents**

1. **Abstract**
2. **Getting Started**
   * **Prerequisites**
   * **Celebrities to Identify**
3. **Methodology**
   1. **Dataset Collection**
   2. **Data Pre-Processing**
   3. **Model Architecture**
   4. **Model Evaluation**
   5. **Model Saving and Making Predictions**
   6. **Web applications**
4. **Project Structure**
   * **Directory Structure**
   * **Explanation of Files and Folders**
5. **Usage**
6. **Architecture**
7. **License**
8. **Authors**
9. **References**

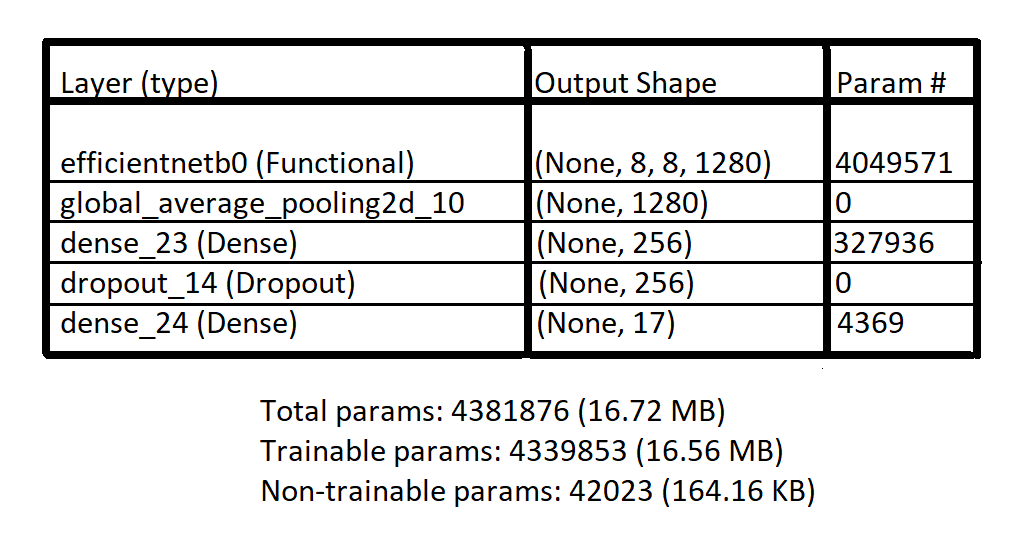
**1. Abstract**

The goal of this project is to develop a deep learning-based system for recognizing celebrities from images using a convolutional neural network (CNN). The dataset comprises images of 17 celebrities, which are pre-processed and divided into training and validation subsets. The model is built using TensorFlow and Keras, leveraging transfer learning with EfficientNetB0 and a CNN architecture to learn distinctive features of celebrity faces. The training process involves data augmentation and regularization techniques to enhance model generalization. The performance of the model is evaluated using standard metrics including accuracy, precision, recall, and F1-score. The final model achieves a 90% accuracy on the validation set, demonstrating its effectiveness in correctly identifying celebrities. Additionally, the project involves techniques for saving and loading the model, as well as generating predictions on new data. Finally, the results of this project are demonstrated in a web application hosted locally through FastAPI and Uvicorn. This system has potential applications in automated celebrity recognition for media, entertainment, and security industries.

**2. Getting Started**

* **Prerequisites**:
  + python
  + TensorFlow
  + Scikit-Learn
  + Matplotlib
  + Seaborn
  + Numpy
  + Pandas
  + os library
  + fast Api
  + uvicorn
  + jinja2
  + bs4
  + requests
  + selenium
  + time
  + PIL
* **Celebrities to Identify:**
  + Brad Pitt
  + Chris Hemsworth
  + Christiano Ronaldo
  + Elon Musk
  + Eminem
  + Garry Kaspov
  + Jeff Bezos
  + Leonardo di Caprio
  + Lionel Messi
  + Magnus Carlsen
  + Mark Zuckerburg
  + Mike Tyson
  + Muhammad Ali
  + Novak Djokovic
  + Sharukh Khan
  + Tom Holland
  + Will Smith

**3. Architecture**

****

Only the last 15 layers of the efficientnetb0 are trainable

* + This number was determined after experimenting with different number of layers. By keeping only 15 layers trainable, the highest accuracy was achieved on validation dataset without the model overfitting to the training dataset
  + Further documentation on efficientnetb0 can be found [here](https://www.mathworks.com/help/deeplearning/ref/efficientnetb0.html)

**4. Methodology**

1. **Dataset Collection**
   1. After shortlisting all the celebrities, I wanted to work on, (See in section 2 of the same document) I created a python script to scrape all the images loaded from a particular search. Multiple python libraries were used in achieving this task, i.e., bs4, requests, selenium, os, time and PIL. Using this script an average of 250 images of each celebrity was downloaded.
   2. Once Images of all the celebrities were downloaded, each picture was edited manually, leaving only the celebrity in the picture. To achieve this either of the three following methods were used:
      1. Other people were cropped out of the picture
      2. Other people were removed using erase tool in images in Windows 10
      3. Some part of the picture was cropped out and some was removed using erase tool

There was no set rule to establish which rule would be applied instead, I decided how to handle each image purely on intuition

1. **Data Pre-Processing**
   1. The images were loaded as [tf.data.Datasets](https://www.tensorflow.org/api_docs/python/tf/data/Dataset) and divided into training and validation subsets using the TensorFlow

[tf.keras.preprocessing.image\_dataset\_from\_directory](https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image_dataset_from_directory) library. As the dataset was relatively small, the training size was 80% while the rest of the 20% were used for validation

* 1. After loading, data exploration was not required as I was already familiar with the dataset so I moved on to augmenting the images using TensorFlow [data augmentation layers](https://tensorflow.org/tutorials/images/data_augmentation) so as to leading the model to generalize better.

Although it is important to note that data was augmented for use after the first model, as I wanted to see how the model fits to only the dataset

1. **Model Architecture**
   1. It is apparent that a basic convolutional model wouldn’t do well on a complex problem with only a small dataset, so instead of working on a basic convolutional model, I directly started out with the idea of transfer learning.
   2. After researching the internet, reading multiple articles, and comparing architecture and performance metrics of multiple pretrained networks, I was able to determine that [EfficientNetB0](https://www.mathworks.com/help/deeplearning/ref/efficientnetb0.html) would be the best choice for the task at hand due to multiple reasons:
      1. **Transfer Learning Capability:** EfficientNetB0, pretrained on large datasets like ImageNet, provides a robust starting point for transfer learning. This pretraining enables the model to leverage learned features that are beneficial for a wide range of image classification tasks, including celebrity recognition.
      2. **Performance:** EfficientNetB0 has demonstrated excellent performance on various image classification benchmarks. Its ability to achieve high accuracy makes it suitable for tasks requiring detailed and accurate recognition, such as distinguishing between celebrity faces.
      3. **Scalability:** The EfficientNet family includes models of varying sizes (from B0 to B7), allowing for scalability depending on the available computational resources and the specific requirements of the task. EfficientNetB0, being the smallest in the family, is particularly useful when computational efficiency is a priority.
      4. **Regularization Techniques:** EfficientNet models incorporate advanced regularization techniques like Dropout and Batch Normalization, which help in reducing overfitting and improving generalization performance on the validation dataset.
      5. **Model Size and Speed:** EfficientNetB0 strikes a good balance between model size and inference speed. This makes it suitable for applications that may need to run on limited hardware resources or in real-time scenarios.
   3. I first started out with the following architecture transfer learning:
      1. removing the top layer and freezing all the other layers of the pretrained EffecientNetB0 model except the last 20 layers
      2. inserting a global average pooling layer and two densely connected neural network layers with a dropout layer on top of the pretrained network.

This network started out well, but after only achieving 80% accuracy on validation set it started overfitting on the training dataset so much so that after about 18 epochs it had almost 100% accuracy on training data, while only 86% accuracy on validation data. This model is referred to in the code as ‘model1’

* 1. To overcome the problem of overfitting I tweaked the model architecture in so that none of the layers of the pretrained EffecientNetB0 model were trainable. This model is referred to in the code as ‘model2’

This overcame the problem of overfitting however the model’s performance on validation dataset levelled out at only 73%

* 1. Next keeping everything else same, I changed the number of trainable layers in the pretrained EfficientNetB0 to 15 (these are the last fifteen layers).

This model, referred to as ‘model3’ in the code, achieved high accuracy on validation dataset, 90%, while not overfitting on the training dataset (94% accuracy).

* 1. The optimizer optimizer, loss function, and metrics used during training were [adam](https://keras.io/api/optimizers/adam/) with an initial learning rate of 0.001, [sparse\_categorical\_crossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/losses/sparse_categorical_crossentropy) and [accuracy](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy) respectively
  2. To optimize model training, I used three callbacks:
     1. **Learning rate scheduler**: Every time the validation loss does not decrease for three consecutive epochs this callback decreases the learning rate by a factor of 10. This allows the model converge faster to the optima by breaking learning plateaus
     2. **Checkpoint:** Each time the validation accuracy increases, this callback is to save the model, hence saving the best model, if the model starts to overfit too much, or the training process unexpectedly interrupts
     3. **Early Stopping:** this callback stops the model if validation loss does not decrease for 10 consecutive epochs, stopping the model training early, hence preventing the model form overfitting

1. **Model evaluation**
   1. I evaluated each model through loss and accuracy graph one plotted for training dataset and one for validation dataset. This allowed me to see the trends of overfitting and hence, decide the next course of action
   2. After finalizing ‘model3’ as the final model, I evaluated it on all the standard metrics such including accuracy, precision, recall, and F1-score, using a [classification report](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html). A [confusion matrix](https://www.tensorflow.org/api_docs/python/tf/math/confusion_matrix) was also generated to assess individual celebrity classification performance.
2. **Model Saving and making predictions**
   1. The ‘not overfit’ version of model3 was already saved by the checkpoint callback at validation accuracy almost 90% and training accuracy almost 94%
   2. I created a function to preprocess single images into a format suitable for model input (none, 225, 225, 3).
   3. The model was tested on new images to check its response to unseen data:
      1. The model classified unknown individuals as celebrities with low confidence, enabling a classification as "Unknown" if confidence was below a threshold.
      2. High resemblance to celebrities resulted in high-confidence misclassifications, for example, Evander Holyfield was classified as Mike Tyson with a confidence of 80%, highlighting a limitation
3. **Web application**
   1. The model was integrated into a web application to classify celebrities in real-time on a user interface
   2. The web applications user interface was designed using HTML, CSS and Java Script
   3. The back-end was developed using python, and [FastAPI](https://fastapi.tiangolo.com/) framework
   4. Uploaded pictures were saved locally, then processed as described above

**5. Project Structure**

* **Directory Structure:**
* **E****xplanation of Files and Folders:** 
  + Dataset: This folder contains all the images of each celebrity separated in their respective folders
  + Code: This folder has a single .ipynb file containing the code for data preprocessing, Model building, training and evaluation
  + Model: This folder contains the trained model for the image recognition task
  + my\_fastapi\_app: This folder contains relevant files and folders pertaining to the front-end and back-end aspects of the websites, Screenshot of which can be seen in the Results folder
    - Static / upload: This folder will contain all the images uploaded to the web server for further processing
    - Template: This folder contains the front-end – HTML, CSS, Java Script – code.
  + Results: This folder contains screenshots showing the final results, demonstrating the model's predictions as displayed through the website
  + Web Scraping : This folder contains the code and the readme file for scraping the internet for pictures

**Note: Most of these folders also contain a readme file containing further information about the contents of that directory**

**6. Usage**

* If you want to retrain the model yourself, comment out the or delete the code cells for model 1 and model 2. The final architecture implemented is in model 3. For more information regarding the code see the readme file in the code folder
* Either save the model you train, or download the model in the “model” folder.
* It is important that you keep the file hierarchy in the “my\_fastapi\_app” the same. If you do decide to change the hierarchy, make sure the changes are reflected in each code file in the “my\_fastapi\_app” folder or any of its subfolder
* Make sure all the paths specified in all the files containing any code are true
* To start the webserver on local host, start a command prompt (or equivalent in whatever operating system you are using) in the folder where the main file is (In the current folder hierarchy, it has to be started in my\_fastapi\_app folder) and type in the following “ uvicorn main:app --reload ” and then navigate to “ <http://127.0.0.1:8000> ,” unless you change it from the code

**7. License**

* This is an open-sourced project. Feel free to copy, use and experiment on it however you like

**8. Acknowledgments**

* **Contributors**: This entire project, including its dataset was compiled and developed by Hamza Amir. For any suggestions feel free to email at [hamzaamir0616@gmail.com](mailto:hamzaamir0616@gmail.com) or connect at [LinkedIn](https://www.linkedin.com/in/hamza-amir-0616m/)
* **References**:
  + [python](https://www.python.org/doc/)
  + [TensorFlow](https://www.tensorflow.org/api_docs)
  + [Scikit-Learn](https://scikit-learn.org/stable/)
  + [Matplotlib](https://matplotlib.org/stable/index.html)
  + [Seaborn](https://seaborn.pydata.org/)
  + [Numpy](https://numpy.org/doc/1.26/)
  + [Pandas](https://pandas.pydata.org/docs/reference/index.html#api)
  + [Python os Module](https://docs.python.org/3/library/os.html)
  + [fast Api](https://fastapi.tiangolo.com/)
  + [uvicorn](https://www.uvicorn.org/)
  + [jinja2](https://jinja.palletsprojects.com/en/2.10.x/)
  + [bs4](https://pypi.org/project/beautifulsoup4/)
  + [requests](https://pypi.org/project/requests/)
  + [Selenium](https://pypi.org/project/selenium/)
  + [time](https://docs.python.org/3/library/time.html)
  + [PIL](https://pypi.org/project/pillow/)
  + [FastAPI](https://fastapi.tiangolo.com/)
  + [confusion matrix](https://www.tensorflow.org/api_docs/python/tf/math/confusion_matrix)
  + [classification report](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)
  + [adam](https://keras.io/api/optimizers/adam/)
  + [sparse\_categorical\_crossentropy](https://www.tensorflow.org/api_docs/python/tf/keras/losses/sparse_categorical_crossentropy)
  + [accuracy](https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy)
  + [EfficientNetB0](https://www.mathworks.com/help/deeplearning/ref/efficientnetb0.html)
  + [data augmentation layers](https://tensorflow.org/tutorials/images/data_augmentation)
  + [tf.data.Datasets](https://www.tensorflow.org/api_docs/python/tf/data/Dataset)
  + [tf.keras.preprocessing.image\_dataset\_from\_directory](https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image_dataset_from_directory)