**BMI Prediction in Real Time**

**Problem Description:**

Body Mass Index (BMI) prediction approaches now in use lack real-time functionality and do not make use of cutting-edge image processing techniques. By creating a straightforward online API that uses camera input to predict a user's BMI in real-time, this project seeks to address these constraints. Pre-trained picture models, such VGG Face, will be used by the API and adjusted with supplied data to increase prediction accuracy.

Currently, available BMI calculators require users to manually input their information, which can be cumbersome and time-consuming. Moreover, these calculators do not take advantage of image analysis techniques, which can provide valuable insights into a person's body composition. Users will have a quick and easy way to get their BMI forecasts by developing a web API. Users can enter their webcam stream for analysis using the API. With these real-time capabilities, users will receive prompt feedback and be able to routinely check their BMI, raising awareness and promoting healthier lifestyle choices.

The web API development will incorporate the fine-tuned model into accessible frameworks like Jupyter Notebook and Flask. These frameworks will enable users to interact smoothly with the API by either using webcam input or uploading images. The API will analyze the input images, extract important features, and employ the fine-tuned model to predict the user's BMI. The results will be presented in a clear and comprehensible format, making it easy for users to understand the outcomes.

By addressing the limitations of existing BMI prediction methods, this project aims to provide a user-friendly and efficient solution for real-time BMI prediction. By leveraging pre-trained image models and webcam input, the web API will empower users to monitor their BMI conveniently, foster awareness about their body composition, and promote healthier lifestyle choices.

**Data Preparation:**

To prepare the dataset containing images and their corresponding BMI values, the user should update the image\_dir variable with the correct directory path. This directory should contain the images. For additional processing, the images will be resized to 224x224 pixels. The resized pictures will be kept in the X list, and their associated BMI values will be kept in the Y list.

The data needs to be split into training and testing sets. This can be done using the train\_test\_split function from the sklearn.model\_selection module. The purpose of this step is to evaluate the model's performance. The training images will be kept in the X\_train and y\_train variables, along with the appropriate BMI values. The testing images and their corresponding BMI values will be stored in X\_test and y\_test, respectively.

Data augmentation strategies are employed to enhance the versatility and robustness of models. These techniques involve introducing artificial variations to the training dataset, thereby increasing its diversity. By applying transformations such as rotation, shift, shear, zoom, and horizontal flip to images, the dataset becomes enriched with a wider range of examples. This augmentation process aims to improve the model's ability to handle different types of inputs and enhance its generalization performance by exposing it to a broader spectrum of variations commonly encountered in real-world scenarios.

To implement data augmentation, the user needs to import the ImageDataGenerator class from the tensorflow.keras.preprocessing.image module. They can then create an instance of ImageDataGenerator and configure the desired augmentation techniques.

Finally, a train\_data\_gen generator can be created by calling the flow method on the ImageDataGenerator object. The flow method takes the training data (X\_train and y\_train) as input, along with the desired batch size. This generator will generate augmented images during the training process.

**Model Architecture:**

The model is built using the Sequential API from TensorFlow's Keras library, allowing the layers to be stacked in a sequential manner. The architecture includes various types of layers such as convolutional, pooling, flatten, dense, dropout, and a prediction layer.

The first layer, Con\_2d\_1, is a convolutional layer that applies 32 filters of size 2x2 to the input data. It utilizes the ReLU activation function to introduce non-linearity and capture complex patterns.

Following Con\_2d\_1, the Pool\_2d\_1 layer performs max pooling with a pool size of 2x2. This operation reduces the spatial dimensions of the previous layer's output by selecting the maximum value within each pooling region.

The next layer, Con\_2d\_2, is another convolutional layer that applies 128 filters of size 2x2. It also uses the ReLU activation function to introduce non-linearity and extract higher-level features from the input.

After Con\_2d\_2, the Pool\_2d\_2 layer performs max pooling with a pool size of 2x2, further decreasing the spatial dimensions of the feature maps.

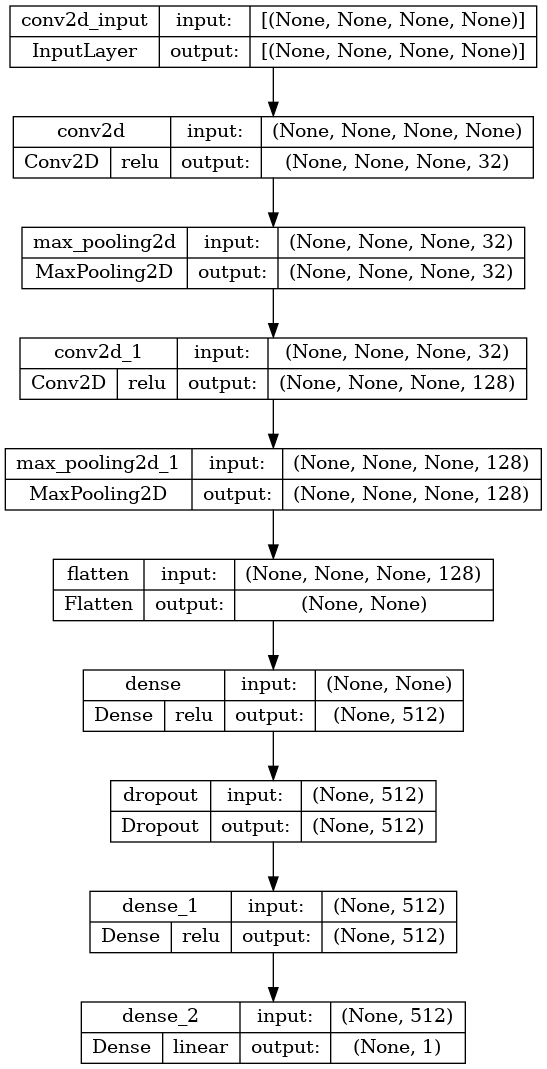
The flatten\_layer comes next and is responsible for flattening the output of the previous layer into a 1D vector. This transformation is necessary to connect the convolutional layers to the subsequent dense layers.

The first dense layer, dense\_layer\_1, consists of 512 neurons and uses the ReLU activation function. It adds more non-linearity to the model and helps in learning complex relationships between the features.

To prevent overfitting, the dropout\_layer is included after dense\_layer\_1. It applies dropout regularization with a rate of 0.3, randomly setting a fraction of input units to 0 during training. This technique helps in reducing the reliance on specific features and improves the generalization capability of the model.

The second dense layer, dense\_layer\_2, also consists of 512 neurons with the ReLU activation function. It further enhances the non-linear modeling capacity of the network.

Finally, the prediction\_layer serves as the output layer of the model. It comprises a single neuron with a linear activation function, producing the predicted BMI value based on the learned representations from the preceding layers.



**Model Evaluation:**

The training process incorporates various callbacks such as Early Stopping, Model

Checkpoint and Reduce Learning Rate on Plateau. These callbacks are set up to configure the training process. The fit method is then used with the model, along with the defined callbacks. The training data is provided through the train\_data\_gen generator, which also includes the defined callbacks. The desired number of training epochs is specified to determine how many times the model should iterate over the complete training dataset.

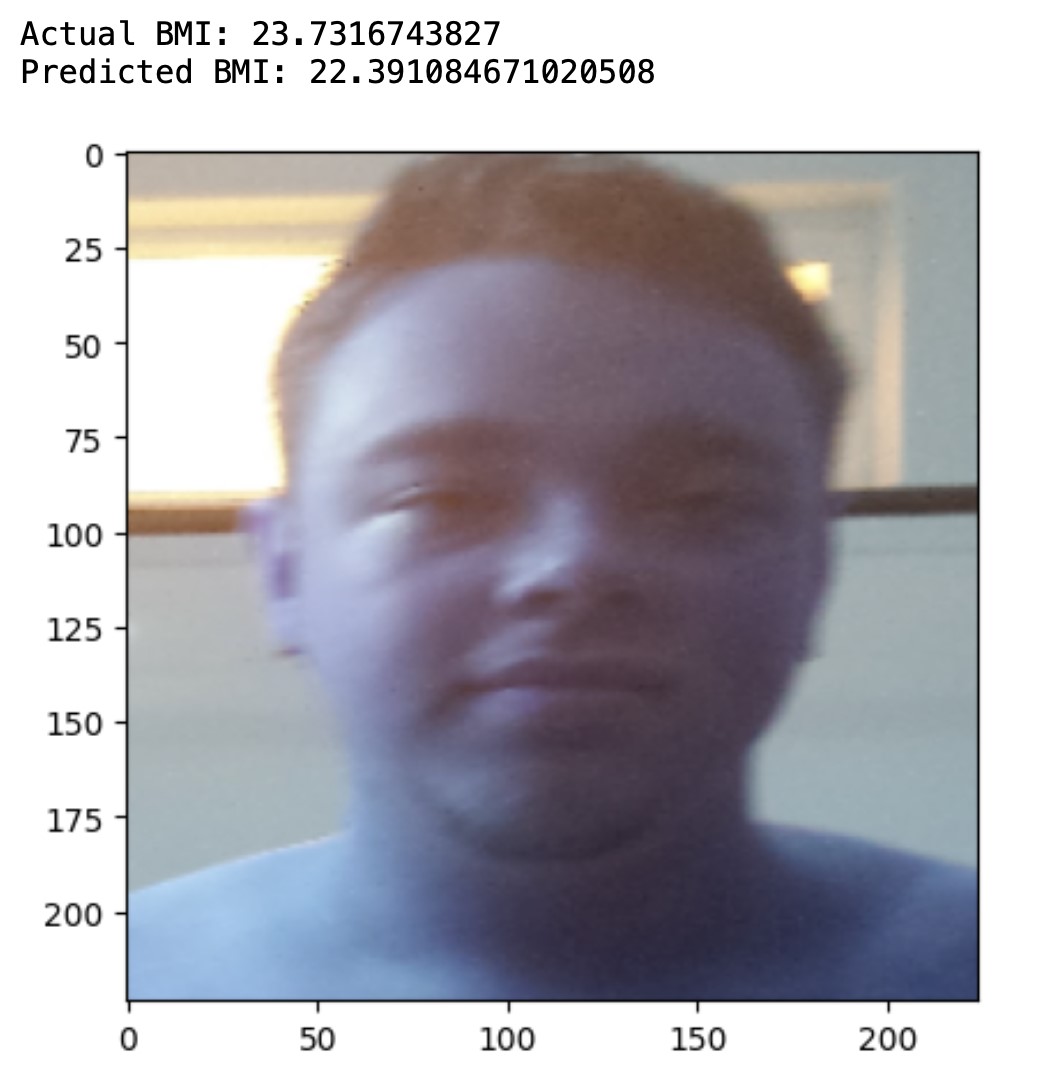
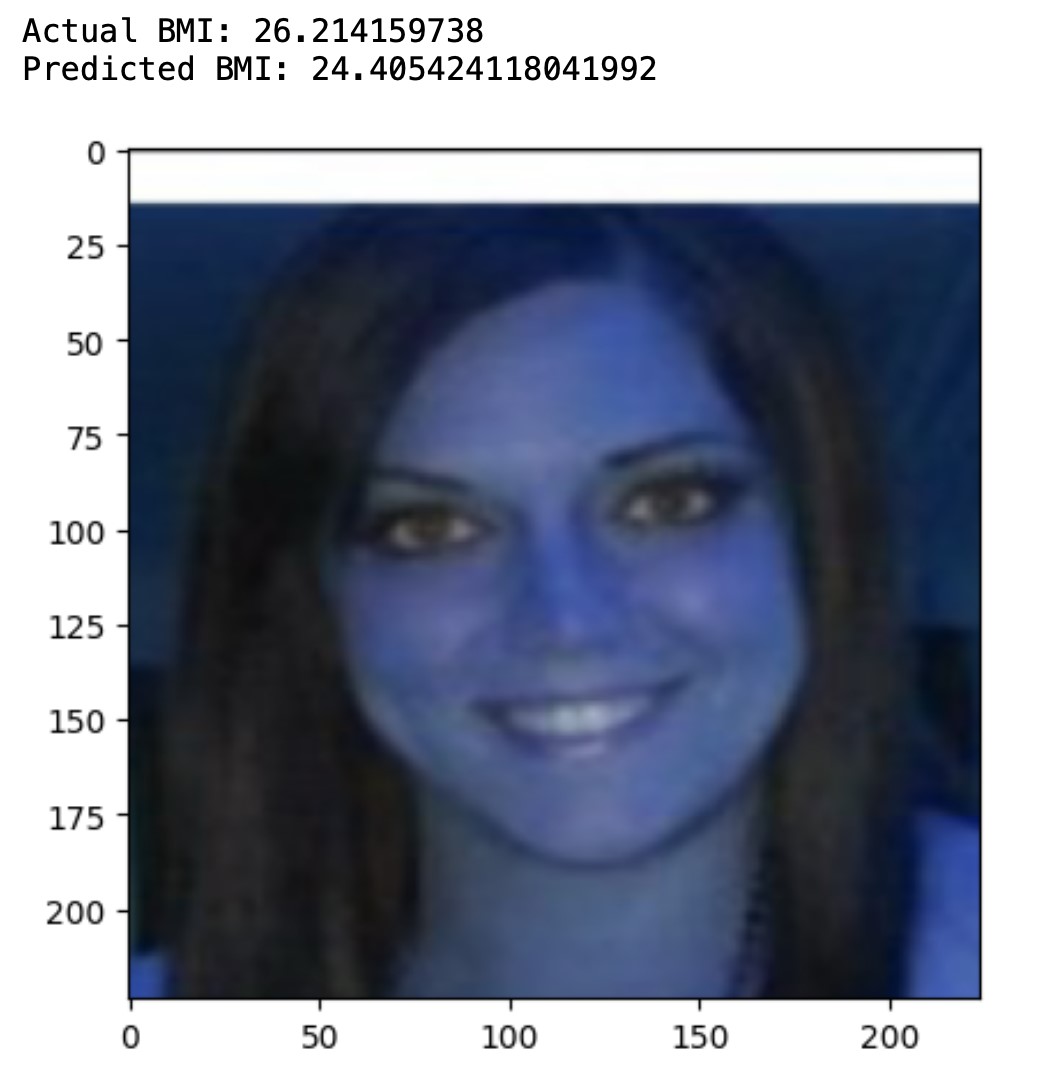
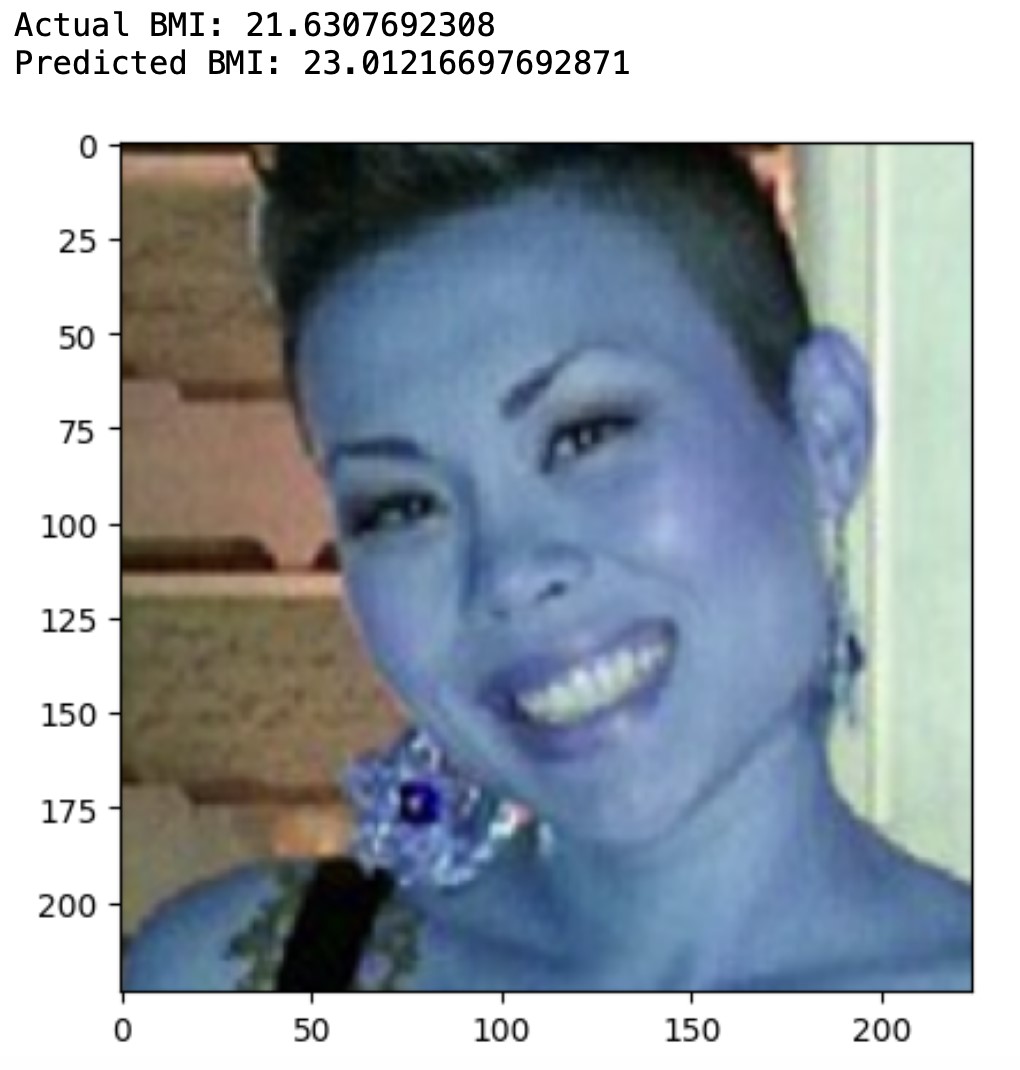
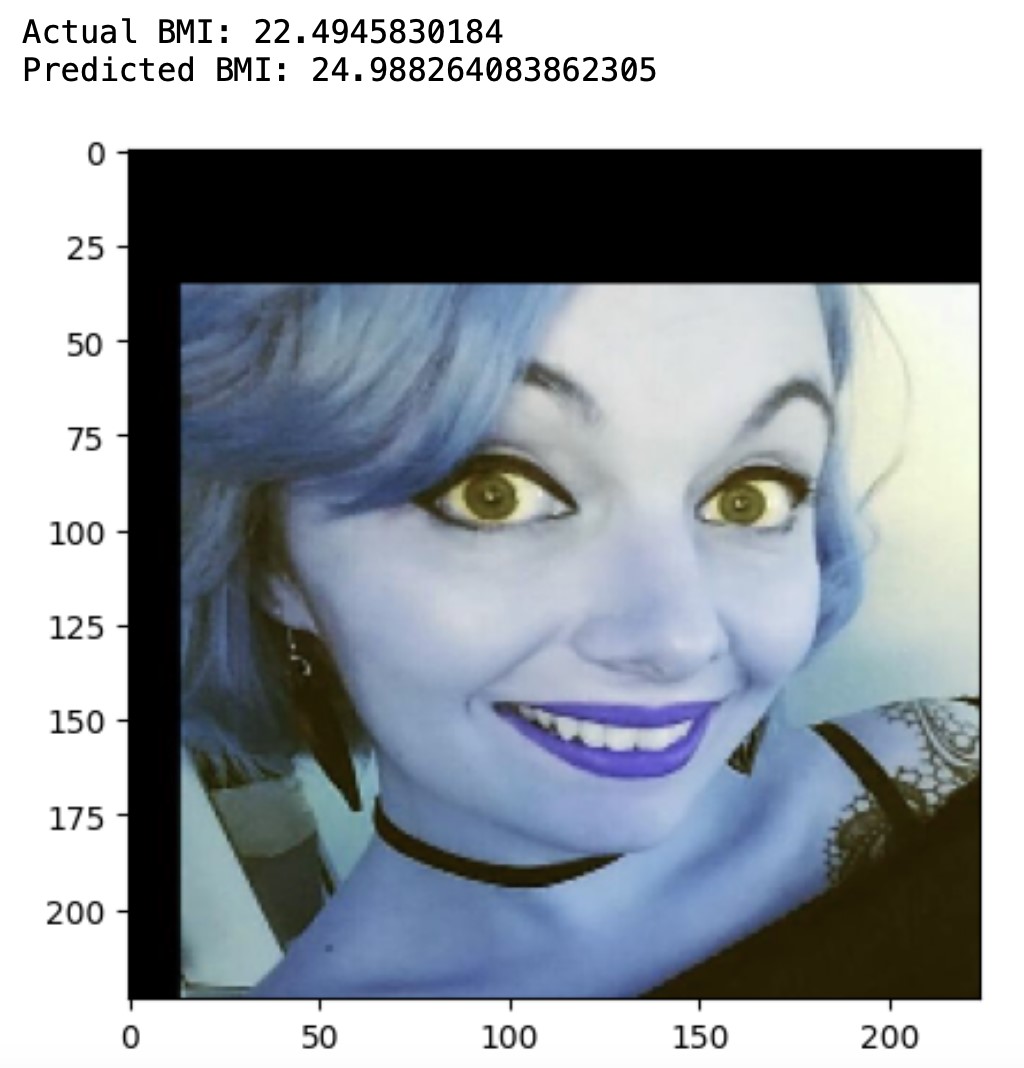
Evaluation of the model's performance on a different testing set is crucial after the training phase. This is usually accomplished by using the evaluate method to the trained model, which accepts as inputs the testing images (X\_test) and the BMI values that correspond to them (y\_test). The evaluate technique computes a variety of evaluation measures, which offer insights into how well the model performs in predicting BMI values.

The evaluation metrics can be saved for later use, allowing comparisons with other models or tracking the model's performance over time. These assessment measures are used to quantify the model's performance and help assess the model's precision and dependability when predicting BMI values from input photos.

To perform real-time predictions using webcam input, the process involves continuously capturing frames from the webcam using OpenCV's cv2.VideoCapture. Each captured frame is then preprocessed by resizing it to the desired input size, typically 224x224, and converting it to a suitable format for the model. Subsequently, the trained model is utilized to predict the Body Mass Index (BMI) for each frame using the model.predict function. Depending on the application's specific requirements, these predictions can be either displayed or processed further. This workflow allows for real-time BMI estimation using webcam input by leveraging computer vision techniques and a pre-trained model.

**Results:**

Here, we can see the actual BMI result from the test set (y\_test), as well as the corresponding predicted BMI result from the implemented model. By comparing these values, users can assess how accurately the model predicts the BMI for different individuals.



**Deployment:**

A web application is developed using Flask, which is a Python web framework known for its simplicity and versatility. The purpose of this application is to deploy a BMI prediction model that utilizes deep learning and computer vision techniques. TensorFlow, a popular deep learning library, is employed for model training and inference. Additionally, the OpenCV library is used for face detection and image preprocessing tasks. The application integrates the webcam feed into the user interface, allowing users to capture live video for BMI prediction. The face detection algorithm provided by OpenCV is utilized to identify faces in each frame of the video. These detected faces are then processed and prepared as input for the BMI prediction model.

The process begins by initializing the Flask application and importing the required modules. Additionally, the BMI prediction model and face cascade classifier are loaded for later use. To enable webcam access, the necessary configurations are set up. A crucial function called "generate\_frames" is defined to ensure a continuous capture of frames from the webcam. Once the BMI is predicted, it is displayed on the captured frames to provide visual feedback. These frames are encoded as JPEG images and sent as HTTP responses to the client.

To establish the necessary routes for the Flask application, two routes are defined: one for the homepage and another for the video feed. Each route is associated with a specific function. The homepage route renders the index.html template, which must be located in the same directory as the Python script. This HTML template can be customized to display the camera video feed and any additional elements required for the web application.

The video feed route corresponds to a function that returns the continuously generated frames. After defining the routes and functions, the Flask app is executed by running the application using the specified SSL context, host, and port. This ensures that the web application is securely served over HTTPS and is accessible on the specified host and port.

**Conclusion:**

In conclusion, this documentation outlines a process for developing a straightforward online API that uses a trained picture model to estimate a user's BMI in real-time. The objective is to develop a simple and user-friendly BMI prediction interface that accepts webcam input.

In order to improve the performance of the model, data augmentation techniques may also be used after loading and resizing the input photographs and dividing the data into training and testing sets. Pre-trained and custom layers are combined in the model architecture, which is also built with the proper optimizer, loss function, and evaluation metrics.

Users can follow the documentation's instructions for data preparation, model training, and deployment. It also draws attention to important factors like model review and data augmentation. The offered Instructions can be modified by users to meet their unique needs and preferences.

By employing this technology, users can build a web API that predicts BMI in real-time based on webcam input, making it simple and accessible for users to keep track of their BMI.