Final Assignment Submission

Machine Learning & Predictive Analytics

Face-to-BMI

Submitted by
Swathi Ganesan
12372237
swathiganesan@uchicago.edu

PREDICTION OF BMI FROM FACIAL IMAGES USING REAL-TIME PROCESSING AND MACHINE LEARNING

Introduction

Brief Overview

Body weight and BMI have received an increased attention in the recent years. The fact that abnormal BMI could cause various diseases has increased research of contactless and low-cost estimation of BMI. This work attempts to emulate the ***Face-to-BMI: Using Computer Vision to Infer Body Mass Index on Social Media paper***, investigating ways to predict near accurate BMI from facial images by implementing the ***VGGFace pre-trained model*** to extract facial features and fine-tuning by adding fully connected layers for BMI and gender. For face detection, ***Multi-task Cascaded Convolutional Neural Networks*** have been employed. A Streamlit application is deployed to evaluate BMI along with gender from human facial real-time images.

Problem Statement and Objectives

Weight and BMI are pertinent indicators for health conditions and excessive weight can be associated to obesity, diabetes, and cardiovascular diseases while lack of proper weight can lead to malnutrition related diseases. In this context, the method presented in this project contributes to image-based automated self-diagnostic which is the current trend of neural networks in the medical field. The goal of this work is to investigate the feasibility of body weight and BMI analysis from the visual appearance of real-time photographs of the human face. The proposed method is useful in establishing the relation between the characteristics of the human face and the body, such as body height and weight. The widely used body fat Indicator the BMI (body mass index) is calculated using the formula :

$$\text{BMI} = \frac{\text{weight (kg)}}{\text{height (m}^2\text{)}}$$

We formulate the BMI prediction from facial features as a computer vision problem, and evaluate our approach on a large database with about 4,000 face images and data containing gender information. A promising result has been obtained, which demonstrates the feasibility of developing a computational system for BMI prediction from face images at a large scale.

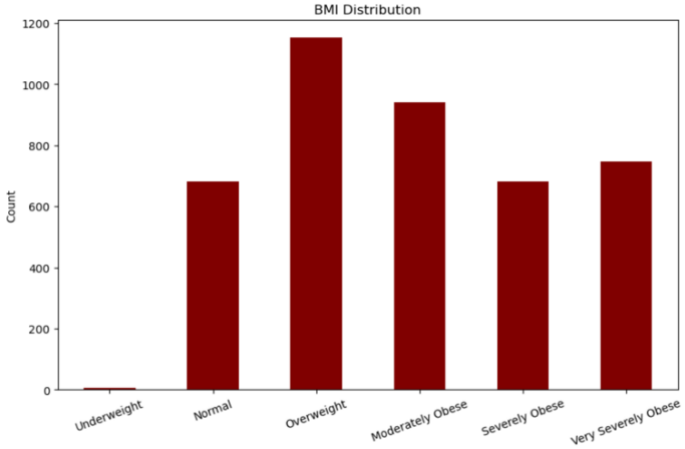
Relevance of face-to-BMI detection

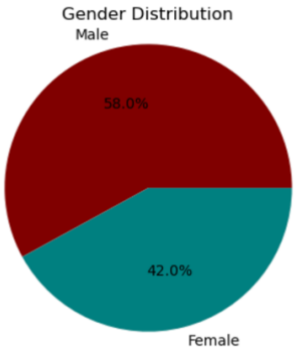
- Early health risk detection: Face-to-BMI detection identifies health risks early, enabling timely interventions.
- Non-invasive and accessible: Unlike traditional methods of measuring BMI that require physical measurements or invasive procedures, face-to-BMI uses computer vision, making it convenient and accessible for widespread use.
- Time and cost efficient: Provides quick and cost-effective BMI assessment compared to traditional methods.
- Privacy-preserving: Analyzes facial features, minimizing the need for personal data collection.
- Personalized health assessments: Offers tailored health assessments based on facial features and BMI estimation.
- Integration with technology: Seamlessly integrates with existing platforms and applications for BMI monitoring and health tracking.
- Automation and scalability: Automate BMI estimation and scales for efficient population-level health monitoring.

Dataset Description

Details of the dataset used for the project

The dataset comprises of 4206 faces with corresponding gender and BMI information extracted from the VisualBMI dataset and collated with relevant information needed for our assessment(as is in our reference paper). Of these, seven were in the underweight range (16 < BMI ≤ 18.5), 680 were normal (18.5 < BMI ≤ 25), 1151 were overweight (25 < BMI ≤ 30), 941 were moderately obese (30 < BMI ≤ 35), 681 were severely obese (35 < BMI ≤ 40) and 746 were very severely obese (40 < BMI). The dataset contained 2438 males and 1768 females.





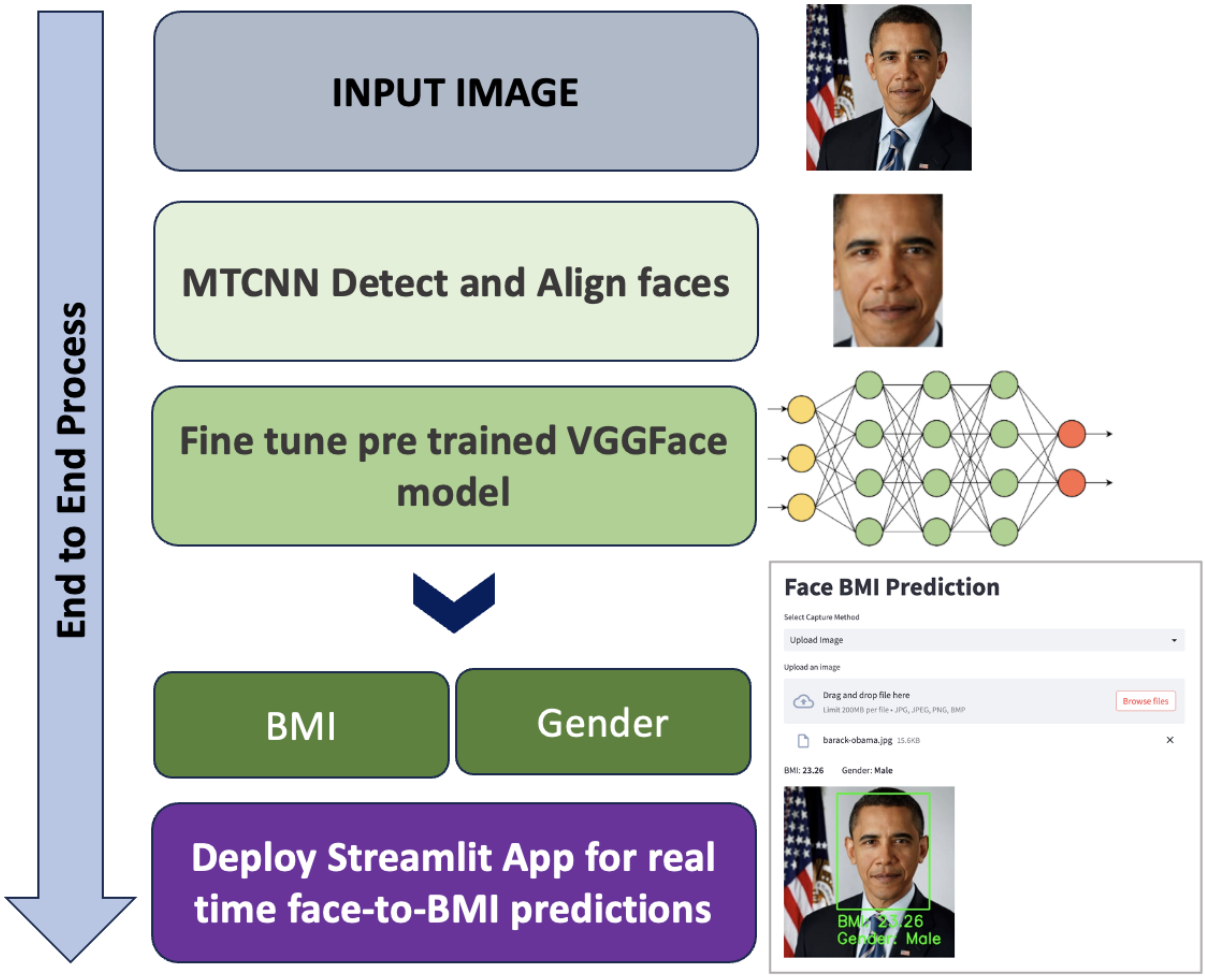
Overview of the attributes and labels in the dataset

Attribute	Label	Description
bmi	Numeric	Body Mass Index (BMI)
gender	Categorical	Gender (Male/Female)
is_training	Numeric	Training flag (1 if in training set, 0 otherwise)
name	Text	Name/identifier of the image

Data pre-processing steps and considerations

- ⇒ Encoded gender to Binary
- ⇒ Removed rows in the data frame that did not have an image file associated. Remaining data : 3962 rows
- ⇒ Train-Test split with 3210 and 752 rows respectively
- ⇒ Applied MTCNN to crop face images. Some images had no face identified.
- ⇒ Final data dimensions : **Train : (3203,3) Test : (750,3)**

Implementation



Since we are going to predict BMI and Gender from the same image, we can share the same backbone for the two different prediction heads and hence only one model will be maintained.

```
[input image] => [VGG16] => [separate dense layers] x2 => weighted([BMI], [GENDER])
```

Methodology

Step 1 : Faces aligned using MTCNN

MTCNN (Multi-task Cascaded Convolutional Networks) is a widely used deep learning model for face detection and facial landmark localization. It is commonly employed in image pre-processing tasks before performing further analysis or tasks related to face recognition. The MTCNN algorithm consists of three cascaded stages:

1. **Face Detection:** The first stage of MTCNN is responsible for detecting the presence of faces in an image. It uses a convolutional neural network (CNN) to scan the image at multiple scales and locations, identifying potential face regions. These regions are then refined using bounding box regression to accurately localize the faces.

2. **Facial Landmark Localization:** Once the faces are detected, the second stage of MTCNN is employed to locate the facial landmarks, such as the eyes, nose, and mouth. It uses another CNN-based model to predict the coordinates of these landmarks. By identifying the facial landmarks, MTCNN obtains more precise information about the facial structure, enabling subsequent analysis and applications.

3. **Face Alignment:** The final stage of MTCNN performs face alignment, which aims to normalize and align the detected faces. By applying transformation techniques based on the detected facial landmarks, the faces are aligned to a standardized pose and scale. This step is crucial for improving the accuracy and robustness of subsequent face-related tasks.

MTCNN is known for its accuracy and efficiency in face detection and facial landmark localization tasks. It is capable of handling faces at different scales, orientations, and angles. MTCNN progressively refines the face detection and landmark localization results, leading to improved performance. Test and train data sets are pre-processed using MTCNN before training and evaluating using our model.

Faces before and after processing using MTCNN

Step 2 : Data Augmentation and Batch Generation using ImageDataGenerator

An **ImageDataGenerator** from the Keras library is created to perform data augmentation, which is a technique used to artificially expand the training dataset and improve model generalization. Data augmentation techniques are applied to the images, such as rescaling the pixel values. The ImageDataGenerator class is used to define the data augmentation settings.

The ***target_size*** parameter sets the dimensions to which the images will be resized during the data augmentation process. In this case, the images are resized to a size of *224x224* pixels. The ***batch_size*** parameter determines the number of images and labels in each batch generated by the data generator. The ***class_mode*** parameter is set to *'multi_output'* since the model has multiple output layers for predicting **'bmi'** and **'gender'** simultaneously. The ***shuffle*** parameter is set to *True* to shuffle the training data before each epoch. The code performs data augmentation and generates batches of training and validation images with their corresponding labels.

```python
# Create an ImageDataGenerator for data augmentation
datagen = ImageDataGenerator(rescale=1./255)

# Generate batches of training images and labels
train_generator = datagen.flow_from_dataframe(
    dataframe=train,
    directory=train_processed_dir,
    x_col='name',
    y_col=['bmi', 'gender'],
    target_size=(224, 224),
    batch_size=32,
    class_mode='multi_output',
    shuffle=True
)


# Generate batches of validation images and labels
valid_generator = datagen.flow_from_dataframe(
    dataframe=valid,
    directory=test_processed_dir,
    x_col='name',
    y_col=['bmi', 'gender'],
    target_size=(224, 224),
    batch_size=32,
    class_mode='multi_output',
    shuffle=False
)
```

Step 3 : Fine tuning pre-trained VGGFace model

This section showcases the process of fine-tuning a pre-trained VGGFace model, with different architectures for the tasks of BMI and gender prediction by leveraging the pre-trained features learned from a large dataset.

Model 1 : VGGFace model with ResNet50 and 64 fully connected layers

- The *VGGFace model with the ResNet50* architecture is loaded with its top layers excluded and takes input images of size *224x224 with 3 channels*.
- A *global average pooling layer* is added to reduce the spatial dimensions of the output.
- Separate *fully connected layers* are added for *BMI and gender* prediction, with *64 units and ReLU activation* for each layer.
- The fine-tuned model is created by defining the input and output layers.
- The *weights of the pre-trained layers* are *frozen* to prevent them from being updated during training.
- The model is compiled with the *Adam optimizer* and appropriate loss functions and metrics for each output.
- *Early stopping* is applied as a callback to stop training if the validation loss does not improve for at least 2 epochs.
- The model is trained using the *train_generator* and validated using the *valid_generator*, with specified number of steps per epoch and validation steps for *10 epochs*.

```
# Load the pre-trained VGGFace model
base_model = VGGFace(model='resnet50', include_top=False, input_shape=(224, 224, 3))

# Add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)

# Add fully connected layers for BMI prediction
bmi_fc = Dense(64, activation='relu')(x)
bmi_fc = Dense(1, name='bmi')(bmi_fc)

# Add fully connected layers for gender prediction
gender_fc = Dense(64, activation='relu')(x)
gender_fc = Dense(1, activation='sigmoid', name='gender')(gender_fc)

# Create the fine-tuned model
model = Model(inputs=base_model.input, outputs=[bmi_fc, gender_fc])

# Freeze the weights of the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

# Compile the model with appropriate loss functions and metrics
model.compile(optimizer=Adam(),
              loss={'bmi': 'mean_squared_error', 'gender': 'binary_crossentropy'},
              metrics={'bmi': 'mse', 'gender': 'accuracy'})

from keras.callbacks import EarlyStopping

# Train the model
model.fit(train_generator,
          steps_per_epoch=len(train_generator),
          epochs=10,
          validation_data=valid_generator,
          validation_steps=len(valid_generator),
          callbacks=[EarlyStopping(patience=2)])

model.save(path+'saved_model/fine_tuned_vggface_model_64.h5')
```

Model 2 : VGGFace model with ResNet50 and 128 fully connected layers (with increased epochs and EarlyStopping patience)

- The *VGGFace model* with the *ResNet50* architecture is loaded, excluding the top layers.
- A *global average pooling layer* is added to the model. This layer aggregates the spatial information from the previous convolutional layers into a single feature vector.
- Two sets of fully connected layers are appended to the model. The first set aims to predict the BMI (Body Mass Index) and consists of a dense layer with *128 units* and a *ReLU* activation function, followed by a single unit dense layer named 'bmi'.
- The second set of fully connected layers predicts the gender. It also has a dense layer with *128 units* and a *ReLU* activation function, followed by a single unit dense layer with a *sigmoid activation function* for binary classification. This layer is named 'gender'.
- The fine-tuned model is created by specifying the inputs and outputs of the modified VGGFace model.

- To prevent the pre-trained layers from being updated during training, their weights are *frozen* by setting them as non-trainable.

- The model is compiled using the *Adam* optimizer. The loss function for BMI prediction is *mean squared error*, while *binary cross-entropy* is used for gender prediction. The metrics for evaluation are *mean squared error (mse) for BMI and accuracy for gender.*
- The model is trained using data generators that generate batches of training and validation images along with their corresponding BMI and gender labels. The specified number of *epochs is 20*, and early stopping is applied using the *EarlyStopping* callback with *patience of 5 epochs* to prevent overfitting.

```
# Load the pre-trained VGGFace model
base_model = VGGFace(model='resnet50', include_top=False, input_shape=(224, 224, 3))

# Add a global average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)

# Add fully connected layers for BMI prediction
bmi_fc = Dense(128, activation='relu')(x)
bmi_fc = Dense(1, name='bmi')(bmi_fc)

# Add fully connected layers for gender prediction
gender_fc = Dense(128, activation='relu')(x)
gender_fc = Dense(1, activation='sigmoid', name='gender')(gender_fc)

# Create the fine-tuned model
model = Model(inputs=base_model.input, outputs=[bmi_fc, gender_fc])

# Freeze the weights of the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

# Compile the model with appropriate loss functions and metrics
model.compile(optimizer=Adam(),
              loss={'bmi': 'mean_squared_error', 'gender': 'binary_crossentropy'},
              metrics={'bmi': 'mse', 'gender': 'accuracy'})

# Train the model
model.fit(train_generator,
          steps_per_epoch=len(train_generator),
          epochs=20,
          validation_data=valid_generator,
          validation_steps=len(valid_generator),
          callbacks=[EarlyStopping(patience=5)])

# Save the fine-tuned model
model.save(path+'saved_model/fine_tuned_vggface_model_128.h5')
```

Model 3 : VGGFace model with ResNet50 and 128 fully connected layers (unfreezing last 5 layers)

Model 2 is further fine-tuned by unfreezing the last 5 layers of our pretrained VGGFace model and training these layers, while keeping the remaining layers frozen. It enables the model to adapt to the specific task of BMI and gender prediction by leveraging the pre-trained features learned from a large dataset.

```
# Load the pre-trained VGGFace model
base_model = VGGFace(model='resnet50', include_top=False, input_shape=(224, 224, 3))

# Unfreeze and train the last few layers of the base model
for layer in base_model.layers[-5:]:
    layer.trainable = True

# Add a global average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)

# Add fully connected layers for BMI prediction
bmi_fc = Dense(128, activation='relu')(x)
bmi_fc = Dense(1, name='bmi')(bmi_fc)

# Add fully connected layers for gender prediction
gender_fc = Dense(128, activation='relu')(x)
gender_fc = Dense(1, activation='sigmoid', name='gender')(gender_fc)

# Create the fine-tuned model
model = Model(inputs=base_model.input, outputs=[bmi_fc, gender_fc])

# Compile the model with appropriate loss functions and metrics
model.compile(optimizer=Adam(),
              loss={'bmi': 'mean_squared_error', 'gender': 'binary_crossentropy'},
              metrics={'bmi': 'mse', 'gender': 'accuracy'})

# Train the model
model.fit(train_generator,
          steps_per_epoch=len(train_generator),
          epochs=20,
          validation_data=valid_generator,
          validation_steps=len(valid_generator),
          callbacks=[EarlyStopping(patience=5)])

# Save the fine-tuned model
model.save(path+'saved_model/fine_tuned_vggface_model_unfreeze5.h5')
```

Model 4 : VGGFace model with VGG16 architecture and 64 fully connected layers (frozen pre-trained layers and training for 10 epochs)

Fine-tuning a VGGFace model with VGG16 architecture by using 64 fully connected layers while freezing weights of pre-trained layers and implementing EarlyStopping callback with a patience of 5 epochs with a total training of 10 epochs.

```
# Load the pre-trained VGGFace model
base_model = VGGFace(model='vgg16', include_top=False, input_shape=(224, 224, 3))

# Add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)

# Add fully connected layers for BMI prediction
bmi_fc = Dense(64, activation='relu')(x)
bmi_fc = Dense(1, name='bmi')(bmi_fc)

# Add fully connected layers for gender prediction
gender_fc = Dense(64, activation='relu')(x)
gender_fc = Dense(1, activation='sigmoid', name='gender')(gender_fc)

# Create the fine-tuned model
model = Model(inputs=base_model.input, outputs=[bmi_fc, gender_fc])

# Freeze the weights of the pre-trained layers
for layer in base_model.layers:
    layer.trainable = False

# Compile the model with appropriate loss functions and metrics
model.compile(optimizer=Adam(),
              loss={'bmi': 'mean_squared_error', 'gender': 'binary_crossentropy'},
              metrics={'bmi': 'mse', 'gender': 'accuracy'})

from keras.callbacks import EarlyStopping

# Train the model
model.fit(train_generator,
          steps_per_epoch=len(train_generator),
          epochs=10,
          validation_data=valid_generator,
          validation_steps=len(valid_generator),
          callbacks=[EarlyStopping(patience=5)])

model.save(path+'saved_model/fine_tuned_vggface_model_vgg16.h5')
```

Model Evaluation and Comparison

During training and evaluation, the model's performance is assessed by computing the defined *loss functions* and *evaluation metrics*. The model has two outputs: BMI (body mass index) and gender predictions. Therefore, the loss function is defined separately for each output. The model aims to ***minimize the loss functions (MSE and binary cross-entropy)*** to improve its predictions and ***optimise the evaluation metrics (RMSE for BMI and accuracy for gender)*** to achieve higher accuracy and predictive performance.

Loss Functions

For the BMI prediction, the mean squared error (MSE) loss function is used. MSE measures the average squared difference between the predicted and true BMI values. Lower values of MSE indicate better accuracy in predicting BMI.

Binary cross-entropy is a common loss function for binary classification tasks, such as predicting gender. It measures the dissimilarity between the predicted probabilities and the true binary labels. Smaller values of binary cross-entropy indicate better accuracy in predicting gender.

Evaluation Metrics

The model uses RMSE as the metric for evaluating the BMI prediction, which calculates the root mean squared difference between the predicted and true BMI values. The accuracy metric is used to evaluate the gender prediction, measuring the percentage of correctly classified gender labels.

Model Comparison

Pre-trained Model	Architecture	#Neurons in Dense layer	#Epochs	#Frozen pre-trained layers	Metrics	
					BMI - RMSE	Gender - Accuracy
VGGFace	resnet50	64	4/10	All	9.36	56.80%
VGGFace	resnet50	128	8/20	All	9.64	56.80%
VGGFace	resnet50	128	14/20	Unfreeze last 5 layers	8.72	56.80%
VGGFace	vgg16	64	10/10	All	10.81	95.80%

Even though **VGGFace** - **vgg16** has the highest RMSE, we select the model as the best as it performs well for Gender as well and the difference is RMSE values is close between models.

Results

We can now use our trained and fine-tuned model to predict BMI and gender from facial images in real time. The output of our project is a deployed Streamlit application that allows users to capture images from a webcam or upload images and obtain BMI and gender predictions based on facial features using a fine-tuned VGGFace model with vgg16 architecture pre-processed using MTCNN.

Python scripts are generated for the Streamlit app and uploaded to git. The app is then deployed using Streamlit cloud here.

Some predictions of facial images generated by our model (**fine_tuned_vggface_model_vgg16.h5**) via the Streamlit's **Upload image** option are as follows :

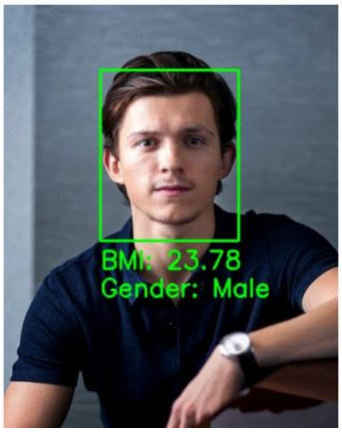
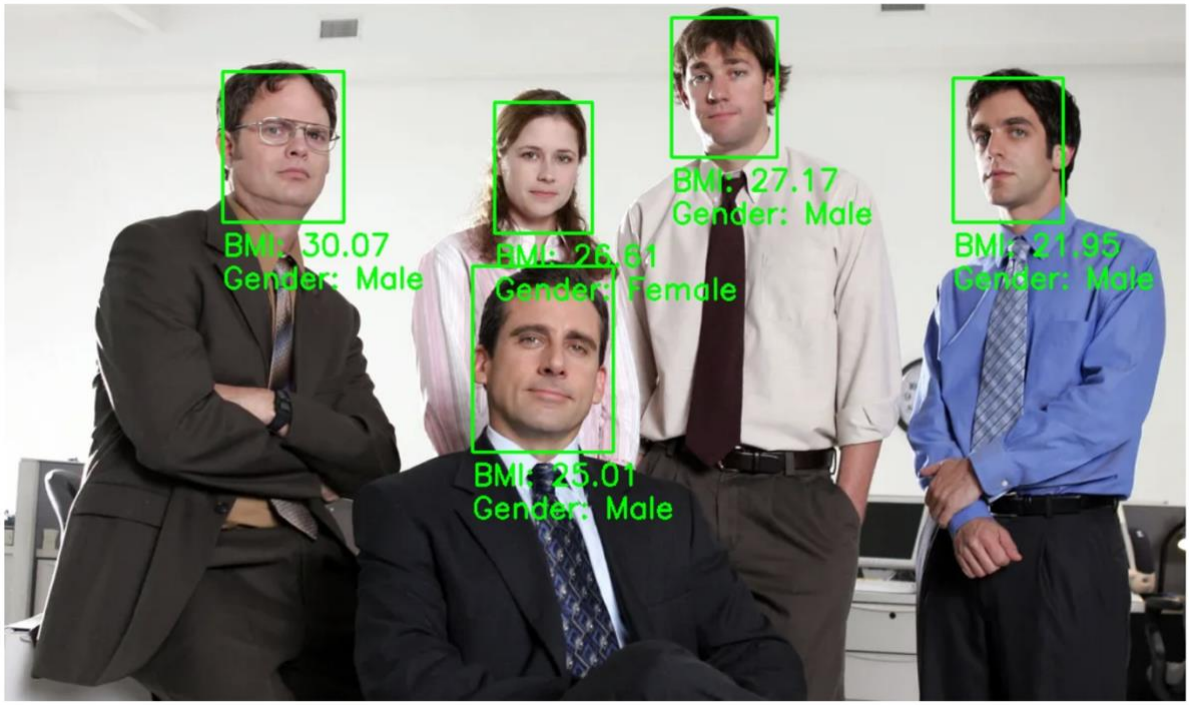


Illustration of ability to predict for multiple faces



More images can be verified at the link. Predictions made by real-time webcam images to be demoed in class.

Conclusion

This project presented a face-to-BMI model that estimates an individual's body mass index (BMI) and identifies gender based on facial images. The model leverages deep learning techniques, specifically a pre-trained VGGFace model with vgg16 architecture, to extract relevant facial features and predict both BMI and gender. The model was trained and fine-tuned using a dataset of labeled facial images with corresponding BMI and gender information.

However, it is important to consider privacy, data security, and ethical considerations when deploying such models. Further fine-tuning and development are necessary to enhance the model's accuracy, generalize it to diverse populations, and address potential biases by adding more input features.

The presented face-to-BMI model demonstrates the potential of leveraging facial images for BMI estimation and opens possibilities for innovative applications in healthcare, security, and personal well-being. Further advancements in this field hold the promise of enhancing self-care, remote diagnostics, and biometric identification systems while promoting positive body image and empowering individuals in their pursuit of a healthy lifestyle.

References

https://medium.com/@leosimmons/estimating-body-mass-index-from-face-images-using-keras-and-transfer-learning-de25e1bc0212

https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/ipr2.12222

https://6chaoran.wordpress.com/2020/01/17/detect-faces-and-predict-age-gender-bmi-using-keras/

https://www.sciencedirect.com/science/article/abs/pii/S0262885613000462

Submitted documents

Script for Streamlit app

1. face-2-bmi-streamlit.zip

-README.md
-fine_tuned_vggface_model_vgg16.h5
-requirements.txt
-streamlit_app.py

Code and Documentation

2. face-2-bmi-final_doc.pdf – writeup on implementation of project

3. face-2-bmi-modelling.zip

-01_pre_processing.ipynb
-02_modelling.ipynb
-Data (original data)
-Train (Train images, MTCNN aligned Train images, Train data .csv)
-Test (Test images, MTCNN aligned Test images, Test data .csv)
-saved_model (all implemented models)

AWS Academy Machine Learning Foundations [34961] Certificate

4. AWS_Course_Completion_Certificate_SwathiGanesan.pdf