Low Level Design (HLD)

FACE MASK DETECTION

Partha Pratim Kalita

July, 2024

Content

Introduction

1. What is Low Level Design Document?

The goal of the Low-level design document (LLDD) is to give the internal logic design of the actual program code for the Heart Disease Diagnostic Analysis dashboard. LLDD describes the class diagrams with the methods and relations between classes and programs specs. It describes the modules so that the programmer can directly code the program from the document.

2. Scope

Low-level design (LLD) is a component-level design process that follows a step by step refinement process. The process can be used for designing data structures, required software architecture, source code and ultimately, performance algorithms. Overall, the data organization may be defined during requirement analysis and then refined during data design work.

Introduction

1.3 Project Introduction

The face mask detection project is a computer vision application that employs deep learning techniques, like Convolutional Neural Networks (CNNs), to automatically detect if a person is wearing a face mask or not. By analyzing images or live video streams, the system can quickly identify individuals who are not following mask-wearing guidelines. The primary objectives include high accuracy, real-time processing, and scalability. This technology has diverse applications in public safety, healthcare, access control, and contact tracing, contributing to disease prevention and safer environments.

Problem Statement

COVID-19 pandemic has rapidly affected our day-to-day life disrupting the world trade and movements. Wearing a protective face mask has become a new normal. In the near future, many public service providers will ask the customers to wear masks correctly to avail of their services. Therefore, face mask detection has become a crucial task to help global society. COVID-19 mask detector could potentially be used to help ensure your safety and the safety of others.

Approach: A simplified approach to achieve this purpose using some basic Machine Learning packages like TensorFlow, Keras, OpenCV and Scikit-Learn. This method detects the face from the image correctly and then identifies if it has a mask on it or not.

Results: Build a robust solution that detect and identify the person is wearing mask or not. With a varying distance and color combination, works for any person.

Dataset Information

Dataset Information:

1. **Number of Images:** 1000 (500 images with masks, 500 images without masks)
2. **Image Variability:** The dataset should cover various scenarios, including different lighting conditions, angles, backgrounds.
3. **Resolution:** The images have consistent and reasonable resolutions to maintain model performance and efficiency.
4. **Data Split:** Divided the dataset into three subsets: training set (80% of the data) and test set (20% of the data). The training set is used to train the model and the test set is used to evaluate the final model's performance.

Model Information

baseModel -> MobileNetV2(weights="imagenet", include_top=False, input_tensor=layers.Input(shape=(224, 224, 3)))

- MobileNetV2 model with pre-trained weights from the "imagenet" dataset.
- **include_top=False** means that the final fully connected layers (the "head") of the MobileNetV2 model, which are responsible for ImageNet classification, are excluded.
- **input_tensor=layers.Input(shape=(224, 224, 3))** specifies the input shape of the model, which is (224, 224, 3) for RGB images with a resolution of 224x224.

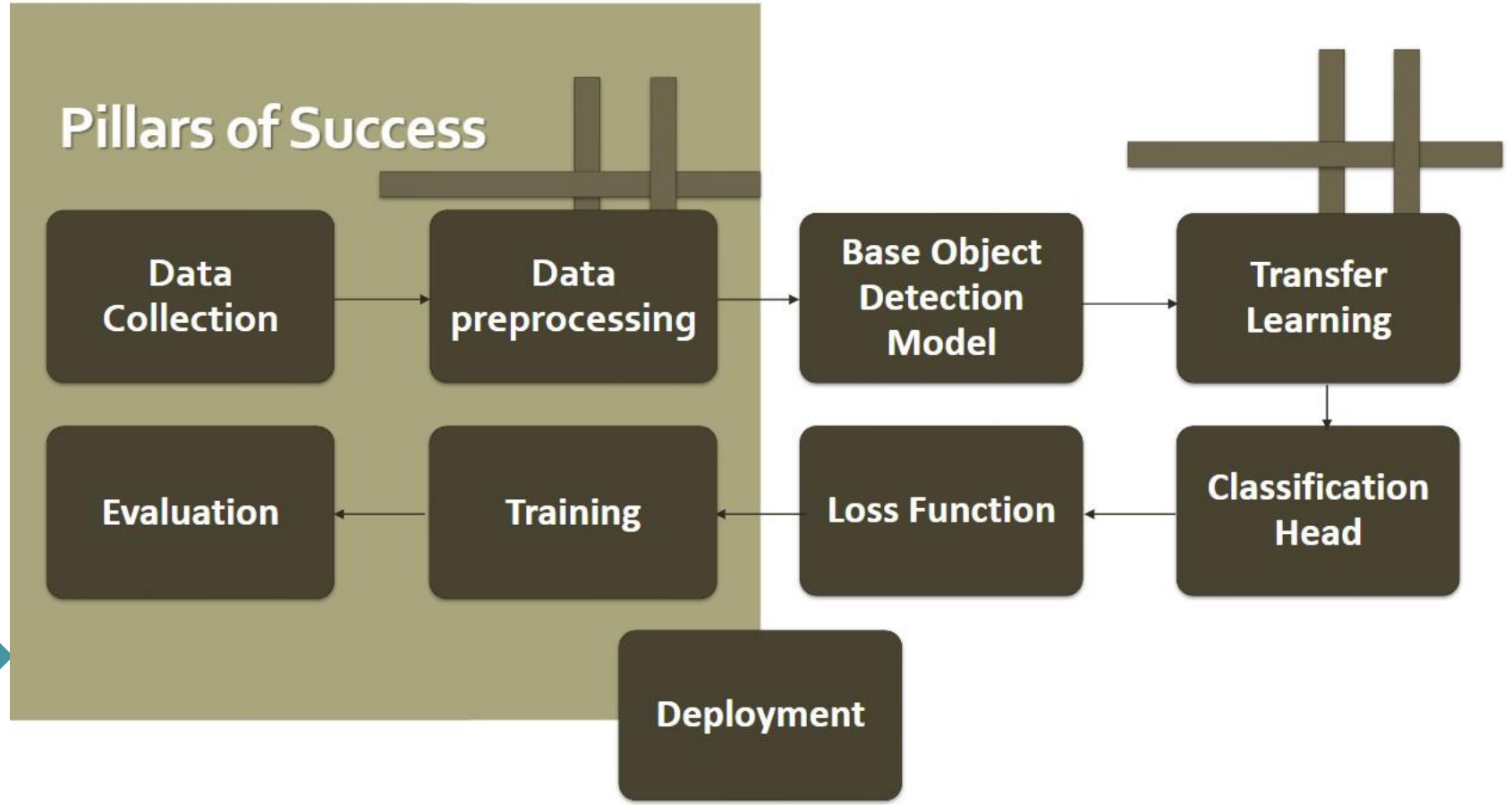
headModel -> baseModel.output

- This line assigns the output of the base model (MobileNetV2) to the variable **headModel**. This output will be used as the input to the custom head model.

Model Information

1. **headModel -> layers.AveragePooling2D(pool_size=(7, 7))(headModel)**
   - This line adds an AveragePooling2D layer to the head model. The AveragePooling2D layer reduces the spatial dimensions of the input by taking the average within each 7x7 window. This operation helps reduce computational complexity.
2. **headModel -> layers.Flatten(name="flatten")(headModel)**
   - This line adds a Flatten layer to the head model, which converts the 2D feature maps into a 1D vector. This step is necessary to connect the head model to the fully connected layers.
3. **headModel -> layers.Dense(128, activation="relu")(headModel)**
   - This line adds a Dense (fully connected) layer with 128 units and ReLU activation function. This layer serves as a hidden layer to learn more complex patterns from the flattened features.
4. **headModel -> layers.Dropout(0.5)(headModel)**
   - This line adds a Dropout layer to the head model. Dropout is a regularization technique that randomly sets a fraction (0.5 in this case) of the input units to 0 during training. It helps prevent overfitting.
5. **headModel -> layers.Dense(2, activation="softmax")(headModel)**
   - This line adds the final Dense layer with 2 units and a softmax activation function. The softmax function converts the output values into probabilities for the two classes: "with_mask" and "without_mask." The class with the highest probability will be the predicted class.

Architecture



Architecture

Architecture description

Dataset Collection : https://www.kaggle.com/datasets/ahmedabdelraouf/face-datasets

Data Preprocessing : Images -> NumPy array -> scaling, or normalization(using `from keras.applications.mobilenet_v2 import preprocess_input`)

Base Object Detection Mode : Single Shot Multibox Detector (SSD) method with the ResNet-10

Transfer Learning : Taking the pre-trained object detection models (Face Models) and fine-tune it on your custom face mask dataset (Mask Model). During this process, the model learns to identify faces and distinguish between masked and unmasked faces.

Architecture

Architecture description

Classification Head: The last layer(s) of the object detection model to include a classification head that can predict two classes: "with mask" and "without mask.“

```
model.compile(loss="binary_crossentropy", optimizer=Adam(lr=0.0001, decay=0.0001 / 20),
metrics=["accuracy"])
```

Loss Function: binary cross-entropy is used as loss function.

Training and Evaluation:

```
history = model.fit(augmentation.flow(X_train, y_train, batch_size=32),
steps_per_epoch=len(X_train) // 32, validation_data=(X_test, y_test),
validation_steps=len(X_test) // 32, epochs=20)
```

Deployment : Deployment on Microsoft Azure.

Thank you

Partha Pratim Kalita

youranalystpartha@gmail.com