FACE RECOGNITION AND DETECTION USING VGG16

By

Dhayanithi T(STB03-T0018)

Submitted to Scifor Technologies

Meta
Scifor Technologies

Script. Sculpt. Socialize

UNDER GUIDIANCE OF

Urooj Khan

TABLE OF CONTENTS

ABSTRACT

Large organizations have been successfully using facial recognition in security systems which were using Convolutional Neural Networks (CNNs). However, small organizations struggle to implement the same security measures like CNNs using smaller training databases and fewer computer resources. By applying transfer learning for facial recognition, which requires less retraining, The research paper provides a solution to resolve this problem. This can be demonstrated by adding a layer of a trained element with a minimal number of neurons in a network can be performed in tiny datasets, allowing for functional and legitimate authentication.Transfer learning is a method of reusing a pre-trained model knowledge for another task. It can be used for classification, regression and clustering problems. It is a long process to collect related training data and rebuild the models. In such cases, Transferring of Knowledge or transfer learning from disparate domains would be desirable.

INTRODUCTION

Face is one of the most widely accepted biometrics, and it has become the most common way to detect human use in their visual interactions. Problem with verification systems based on fingerprints, voice, iris and recent genetic structure (DNA fingerprints) data acquisition problem. For example, with fingerprints the person concerned should keep his or her finger in the correct position and direction and when the speaker is known the microphone should be kept in the correct position along with the speaker and distance.

However, the process of getting facial images does not interfere with the face being used as a biometric encryption (where the user is unaware that they are being deceived) structure. The face is a normal human trait. Face recognition is important not only because of the power of your many potential applications in the field of research but also because of the power of your solution that can help solve other divisive issues such as object recognition.

In verifying a face or proving authenticity there is one similarity and that compares the question of the face image with the face image of the image sought by its ownership. In face recognition or eye contact there is one of the many comparisons that compares the face and question to all the image templates on the website to determine the identity of the face of the question. Another type of facial recognition involves the checklist check, in which the question face is compared to a list of suspects (one-to-one matching).

Face recognition and detection have garnered significant attention due to their practical applications in security, surveillance, and biometric authentication systems. This report outlines a comprehensive approach to develop a face recognition and detection system utilizing the VGG16 architecture, a widely used deep learning model for image classification tasks.

TECHNOLOGY USED

The technology stack used in this projects can be summarized as follows:

1. Programming Language: Python
2. Deep Learning Framework: TensorFlow
3. Libraries:
    - ❖ tensorflow: Deep learning library for building neural networks.
    - ❖ matplotlib: Data visualization library for plotting graphs and images.
    - ❖ labelme: A graphical image annotation tool used for labeling objects in images, commonly used in computer vision tasks.
    - ❖ opencv-python: OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. The Python version provides an easy-to-use interface for computer vision tasks.
    - ❖ albumentations: An image augmentation library for machine learning tasks, providing a wide range of augmentations to enhance training data and improve model generalization.
4. Data Handling and Manipulation:
    - ❖ Loading and preprocessing the images.
    - ❖ Manipulating arrays and images using NumPy and TensorFlow libraries.
5. Model Building:
    - ❖ Utilization of VGG16, a high-level neural networks API running on top of TensorFlow, for building and training neural network models.
    - ❖ Sequential model architecture with densely connected layers.
    - ❖ Activation functions such as ReLU and Softmax.
    - ❖ Compilation of the model with specified loss function, optimizer, and evaluation metrics.
6. Training and Evaluation:

- Training the model using the `fit` method.
- Evaluation of the model using accuracy metrics.
- Evaluating by using the live webcam for face recognition.

These technologies collectively enable the creation of a complete pipeline for training, evaluating of face recognition using VGG16.

METHODOLOGY

1.1 Install Dependencies and Setup:

The initial step involves setting up the development environment by installing necessary dependencies and configuring the project environment for seamless execution.

1.2 Collect Images Using OpenCV:

Data acquisition is crucial for training a robust face recognition model. Images are collected using OpenCV, a popular library for computer vision tasks.

1.3 Annotate Images with LabelMe:

To enhance the dataset quality and facilitate supervised learning, images are annotated using LabelMe, enabling precise labeling of facial regions.

2. Data Preparation:

Before feeding the data into the model, it undergoes preprocessing steps to ensure compatibility and optimal training performance.

2.1 Review Dataset and Build Image Loading Function:

An overview of the dataset is conducted, followed by the development of functions to load images into the training pipeline efficiently.

2.2 Limit GPU Memory Growth:

To avoid memory overflow during model training, GPU memory growth is restricted to ensure stable performance.

2.3 Load Image into TF Data Pipeline:

Images are loaded into TensorFlow data pipelines, facilitating seamless integration with the deep learning model.

2.4 View Raw Images with Matplotlib:

A visualization step is performed to inspect raw images and verify data integrity.

3. Data Partitioning:

The dataset is partitioned into training, testing, and validation sets to enable model evaluation and prevent overfitting.

3.1 Manually Split Data into Train, Test, and Validation Sets:

Data partitioning is performed manually, ensuring representative distribution across all sets.

3.2 Move the Matching Labels:

Corresponding labels are moved along with the images to maintain data integrity across partitions.

4. Image Augmentation:

To enhance model generalization and robustness, image augmentation techniques are applied to diversify the dataset.

4.1 Setup Albumentations Transform Pipeline:

Albumentations library is utilized to define augmentation pipelines, incorporating various transformations.

4.2 Load a Test Image and Annotation with OpenCV and JSON:

Test images and annotations are loaded using OpenCV and JSON formats, facilitating augmentation.

4.3 Extract Coordinates and Rescale to Match Image Resolution:

Coordinates are extracted from annotations and rescaled to match image resolutions post-augmentation.

4.4 Apply Augmentations and View Results:

Augmentations are applied to images, and the results are visualized to ensure augmentation effectiveness.

5. Augmentation Pipeline Execution:

The augmentation pipeline is executed to generate augmented images and labels for training.

5.1 Run Augmentation Pipeline:

The defined augmentation pipeline is executed to generate augmented data samples.

5.2 Load Augmented Images to TensorFlow Dataset:

Augmented images are loaded into TensorFlow datasets for training.

6. Label Preparation:

Label loading functions are developed to facilitate the integration of labels with image data.

6.1 Build Label Loading Function:

Functions are developed to load labels into TensorFlow datasets for model training.

6.2 Load Labels to TensorFlow Dataset:

Labels are loaded into TensorFlow datasets, ensuring alignment with corresponding images.

7. Data Integration:

Images and labels are combined to create final datasets for model training.

7.1 Check Partition Lengths:

The lengths of training, testing, and validation sets are verified to ensure consistency.

7.2 Create Final Datasets (Images/Labels):

Final datasets comprising images and labels are created for model training and evaluation.

7.3 View Images and Annotations:

Final datasets are visualized to ensure proper alignment between images and corresponding annotations.

8. Model Development:

The VGG16 architecture is employed to build the deep learning model for face recognition and detection.

8.1 Import Layers and Base Network:

Necessary layers and the base VGG16 network are imported for model construction.

8.2 Download VGG16:

The pre-trained VGG16 model weights are downloaded to initialize the base network.

8.3 Build Instance of Network:

An instance of the VGG16 model is constructed, with modifications for the specific task of face recognition and detection.

8.4 Test out Neural Network:

The constructed neural network is tested to ensure proper initialization and functionality.

9. Model Optimization:

Loss functions and optimizers are defined to optimize the model during training.

9.1 Define Optimizer and Learning Rate:

An optimizer and learning rate are defined to train the model efficiently.

9.2 Create Localization Loss and Classification Loss:

Localization and classification losses are defined to optimize model performance for face detection and recognition tasks.

9.3 Test out Loss Metrics:

Loss metrics are evaluated to ensure proper computation and alignment with model objectives.

10. Model Training:

The constructed model is trained using the prepared datasets and optimization techniques.

10.1 Create Custom Model Class:

A custom model class is developed to encapsulate the model architecture and training process.

10.2 Train:

The model is trained on the prepared datasets using the defined optimization strategies.

10.3 Plot Performance:

Training performance metrics such as loss and accuracy are plotted to monitor model training progress.

11. Model Evaluation:

The trained model is evaluated on test data to assess its performance in face recognition and detection.

11.1 Make Predictions on Test Set:

Predictions are made on the test set to evaluate model accuracy and performance.

11.2 Save the Model:

The trained model is saved for future use and deployment in real-world applications.

11.3 Real-Time Detection:

The trained model is capable of real-time face detection, demonstrating its practical utility in various applications.

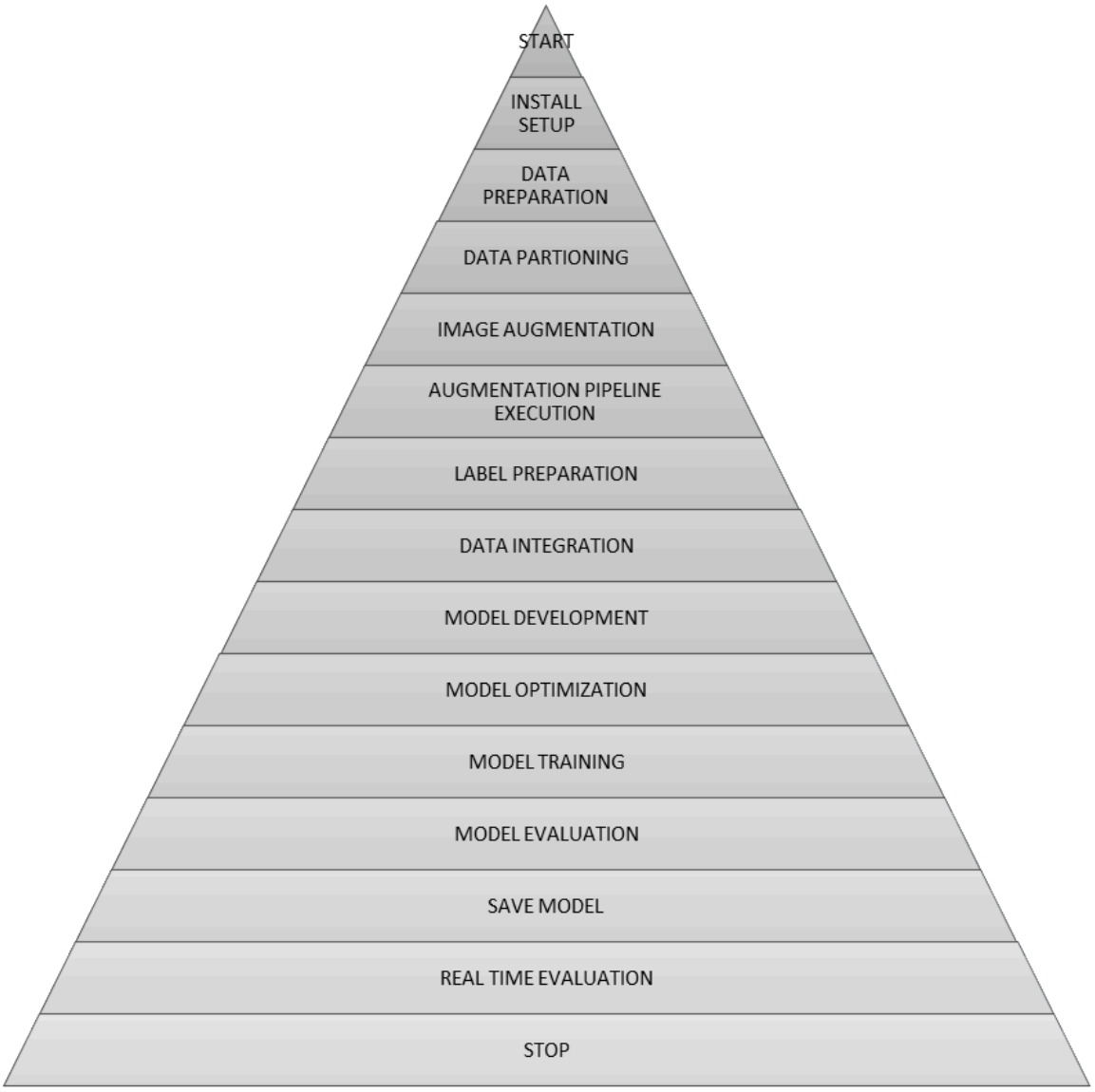


FIG-1: Workflow of the Model

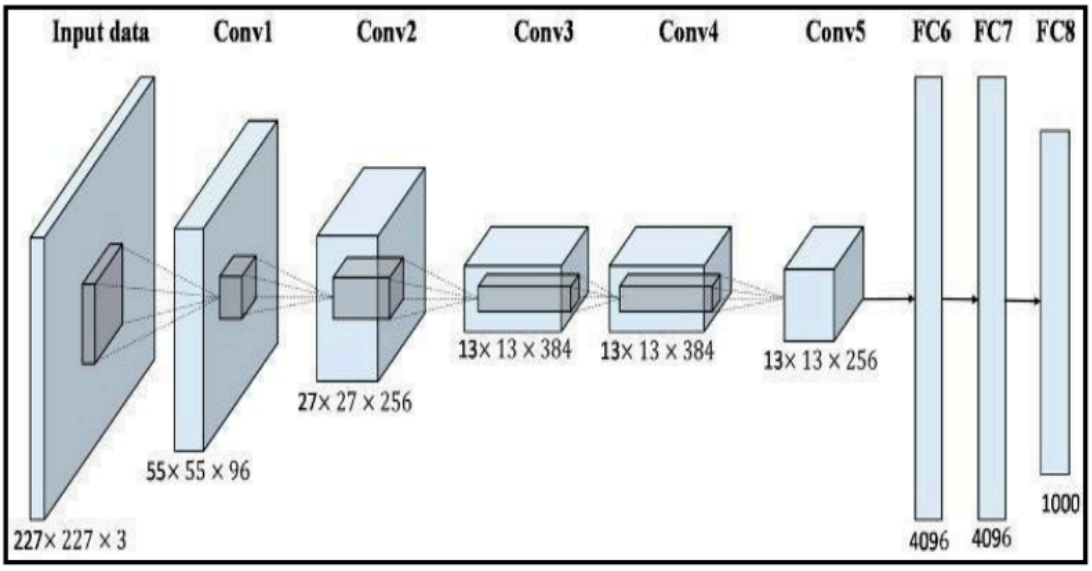


FIG-2: VGG - 16 Architecture

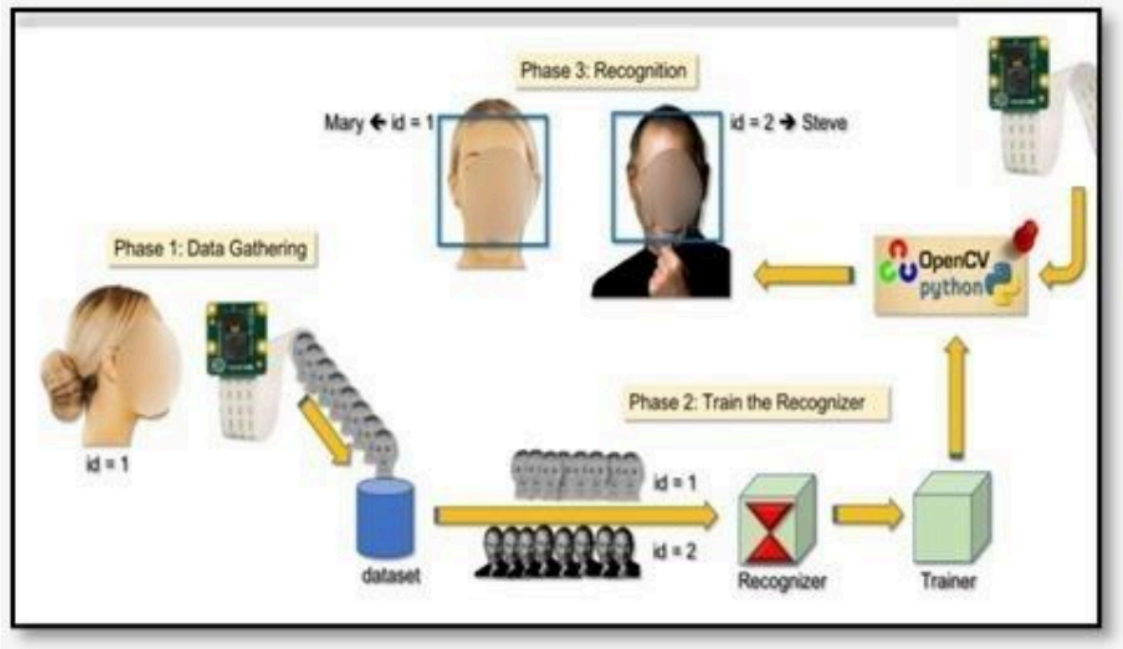


FIG-3: Face Recognition Process Flow

CODE SNIPPET

jupyter Face Recognition and Detection Last Checkpoint: 3 days ago

File Edit View Run Kernel Settings Help — Trusted

Markdown — JupyterLab — Python 3 (ipykernel)

1.2 Collect Images Using OpenCV

[2]:
```
import os
import time
import uuid
import cv2
```

[3]:
```
IMAGES_PATH = os.path.join('data','images')
number_images = 30
```

[7]:
```
cap = cv2.VideoCapture(0)
for imgnum in range(number_images):
    print('Collecting image {}'.format(imgnum))
    ret, frame = cap.read()
    imgname = os.path.join(IMAGES_PATH,f'{str(uuid.uuid1())}.jpg')
    cv2.imwrite(imgname, frame)
    cv2.imshow('frame', frame)
    time.sleep(0.5)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

```
Collecting image 0
Collecting image 1
Collecting image 2
Collecting image 3
Collecting image 4
Collecting image 5
```

jupyter Face Recognition and Detection Last Checkpoint: 3 days ago

File Edit View Run Kernel Settings Help — Trusted

Markdown — JupyterLab — Python 3 (ipykernel)

2.3 Load Image into TF Data Pipeline ¶

[12]:
```
images = tf.data.Dataset.list_files('data\\images\\*.jpg')
```

[13]:
```
images.as_numpy_iterator().next()
```

[13]:
```
b'data\\images\\b96af8c3-fe1a-11ee-a041-f889d2840d92.jpg'
```

[14]:
```
def load_image(x):
    byte_img = tf.io.read_file(x)
    img = tf.io.decode_jpeg(byte_img)
    return img
```

[15]:
```
images = images.map(load_image)
```

[16]:
```
images.as_numpy_iterator().next()
```

[16]:
```
array([[[ 75,  85,  74],
        [ 84,  94,  85],
        [ 93, 104,  96],
        ...,
        [106, 109, 118],
        [106, 106, 114],
        [105, 104, 112]],

       [[ 81,  91,  83],
        [ 88,  98,  90],
        [ 94, 105,  99],
        ...,
```

jupyter Face Recognition and Detection Last Checkpoint: 3 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

3. Partition Unaugmented Data ¶

3.1 MANUALLY SPLT DATA INTO TRAIN TEST AND VAL

[21]:
```
90*.7 # 63 to train
```

[21]:
```
62.99999999999999
```

[22]:
```
90*.15 # 14 and 13 to test and val
```

[22]:
```
13.5
```

3.2 Move the Matching Labels

[23]:
```
for folder in ['train','test','val']:
    for file in os.listdir(os.path.join('data', folder, 'images')):

        filename = file.split('.')[0]+'.json'
        existing_filepath = os.path.join('data','labels', filename)
        if os.path.exists(existing_filepath):
            new_filepath = os.path.join('data',folder,'labels',filename)
            os.replace(existing_filepath, new_filepath)
```

4. Apply Image Augmentation on Images and Labels using Albumentations

4.1 Setup Albumentations Transform Pipeline

jupyter Face Recognition and Detection Last Checkpoint: 3 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

5. Build and Run Augmentation Pipeline

5.1 Run Augmentation Pipeline

[38]:
```
for partition in ['train','test','val']:
    for image in os.listdir(os.path.join('data', partition, 'images')):
        img = cv2.imread(os.path.join('data', partition, 'images', image))

        coords = [0,0,0.00001,0.00001]
        label_path = os.path.join('data', partition, 'labels', f'{image.split(".")[0]}.json')
        if os.path.exists(label_path):
            with open(label_path, 'r') as f:
                label = json.load(f)

            coords[0] = label['shapes'][0]['points'][0][0]
            coords[1] = label['shapes'][0]['points'][0][1]
            coords[2] = label['shapes'][0]['points'][1][0]
            coords[3] = label['shapes'][0]['points'][1][1]
            coords = list(np.divide(coords, [640,480,640,480]))

        try:
            for x in range(60):
                augmented = augmentor(image=img, bboxes=[coords], class_labels=['face'])
                cv2.imwrite(os.path.join('aug_data', partition, 'images', f'{image.split(".")[0]}.{x}.jpg'), augmented['image'])

                annotation = {}
                annotation['image'] = image

                if os.path.exists(label_path):
```

jupyter Face Recognition and Detection Last Checkpoint: 3 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

5.2 Load Augmented Images to Tensorflow Dataset

[39]:
```
train_images = tf.data.Dataset.list_files('aug_data\\train\\images\\*.jpg', shuffle=False)
train_images = train_images.map(load_image)
train_images = train_images.map(lambda x: tf.image.resize(x, (120,120)))
train_images = train_images.map(lambda x: x/255)
```

[40]:
```
test_images = tf.data.Dataset.list_files('aug_data\\test\\images\\*.jpg', shuffle=False)
test_images = test_images.map(load_image)
test_images = test_images.map(lambda x: tf.image.resize(x, (120,120)))
test_images = test_images.map(lambda x: x/255)
```

[41]:
```
val_images = tf.data.Dataset.list_files('aug_data\\val\\images\\*.jpg', shuffle=False)
val_images = val_images.map(load_image)
val_images = val_images.map(lambda x: tf.image.resize(x, (120,120)))
val_images = val_images.map(lambda x: x/255)
```

[42]:
```
train_images.as_numpy_iterator().next()
```

[42]:
```
array([[[0.22156863, 0.25245097, 0.29313725],
        [0.21544118, 0.23210785, 0.30661765],
        [0.24779412, 0.25563726, 0.34289217],
        ...,
        [0.15900736, 0.09950981, 0.09675245],
        [0.18186274, 0.12303922, 0.09852941],
        [0.14368872, 0.08278187, 0.08431373]],

       [[0.21856618, 0.2382353 , 0.29993874],
        [0.22971813, 0.24295343, 0.3091299 ],
        [0.25563726, 0.26838234, 0.33688724],
```

jupyter Face Recognition and Detection Last Checkpoint: 3 days ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

```
        [0.5464 , 0.0642 , 0.987  , 0.748  ],
        [0.2394 , 0.1165 , 0.743  , 0.7686 ]], dtype=float16))
```

7.3 View Images and Annotations

[53]:
```
data_samples = train.as_numpy_iterator()
```

[59]:
```
res = data_samples.next()
```

[60]:
```
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx in range(4):
    sample_image = res[0][idx]
    sample_coords = res[1][1][idx]

    # Convert to BGR color format and make a copy
    sample_image = cv2.cvtColor(sample_image, cv2.COLOR_RGB2BGR).copy()

    cv2.rectangle(sample_image,
                  tuple(np.multiply(sample_coords[:2], [120,120]).astype(int)),
                  tuple(np.multiply(sample_coords[2:], [120,120]).astype(int)),
                  (255,0,0), 2)

    ax[idx].imshow(sample_image)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

jupyter Face Recognition and Detection Last Checkpoint: 3 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

8. Build Deep Learning using the Functional API

8.1 Import Layers and Base Network

[61]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, Dense, GlobalMaxPooling2D
from tensorflow.keras.applications import VGG16
```

8.2 Download VGG16

[62]:
```
vgg = VGG16(include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 10s 0us/step
```

[63]:
```
vgg.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, None, 3)]   0

 block1_conv1 (Conv2D)       (None, None, None, 64)    1792

 block1_conv2 (Conv2D)       (None, None, None, 64)    36928

 block1_pool (MaxPooling2D)  (None, None, None, 64)    0

 block2_conv1 (Conv2D)       (None, None, None, 128)   73856
```

jupyter Face Recognition and Detection Last Checkpoint: 3 days ago

File Edit View Run Kernel Settings Help

Trusted

Code

JupyterLab Python 3 (ipykernel)

8.4 Test out Neural Network

[65]:
```
facetracker = build_model()
```

[66]:
```
facetracker.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_2 (InputLayer)        [(None, 120, 120, 3)]     0         []

 vgg16 (Functional)          (None, None, None, 512)   14714688  ['input_2[0][0]']

 global_max_pooling2d (GlobalMaxPooling2D)  (None, 512)  0  ['vgg16[0][0]']

 global_max_pooling2d_1 (GlobalMaxPooling2D)  (None, 512)  0  ['vgg16[0][0]']

 dense (Dense)               (None, 2048)              1050624   ['global_max_pooling2d[0][0]']

 dense_2 (Dense)             (None, 2048)              1050624   ['global_max_pooling2d_1[0][0]']

 dense_1 (Dense)             (None, 1)                 2049      ['dense[0][0]']

 dense_3 (Dense)             (None, 4)                 8196      ['dense_2[0][0]']

==================================================================================================
Total params: 16826181 (64.19 MB)
```

jupyter Face Recognition and Detection Last Checkpoint: 3 days ago

File Edit View Run Kernel Settings Help

Trusted

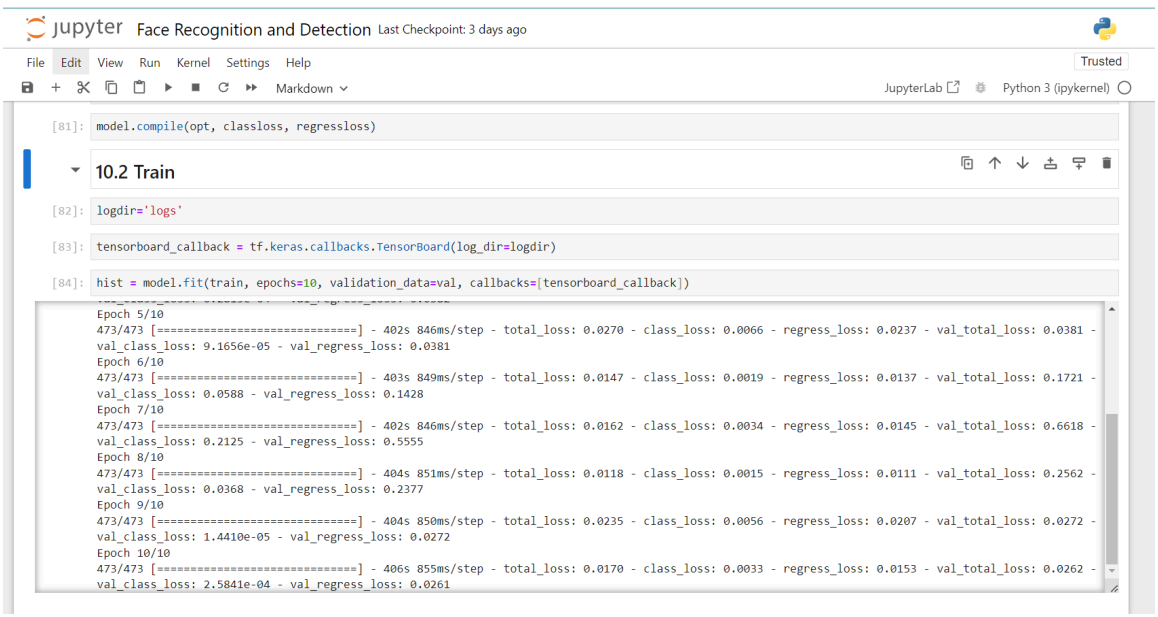
Code

JupyterLab Python 3 (ipykernel)

10. Train Neural Network

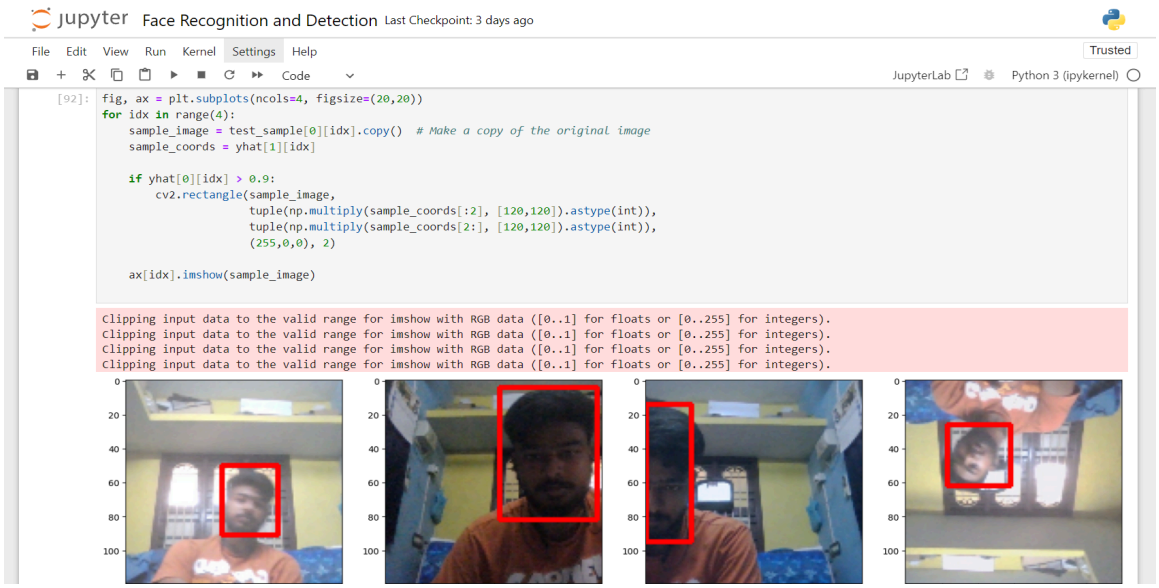
10.1 Create Custom Model Class ¶

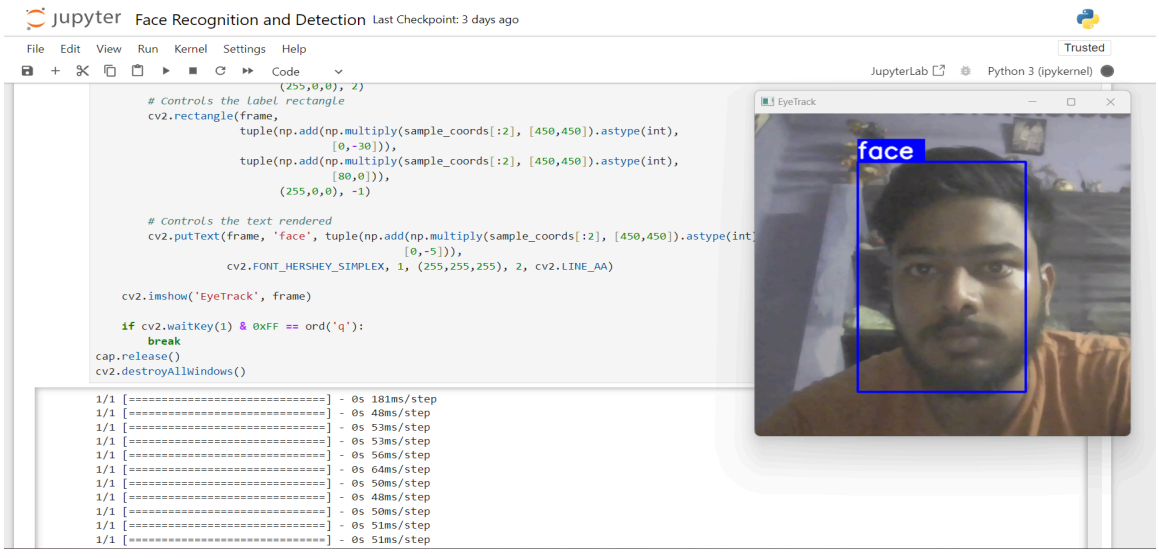
[79]:
```
class FaceTracker(Model):
    def __init__(self, eyetracker,  **kwargs):
        super().__init__(**kwargs)
        self.model = eyetracker

    def compile(self, opt, classloss, localizationloss, **kwargs):
        super().compile(**kwargs)
        self.closs = classloss
        self.lloss = localizationloss
        self.opt = opt

    def train_step(self, batch, **kwargs):

        X, y = batch

        with tf.GradientTape() as tape:
            classes, coords = self.model(X, training=True)

            batch_classloss = self.closs(y[0], classes)
            batch_localizationloss = self.lloss(tf.cast(y[1], tf.float32), coords)

            total_loss = batch_localizationloss+0.5*batch_classloss

            grad = tape.gradient(total_loss, self.model.trainable_variables)
```



RESULT AND DISCUSSION

The implementation of a face recognition and detection system using the VGG16 architecture demonstrates the feasibility and effectiveness of deep learning techniques in computer vision tasks. Through meticulous data preparation, model development, and training, accurate and efficient face recognition models can be developed for diverse applications.The total loss during training is 0.0170, comprised of a class loss of 0.0033 and a regression loss of 0.0153. During validation, the total loss slightly increases to 0.0262, with a negligible class loss of 2.5841e-04 and a regression loss of 0.0261.





CONCLUSION

The face recognition model trained using VGG16 architecture demonstrates promising performance, as indicated by the low total loss during training. The class loss, representing the accuracy of class predictions, is relatively low, indicating effective classification of facial features. The regression loss, reflecting the precision of bounding box predictions, is also acceptable. However, during validation, a slight increase in total loss is observed, primarily driven by a higher regression loss. Further investigation is warranted to understand the underlying factors contributing to this discrepancy and potential strategies for improvement. Overall, the model shows potential for accurate face recognition, albeit with room for optimization in regression tasks during validation.

REFERENCES

1. Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. ”siamese neural networks for one- shot image recognition ”. In ICML Deep Learning workshop, 2015.

2. Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. ”learning and transferring mid-level image representations using convolutional neural networks.”. Computer Vision and Pattern Recognition, 2014.

3. ”the database of faces,” att laboratories cambridge, (2002). [online] avail- able: http://www.cl.cam.ac.uk/research/dtg/attarchive/ facedatabase.html.

4. Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. ”CNN features off- the-shelf: an astounding baseline for recognition”. CoRR, abs/1403.6382, 2014.

5. ”description of the collection of facial images” dr libor spacek [online] available: http://cswww.essex.ac.uk/mv/allfaces/index.html

6. ”georgia tech face database” [online] available: http://www.anefian.com/research/facereco.htm.

7. G. B. Huang, M. Ramesh, and T. Berg. ”labeled faces in the wild: A database for studying face recognition in unconstrained environments.”.

8. O. M. Parkhi, A. Vedaldi, and A. Zisserman. ”deep face recognition”. A In British Machine Vision Conference, 2015.