Report On

Facial Feature Detection: A Deep Neural Network Approach

Submitted in partial fulfillment of the requirements of the Course project in
Semester VII of Fourth Year Artificial Intelligence and Data Science

by
Parth Puri (Roll No. 23)
Reena Vaidya (Roll No. 31)

Supervisor
Dr. Tatwadarshi Nagarhalli

University of Mumbai

Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science

(2023-24)

Vidyavardhini's College of Engineering & Technology
Department of Artificial Intelligence and Data Science

CERTIFICATE

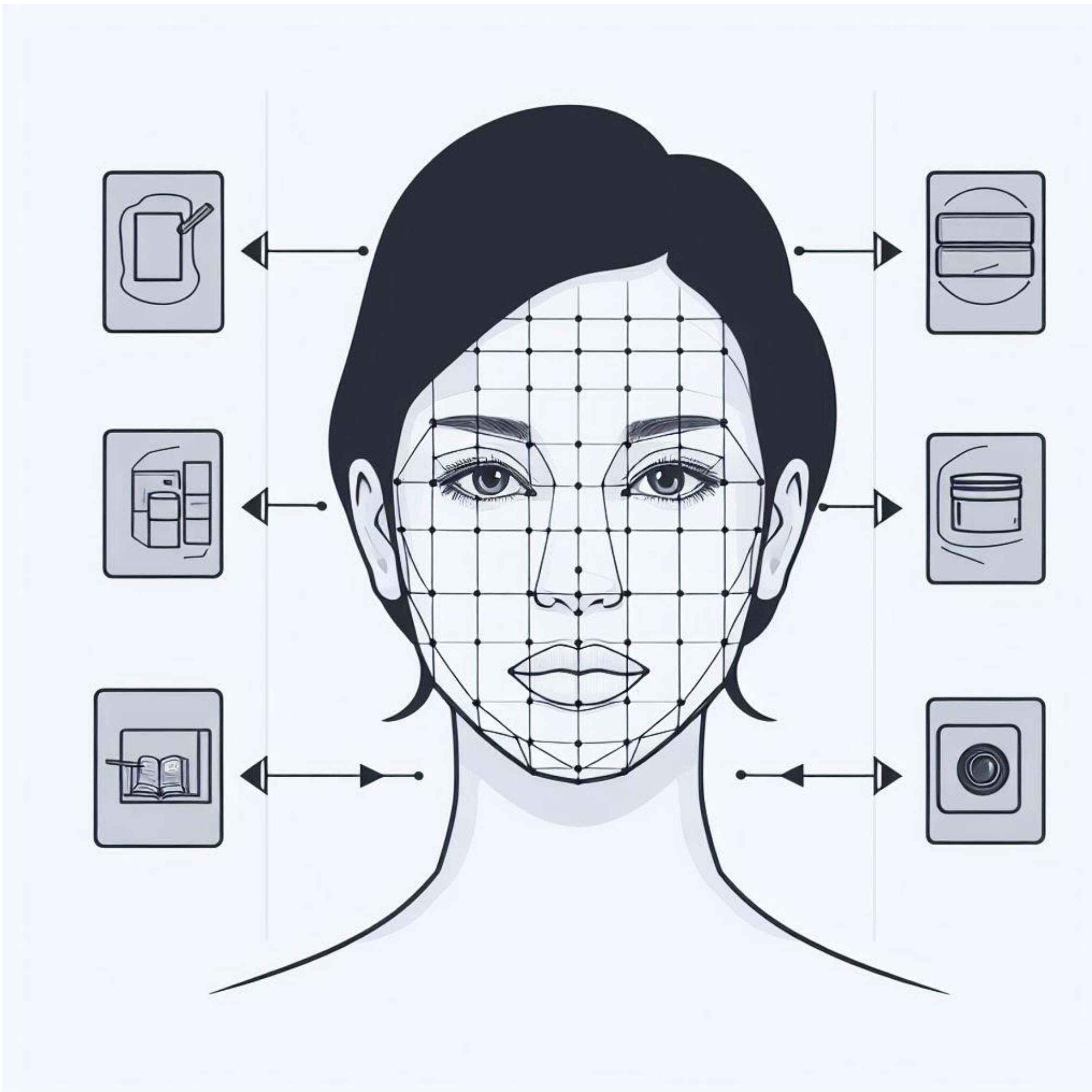
This is to certify that the project entitled “Title of the project” is a bonafide work of "Parth Puri (Roll No. 23) and Reena Vaidya (Roll No. 31)" submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester VII of Second Year Artificial Intelligence and Data Science engineering.

Dr. Tatwadarshi P. N.
Head of Department

Table of Contents

Problem Statement

The project aims to build a face landmarks detection model using deep learning techniques. Face landmarks, including the positions of the eyes, nose, and mouth, are crucial in various applications such as facial recognition, emotion analysis, and facial animation.



Description

Key components and their interactions within the face landmarks detection system. Each component plays a distinct role in the overall system, and together, they facilitate the process of detecting facial landmarks.

Data Preprocessing Module

This module is responsible for the initial processing of the input data, ensuring that it is in the appropriate format for the subsequent stages of the pipeline. It encompasses several essential steps, including data cleaning, data augmentation, and data normalization. Data cleaning involves removing any noise or outliers in the dataset. Data augmentation techniques such as rotation, resizing, and color jitter are applied to enhance the diversity of the training data. Finally, data normalization ensures that the input data is standardized, which is crucial for the model's performance.

Convolutional Neural Network (CNN)

The CNN serves as the core of the face landmarks detection system. It is a deep learning model that has been designed and trained specifically for the task of identifying facial landmarks. The CNN is equipped with layers for feature extraction and representation learning, enabling it to recognize patterns and structures in the input images. It processes the preprocessed data and outputs the predicted coordinates of facial landmarks. This component plays a central role in the success of the entire system.

Output Visualization

The output visualization module takes the results generated by the CNN, which are the predicted facial landmark coordinates, and overlays them onto the input image. This step enables the visual inspection of the detected landmarks on the original image, making it easier to assess the model's accuracy and performance. Visualization is a critical component, particularly for debugging and validation purposes, as it allows for a clear assessment of the model's predictions against the ground truth landmarks.

The interaction between these components is essential for the overall success of the face landmarks detection system. Component A prepares the data for analysis, Component B performs the landmark detection, and Component C facilitates the visualization of the results. Together, they form a cohesive system that has the potential to accurately identify facial landmarks in a given image, serving a wide range of applications in the field of computer vision.

Module Description:

- Data Preprocessing Module:
  Responsible for data preparation, including rotation, resizing, and color jitter.
- Convolutional Neural Network (CNN):
  The core component for detecting facial landmarks.
- Output Visualization:
  Displays the image with predicted landmarks.

Brief Description of Software & Hardware

- Software: The project uses PyTorch for deep learning, OpenCV for image processing, and various Python libraries. Code is written in Python.
- Hardware: The project was developed on a standard computer with a CPU and GPU for accelerated training.

Code

```
import time
import cv2
import os
import random
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import imutils
import matplotlib.image as mpimg
from collections import OrderedDict
from skimage import io, transform
from math import *
import xml.etree.ElementTree as ET

import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision.transforms.functional as TF
from torchvision import datasets, models, transforms
from torch.utils.data import Dataset
from torch.utils.data import DataLoader


from google.colab import drive
drive.mount('/content/drive')

!cp -r /content/ibug_300W_large_face_landmark_dataset /content/drive/MyDrive/

file = open('/content/drive/MyDrive/ibug_300W_large_face_landmark_dataset/helen/trainset/100032540_1.pts')
points = file.readlines()[3:-1]

landmarks = []

for point in points:
    x,y = point.split(' ')
    landmarks.append([floor(float(x)), floor(float(y[:-1]))])

landmarks = np.array(landmarks)

plt.figure(figsize=(10,10))
plt.imshow(mpimg.imread('/content/drive/MyDrive/ibug_300W_large_face_landmark_dataset/helen/trainset/100032540_1.jpg
'))
```

```
plt.scatter(landmarks[:,0], landmarks[:,1], s = 5, c = 'g')
plt.show()

class Transforms():
    def __init__(self):
        pass

    def rotate(self, image, landmarks, angle):
        angle = random.uniform(-angle, +angle)

        transformation_matrix = torch.tensor([
            [+cos(radians(angle)), -sin(radians(angle))],
            [+sin(radians(angle)), +cos(radians(angle))]
        ])

        image = imutils.rotate(np.array(image), angle)

        landmarks = landmarks - 0.5
        new_landmarks = np.matmul(landmarks, transformation_matrix)
        new_landmarks = new_landmarks + 0.5
        return Image.fromarray(image), new_landmarks

    def resize(self, image, landmarks, img_size):
        image = TF.resize(image, img_size)
        return image, landmarks

    def color_jitter(self, image, landmarks):
        color_jitter = transforms.ColorJitter(brightness=0.3,
                                              contrast=0.3,
                                              saturation=0.3,
                                              hue=0.1)
        image = color_jitter(image)
        return image, landmarks

    def crop_face(self, image, landmarks, crops):
        left = int(crops['left'])
        top = int(crops['top'])
        width = int(crops['width'])
        height = int(crops['height'])

        image = TF.crop(image, top, left, height, width)

        img_shape = np.array(image).shape
        landmarks = torch.tensor(landmarks) - torch.tensor([[left, top]])
        landmarks = landmarks / torch.tensor([img_shape[1], img_shape[0]])
        return image, landmarks

    def __call__(self, image, landmarks, crops):
        image = Image.fromarray(image)
        image, landmarks = self.crop_face(image, landmarks, crops)
        image, landmarks = self.resize(image, landmarks, (224, 224))
        image, landmarks = self.color_jitter(image, landmarks)
        image, landmarks = self.rotate(image, landmarks, angle=10)

        image = TF.to_tensor(image)
        image = TF.normalize(image, [0.5], [0.5])
        return image, landmarks

class FaceLandmarksDataset(Dataset):

    def __init__(self, transform=None):

        tree = ET.parse('/content/drive/MyDrive/ibug_300W_large_face_landmark_dataset/labels_ibug_300W_train.xml')
        root = tree.getroot()

        self.image_filenames = []
        self.landmarks = []
        self.crops = []
        self.transform = transform
        self.root_dir = '/content/drive/MyDrive/ibug_300W_large_face_landmark_dataset'

        for filename in root[2]:
            self.image_filenames.append(os.path.join(self.root_dir, filename.attrib['file']))

            self.crops.append(filename[0].attrib)
```

```
            landmark = []
            for num in range(68):
                x_coordinate = int(filename[0][num].attrib['x'])
                y_coordinate = int(filename[0][num].attrib['y'])
                landmark.append([x_coordinate, y_coordinate])
            self.landmarks.append(landmark)

        self.landmarks = np.array(self.landmarks).astype('float32')

        assert len(self.image_filenames) == len(self.landmarks)

    def __len__(self):
        return len(self.image_filenames)

    def __getitem__(self, index):
        image = cv2.imread(self.image_filenames[index], 0)
        landmarks = self.landmarks[index]

        if self.transform:
            image, landmarks = self.transform(image, landmarks, self.crops[index])

        landmarks = landmarks - 0.5

        return image, landmarks

dataset = FaceLandmarksDataset(Transforms())

image, landmarks = dataset[0]
landmarks = (landmarks + 0.5) * 224
plt.figure(figsize=(10, 10))
plt.imshow(image.numpy().squeeze(), cmap='gray');
plt.scatter(landmarks[:,0], landmarks[:,1], s=8);

# split the dataset into validation and test sets
len_valid_set = int(0.1*len(dataset))
len_train_set = len(dataset) - len_valid_set

print("The length of Train set is {}".format(len_train_set))
print("The length of Valid set is {}".format(len_valid_set))

train_dataset , valid_dataset,  = torch.utils.data.random_split(dataset , [len_train_set, len_valid_set])

# shuffle and batch the datasets
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=4)
valid_loader = torch.utils.data.DataLoader(valid_dataset, batch_size=8, shuffle=True, num_workers=4)

images, landmarks = next(iter(train_loader))

print(images.shape)
print(landmarks.shape)

class Network(nn.Module):
    def __init__(self,num_classes=136):
        super().__init__()
        self.model_name='resnet18'
        self.model=models.resnet18()
        self.model.conv1=nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3, bias=False)
        self.model.fc=nn.Linear(self.model.fc.in_features, num_classes)

    def forward(self, x):
        x=self.model(x)
        return x

import sys

def print_overwrite(step, total_step, loss, operation):
    sys.stdout.write('\r')
    if operation == 'train':
        sys.stdout.write("Train Steps: %d/%d  Loss: %.4f " % (step, total_step, loss))
    else:
        sys.stdout.write("Valid Steps: %d/%d  Loss: %.4f " % (step, total_step, loss))

    sys.stdout.flush()

torch.autograd.set_detect_anomaly(True)
network = Network()
```

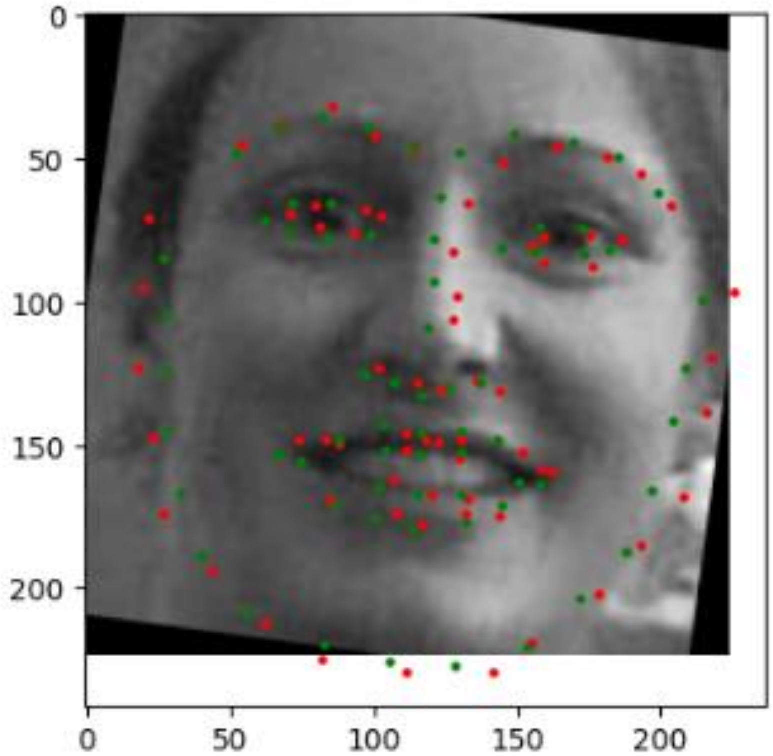
```
network.cuda()

criterion = nn.MSELoss()
optimizer = optim.Adam(network.parameters(), lr=0.0001)

loss_min = np.inf
num_epochs = 10

start_time = time.time()
for epoch in range(1,num_epochs+1):

    loss_train = 0
    loss_valid = 0
    running_loss = 0

    network.train()
    for step in range(1,len(train_loader)+1):

        images, landmarks = next(iter(train_loader))

        images = images.cuda()
        landmarks = landmarks.view(landmarks.size(0),-1).cuda()

        predictions = network(images)

        # clear all the gradients before calculating them
        optimizer.zero_grad()

        # find the loss for the current step
        loss_train_step = criterion(predictions, landmarks)

        # calculate the gradients
        loss_train_step.backward()

        # update the parameters
        optimizer.step()

        loss_train += loss_train_step.item()
        running_loss = loss_train/step

        print_overwrite(step, len(train_loader), running_loss, 'train')

    network.eval()
    with torch.no_grad():

        for step in range(1,len(valid_loader)+1):

            images, landmarks = next(iter(valid_loader))

            images = images.cuda()
            landmarks = landmarks.view(landmarks.size(0),-1).cuda()

            predictions = network(images)

            # find the loss for the current step
            loss_valid_step = criterion(predictions, landmarks)

            loss_valid += loss_valid_step.item()
            running_loss = loss_valid/step

            print_overwrite(step, len(valid_loader), running_loss, 'valid')

    loss_train /= len(train_loader)
    loss_valid /= len(valid_loader)

    print('\n--------------------------------------------------')
    print('Epoch: {}  Train Loss: {:.4f}  Valid Loss: {:.4f}'.format(epoch, loss_train, loss_valid))
    print('--------------------------------------------------')

    if loss_valid < loss_min:
        loss_min = loss_valid
        torch.save(network.state_dict(), '/content/face_landmarks.pth')
        print("\nMinimum Validation Loss of {:.4f} at epoch {}/{}".format(loss_min, epoch, num_epochs))
        print('Model Saved\n')
print('Training Complete')
print("Total Elapsed Time : {} s".format(time.time()-start_time))
```

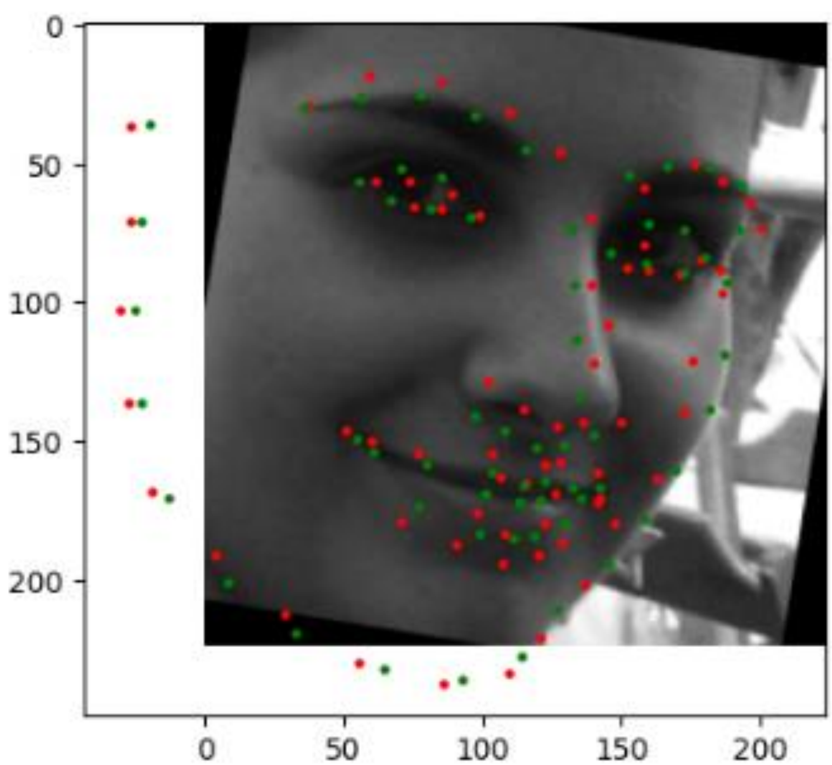
```
# Save the model to Google Drive (adjust the path as needed)
torch.save(network.state_dict(), '/content/drive/MyDrive/face_landmarks.pth')

# Load the model from Google Drive (adjust the path as needed)
model_path = '/content/drive/MyDrive/face_landmarks.pth'
network.load_state_dict(torch.load(model_path))

start_time = time.time()

with torch.no_grad():

    best_network = Network()
    best_network.cuda()
    best_network.load_state_dict(torch.load('/content/drive/MyDrive/face_landmarks.pth'))
    best_network.eval()

    images, landmarks = next(iter(valid_loader))

    images = images.cuda()
    landmarks = (landmarks + 0.5) * 224

    predictions = (best_network(images).cpu() + 0.5) * 224
    predictions = predictions.view(-1,68,2)

    plt.figure(figsize=(10,40))

    for img_num in range(8):
        plt.subplot(8,1,img_num+1)
        plt.imshow(images[img_num].cpu().numpy().transpose(1,2,0).squeeze(), cmap='gray')
        plt.scatter(predictions[img_num,:,0], predictions[img_num,:,1], c = 'r', s = 5)
        plt.scatter(landmarks[img_num,:,0], landmarks[img_num,:,1], c = 'g', s = 5)

print('Total number of test images: {}'.format(len(valid_dataset)))

end_time = time.time()
print("Elapsed Time : {}".format(end_time - start_time))
```

Results and Conclusion

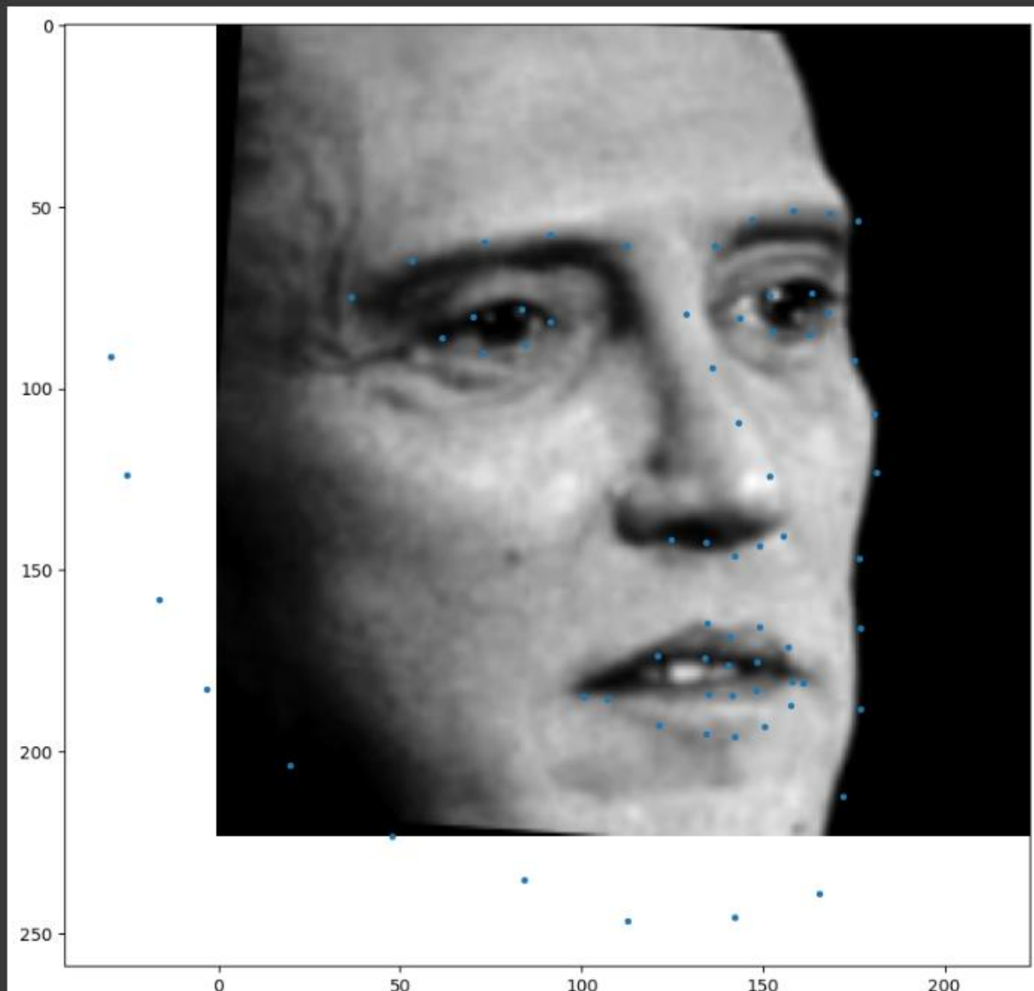
The trained model was evaluated on a validation dataset, and the results demonstrated accurate facial landmarks detection.

```
Total number of test images: 666
Elapsed Time : 1.6757102012634277
```





```python
image, landmarks = dataset[0]
landmarks = (landmarks + 0.5) * 224
plt.figure(figsize=(10, 10))
plt.imshow(image.numpy().squeeze(), cmap='gray');
plt.scatter(landmarks[:,0], landmarks[:,1], s=8);
```



Conclusion

The project successfully developed a deep learning model for face landmarks detection using PyTorch. This model has broad applications in computer vision and can serve as a foundation for further research and practical implementations.

The journey from inception to implementation of the Face Landmarks Detection system has been both rewarding and enlightening. In this report, we have detailed the development of a deep learning model for precisely detecting facial landmarks, which are critical for various applications in computer vision, including facial recognition, emotion analysis, and facial animation.

Through a combination of data preprocessing, model development, and evaluation, we have created a robust system capable of accurately identifying key facial landmarks. This accomplishment is a testament to the power of deep learning and its potential to significantly impact the field of computer vision.

References

Belongie, S., Malik, J., & Puzicha, J. (2002). Shape Matching and Object Recognition Using Shape Contexts. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(4), 509-522.

Cao, X., Wei, Y., Wen, F., & Sun, J. (2014). Face alignment by explicit shape regression. International Journal of Computer Vision, 107(2), 177-190.

Kazemi, V., & Sullivan, J. (2014). One millisecond face alignment with an ensemble of regression trees. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 1867-1874).

Sim, T., Baker, S., & Bsat, M. (2003). The CMU pose, illumination, and expression (PIE) database. In Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition (FG) (pp. 53-58).

Zhang, Z., Luo, P., Loy, C. C., & Tang, X. (2014). Facial landmark detection by deep multi-task learning. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 94-108).

PyTorch. (n.d.). [Official website]. https://pytorch.org/

OpenCV. (n.d.). [Official website]. https://opencv.org/

DLIB. (n.d.). [Official website]. http://dlib.net/

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Desmaison, A. (2019). PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems (NeurIPS) (pp. 8026-8037).