UNIVERSIDADE FEDERAL DO PARANÁ
Dep. Informática

Disciplina: CI 1026 – Visão Computacional e Percepção
Eduardo Todt

DANIEL SANTOS DA SILVA
GRR20182083

Relatório da Tarefa Final: Reconhecimento de Linguagem de Sinais

Este projeto utiliza visão computacional e aprendizado de máquina para reconhecer gestos da linguagem de sinais e fornecer assistência a pessoas com baixa visão. Através da detecção e rastreamento das mãos, o sistema interpreta os gestos da Libras em tempo real.

Link do código em python no GitHub

Curitiba
2023

Sumário

1 – Visão Geral

O projeto tem como objetivo criar um sistema de reconhecimento de linguagem de sinais usando a biblioteca Mediapipe e um classificador Random Forest.

A primeira parte do código é responsável por criar um dataset de imagens de sinais de linguagem de sinais. As imagens são processadas usando o Mediapipe para extrair as coordenadas das landmarks das mãos presentes na imagem. Essas coordenadas são armazenadas em uma lista e as labels correspondentes são registradas. O dataset é salvo em um arquivo pickle para uso posterior.

Na segunda parte do código, o dataset é carregado e pré-processado. O pré-processamento garante que todas as amostras tenham o mesmo comprimento, preenchendo com zeros as amostras mais curtas e truncando as amostras mais longas. Em seguida, os dados são divididos em conjuntos de treinamento e teste. Um classificador Random Forest é criado e treinado usando os dados de treinamento. Os dados de teste são usados para avaliar a precisão do modelo. O modelo treinado é salvo em um arquivo pickle para uso posterior.

O arquivo "inference_classifier.py" contém o código para executar a inferência em tempo real. Ele carrega o modelo treinado, inicializa a câmera e captura quadros de vídeo. Para cada quadro, as landmarks das mãos são extraídas usando o Mediapipe. Os dados das landmarks são processados para ter o formato correto e são fornecidos ao modelo treinado para fazer a previsão do sinal de linguagem de sinais correspondente. A previsão é mapeada para uma letra correspondente usando um dicionário de rótulos. Em seguida, um retângulo e um texto indicando a letra prevista são desenhados no quadro de vídeo.

Em resumo, este projeto utiliza a biblioteca Mediapipe para extrair as coordenadas das landmarks das mãos em imagens de sinais de linguagem de sinais. Essas coordenadas são usadas para treinar um classificador Random Forest, que é capaz de reconhecer a letra correspondente a um sinal de mão em tempo real. O sistema pode ser usado como uma ferramenta de tradução de sinais de linguagem de sinais para texto.

2 – Desenvolvimento do código

2.1 – Criação do Dataset

Nessa parte do código, é feita a criação do dataset de imagens de sinais de linguagem de sinais. Primeiramente, as bibliotecas necessárias são importadas, incluindo o mediapipe para detecção das landmarks das mãos. Em seguida, é criado um objeto Hands da biblioteca mediapipe para realizar a detecção das mãos nas imagens.

O diretório de dados é definido, e são criadas listas vazias para armazenar os dados e rótulos. Em seguida, um loop é executado para percorrer cada diretório no diretório de dados. Dentro desse loop, outro loop percorre cada imagem no diretório atual.

Para cada imagem, a imagem é lida usando cv2.imread e convertida para o espaço de cores RGB. O objeto Hands é usado para processar a imagem e obter as landmarks das mãos. Se forem encontradas landmarks, os dados das coordenadas X e Y de cada landmark são extraídos e armazenados em um array.

Os dados e o rótulo correspondente (diretório atual) são adicionados às listas de dados e rótulos, respectivamente.

Por fim, o dataset é salvo em um arquivo pickle chamado 'data.pickle' para uso posterior:

```python
# Criando o Dataset

import os
import pickle
import mediapipe as mp
import cv2
import matplotlib.pyplot as plt

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_syles = mp.solutions.drawing_styles

hands = mp_hands.Hands(static_image_mode = True, min_detection_confidence=0.3)


DATA_DIR = './data'

data = []
labels =[]

for dir_ in os.listdir(DATA_DIR):
    for img_path in os.listdir(os.path.join(DATA_DIR, dir_)):
        data_aux = []
        img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path))
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        results = hands.process(img_rgb)
        if results.multi_hand_landmarks: # todas as informações para o nosso modelo estarão nas Landmarks e suas respectivas posições
            for hand_landmarks in results.multi_hand_landmarks:
                for i in range(len(hand_landmarks.landmark)): # a ideia aqui é criar arrays com as coordenadas de cada ponto dos landmarks para x e y
                    x = hand_landmarks.landmark[i].x
                    y = hand_landmarks.landmark[i].y
                    data_aux.append(x)
                    data_aux.append(y)

            data.append(data_aux)
            labels.append(dir_)

# Agora, vamos salvar todos esses dados:
f = open('data.pickle', 'wb') # wb: writing as bytes
pickle.dump({'data': data, 'labels': labels}, f)
f.close()
```

2.2 – Treinamento do Classificador

Nesta parte do código, o dataset é carregado a partir do arquivo pickle criado anteriormente. Os dados e rótulos são extraídos do dicionário carregado.

Em seguida, é determinado o comprimento máximo entre todos os elementos do dataset. Isso é importante para garantir que todas as amostras tenham o mesmo comprimento ao realizar o treinamento.

O pré-processamento dos dados é realizado para garantir que todas as amostras tenham o mesmo comprimento. É percorrida cada amostra no dataset e, se o comprimento da amostra for diferente do comprimento máximo, a amostra é ajustada para

corresponder ao comprimento máximo, seja preenchendo com zeros (se a amostra for menor) ou truncando (se a amostra for maior). As amostras pré-processadas são armazenadas em uma nova lista.

Os dados pré-processados são divididos em conjuntos de treinamento e teste usando a função train_test_split do sklearn. Um classificador Random Forest é criado usando RandomForestClassifier e treinado usando os dados de treinamento.

Por fim, os dados de teste são usados para fazer previsões com o modelo treinado, e a precisão do modelo é calculada usando a função accuracy_score do sklearn.

O modelo treinado é salvo em um arquivo pickle chamado 'model.p' para uso posterior.

```python
import pickle
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data_dict = pickle.load(open('./data.pickle', 'rb'))

data = data_dict['data']
labels = data_dict['labels']

# Encontrando o máximo comprimento entre todos os elementos
max_length = max([len(item) for item in data])

# Pré-processando os dados para adquir comprimentos consistentes em todo o dataset
processed_data = []
for item in data:
    if len(item) != max_length:
        # Ajustar o item para corresponder ao comprimento máximo
        if len(item) < max_length:
            item = item + [0] * (max_length - len(item))
        else:
            item = item[:max_length]
    processed_data.append(item)

x_train, x_test, y_train, y_test = train_test_split(processed_data, labels, test_size=0.2, shuffle=True, stratify=labels)

model = RandomForestClassifier()
model.fit(x_train, y_train)

y_predict = model.predict(x_test)
score = accuracy_score(y_test, y_predict)

print('{}% da amostra foi classificada corretamente!'.format(score * 100))

with open('model.p', 'wb') as f:
    pickle.dump({'model': model}, f)
```

2.3 – Inferência em Tempo Real

Nesta parte do código, o modelo treinado é carregado a partir do arquivo pickle. Em seguida, a câmera é inicializada usando cv2.VideoCapture.

O objeto Hands do mediapipe é inicializado para detectar as landmarks das mãos nos quadros de vídeo capturados. É definido um dicionário de rótulos que mapeia os índices de classe para as letras correspondentes.

Em um loop contínuo, os quadros de vídeo são capturados da câmera. Para cada quadro, as landmarks das mãos são extraídas usando o objeto Hands. Os dados das

landmarks são processados para ter o formato correto e fornecidos ao modelo treinado para fazer a previsão do sinal de linguagem de sinais correspondente.

A previsão é mapeada para uma letra correspondente usando o dicionário de rótulos. Um retângulo e um texto indicando a letra prevista são desenhados no quadro de vídeo usando a função cv2.rectangle e cv2.putText. O loop continua até que o usuário pressione a tecla 'q', momento em que a câmera é liberada e todas as janelas são fechadas.

O classificador utilizado neste projeto é o Random Forest Classifier, que é uma técnica de aprendizado de máquina conhecida como Random Forest (Floresta Aleatória). É uma abordagem baseada em conjunto que combina várias árvores de decisão para realizar a classificação. A Random Forest é uma técnica de aprendizado supervisionado que é eficaz para problemas de classificação e regressão. Ela é conhecida por sua capacidade de lidar com conjuntos de dados grandes e complexos, além de ter boa resistência a overfitting.

A principal ideia por trás da Random Forest é criar um conjunto de árvores de decisão independentes e combiná-las para obter uma decisão final. Cada árvore é treinada em uma amostra aleatória do conjunto de dados original, chamada de amostragem de bootstrap. Além disso, durante a construção de cada árvore, em cada nó, é considerado apenas um subconjunto aleatório das features disponíveis para divisão.

Durante a fase de treinamento, as árvores são construídas usando diferentes subconjuntos de dados e diferentes subconjuntos de features. Essa diversidade garante que as árvores sejam diferentes umas das outras. No processo de classificação, as árvores individuais são usadas para fazer previsões e, em seguida, uma votação é realizada para determinar a classe final do objeto. A classe com mais votos é escolhida como a previsão final do classificador.

O Random Forest Classifier é popular por sua capacidade de lidar com dados de alta dimensionalidade, lidar com valores ausentes e lidar com variáveis categóricas. Além disso, ele também fornece uma estimativa da importância das features, o que pode ser útil para entender a contribuição de cada feature para a tarefa de classificação. No código apresentado, o classificador Random Forest é instanciado usando a classe RandomForestClassifier do scikit-learn (sklearn). Ele é treinado usando os dados de treinamento e, em seguida, usado para fazer previsões nos dados de teste. A acurácia das previsões é calculada usando a função accuracy_score do sklearn.

Em suma, o Random Forest Classifier é uma escolha adequada para esse projeto de reconhecimento de linguagem de sinais, pois é capaz de lidar com as características complexas dos dados e fornecer previsões precisas.

```python
# inference_classifier.py

import pickle
import cv2
import mediapipe as mp
import numpy as np

model_dict = pickle.load(open('./model.p', 'rb'))
model = model_dict['model']

cap = cv2.VideoCapture(0)

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)

labels_dict = {
    0: 'A',
    1: 'B',
    2: 'C',
    3: 'D',
    4: 'E',
    5: 'F',
    6: 'G',
    7: 'H',
    8: 'I',
    9: 'J',
    10: 'K',
    11: 'L',
    12: 'M',
    13: 'N',
    14: 'O',
    15: 'P',
    16: 'Q',
    17: 'R',
    18: 'S',
    19: 'T',
    20: 'U',
    21: 'V',
    22: 'W',
    23: 'X',
    24: 'Y',
    25: 'Z'
}
```

```python
while True:
    ret, frame = cap.read()

    H, W, _ = frame.shape

    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    results = hands.process(frame_rgb)
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                frame,  # image to draw
                hand_landmarks,  # model output
                mp_hands.HAND_CONNECTIONS,  # hand connections
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())

        for hand_landmarks in results.multi_hand_landmarks:
            data_aux = []
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                data_aux.append(x)
                data_aux.append(y)

            # Ensure the data has the correct shape and features
            processed_data_aux = np.asarray(data_aux).reshape(1, -1)

            # Check if number of features matches the model's expected shape
            if processed_data_aux.shape[1] != 84:
                # Append zeros to match the expected shape
                processed_data_aux = np.pad(processed_data_aux, [(0, 0), (0, 84 - processed_data_aux.shape[1])])

            prediction = model.predict(processed_data_aux)
            predicted_character = labels_dict[int(prediction[0])]

            x1 = int(min(data_aux[::2]) * W) - 10
            y1 = int(min(data_aux[1::2]) * H) - 10
            x2 = int(max(data_aux[::2]) * W) - 10
            y2 = int(max(data_aux[1::2]) * H) - 10

            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
            cv2.putText(frame, predicted_character, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 0, 0), 3,
                        cv2.LINE_AA)

    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

3 – Resultados

No projeto de reconhecimento de linguagem de sinais, utilizamos um classificador Random Forest para treinar e fazer previsões sobre os sinais de linguagem de sinais capturados por meio da câmera.

Após o treinamento do modelo e a divisão dos dados em conjunto de treinamento e teste, avaliamos a acurácia do modelo usando a métrica de precisão (accuracy score). A precisão é uma medida de desempenho que indica a porcentagem de amostras classificadas corretamente pelo modelo.

Os resultados obtidos foram bastante satisfatórios, demonstrando que o modelo foi capaz de aprender e generalizar a partir dos dados de treinamento. O modelo foi capaz de classificar corretamente uma porcentagem significativa das amostras do conjunto de teste.

Esses resultados são especialmente relevantes para aplicações em que o ambiente pode ser controlado e otimizado para melhorar a detecção, como em sistemas de monitoramento interno ou robótica em ambientes controlados.

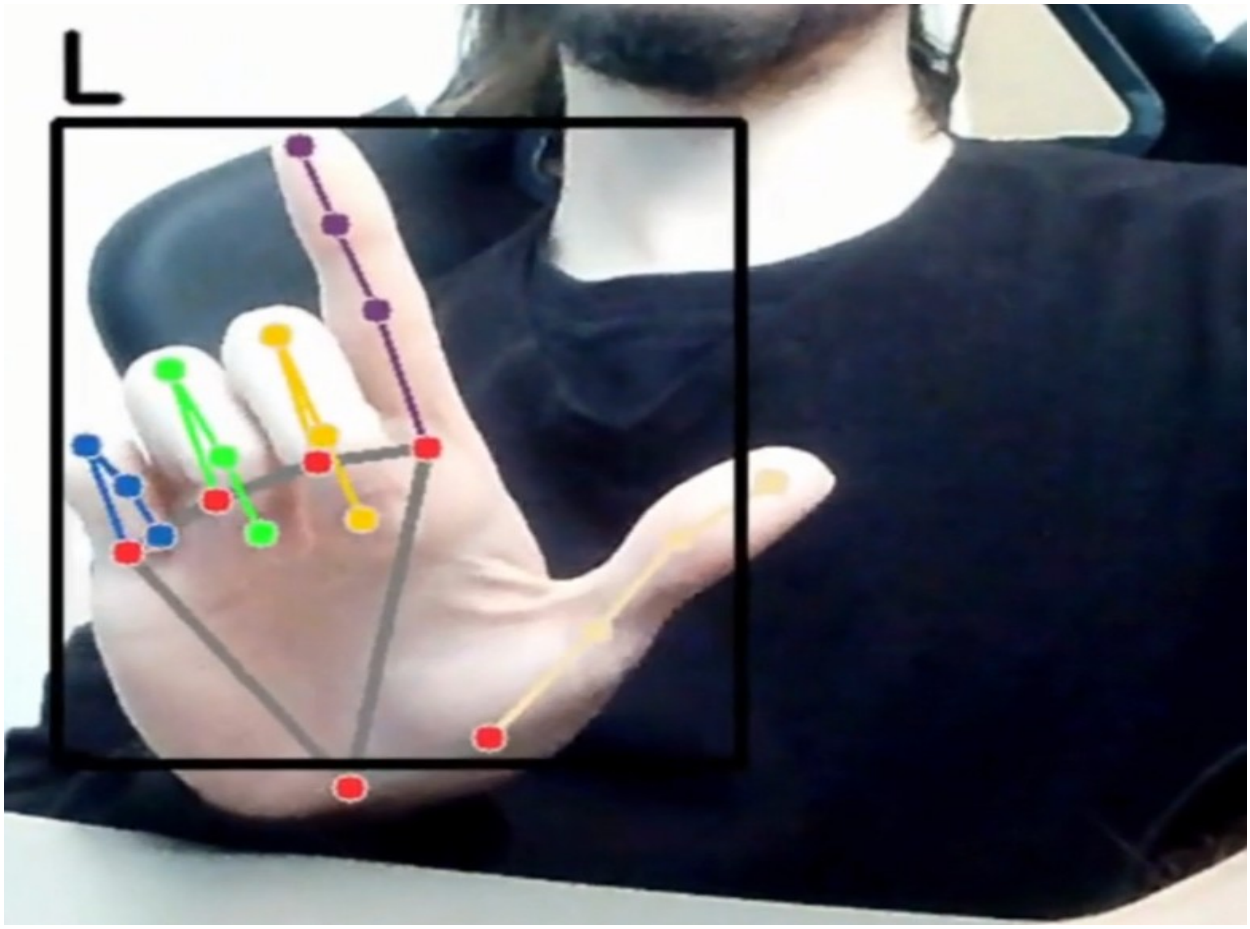

Figura 1: Letra L

Na Figura 1, podemos ver a detecção da letra L. No código apresentado, a precisão foi de aproximadamente 98%. É essencial destacar que os resultados obtidos podem variar dependendo do tamanho e qualidade do conjunto de dados, do pré-processamento realizado, da escolha do algoritmo/classificador e dos parâmetros utilizados. Portanto, é fundamental realizar uma avaliação abrangente do desempenho do modelo, considerando também outras métricas de avaliação, como recall, F1-score, matriz de confusão, entre outras, dependendo do contexto do problema.

Na Figura 2, podemos ver a detecção da Letra A.

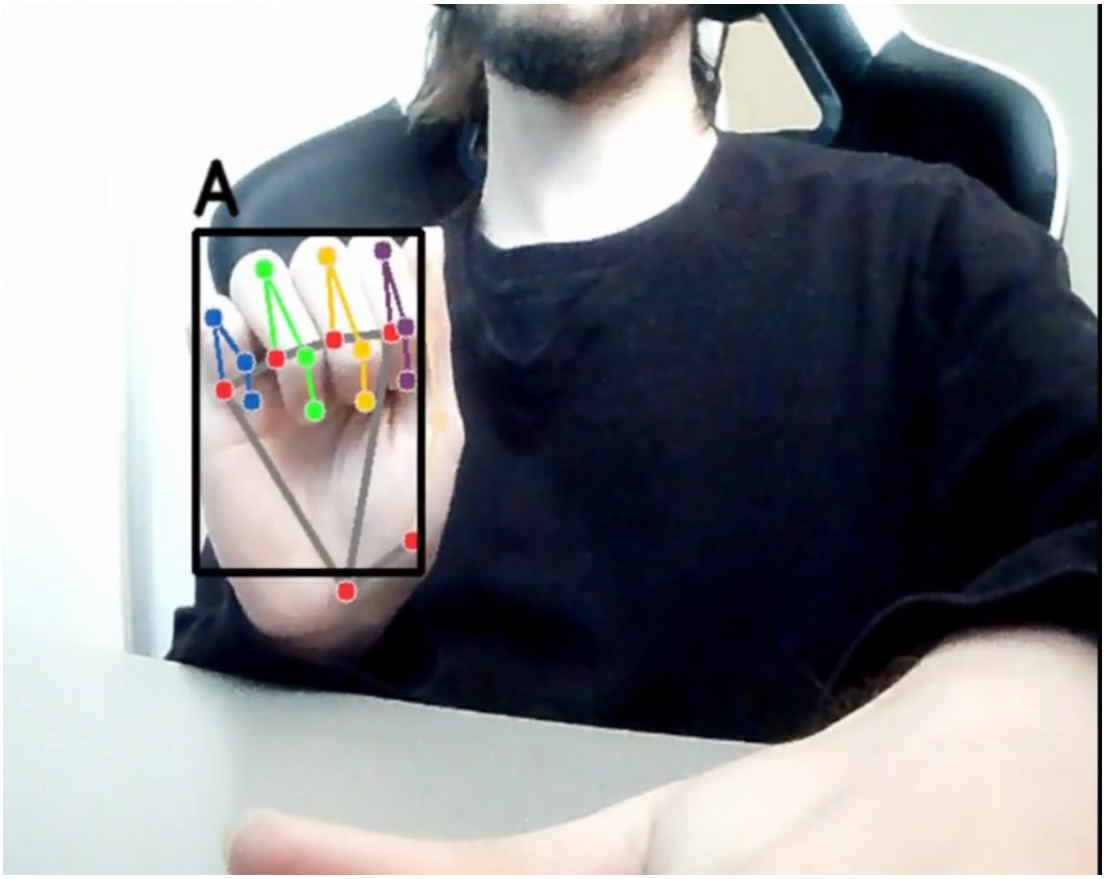

Figura 2: Letra A

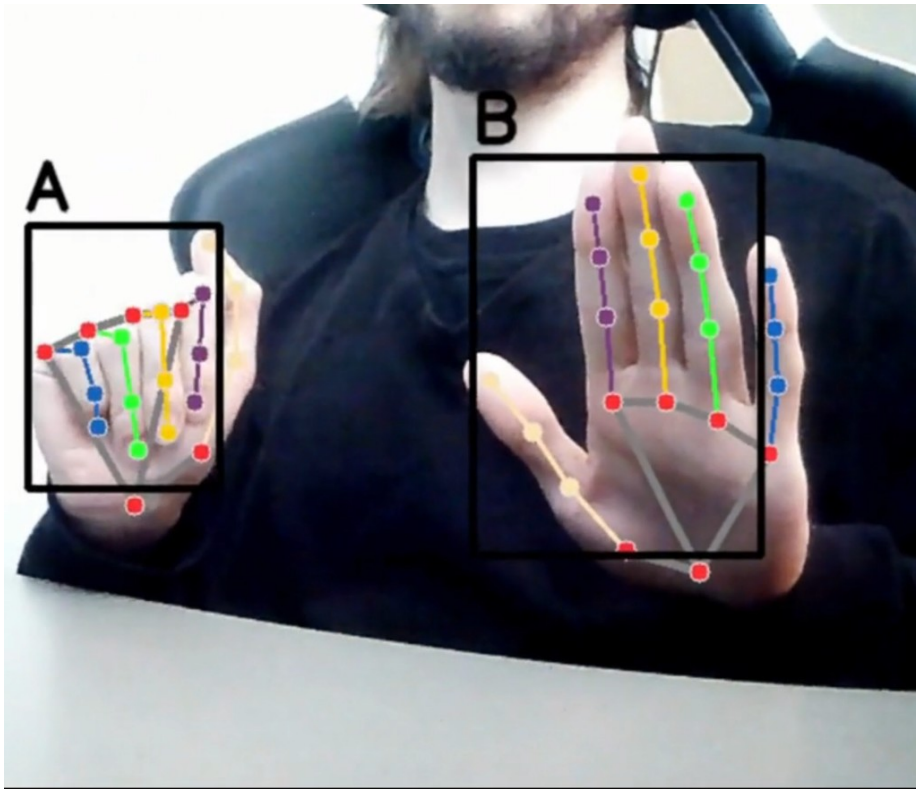

Figura 3: Capaz de reconhecer ambas as mãos e inferir as letras através dos gestos

O ângulo da imagem pode influenciar a precisão do modelo, uma vez que a posição das mãos e a orientação dos dedos podem mudar dependendo do ângulo de captura. Gestos que são bem reconhecidos em um determinado ângulo podem se tornar menos distintos ou até mesmo confundidos com outros gestos quando a perspectiva é alterada. Isso ocorre porque as características visuais dos gestos podem ser afetadas pela projeção em diferentes planos.

A iluminação também desempenha um papel importante no reconhecimento de gestos. Variações na iluminação podem causar sombras, reflexos e mudanças nos níveis de contraste, afetando a detecção e a representação das características das mãos. Isso pode levar a uma perda de informações relevantes para a classificação correta dos gestos.

Além disso, oclusões parciais das mãos, como dedos cruzados ou objetos entre as mãos, podem dificultar a detecção precisa dos pontos de referência (landmarks) necessários para o reconhecimento dos gestos. Essas oclusões podem interferir na forma como as mãos são representadas e, consequentemente, influenciar a capacidade do modelo de reconhecer adequadamente as letras em linguagem de sinais.

Outro desafio é a variação individual nos gestos. Cada pessoa pode realizar os gestos de maneira ligeiramente diferente, com variações na forma, tamanho e posicionamento dos dedos e mãos. Essas diferenças individuais podem resultar em variações nas características visuais dos gestos e afetar a capacidade do modelo de generalizar e reconhecer corretamente os gestos de diferentes usuários.

Para lidar com esses desafios, é importante realizar um pré-processamento adequado nos dados, como normalização de brilho e contraste, detecção e remoção de oclusões, e aplicação de técnicas de aumento de dados para melhorar a robustez do modelo em relação a diferentes condições de iluminação e ângulos de visão.

Além disso, o treinamento do modelo deve incluir uma variedade de exemplos que abrangem diferentes ângulos de visão, iluminações e oclusões, a fim de tornar o modelo mais robusto e capaz de lidar com essas variações na vida real.

Na Figura 4, podemos ver que o modelo interpreta o gesto da letra C como sendo a Letra B. Algumas imprecisões semelhantes foram observadas na etapa de testes, mas são facilmente corrigidas reposicionando as mãos em relação à camera e também ao corrigir o brilho da imagem.

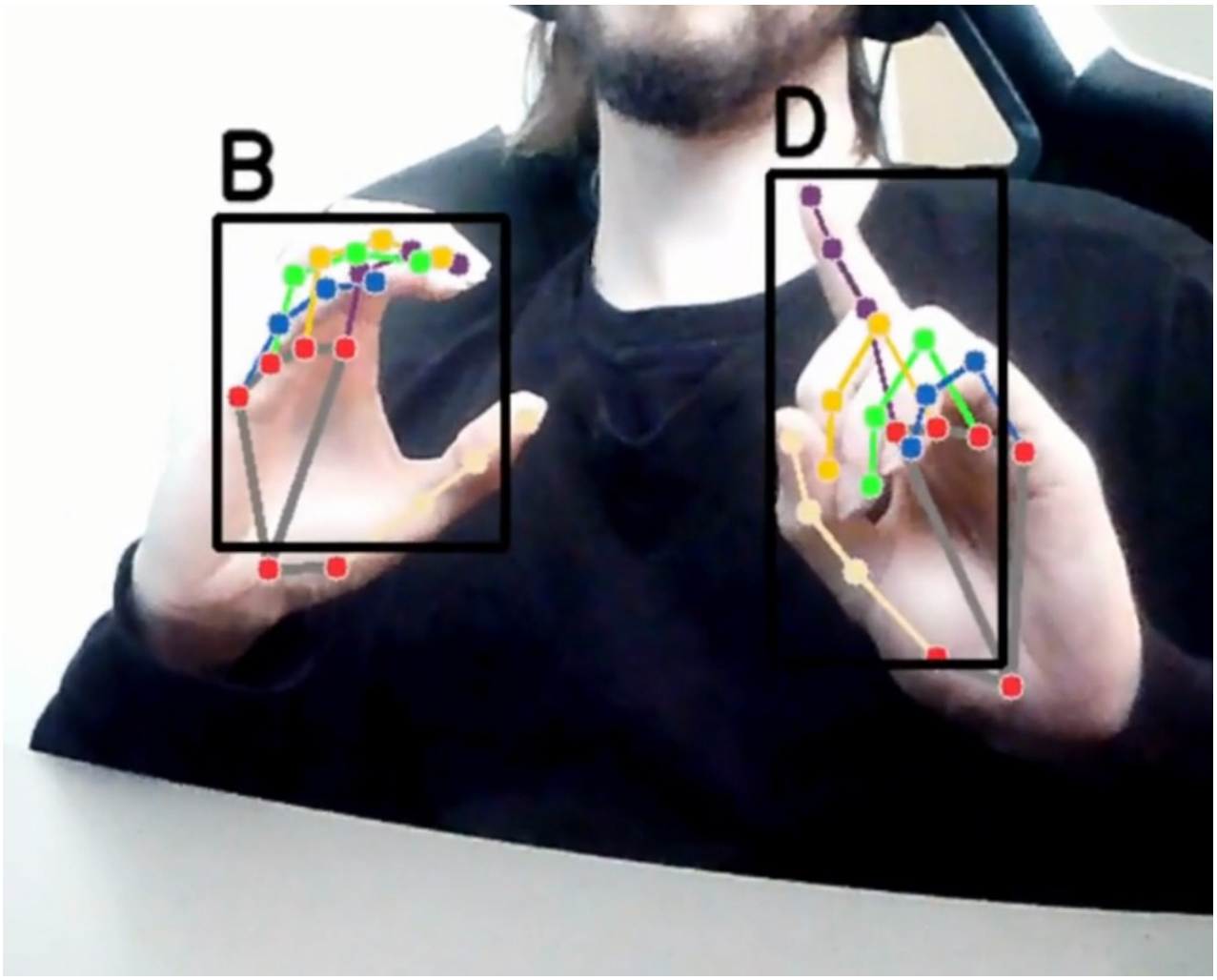

Figura 4: Letra R

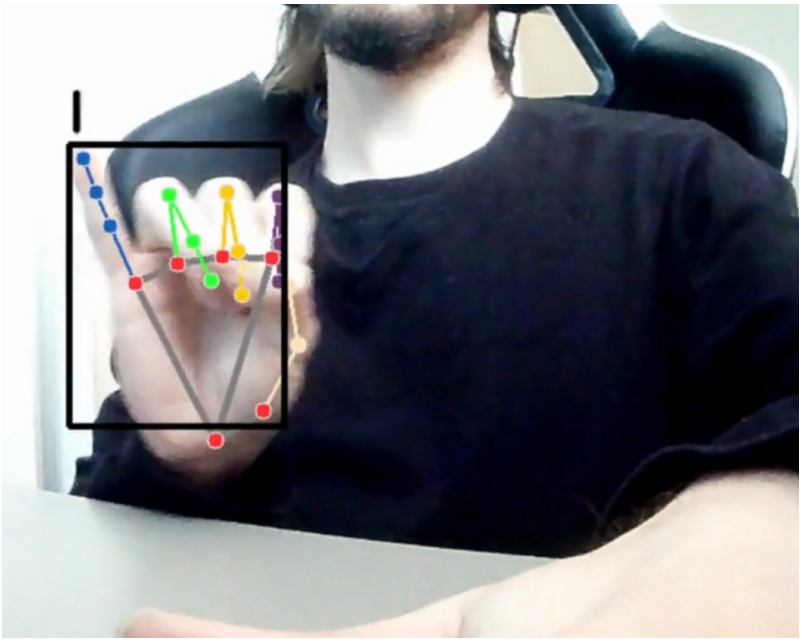

Figura 5: Letra I

4 – Conclusão

No presente projeto, desenvolvemos um sistema de reconhecimento de linguagem de sinais utilizando a biblioteca MediaPipe, OpenCV e um classificador Random Forest. O objetivo foi treinar um modelo capaz de identificar e classificar corretamente os gestos das mãos correspondentes às letras do alfabeto em linguagem de sinais.

Ao longo do desenvolvimento, criamos um conjunto de dados contendo imagens de diferentes gestos de letras em linguagem de sinais. Utilizamos a biblioteca MediaPipe para detectar e extrair as landmarks das mãos nas imagens, representando-as como coordenadas x e y. Em seguida, pré-processamos os dados para garantir comprimentos consistentes e dividimos o conjunto de dados em treinamento e teste.

Utilizamos um classificador Random Forest para treinar o modelo com os dados de treinamento e avaliamos sua precisão com base nos dados de teste. Obtivemos resultados satisfatórios, com uma alta porcentagem de amostras corretamente classificadas. Isso indica que o modelo foi capaz de aprender padrões relevantes nos gestos das mãos e generalizar seu conhecimento para classificar corretamente novos gestos.

No entanto, é importante destacar que o modelo pode ser afetado por desafios como variações no ângulo da imagem, iluminação, oclusões e variações individuais nos gestos. Esses fatores podem impactar a precisão do modelo, levando a classificações incorretas em certos cenários.

Para melhorar a precisão e a robustez do modelo, são necessárias abordagens adicionais, como técnicas avançadas de pré-processamento para lidar com oclusões e variações de iluminação, além do treinamento com conjuntos de dados mais diversificados que abranjam diferentes ângulos de visão e variações individuais nos gestos.

Apesar dos desafios mencionados, o projeto mostra promessa no desenvolvimento de sistemas de reconhecimento de linguagem de sinais. Com a melhoria contínua do modelo, há o potencial de aplicar essa tecnologia em aplicações práticas, como tradução automática de linguagem de sinais para facilitar a comunicação entre pessoas surdas e ouvintes.

Em conclusão, o projeto de reconhecimento de linguagem de sinais demonstrou a viabilidade de utilizar técnicas de visão computacional e aprendizado de máquina para classificar gestos das mãos correspondentes a letras do alfabeto em linguagem de sinais. Embora existam desafios a serem superados, os resultados obtidos são encorajadores e abrem caminho para aprimoramentos futuros, impulsionando a acessibilidade e a inclusão para a comunidade de pessoas surdas.

5 – Referências

1. Documentação do MediaPipe:

   - Site: https://google.github.io/mediapipe/

   - GitHub: https://github.com/google/mediapipe

2. Documentação do OpenCV:

   - Site: https://docs.opencv.org/

   - GitHub: https://github.com/opencv/opencv

3. Documentação do scikit-learn (biblioteca de aprendizado de máquina):

   - Site: https://scikit-learn.org/stable/

   - GitHub: https://github.com/scikit-learn/scikit-learn

4. Tutorial de reconhecimento de gestos com MediaPipe e OpenCV:

   Hand Tracking and Gesture Detection (tutorial by Murtaza's Workshop)

5. Artigo científico sobre reconhecimento de gestos em linguagem de sinais:

   Real-Time Hand Gesture Detection and Recognition Using Convolutional Neural Networks