**Análise dos resultados de predição dos algoritmos de classificação para a base de dados *American Sign Language***

**Levi Matheus Martins Sange**

**Resumo**

A base de dados de língua de sinais americana (ASL) possui 3000 imagens para cada sinal de letra de A à Z, sendo ao total, 78 mil imagens. Estas imagens são fotos quadro a quadro de um vídeo, possuem fundo e iluminação diferente a cada quadro. O objetivo deste trabalho é, utilizando de técnicas de processamento de imagens aliadas ao aprendizado de máquina, analisar os resultados dos algoritmos na predição da classe (letra) correta de sinal na imagem. Os algoritmos testados neste trabalho foram o *Support Vector Machine*, *K-nearest Neighbors* e *Multi-Layer Perceptron*, estes estão disponíveis no programa *Waikato Environment for Knowledge Analysis* (WEKA). Os algoritmos recebem como entrada dados das características extraídas das imagens através de código em linguagem Python, as características selecionadas para serem extraídas das imagens são estatísticas e de matriz de coocorrência (*Gray Level Co-occurrence Matrix*). Os resultados obtidos são discutidos e apresentados, assim como possíveis melhorias e aprimoramentos.

**Palavras-chave:** Aprendizado de máquina, Máquina de vetor suporte, K-ésimo Vizinho mais Próximo, Multi-Layer Perceptron

**Abstract**

The American Sign Language (ASL) database has 3000 images for each letter sign from A to Z, with a total of 78 thousand images. These images are frame-by-frame photos from a video, have a different background and lighting for each frame. The objective of this work is, using image processing techniques combined with machine learning, to analyze the results of the algorithms in the prediction of the correct class (letter) of the signal in the image. The algorithms tested in this work were Support Vector Machine, K-nearest Neighbors and Multi-Layer Perceptron, these are available in the program Waikato Environment for Knowledge Analysis (WEKA). The algorithms receive as input data of the characteristics extracted from the images through code in Python language, the characteristics selected to be extracted from the images are statistics and co-occurrence matrix (Gray Level Co-occurrence Matrix). The results obtained are discussed and presented, as well as possible improvements and improvements.

**Keywords:** Machine Learning, Support Vector Machine, K-nearest Neighbors, Multi-Layer Perceptron

**Etapas de pré-processamento**

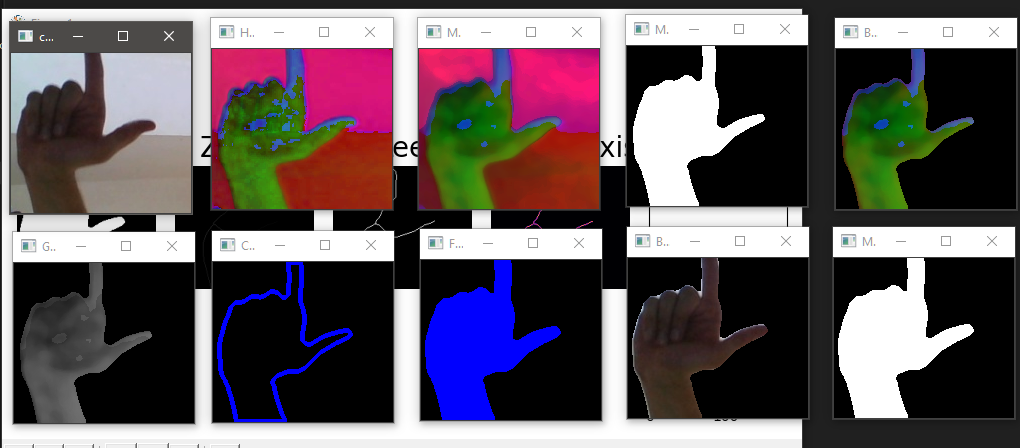
As imagens possuem a seguinte formatação e aparência, por exemplo, para a letra A:



**Imagem 1 – Imagens da base de dados – Disponível em: https://www.kaggle.com/grassknoted/asl-alphabet**

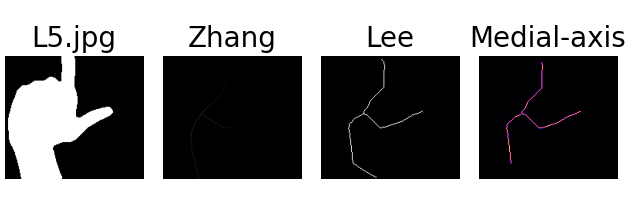
Possuem iluminação, fundo e brilho diferentes, tais diferenças impactam no resultado da extração e consequentemente na precisão dos algoritmos em predizer a classe da imagem. Perante este problema, utiliza-se técnicas e bibliotecas de pré-processamento de imagens para a linguagem Python. As bibliotecas utilizadas são OpenCV, SkImage, Pillow, NumPy e Pandas, todas podem ser instaladas via pip gerenciador de pacotes padrão do Python.

A imagem 2 apresenta as etapas de pré-processamento 1 a 1. As imagens possuem uma borda azul ao redor delas, é necessário retirar ou cortar a imagem para que esse detalhe não seja levado em consideração, aplica-se um corte nessa imagem que inicialmente possui dimensão 200x200, para a dimensão 160x180, desta maneira mantém-se o elemento central, a mão, e remove-se a maioria do entorno, após isso foi escolhido trabalhar com a imagem em canais HSV (Hue, Saturation, Value) devido a apresentar características interessantes, então converte-se a imagem para HSV, quando convertida, a imagem apresenta várias intensidades de cores e ruídos, para aliviar esta condição aplica-se uma suavização de filtro mediano, como o objetivo é extrair característica da mão na imagem somente, aplica-se uma limiarização e operação algébrica binária AND entre a mão em branco e a imagem original cortada, desta forma, o resultado é a mão destacada na imagem em canal HSV e possivelmente alguns ruídos de mesma faixa, estes “ruídos” ou outros elementos presentes na imagem que não sejam a mão não são desejados, então como próximo passo, deseja-se pegar somente o contorno da mão para eliminar de vez todo o resto que não pertence a mão na imagem, encontrando o contorno da mão em azul, preenche-o para poder aplicar novamente a operação algébrica binária AND entre a mão em azul e a imagem original, resultando, agora, somente na mão com o fundo da imagem em preto.



**Imagem 2: Etapas de pré-processamento - Autoria Prória**

Para o trabalho a abordagem de esqueletização da imagem foi optada, com o resultado da etapa anterior e aplicando a esquematização de três maneiras diferentes, obtém-se as seguintes imagens:

**Imagem 3: Resultado do processo de esqueletonização - Autoria Prória**

Com estes resultados é possível extrair características, que para o trabalho, foram, para GLCM, dissimilaridade, correlação, homogeneidade e energia, e para as medidas estatísticas, desvio padrão, variância, média e uniformidade.

Abaixo pode-se encontrar o código de todo o projeto para a extração de características, que itera dentro de um diretório trabalhando sobre cada imagem individualmente e extraindo a características de cada uma, guarda estas saídas como resultado em uma lista e gera um *dataframe* como saída, posteriormente convertida em formato Comma-separated values (CSV).

import numpy as np

import cv2

import skimage

import pandas as pd

from skimage import img\_as\_ubyte

from skimage.feature import greycomatrix, greycoprops

from skimage.morphology import skeletonize, disk, convex\_hull\_image, medial\_axis

from skimage import io, color, img\_as\_ubyte, data

from skimage.filters.rank import entropy

from skimage.segmentation import flood, flood\_fill

from skimage.util import invert, img\_as\_float

from sklearn.preprocessing import normalize, StandardScaler

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

from PIL import Image

#from resizeimage import resizeimage

import os, os.path

import json, csv, sys

from pathlib import Path

from pprint import pprint

# inicializando as variaveis

nomes=[] #vetor com os nomes das imagens

path = r'baseTreinamento/' #pasta a ser usada no processamento

#print(os.getcwd()) #checar se o projeto esta no diretorio correto

os.chdir(path) #mudar para o diretorio desejado

#print(os.getcwd())

imagens\_validas = [".jpg"] #extensao valida das imagens a serem usadas no processamento

listaResult = []#array com os resultados do processamento

#inicio a extracao de caracteristica da imagem

def get\_textural\_features(img, isMultidirectional=False, distance=1):

    img = np.array(img.convert('L', colors=8))

    glcm = greycomatrix(img, [distance], [0], 256, symmetric=False, normed=False)

    dissimilarity = greycoprops(glcm, 'dissimilarity')[0][0]

    correlation = greycoprops(glcm, 'correlation')[0][0]

    homogeneity = greycoprops(glcm, 'homogeneity')[0][0]

    energy = greycoprops(glcm, 'energy')[0][0]

    feature = np.array([dissimilarity, correlation, homogeneity, energy])

    return feature

#pegando todas as imagens do diretorio

for im in os.listdir(os.getcwd()):

    filename = os.path.splitext(im)#separa o nome da extensao

    ext = filename[1]#pega a extensao

    if ext.lower() not in imagens\_validas:#checa se esta dentro das extensoes validas

        continue

    nomeOrigem = filename[0] + ext #nome original da imagem

    imgOriginal = cv2.imread(nomeOrigem, cv2.IMREAD\_ANYCOLOR)#le a imagem

    cortada = imgOriginal[30:190, 10:190]#cortando imagens para deixar (160, 180)

    imgHSV = cv2.cvtColor(cortada, cv2.COLOR\_BGR2HSV)#HSV

    imgHSV[:,:,2] = cv2.equalizeHist(imgHSV[:,:,2])#equalizar

    imgBlur = cv2.medianBlur(imgHSV,11)#blur

    mascara = cv2.inRange(imgBlur,(0,20,0),(180,170,85))#pegar a mão thresh, inRange UTF = (imgHSV,(0,20,0),(180,170,60))

    btws = cv2.bitwise\_and(imgBlur, imgBlur, mask=mascara)#pegar a mão colorida

    maoGray = cv2.cvtColor(btws, cv2.COLOR\_BGR2GRAY)#mao em gray

    \_,maoThresh = cv2.threshold(maoGray,0,255,0)#thresh, otsu deixa espaços em branco na mão

    contours, hierarchy = cv2.findContours(maoThresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)#encontrando contornos list

    maxContour = max(contours, key = cv2.contourArea)#pegando o maior contorno da lista

    imgContornos = np.zeros([160,180,3],dtype=np.uint8)#imagem com mesmo formato

    cv2.drawContours(imgContornos, maxContour, -1, (255,0,0), 3)#desenha o contorno da mão em azul

    #pegar centro dos contornos

    for c in maxContour:

        M = cv2.moments(maxContour)

        cX = int(M["m10"] / M["m00"])#centro X

        cY = int(M["m01"] / M["m00"])#centro Y

    #floodfill a partir do centro do contorno

    cv2.floodFill(imgContornos, None, seedPoint=(cX,cY), newVal=(255,0,0))#preenchendo a imagem

    mascaraContorno = cv2.inRange(imgContornos,(250,0,0),(255,0,0))#criando a máscara para a imagem

    btwsContorno = cv2.bitwise\_and(cortada, cortada, mask=mascaraContorno)#pegar a mão colorida

    btwsGray = cv2.cvtColor(btwsContorno, cv2.COLOR\_BGR2GRAY)#mao em gray novamente

    \_,btwsThresh = cv2.threshold(btwsGray,0,255,0)#thresh, otsu deixa espaços em branco na mão

    kernel = np.ones((3,3),np.uint8)#elemento estruturante

    imgAbertura = cv2.morphologyEx(btwsThresh, cv2.MORPH\_OPEN, kernel)#opening na img thresh

    imgFechamento = cv2.morphologyEx(imgAbertura, cv2.MORPH\_CLOSE, kernel)#closing na img thresh

    \_,btwsThresh2 = cv2.threshold(imgFechamento,127,1,cv2.THRESH\_OTSU)#thresh novamente

    #cv2.imshow("Thresh", imgFechamento)

    skel, distance = medial\_axis(btwsThresh2, return\_distance=True)

    medial\_skeleton = distance \* skel#medial skeleton

    zhang\_skeleton = skeletonize(btwsThresh2, method='zhang')#esqueleton zhang

    lee\_skeleton = skeletonize(btwsThresh2, method='lee')#esqueleton lee

    mdarray = np.array(medial\_skeleton)

    zgarray = np.array(zhang\_skeleton)

    learray = np.array(lee\_skeleton)

    img\_medial = Image.fromarray(mdarray)

    img\_zhang = Image.fromarray(zgarray)

    img\_lee = Image.fromarray(learray)

    #extrair características dos 3 skeletons

    medial\_feature = get\_textural\_features(img\_medial)#extraindo da img medial skeleton

    zhang\_feature = get\_textural\_features(img\_zhang)#extraindo da img zhang skeleton

    lee\_feature = get\_textural\_features(img\_lee)#extraindo da img lee skeleton

    #criar lista dessas caracteristicas

    listaResult.append({

        "Imagem" : nomeOrigem,#nome da imagem com extensao

        "MD\_Dissimilaridade" : medial\_feature[0],#GLCM dissimilaridade

        "MD\_Correlação" : medial\_feature[1],#GLCM correlação

        "MD\_Homogeneidade" : medial\_feature[2],#GLCM homogeneidade

        "MD\_Energia" : medial\_feature[3],#GLCM energia

        "MD\_DesvioPadrao" : np.std(mdarray.flatten()),#Estatísticas desvio padrão

        "MD\_Variancia" : np.var(mdarray.flatten()),#Estatísticas variância

        "MD\_Media" : np.mean(mdarray.flatten()),#Estatísticas média

        "MD\_Uniformidade" : 1 - 0.5\*sum(abs(mdarray.flatten() - np.average(mdarray.flatten())))/(len(mdarray.flatten())\*np.average(mdarray.flatten())),#Estatísticas uniformidade

        "ZG\_Dissimilaridade" : zhang\_feature[0],#GLCM dissimilaridade

        "ZG\_Correlação" : zhang\_feature[1],#GLCM correlação

        "ZG\_Homogeneidade" : zhang\_feature[2],#GLCM homogeneidade

        "ZG\_Energia" : zhang\_feature[3],#GLCM energia

        "ZG\_DesvioPadrao" : np.std(zgarray.flatten()),#Estatísticas desvio padrão

        "ZG\_Variancia" : np.var(zgarray.flatten()),#Estatísticas variância

        "ZG\_Media" : np.mean(zgarray.flatten()),#Estatísticas média

        "ZG\_Uniformidade" : 1 - 0.5\*sum(abs(zgarray.flatten() - np.average(zgarray.flatten())))/(len(zgarray.flatten())\*np.average(zgarray.flatten())),#Estatísticas uniformidade

        "LE\_Dissimilaridade" : lee\_feature[0],#GLCM dissimilaridade

        "LE\_Correlação" : lee\_feature[1],#GLCM correlação

        "LE\_Homogeneidade" : lee\_feature[2],#GLCM homogeneidade

        "LE\_Energia" : lee\_feature[3],#GLCM energia

        "LE\_DesvioPadrao" : np.std(learray.flatten()),#Estatísticas desvio padrão

        "LE\_Variancia" : np.var(learray.flatten()),#Estatísticas variância

        "LE\_Media" : np.mean(learray.flatten()),#Estatísticas média

        "LE\_Uniformidade" : 1 - 0.5\*sum(abs(learray.flatten() - np.average(learray.flatten())))/(len(learray.flatten())\*np.average(learray.flatten())),#Estatísticas uniformidade

        "Classe" : nomeOrigem[0]#nome da classe

    })

#gerando dataframe

dataframe = pd.DataFrame(listaResult)

classes = dataframe.iloc[:,-1]#pegando as classes(26) (-1 por ser o ultimo elemento)

somente\_dados = dataframe.iloc[:,1:25]#tirando nome (0) e classe (26)

#normalizando

normalizar = StandardScaler()

dados\_normalizados = normalizar.fit\_transform(somente\_dados)

base = pd.DataFrame(dados\_normalizados)

base['z'] = classes#adicionando uma coluna a base com as classes

#saida em CSV

os.chdir("..")#subir um diretorio

base.to\_csv('saidaCSV.csv',header=False,index=False,sep=',',encoding='utf-8-sig')

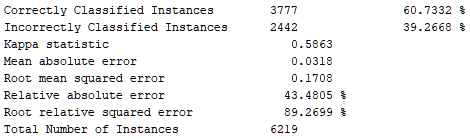
cv2.waitKey()

O código gera como saída o arquivo abaixo:

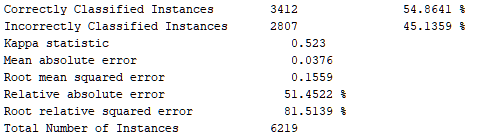


Após abrir o arquivo CSV no ambiente WEKA, e aplicá-lo nos algoritmos, com 10% das instâncias separadas para teste, obtém-se os seguintes resultados:

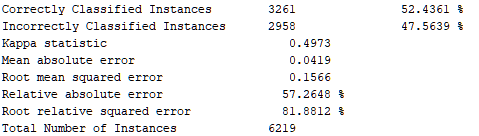
KNN (K=1):



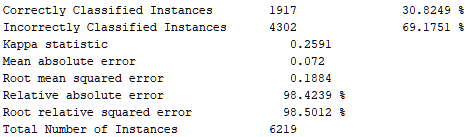
KNN (K=3):



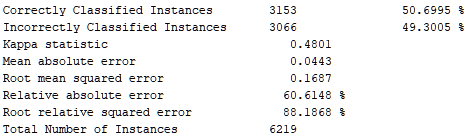
KNN (K=5):



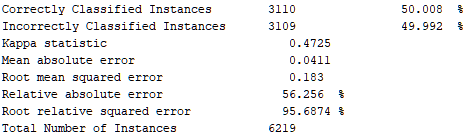
SMO (SVM):



MLP:



J48 (Árvore de decisão):



**Considerações Finais e discussão**

A precisão para diferentes valores e porcentagem de instâncias de teste, aproximam-se de 50%, atingindo o valor esperado para o objetivo do trabalho. Etapas de pré-processamento e abordagens diferentes para o problema podem resultar em alteração na precisão do resultado, tanto para valores superiores quanto inferiores, abre-se então, espaço para outros testes e pesquisas, tais como a utilização de *Local Binary Patterns* (LBP)*,* padrões binários locais que são descritores visuais usados para classificação em visual computacional, assim como o *Local Phase Quantization* (LPQ), que pode ser usado para caracterizar a textura da imagem subjacente, este métodos são mais recentes e podem apresentar resultados melhores.

Detalhes interessantes sobre os resultados, segundo o ambiente WEKA, os sinais de letras que os algoritmos mais acertaram foram das seguintes letras: C, F, K, O, P, Q, Z, sendo P, a com maior precisão, aproximadamente 85%. Os sinais de letras que os algoritmos menos acertaram foram das seguintes letras: E, H, J, M, N, R, T, W, X, Y, sendo X, a com menor precisão, aproximadamente 12%.

Quanto aos algoritmos selecionados para a classificação, como o objetivo do trabalho é a utilização dos mesmos, não se expandiu a prática e desenvolvimento para as redes neurais. As redes possuem camadas de convolução que facilitam etapas de pré-processamento que para o processo deste trabalho, por exemplo, acabam sendo realizadas por heurísticas humanas em todo o processo, já nas redes elas testam por si próprias abordagens que melhor se aplicam em cada camada desta forma aprimorando os resultados.

**Referências**

Kaggle: ASL (American Sign Language). Disponível em: https://www.kaggle.com/grassknoted/asl-alphabet. Acesso em 08 maio. 2021.