EXTRAÇÃO DE ATRIBUTOS

E

CLASSIFICAÇÃO

Leonardo Costa de Sousa

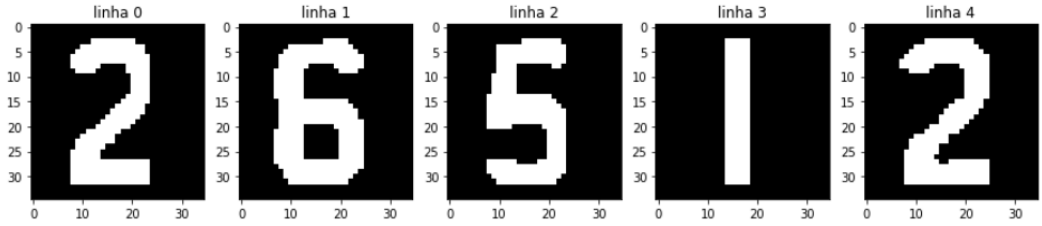
Trabalho 3 - Extração de atributos e Classificação

1. EXTRAÇÃO DE ATRIBUTOS

A base de dados é uma matriz 1226 x 3352, onde todos os 1225 primeiros números de cada linha (0 ou 1) formam numero de 0 a 9 e a ultimo numero da linha revela esse numero, a base então é constituída por 3352 números, para exibir esses númerosfoi usado a função getImageque recebe como argumento uma linha composta por 1225 e retorna uma imagem 35x35 desse numero.

```
def getImage(data, row):
    image = data[row, :-1] #seleciona a linha e apaga o útima coluna
    image.shape = (35, 35) #converte para duas dimensões
    image[image==1]=255
    image = np.uint8(image)
    return image
```

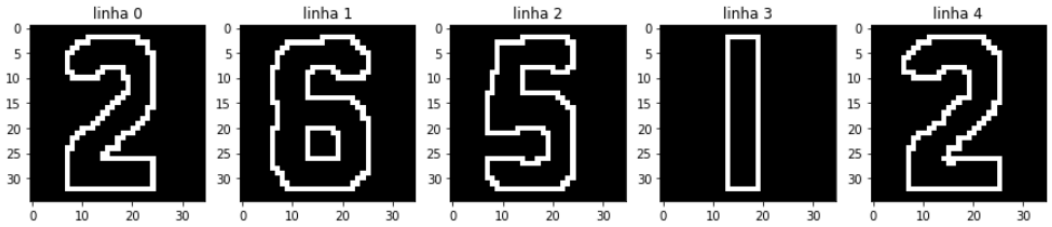
Abaixo foi plotado os cinco primeiros números dessa base de dados:



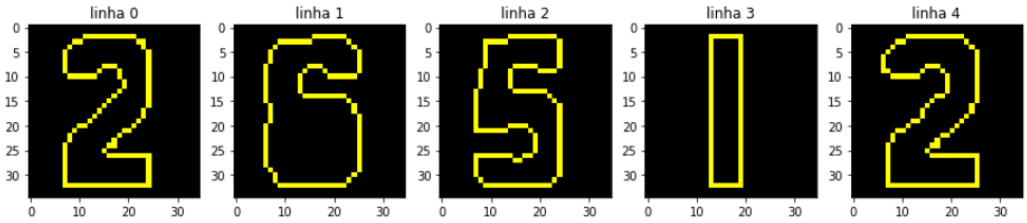
Extrator 01 - Código da Cadeia

O primeiro extrator utilizado e o código da Cadeia que consistem em montar uma seqüência de direções entre cada pixel e seu vizinho, até que todos os pixels da borda da imagem sejam considerados. Primeiramente e obtido o contorno da imagem usando para isso a diferença entre a sua dilatação e a imagem original, o código abaixo monta uma lista composta pelos contornos das imagens, também podemos ver o contorno dos cinco primeiros números.

```
kernel = np.ones((3,3), np.uint8)
edges = [cv2.dilate(img, kernel, iterations=1) - img for img in imgs] #array de bordas do array de imagens
```



Logo em seguida é usado a função cv2.findContours com o parâmetro cv2.RETR_EXTERNAL para capturar as coordenadas do contorno mais externo, Logo abaixo podemos ver a plotagem desses contornos



A função generateChainCode abaixo e responsável por montar o código da cadeia recebendo como parâmetro os pontos do contorno da imagem, a direção de cada segmento é codificado por um esquema de numeração baseado na visinhança-8.

```
def generateChainCode(ListOfPoints):
    chainCode = []
    textChain = " "
    for i in range(len(ListOfPoints)):
        a = ListOfPoints[i][0]
        #print("valor de A = ",a[0],a[1])
        if i != len(ListOfPoints)-1:
            b = ListOfPoints[i + 1][0]
            #print("valor de B = ",b[0],b[1])
        else :
            b = ListOfPoints[0][0]
        chainCode.append(getChainCode(a[0], a[1], b[0], b[1]))
        textChain = textChain + str(getChainCode(a[0], a[1], b[0], b[1]))
        textChainT = str(getChainCode(a[0], a[1], b[0], b[1]))

    return chainCode,textChain
```

A função getChainCode auxilia a função generateChainCode onde ele recebe as coordenadas de dois pontos e retorna uma de oito direções.

```
def getChainCode(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1

    if dx>0: #Significa que as posições podem ser (5,4,3)
        if dy <0 :
            code = 5
        elif dy==0:
            code = 4
        else:
            code = 3

    elif dx==0: # Significa que as posicoes podem ser (6,2)
        if dy<0:
            code = 6
        elif dy>0:
            code = 2
        else:
            code=8
    else: #Significa que as posicoes podem ser (7,0,1)
        if dy<0:
            code = 7
        elif dy==0:
            code = 0
        else:
            code = 1

    return code
```

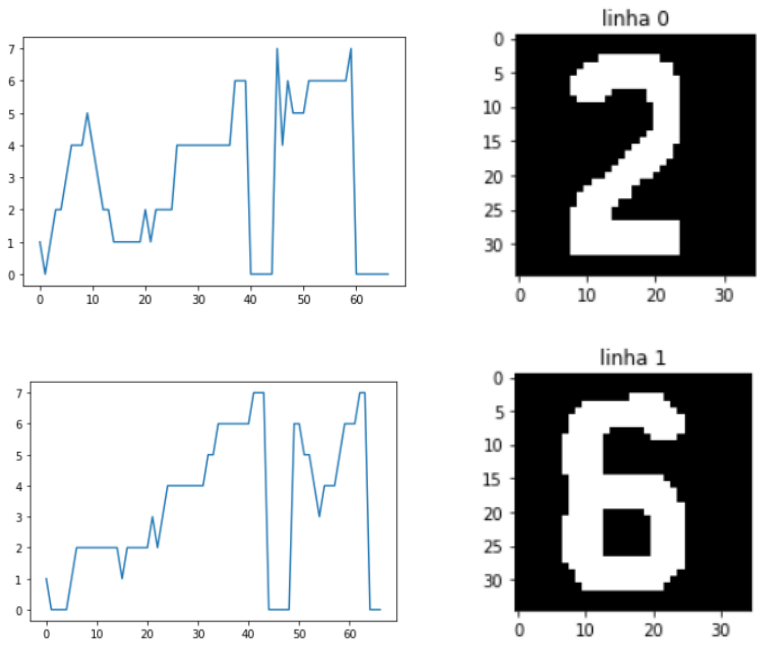
E a função normalizar e utilizado para que todas as cadeias tenham o tamanho da menor cadeia, ela recebe como parâmetro a cadeia e um valor pra limitar o numero de elementos que e retornado no final.

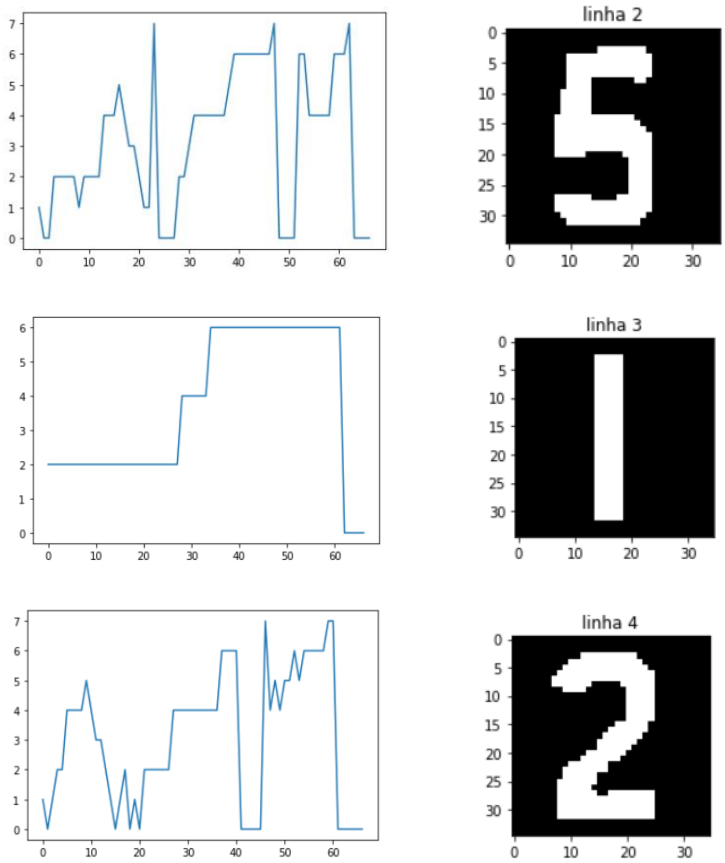
```
def normalizar(signal, smaller):

    index = 0
    newSignal = []
    prop = len(signal)/smaller
    d = prop

    while len(newSignal) < smaller:

        newSignal.append(signal[index])
        i, d = divmod(d, 1) #separa a pater inteira da parte decimal
        index += int(i) #adiciona a parte inteira ao indice
        d += prop #adiciona ao resto a proporcao

    return newSignal
```

Então é montado um dataset com os códigos da cadeia desses números conforme trecho abaixo:

```
chainCodes = []

for contours in n_contours:
    chainCode,tsx = generateChainCode(contours[0])
    chainCodes.append(chainCode)
    #print("chain code ",ind_contour," = ", tsx)
    #print("sum ",ind_contour," = ",sum(chainCode))
    #print("Quantidade de pontos =",len(contours[ind_contour]))
```

Segue ploagem dos cinco primeiros números:





Observa-se a similaridade entre os dois números dois da linha 0 e da linha 4, e o gráfico mais simples do número um da linha 3.

Extrator 2 – Assinatura

Uma assinatura é uma representação unidimensional da borda de um objeto, nesse caso, formada pela distância dos pontos da borda ao centróide em função do ângulo.

Temos a função centroid que retorna o centróide de um conjunto de pontos, ou seja, o centro de massa desses pontos e a função signature que recebe uma imagem calcula seu contorno com a função cv2.findContours e o centróide com a função centroid e então calcula a distância de cada ponto do contorno ao centróide retornando a assinatura da imagem em uma lista.

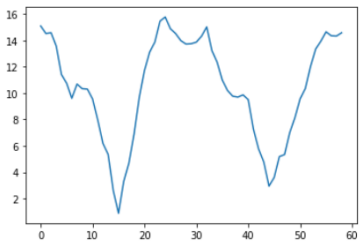
```
def centroid(img):
    ret,thresh = cv2.threshold(img,200,255,cv2.THRESH_BINARY)
    height, width = thresh.shape[:2]

    mass = 0
    Xcm  = 0.0
    Ycm  = 0.0

    for i in range(width) :
        for j in range(height) :
            if not thresh[j][i] :
                mass += 1
                Xcm  += i
                Ycm  += j

    return  [Ycm/mass, Xcm/mass]
```

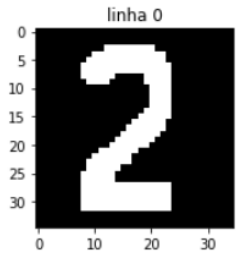
```
def signature(image):
    sign = []
    contours, _ = cv2.findContours(image,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
    center = centroid(image)
    for i in range(len(contours[0])):
        point = contours[0][i][0]
        dist = np.linalg.norm(point-center)
        sign.append(dist)

    return sign
```

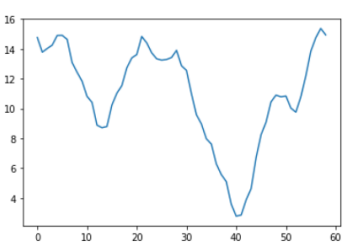
Criando do dataset de assinaturas normalizado pela menor assinatura

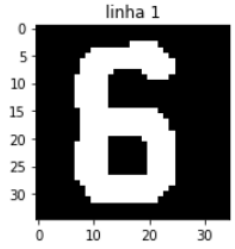
```
signatures = [signature(img) for img in imgs]

smaller = min([len(signature) for signature in signatures]) #pega o comprimento da menor cadeia
signatures = [box(signature, smaller) for signature in signatures] #faz a normalização das cadeias

signatures = np.array(signatures)
```

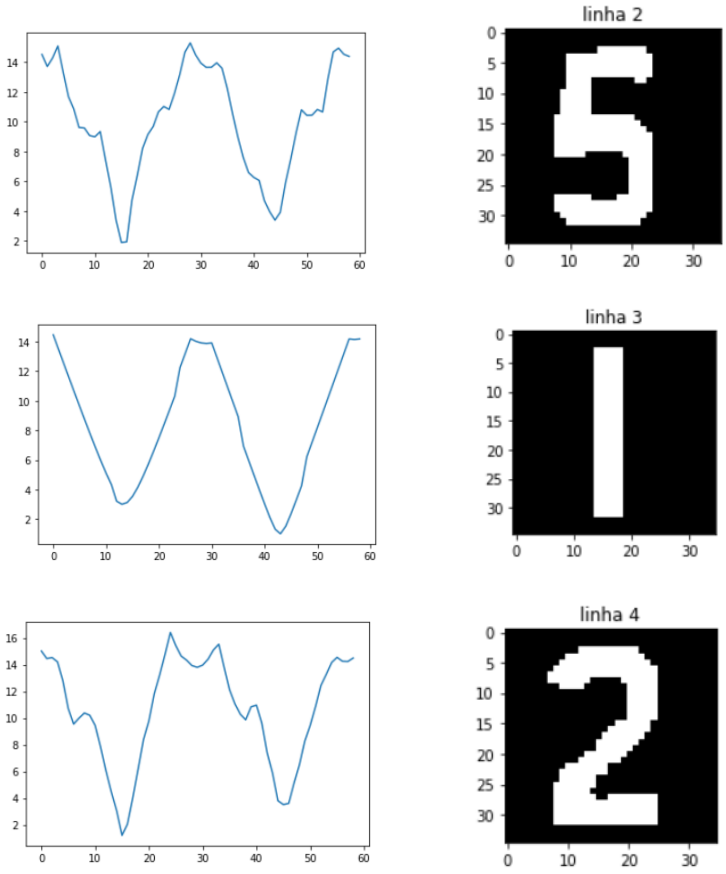
Segue a plotagem da assinatura dos primeiros cinco números:







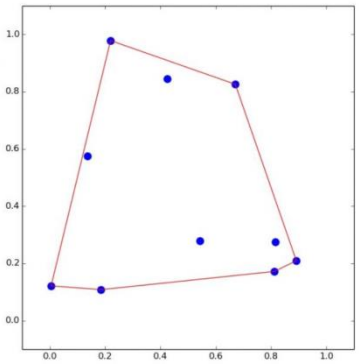




Assim como no código da cadeia aqui também se pode notar a semelhança entre os gráficos que representam o numero dois assim com uma maior simplicidade do gráfico do numero um.

Extrartor3 - Convex Hull

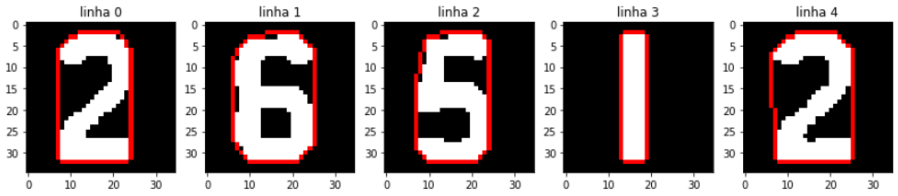
Dado um conjunto de pontos Convex Hull consiste em encontrar o menor número de pontos que gerem um polígono convexo que abrange todos esses pontos, como no exemplo abaixo.



Para se obter os pontos mais externos, primeiro foi calculada as coordenadas dos contornos da imagem através da função cv2.findContours com o parâmetro cv2.RETR_TREE, que retorna todos os contornos, e então aplicada a função cv2.convexHull para obter esses números.

```
convexHulls = []

for img in imgs:
    blur = cv2.blur(img, (3, 3))
    ret, thresh = cv2.threshold(blur, 50, 255, cv2.THRESH_BINARY)
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    hull = []
    for i in range(len(contours)):
        hull.append(cv2.convexHull(contours[i], False))

    convexHulls.append(hull)
```

Abaixo podemos ver o Convex Hull aplicado aos cinco primeiros números.



Entretanto esse atributo neste contesto não apresenta muita variabilidade, pois a maioria dos números aqui apresenta um formato oval, portanto não foi usado no processo de seleção.

2. CLASSIFICAÇÃO

Classificador 01 – MÁQUINA DE VETORES DE SUPORTE (SVM)

O SVM (Support Vector Machine) é um algoritmo de aprendizado de máquina supervisionado usado em classificação ou regressão. Seu foco principal é no treinamento e classificação de um dataset ele atua identificando o melhor hiperplano com a maior margem de separação entre as classes.

Usando SVM com o dataset chainCodes

Carregando as bibliotecas necessárias

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
```

Carregado o dataset e separando os dados onde 80% dos dados da base serão usadas para treino e 20% para teste e o random_state de 30.

```
X = chainCodes
y = dados[:,-1]

# Separando os dados de treino e teste:
X_train, X_test, y_train, y_test = train_test_split(X, y ,test_size=0.2, random_state=30)
```

Com Pipeline é criado um objeto que encapsula a tarefa de pré-processamento e classificação depois entregamos para o método GridSearchCV o pipeline criado anteriormente, os valores dos parâmetros e o número de validações cruzadas. O GridSearchCV faz uma combinação de todos os valores de hiperparâmetros e para cada combinação ele treina o modelo e gera os dados de teste

```
# Pipeline:
steps = [('scaler', StandardScaler()), ('SVM', SVC(kernel='poly'))]
pipeline = Pipeline(steps)

# Utilizando o GridSearchCV para o ajuste dos parâmetros:
parameters = {'SVM__C':[0.001, 0.1, 100, 10e5], 'SVM__gamma':[10,1,0.1,0.01]}
grid = GridSearchCV(pipeline, param_grid=parameters, cv=5)

# Treinando o modelo:
grid.fit(X_train, y_train)
print ("score = %3.2f" %(grid.score(X_test, y_test)))
print ("best parameters from train data: ", grid.best_params_)
```

Melhores parâmetros fornecidos pelo GridSearchCV:

```
score = 0.99
best parameters from train data:  {'SVM__C': 0.001, 'SVM__gamma': 1}
```

O modelo é então instanciado com os melhores parâmetros e usando um kernel polinomial e então treinado e posteriormente testado para ver como se comporta com um conjunto novo de dados.

```
# Treinando o modelo:
svc.fit(X_train,y_train)

# Realizando as predições:
resultado = svc.predict(X_test)
```

As classes 0 e 8 apresentaram a menor margem de acerto.

	precision	recall	f1-score	support
0.0	0.89	0.92	0.90	59
1.0	0.99	1.00	0.99	81
2.0	1.00	1.00	1.00	54
3.0	1.00	0.99	0.99	74
4.0	1.00	0.99	0.99	82
5.0	1.00	1.00	1.00	73
6.0	1.00	1.00	1.00	79
7.0	1.00	1.00	1.00	59
8.0	0.91	0.89	0.90	56
9.0	1.00	1.00	1.00	54
accuracy			0.98	671
macro avg	0.98	0.98	0.98	671
weighted avg	0.98	0.98	0.98	671

O que é confirmado observando a matriz de confusão abaixo, na classe zero (389 – 378 = 11) tem 11 erros.

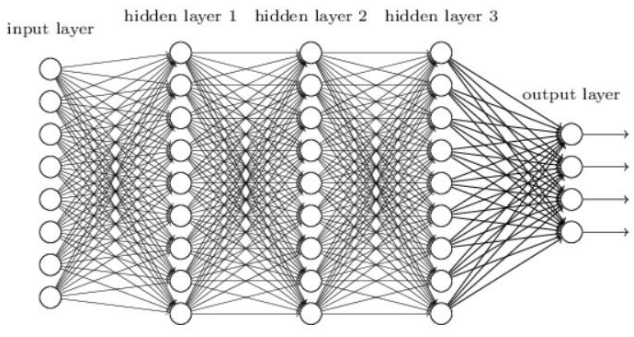
Predito	0	1	2	3	4	5	6	7	8	9	All
Real											
0	378	0	2	0	0	1	0	1	1	0	383
1	0	441	1	1	0	0	0	1	0	0	444
2	2	3	392	0	1	0	0	4	4	1	407
3	1	3	4	388	0	8	2	3	4	0	413
4	0	3	0	0	359	0	0	0	1	7	370
5	2	2	1	6	1	361	2	1	3	1	380
6	0	0	1	0	0	3	379	0	5	0	388
7	1	7	1	2	6	0	0	413	0	3	433
8	2	6	2	3	3	4	1	0	358	2	381
9	3	0	0	4	7	2	2	7	1	375	401
All	389	465	404	404	377	379	386	430	377	389	4000

O classificador SVM com o dataset signature apresentou um resultado ainda melhor como pode ser visto abaixo:

	precision	recall	f1-score	support
0.0	0.97	0.97	0.97	59
1.0	0.99	1.00	0.99	81
2.0	1.00	1.00	1.00	54
3.0	1.00	0.99	0.99	74
4.0	1.00	1.00	1.00	82
5.0	1.00	1.00	1.00	73
6.0	1.00	1.00	1.00	79
7.0	1.00	1.00	1.00	59
8.0	0.96	0.96	0.96	56
9.0	1.00	1.00	1.00	54
accuracy			0.99	671
macro avg	0.99	0.99	0.99	671
weighted avg	0.99	0.99	0.99	671

Classificador 02 – Rede Neural Artificial (RNA)

Uma Rede Neural Artificial pode ser definida, muito resumidamente, como uma estrutura complexa interligada por elementos de processamento simples (neurônios) organizados em camadas, capazes de realizar cálculos paralelos, para processamento de dados e representação de conhecimento.



Rede Neural com dataset do chainCodes

Carregando as bibliotecas necessárias

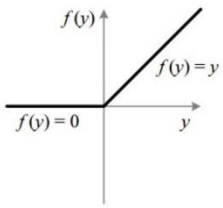
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.metrics import confusion_matrix
```

Depois de carregado o dataset, e montado uma seqüência de camadas com a biblioteca Keras onde primeiro e definida a quantidade correta de inputs de entrada, no caso 67, utilizando o argumento input_dim. Foram utilizadas seis camadas, as quais as quatro primeiras utilizarão a função de ativação ReLU e a função softmax na última, alem do otimizador rmsprop.

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_25 (Dense)            (None, 784)               53312

 dense_26 (Dense)            (None, 512)               401920

 dense_27 (Dense)            (None, 256)               131328

 dense_28 (Dense)            (None, 128)               32896

 dense_29 (Dense)            (None, 32)                4128

 dense_30 (Dense)            (None, 10)                330

=================================================================
Total params: 623,914
```

A função de ativação ReLU é não linear, o que significa que podemos facilmente copiar os erros para trás e ter várias camadas de neurônios ativos.

$$ReLU(x) = max\{0, x\} \qquad ReLU'(x) = \begin{cases} 1, & \text{se } x \geq 0 \\ 0, & \text{c.c.} \end{cases}$$



A função softmax também é um tipo de função sigmóide ela transforma as saídas para cada classe para valores entre 0 e 1 e também divide pela soma das saídas. Isso essencialmente dá a probabilidade de a entrada estar em uma determinada classe

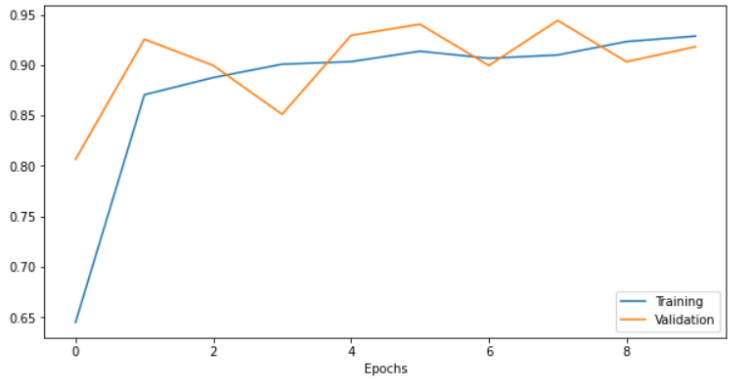
Depois de treinado o modelo é então testado:

```
h = model.fit(X_train, y_train_cat, epochs=10, verbose=1, validation_split=0.1)
```

```
results = model.evaluate(X_test, y_test_cat)
```

O gráfico abaixo exibe a evolução dos dados de treino e os dados de validação em torno das épocas, como e esperado no caso do treino a uma evolução do gráfico conformo o modelo aprende com a base de treino.

```
21/21 [==============================] - 0s 4ms/step - loss: 0.1733 - accuracy: 0.9121
Train:  0.9120715260505676
```



Rede Neural com dataset do signatures

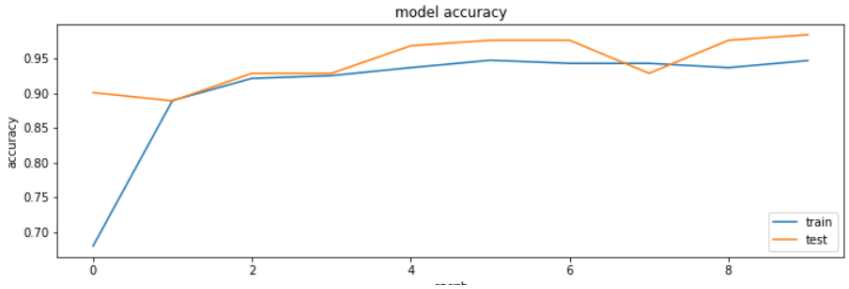
Nesse caso foi utilizado um modelo com três camadas, um input de entrada de 59 (número de classes), e mais duas camadas de Drouput que evita que partes da rede neural tenham mais responsabilidade que o necessário e aqui foi usao um otimizador Adam.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               30720

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 512)               262656

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 10)                5130

=================================================================
Total params: 298,506
Trainable params: 298,506
Non-trainable params: 0
```

Apresentando um desempenho bem melhor que no caso anterior

```
27/27 [==============================] - 0s 5ms/step - loss: 0.0969 - accuracy: 0.9821
Train:  0.982100248336792
```

Gráfico que mostra a evolução da acuracia dos dados de treinamento x teste

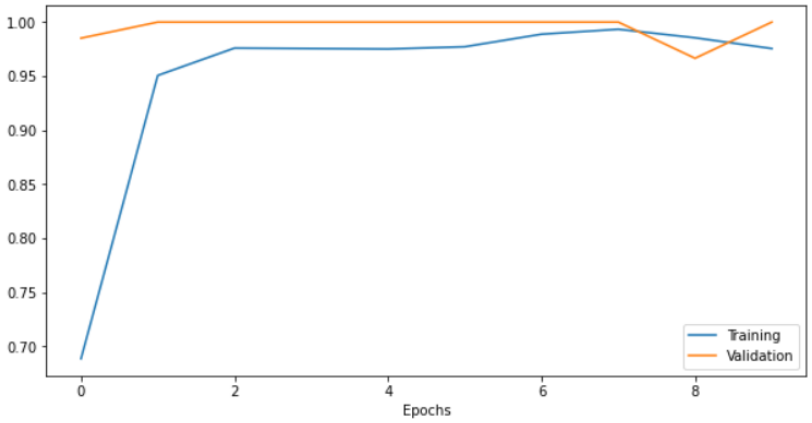


Rede Neural com dataset composto pelos dados originais

Como o Convex Hull não deu um bom dataset, utilizei nesse item os valores da própria base dados, ou seja, os 1225 valores (0 ou 1). E apresentou o melhor valor de acuracia ate aqui, mas ao custo de um dataset relativamente maior.

O modelo tem cinco camadas semelhante ao usado no dataset chainCode com um otimizador Adam, abaixo temos o acuracia media e o gráfico.

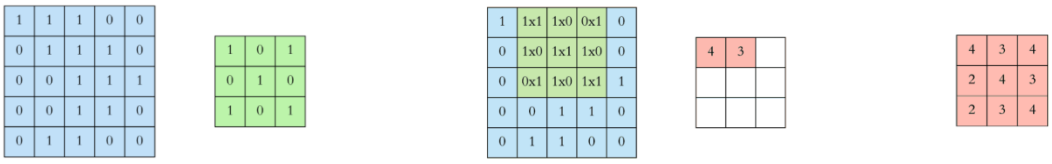
```
21/21 [==============================] - 0s 7ms/step - loss: 2.5200 - accuracy: 0.9985
Train:  0.9985097050666809
```



Classificador 03 – Rede Neural Convolucional (CNN)

Uma CNN é um tipo de rede neural normalmente utilizado para classificação de imagens, podendo ser divida em duas partes: extração de atributos (Convolução, Padding, Relu, Pooling) e uma rede neural tradicional.

No contesto de imagens convolução é um processo que usa um filtro/kernel para transformar uma imagem de entrada através de um produto de matrizes, no exemplo abaixo temos em azul o kernel e em verde a imagem em seguida a convolução e por fim a imagem transformada em vermelho.



O Padding é um processo auxiliar que ocorre antes da operação de convolução adicionando pixels ao redor da imagem para manter a dimensionalidade na imagem resultante durante toda a operação enquanto o Pooling é usado para diminuir a variância a pequenas alterações trata-se de um processo simples de redução da dimensionalidade. Para a divisão em duas partes da CNN (extração de características e rede neural) é usada uma camada denominada Flatten que transforma na matrix da imagem 2D em um formato de array 1D.

Implementação:

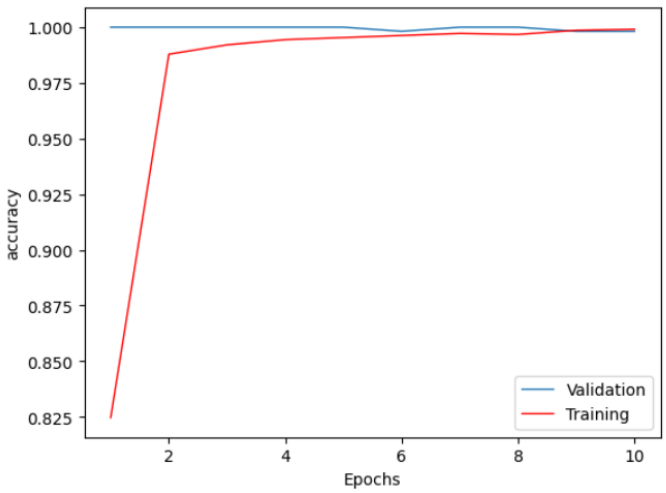
Inicialmente é aplicado um processo de convolução com 32 filtros na imagem de entrada. D depois é aplicado outro processo de convolução com 64 filtros. Na camada subseqüente, é aplicado a operação de *MaxPooling*, que reduzirá o tamanho da imagem em 17x17. Finalmente, esses valores são unificados em um *array* pela operação *Flatten* (64x17x17 = 18496) para serem à entrada de uma rede neural tradicional (Dense Layers).

```
model = Sequential()
model.add(Conv2D(32, (5,5), activation='relu', padding='same', input_shape=(35, 35,1)))
model.add(Conv2D(64, (5,5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

```
optimizer = Adam()
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
```

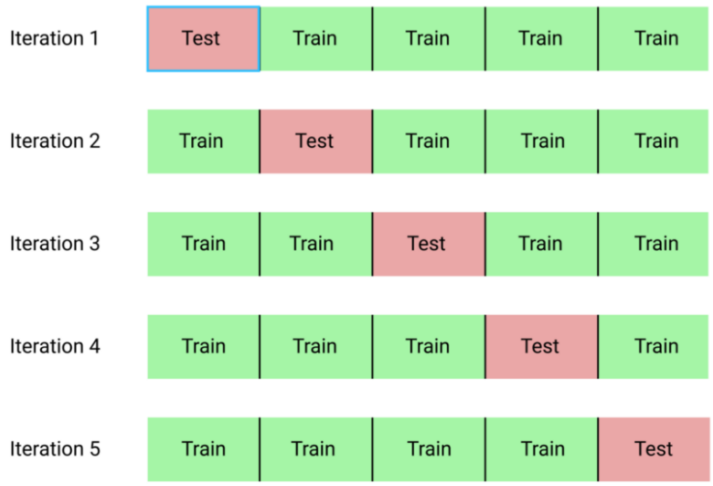
```
Epoch 10/10
67/67 [==============================] - ETA: 0s - loss: 0.0136 - accuracy: 0.9991
Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
67/67 [==============================] - 18s 270ms/step - loss: 0.0136 - accuracy: 0.9991 - val_loss: 0.0062 - val_accurac
y: 0.9981 - lr: 2.5000e-04
```

O gráfico abaixo mostra a evolução das acurácias no conjunto de treino em vermelho e no conjunto de treino em azul.



validação cruzada K-fold

K-fold é um método de cross-validation que consiste em dividir o dataset em k partes, usando k-1 partes para treino e a parte restante para teste, fazendo isso k vezes. Em cada uma das k vezes testam-se o modelo com um fold diferente calculando a métrica escolhida para a avaliação do modelo.



Representação visual do K-Fold Cross-Validation

No exemplo foi usado uma rede neural de três camadas sobre o dataset chainCode, e então instanciado um objeto Kfold da biblioteca sklearn com 10 partições e o parâmetro shuffle = True isso garante que o modelo não irá aprender alguma relação de ordem entrem os registros do *dataset evitando* overfitting .

```
kfold = KFold(n_splits=10, shuffle=True)
```

Em um loop controlado pelo numero de partições, onde em cada interação é criado o modelo, treinada e testado com os intervalos de treino e teste separado no kfold.

```
fold_no = 1
for train, test in kfold.split(inputs, targets):
    model = Sequential()

    model.add(Dense(512, input_dim=67, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']) #Otimização do modelo
```

Esse processo faz que o treino do modelo demore bem mais, mas é importante para ter certeza de que os dados estão generalizando bem.