EFC3_173096

November 4, 2022

Luciano Pinheiro Batista 173096

Para a atividade foram utilizadas algumas funções prontas da biblioteca ScikitLearn, já que algumas delas demandariam uma complexidade razoavelmente grande para serem feitas do zero.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')


from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
```

Os conjuntos de imagens de treino, teste e validação foram carregados e normalizados. Em seguida as imagens, que são compostas de 3 canais de cor, foram redimensionadas com o objetivo de servirem como entrada para a rede neural MLP.

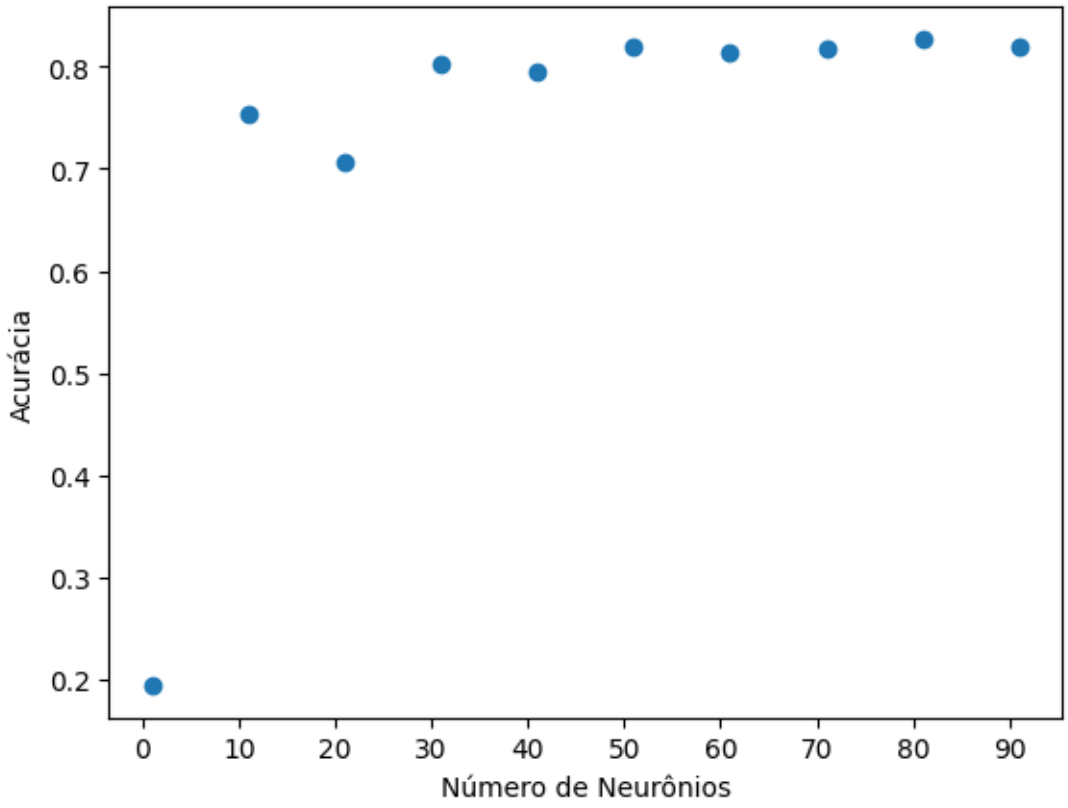
[2]:
```
Train_Images = np.load('train_images.npy')/255
Train_Labels = np.load('train_labels.npy')

Val_Images = np.load('val_images.npy')/255
Val_Labels = np.load('val_labels.npy')

Test_Images = np.load('test_images.npy')/255
Test_Labels = np.load('test_labels.npy')

Train_Images = np.reshape(Train_Images, (len(Train_Images), 28*28*3))
Train_Labels = np.reshape(Train_Labels, len(Train_Labels))

Val_Images = np.reshape(Val_Images, (len(Val_Images), 28*28*3))
Val_Labels = np.reshape(Val_Labels, len(Val_Labels))

Test_Images = np.reshape(Test_Images, (len(Test_Images), 28*28*3))
Test_Labels = np.reshape(Test_Labels, len(Test_Labels))
```

A primeira rede neural consistiu de apenas uma camada intermediária e foi configurada da seguinte

maneira: o número de neurônios da camada foi aumentado 1 a até 100, de 10 em 10. O algoritmo de otimização utilizado foi o *Adam*, que se adequa a datasets relativamente grandes e a função de ativação utilizada foi a *Relu*, que apresenta um melhor custo benefício entre eficiência no ajuste de pesos e desempenho da aproximação. A acurácia obtida com o conjunto de validação foi calculada para cada configuração e o gráfico de acurácia em função de número de neurônios foi gerado.

[5]:
```
#Treinamento e Validação
Num_Neuronios = np.arange(1, 100, 10)
Acuracias = []

for Numero in Num_Neuronios:

    print(Numero)

    Neuronio_1 = MLPClassifier(hidden_layer_sizes = (Numero),
                        max_iter = 50,
                        tol = 0.01,
                        learning_rate_init = .001,
                        solver = "adam",
                        activation = "relu",
                        learning_rate = "constant",
                       )

    Neuronio_1.fit(Train_Images, Train_Labels)

    Pred_1 = Neuronio_1.predict(Val_Images)

    Acuracias.append(accuracy_score(Val_Labels, Pred_1))


fig, ax = plt.subplots()
ax.scatter(Num_Neuronios, Acuracias)
ax.set_xlabel('Número de Neurônios')
ax.set_ylabel('Acurácia')
plt.locator_params ('x', nbins = len(Num_Neuronios))
plt.show()
```

```
1
11
21
31
41
51
61
71
81
91
```



Para a etapa de teste da primeira rede neural foram utilizados 41 neurônios. Observando o gráfico, há relativamente pouca melhora da acurácia ao se aumentar ainda mais o número de neuronios, o que implicaria também em mais parâmetros para serem ajustados e um maior esforço computacional.

[22]:
```
#Teste
Neuronio_1 = MLPClassifier(hidden_layer_sizes = (41),
                           max_iter = 50,
                           tol = 0.01,
                           learning_rate_init = .001,
                           solver = "adam",
                           activation = "relu",
                           learning_rate = "constant",
                          )

Neuronio_1.fit(Train_Images, Train_Labels)

Pred_1 = Neuronio_1.predict(Test_Images)

#Acurácia
print("Acurácia MLP 1 camada:", accuracy_score(Test_Labels, Pred_1))
```

```
#Matriz de Confusão
mc_1 = confusion_matrix(Test_Labels, Pred_1, labels=[0, 1, 2, 3, 4, 5, 6, 7])
Formata_Matriz(mc_1)
```

```
Acurácia MLP 1 camada: 0.7930429698918445
```

[22]:

	Basófilos	Eosinófilos	Eritroblastos	\
Basófilos	139	5	0	
Eosinófilos	2	592	0	
Eritroblastos	5	0	237	
Granulócitos Imaturos	27	28	8	
Linfócitos	12	1	21	
Monócitos	6	7	1	
Neutrófilos	0	28	19	
Plaquetas	0	0	5	

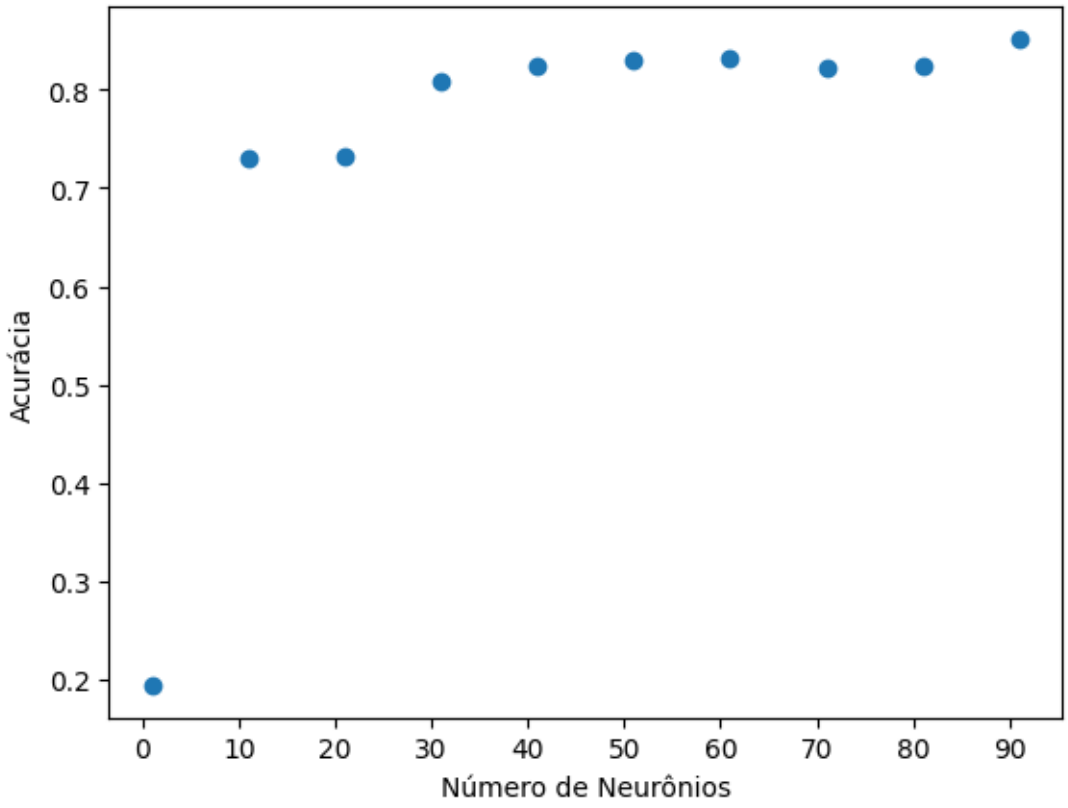
	Granulócitos Imaturos	Linfócitos	Monócitos	\
Basófilos	74	5	20	
Eosinófilos	13	1	3	
Eritroblastos	27	6	3	
Granulócitos Imaturos	429	9	50	
Linfócitos	35	166	1	
Monócitos	135	1	128	
Neutrófilos	45	2	10	
Plaquetas	0	0	0	

	Neutrófilos	Plaquetas
Basófilos	1	0
Eosinófilos	13	0
Eritroblastos	20	13
Granulócitos Imaturos	27	1
Linfócitos	7	0
Monócitos	6	0
Neutrófilos	560	2
Plaquetas	3	462

A segunda rede neural foi construída de maneira similar a primeira, com os neurônios aumentado de 1 até 100, a passos de 10. Por questão de simplicidade os mesmos números de neurônios foram utilizados em cada iteração. Um novo gráfico de acurácia por número de neurônios foi, então, gerado. É interessante pontuar que para a segunda rede foi possível obter uma melhor acurácia com menos neurônios por camada em comparação à primeira.

[24]:
```
#Treinamento e Validação

Num_Neuronios = np.arange(1, 100, 10)
Acuracias = []
```

```
for Numero in Num_Neuronios:

    Neuronio_2 = MLPClassifier(hidden_layer_sizes = (Numero, Numero),
                        max_iter = 50,
                        tol = 0.01,
                        learning_rate_init = .001,
                        solver = "adam",
                        activation = "relu",
                        learning_rate = "constant",
                       )

    Neuronio_2.fit(Train_Images, Train_Labels)

    Pred_2 = Neuronio_2.predict(Val_Images)

    Acuracias.append(accuracy_score(Val_Labels, Pred_2))


fig, ax = plt.subplots()
ax.scatter(Num_Neuronios, Acuracias)
ax.set_xlabel('Número de Neurônios')
ax.set_ylabel('Acurácia')
plt.locator_params ('x', nbins = len(Num_Neuronios))
plt.show()
```



Para a etapa de teste da segunda rede neural foram utilizados 91 neurônios na primeira e na segunda camada, que se mostrou o valor com a melhor acurácia obtida na etapa de validação.

[27]:
```
#Teste
Neuronio_2 = MLPClassifier(hidden_layer_sizes = (91, 91),
                           max_iter = 50,
                           tol = 0.01,
                           learning_rate_init = .001,
                           solver = "adam",
                           activation = "relu",
                           learning_rate = "constant",
                          )

Neuronio_2.fit(Train_Images, Train_Labels)

Pred_2 = Neuronio_2.predict(Test_Images)

#Acurácia
print("Acurácia MLP 2 camadas:", accuracy_score(Test_Labels, Pred_2))

#Matriz de Confusão
```

```
mc_2 = confusion_matrix(Test_Labels, Pred_2, labels=[0, 1, 2, 3, 4, 5, 6, 7])
Formata_Matriz(mc_2)
```

```
Acurácia MLP 2 camadas: 0.8427360420929553
```

[27]:

	Basófilos	Eosinófilos	Eritroblastos	\
Basófilos	138	1	0	
Eosinófilos	1	592	1	
Eritroblastos	3	1	249	
Granulócitos Imaturos	25	15	4	
Linfócitos	4	0	9	
Monócitos	5	0	1	
Neutrófilos	0	12	11	
Plaquetas	0	0	1	

	Granulócitos Imaturos	Linfócitos	Monócitos	\
Basófilos	80	11	13	
Eosinófilos	9	1	2	
Eritroblastos	15	12	3	
Granulócitos Imaturos	442	15	36	
Linfócitos	26	199	2	
Monócitos	89	3	184	
Neutrófilos	23	2	2	
Plaquetas	0	0	0	

	Neutrófilos	Plaquetas
Basófilos	1	0
Eosinófilos	17	1
Eritroblastos	9	19
Granulócitos Imaturos	42	0
Linfócitos	3	0
Monócitos	2	0
Neutrófilos	610	6
Plaquetas	0	469

Houve um aumento de aproximadamente **5%** na acurácia na segunda rede neural em relação a primeira. Isso ocorreu pois com um maior número de parâmetros disponíveis para treinamento a rede adquiriu uma maior flexibilidade para a aproximação. Observando as matrizes de confusão, fica evidente que houve um aumento no acerto em relação a classe Monócitos: de 128 na primeira para 184 na segunda.

Obs: A função Formata_Matriz foi utilizada para formatar a matriz de confusão com o nome das células dos bancos de imagem.

[19]:
```
def Formata_Matriz(mc):

    Colunas = {"Basófilos": mc[:, 0],
    "Eosinófilos": mc[:, 1] ,
```

```
    "Eritroblastos": mc[:, 2],
    "Granulócitos Imaturos": mc[:, 3],
    "Linfócitos": mc[:, 4],
    "Monócitos": mc[:, 5],
    "Neutrófilos": mc[:, 6],
    "Plaquetas": mc[:, 7]}
    Indice = ["Basófilos", "Eosinófilos", "Eritroblastos", "Granulócitos␣
↪Imaturos", "Linfócitos", "Monócitos", "Neutrófilos", "Plaquetas"]
    mc_df = pd.DataFrame(Colunas, index = Indice)

    return mc_df
```

CNN

November 4, 2022

Novamente uma biblioteca auxiliar foi utilizada devido a complexidade necessária para a implementação de algumas rotinas que englobam a rede neural convolucional. Para os itens b), c) e d), foi utilizado o TensorFlow juntamente com o Keras.

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, AveragePooling2D, MaxPooling2D,␣
↪Dense, Flatten
from tensorflow.keras.optimizers import Adam

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

np.set_printoptions(precision = 4)
```

Os conjuntos de treino, validação e teste foram carregados e normalizados. Em contraste com as redes MLP, não foi preciso em um primeiro momento redimensionar as imagens para usá-las como entrada para a rede convolucional, devido à maneira como os dados são tratados nessas redes.

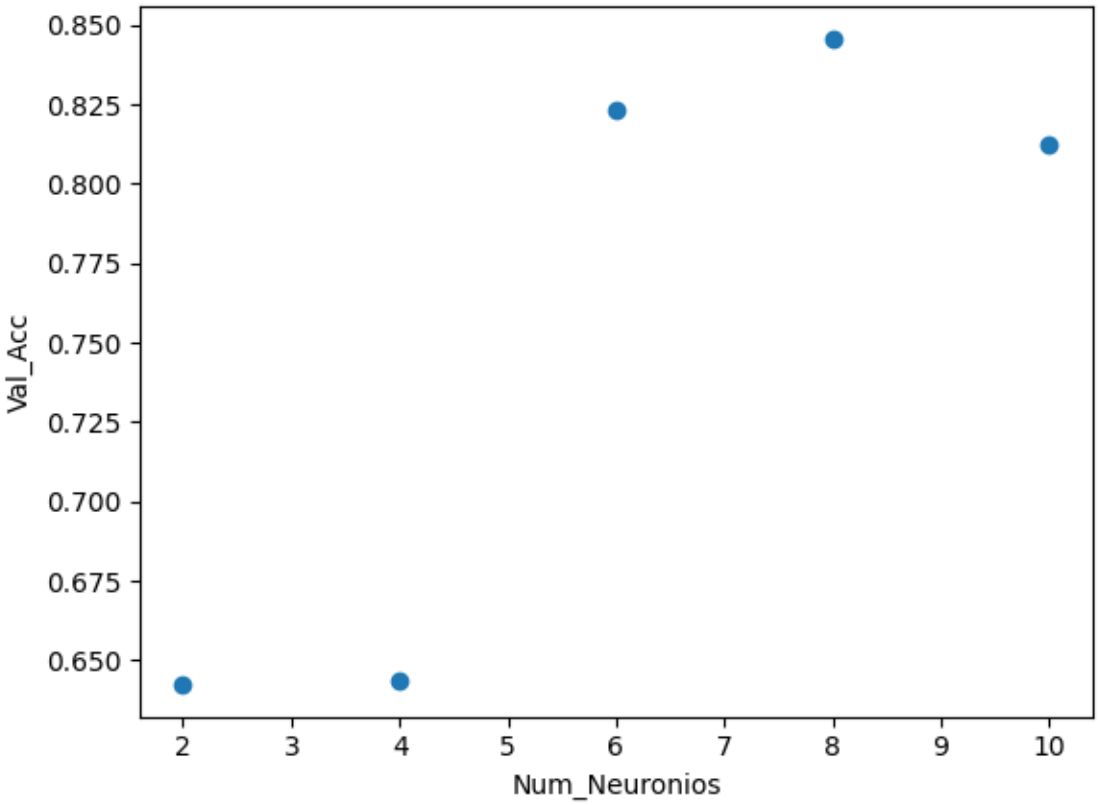
[13]:
```
Train_Images = np.load('train_images.npy')/255
Train_Labels = np.load('train_labels.npy')

Val_Images = np.load('val_images.npy')/255
Val_Labels = np.load('val_labels.npy')

Test_Images = np.load('test_images.npy')/255
Test_Labels = np.load('test_labels.npy')

One_Hot_Train = One_Hot_Encoding(Train_Labels)
One_Hot_Val = One_Hot_Encoding(Val_Labels)
```

Para manter a coerência com o item a) os métodos de treinamento e a função de ativação utilizadas foram os mesmos. A primeira camada consiste em um camada convolucional com kernels de tamanho fixo 3x3. Essa escolha foi feita tendo em vista as dimensões de entrada da imagem 28x28, e a dimensão resultante após o processo de convolução. Apesar da imagem possuir 3 canais

de cor, uma camada de convolução 2d foi utilizada. Isto provoca, na verdade, a geração de 3 feature maps, um proveniente de cada canal de entrada após sua convolução com o mesmo kernel 3x3. A segunda camada é uma camada de Pooling, do tipo Max Pooling, de dimensão 5x5. Melhores resultados foram obtidos com essa configuração em comparação ao Average Pooling e outras dimensões. Em seguida há uma camada Flatten, que serve para empilhar os dados em um vetor para em seguida aplicar uma camada densa do tipo softmax, que gera as probabilidades de um dado de entrada pertencer a uma determinada classe. O número de kernels da primeira camada foi variado e o gráfico de acurácia obtida na etapa de validação por número de kernels foi levantado. Foram utilizadas apenas 10 épocas de treinamento em cada iteração para o custo computacional não ficar muito elevado. Observa-se que 6 kernels bastam para garantir a flexibilidade necessária para a rede alcançar níveis de acurácia semelhantes aos obtidos com a MLP, isto é, em torno de 80%.

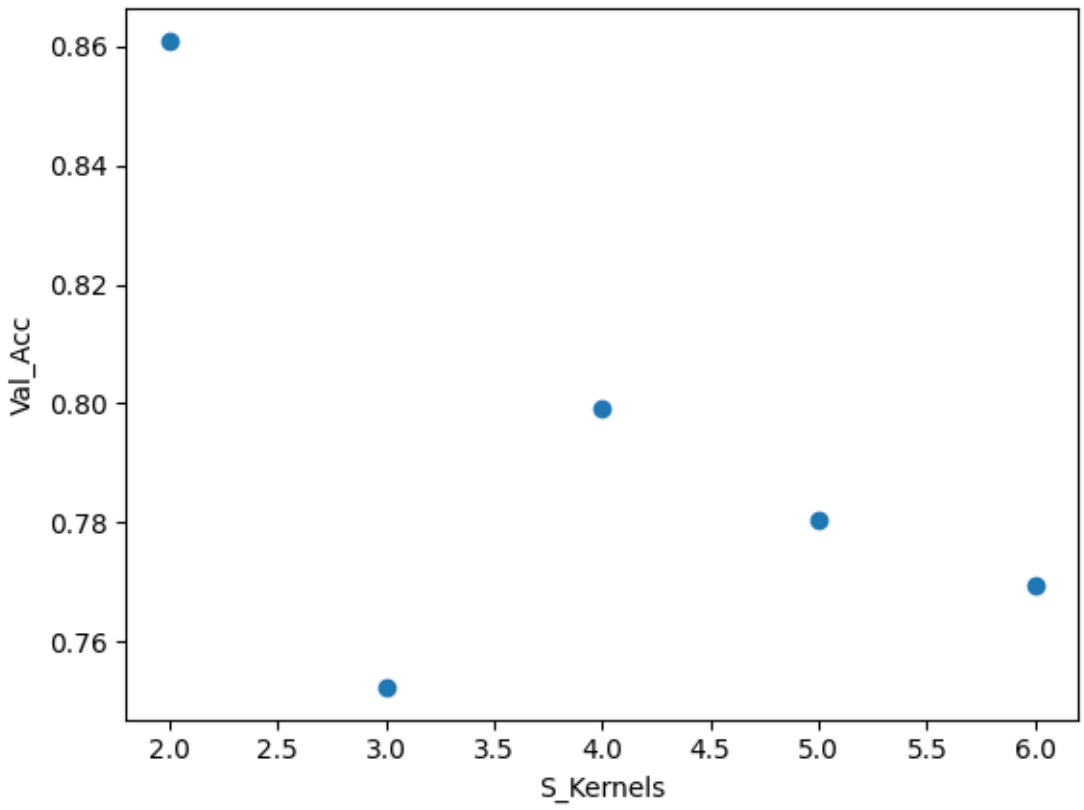
[15]:
```
# CNN Simples:
# 1. Camada convolucional com função de ativação
# 2. Camada de pooling
# 3. Camada de saída softmax
# Analisar Acurácia x N_Kernels e Acurácia x Tam_Kernels


N_Kernels = np.arange(2, 11, 2)
Val_Acc = []

for N_Kernel in N_Kernels:

    #print(N_Kernel)

    CNN = Sequential()


    CNN.add(Conv2D(filters = N_Kernel,
                   kernel_size = (3, 3),
                   activation ='relu',
                   input_shape = (28,28,3)))

    CNN.add(MaxPooling2D(pool_size=(5, 5)))

    CNN.add(Flatten())

    CNN.add(Dense(8, activation='softmax'))


    CNN.compile(loss = 'categorical_crossentropy',
                optimizer = Adam(learning_rate = 1e-2),
                metrics = ['accuracy'])

    history = CNN.fit(Train_Images, One_Hot_Train,
            batch_size = 100,
```

```
            epochs = 10,
            verbose = 0,
            validation_data=(Val_Images, One_Hot_Val))

    Val_Acc.append(max(history.history['val_accuracy']))

Melhor_N_Kernel = np.argmax(Val_Acc)*2 + 2

fig, ax = plt.subplots()
ax.scatter(N_Kernels, Val_Acc)
ax.set_xlabel('Num_Neuronios')
ax.set_ylabel('Val_Acc')
plt.show()
```



Com posse do melhor número de Kernels, a mesma CNN foi montada novamente, mas desta vez variou-se a dimensão dos neurônios. Pelo gráfico gerado, é possível perceber que há uma tendência de queda da acurácia a medida em que a dimensão dos Kernels aumenta. Isso ocorre provavelmente por conta da progressiva redução da dimensão do featuremap resultante da convolução ao se aumentar a dimensão do kernel, o que pode acarretar em menos parâmetros para serem ajustados.

[17]:
```
S_Kernels = np.arange(2, 7, 1)
Val_Acc_2 = []

for S_Kernel in S_Kernels:

    #print(N_Kernel)

    CNN = Sequential()


    CNN.add(Conv2D(filters = Melhor_N_Kernel,
                   kernel_size = (S_Kernel, S_Kernel),
                   activation ='relu',
                   input_shape = (28,28,3)))

    CNN.add(MaxPooling2D(pool_size=(5, 5)))

    CNN.add(Flatten())

    CNN.add(Dense(len(np.unique(Train_Labels)), activation='softmax'))


    CNN.compile(loss = 'categorical_crossentropy',
                optimizer = Adam(learning_rate = 1e-2),
                metrics = ['accuracy'])

    history = CNN.fit(Train_Images, One_Hot_Train,
            batch_size = 100,
            epochs = 10,
            verbose = 0,
            validation_data=(Val_Images, One_Hot_Val))

    Val_Acc_2.append(max(history.history['val_accuracy']))




Melhor_S_Kernel = np.argmax(Val_Acc_2) + 2

fig, bx = plt.subplots()
bx.scatter(S_Kernels, Val_Acc_2)
bx.set_xlabel('S_Kernels')
bx.set_ylabel('Val_Acc')
plt.show()
```



Com posse do melhor número de Kernels e da sua melhor dimensão, a rede foi treinada novamente e o conjunto de teste foi nela aplicado. Obteve-se uma acurácia de 84.4%, um nível semelhante ao obtido no item a) com a MLP de duas camadas intermediárias. Contudo uma das vantagens da rede convolucional tende a ser proporcionar os mesmos resultados com um menor número de parâmetros para o ajuste. A matriz de confusão foi levantada e percebe-se que há uma prevalência da rede em classificar os dados na classe Glanulócitos Imaturos. Isto é evidenciado também ao se escolher 6 amostras classificadas erroneamente. No caso metade foi classificada na classe 3 - Glanulócitos Imaturos.

[32]:
```
CNN = Sequential()

CNN.add(Conv2D(filters = Melhor_N_Kernel,
               kernel_size = (Melhor_S_Kernel, Melhor_S_Kernel),
               activation ='relu',
               input_shape = (28,28,3)))

CNN.add(MaxPooling2D(pool_size=(5, 5)))

CNN.add(Flatten())

CNN.add(Dense(8, activation='softmax'))
```

```
CNN.compile(loss = 'categorical_crossentropy',
            optimizer = Adam(learning_rate = 1e-2),
            metrics = ['accuracy'])

CNN.fit(Train_Images, One_Hot_Train,
        batch_size = 100,
        epochs = 10,
        verbose = 0,
        validation_data=(Val_Images, One_Hot_Val))

Preds = []
Soft_Preds = CNN.predict(Test_Images, verbose = 0)
for Pred in Soft_Preds:
    Preds.append(np.argmax(Pred))

print("Acurácia:", accuracy_score(Preds, Test_Labels))

CM = confusion_matrix(Test_Labels, Preds, labels=[0, 1, 2, 3, 4, 5, 6, 7])
Formata_Matriz(CM)
```

```
Acurácia: 0.8441976030400468
```

[32]:

	Basófilos	Eosinófilos	Eritroblastos	\
Basófilos	203	2	0	
Eosinófilos	16	576	0	
Eritroblastos	5	2	220	
Granulócitos Imaturos	50	13	4	
Linfócitos	9	0	2	
Monócitos	12	1	1	
Neutrófilos	2	6	3	
Plaquetas	0	0	6	

	Granulócitos Imaturos	Linfócitos	Monócitos	\
Basófilos	27	6	4	
Eosinófilos	18	2	0	
Eritroblastos	30	25	1	
Granulócitos Imaturos	448	21	25	
Linfócitos	27	199	0	
Monócitos	103	1	164	
Neutrófilos	32	6	2	
Plaquetas	0	0	0	

	Neutrófilos	Plaquetas
Basófilos	2	0
Eosinófilos	12	0

Eritroblastos	8	20
Granulócitos Imaturos	18	0
Linfócitos	6	0
Monócitos	2	0
Neutrófilos	615	0
Plaquetas	1	463

[47]:
```
i = 0
for Pred, Test_Label, Soft_Pred in zip(Preds, Test_Labels, Soft_Preds):
    if Pred != Test_Label and i <= 5:
        print("Classe Esperada:", Test_Label[0])
        print("Classe Predita:", Pred)
        print("Probabilidades", np.array(Soft_Pred))
        print("\n\n")
        i += 1
```

```
Classe Esperada: 3
Classe Predita: 5
Probabilidades [6.7480e-02 5.4590e-06 6.3741e-04 2.2104e-01 2.1432e-04
7.1006e-01
 5.5509e-04 5.0912e-13]



Classe Esperada: 1
Classe Predita: 6
Probabilidades [2.4626e-01 2.1591e-01 3.0611e-04 1.6388e-01 3.4664e-03
4.7849e-02
 3.2232e-01 3.5655e-09]



Classe Esperada: 1
Classe Predita: 3
Probabilidades [4.2439e-02 1.9298e-01 2.2259e-03 6.2998e-01 4.0712e-02
3.1324e-02
 6.0341e-02 3.5287e-08]



Classe Esperada: 1
Classe Predita: 3
Probabilidades [1.9101e-01 1.8124e-01 7.8962e-03 4.0191e-01 1.0294e-01
9.8021e-03
 1.0519e-01 8.1468e-06]
```

```
Classe Esperada: 2
Classe Predita: 3
Probabilidades [5.3945e-02 3.7585e-04 7.0155e-02 6.1988e-01 2.1619e-01
3.4829e-02
 4.6234e-03 1.0918e-07]



Classe Esperada: 4
Classe Predita: 3
Probabilidades [3.2263e-02 1.6306e-04 3.1352e-01 3.3845e-01 3.0532e-01
4.5917e-03
 5.6964e-03 5.6126e-06]
```

[30]:
```python
def Formata_Matriz(mc):

    Colunas = {"Basófilos": mc[:, 0],
    "Eosinófilos": mc[:, 1] ,
    "Eritroblastos": mc[:, 2],
    "Granulócitos Imaturos": mc[:, 3],
    "Linfócitos": mc[:, 4],
    "Monócitos": mc[:, 5],
    "Neutrófilos": mc[:, 6],
    "Plaquetas": mc[:, 7]}
    Indice = ["Basófilos", "Eosinófilos", "Eritroblastos", "Granulócitos␣
 ↪Imaturos", "Linfócitos", "Monócitos", "Neutrófilos", "Plaquetas"]
    mc_df = pd.DataFrame(Colunas, index = Indice)

    return mc_df
```

[12]:
```python
def One_Hot_Encoding(Labels):
    One_Hot_Labels = []
    for Label in Labels:
        One_Hot_Label = np.zeros(8)
        One_Hot_Label[Label] = 1
        One_Hot_Labels.append(One_Hot_Label)
    return np.array(One_Hot_Labels)
```

CNN d)

November 4, 2022

Para o item d) foi proposta uma camada convolucional adicional com uma redução na dimensão do Kernel, a fim de não reduzir tanto a dimensão dos feature maps resultantes da convolução. Além disso, o número de Kernels de cada camada foi aumentado para 15, para que mais feature maps fossem gerados. A camada de Pooling permaneceu sendo do tipo MaxPooling, contudo após a camada de Flatten foi feito um processo de Dropout. O Dropout consiste em atribuir uma probabilidade para que os neurônios sejam desativados a cada passo de treinamento. Isto contribui para que cada neurônio desempenhe um papel útil para a rede por conta própria, diminuindo a dependencia dos seus vizinhos. No caso desta atividade a probabilidade que resultou em um melhor desempenho foi de 0.01. Usando esta configuração a rede neural conseguiu atingir uma acurácia de 90% junto aos dados de teste em 20 épocas de treinamento, representando um ganho de 6% em relação aos itens c) e a).

[6]:
```
CNN = Sequential()

CNN.add(Conv2D(filters = 15,
               kernel_size = (2, 2),
               activation ='relu',
               input_shape = (28,28,3)))

CNN.add(Conv2D(filters = 15,
               kernel_size = (2, 2),
               activation ='relu',
               input_shape = (28,28,3)))


CNN.add(MaxPooling2D(pool_size=(5, 5)))

CNN.add(Flatten())

CNN.add(Dropout(0.01))

CNN.add(Dense(8, activation='softmax'))


CNN.compile(loss = 'categorical_crossentropy',
            optimizer = Adam(learning_rate = 1e-2),
            metrics = ['accuracy'])
```

```
CNN.fit(Train_Images, One_Hot_Train,
        batch_size = 100,
        epochs = 20,
        verbose = 0,
        validation_data=(Val_Images, One_Hot_Val))

Preds = []
Soft_Preds = CNN.predict(Test_Images, verbose = 0)
for Pred in Soft_Preds:
    Preds.append(np.argmax(Pred))

print("Acurácia:", accuracy_score(Preds, Test_Labels))

CM = confusion_matrix(Test_Labels, Preds, labels=[0, 1, 2, 3, 4, 5, 6, 7])
Formata_Matriz(CM)
```

```
Acurácia: 0.9017831043554516
```

[6]:

	Basófilos	Eosinófilos	Eritroblastos	\
Basófilos	215	1	1	
Eosinófilos	10	603	0	
Eritroblastos	4	1	288	
Granulócitos Imaturos	54	5	12	
Linfócitos	2	0	9	
Monócitos	11	0	2	
Neutrófilos	7	4	9	
Plaquetas	0	0	1	

	Granulócitos Imaturos	Linfócitos	Monócitos	\
Basófilos	18	2	7	
Eosinófilos	3	0	2	
Eritroblastos	10	2	1	
Granulócitos Imaturos	445	7	38	
Linfócitos	6	222	4	
Monócitos	44	2	224	
Neutrófilos	22	3	2	
Plaquetas	0	0	0	

	Neutrófilos	Plaquetas
Basófilos	0	0
Eosinófilos	6	0
Eritroblastos	3	2
Granulócitos Imaturos	18	0
Linfócitos	0	0
Monócitos	1	0
Neutrófilos	619	0

```
Plaquetas                       0         469
```

[5]:
```
i = 0
for Pred, Test_Label, Soft_Pred in zip(Preds, Test_Labels, Soft_Preds):
    if Pred != Test_Label and i <= 5:
        print("Classe Esperada:", Test_Label[0])
        print("Classe Predita:", Pred)
        print("Probabilidades", np.array(Soft_Pred))
        print("\n\n")
        i += 1
```

```
Classe Esperada: 1
Classe Predita: 3
Probabilidades [5.9318e-02 1.3811e-02 1.0516e-02 9.0688e-01 3.4214e-04
1.4075e-04
 8.9884e-03 4.2572e-15]



Classe Esperada: 1
Classe Predita: 0
Probabilidades [8.7337e-01 1.2010e-02 6.8291e-03 3.0490e-02 4.8740e-05
3.0909e-04
 7.6940e-02 6.0836e-11]



Classe Esperada: 2
Classe Predita: 3
Probabilidades [1.7396e-02 1.6811e-03 3.1182e-01 3.3009e-01 3.1264e-01
2.5037e-02
 1.3303e-03 1.4767e-09]



Classe Esperada: 3
Classe Predita: 0
Probabilidades [5.9438e-01 3.6710e-01 6.4155e-03 1.1037e-02 8.7633e-09
1.0978e-04
 2.0957e-02 4.0049e-16]



Classe Esperada: 3
Classe Predita: 0
Probabilidades [8.8257e-01 2.2950e-04 1.0444e-03 4.9080e-02 5.5609e-06
6.6325e-02
```

```
 7.4455e-04 1.6487e-15]



Classe Esperada: 5
Classe Predita: 0
Probabilidades [8.8638e-01 2.6673e-05 1.6109e-03 3.4855e-03 2.6557e-02
8.1800e-02
 1.3786e-04 5.2798e-11]
```