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.

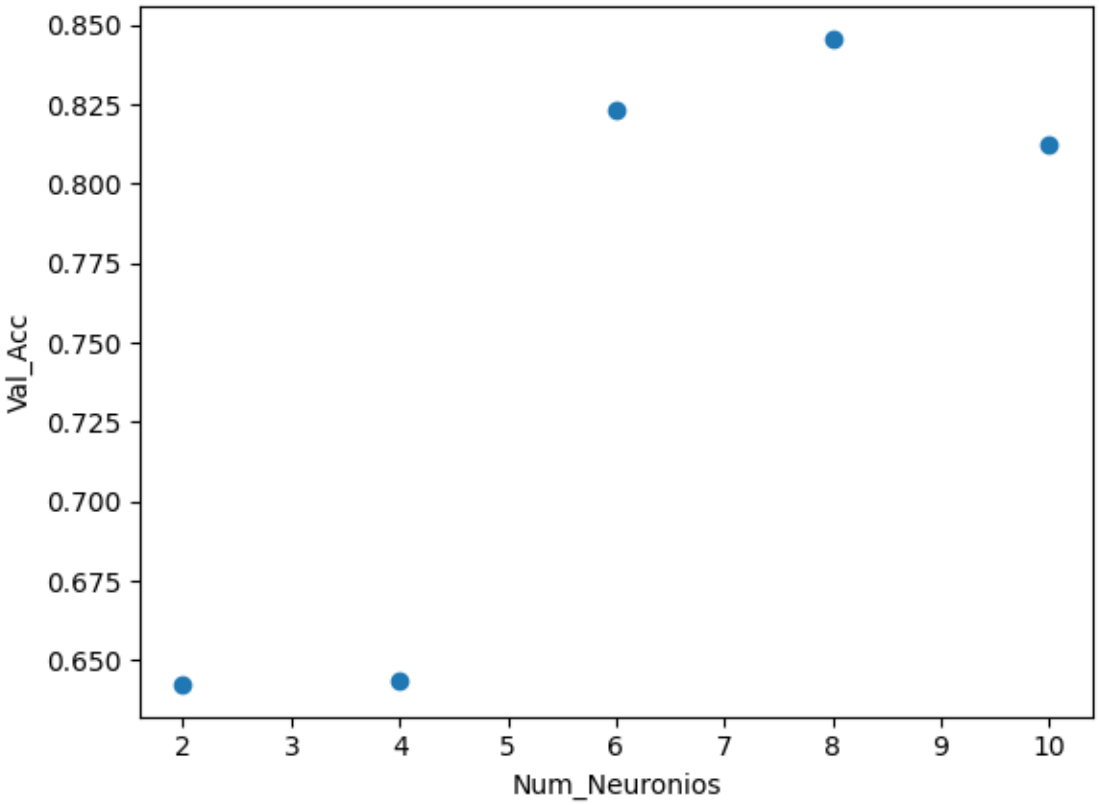
[13]:
```
Train_Images = np.load('train_images.npy')/255
Train_Labels = np.load('train_labels.npy')

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%.

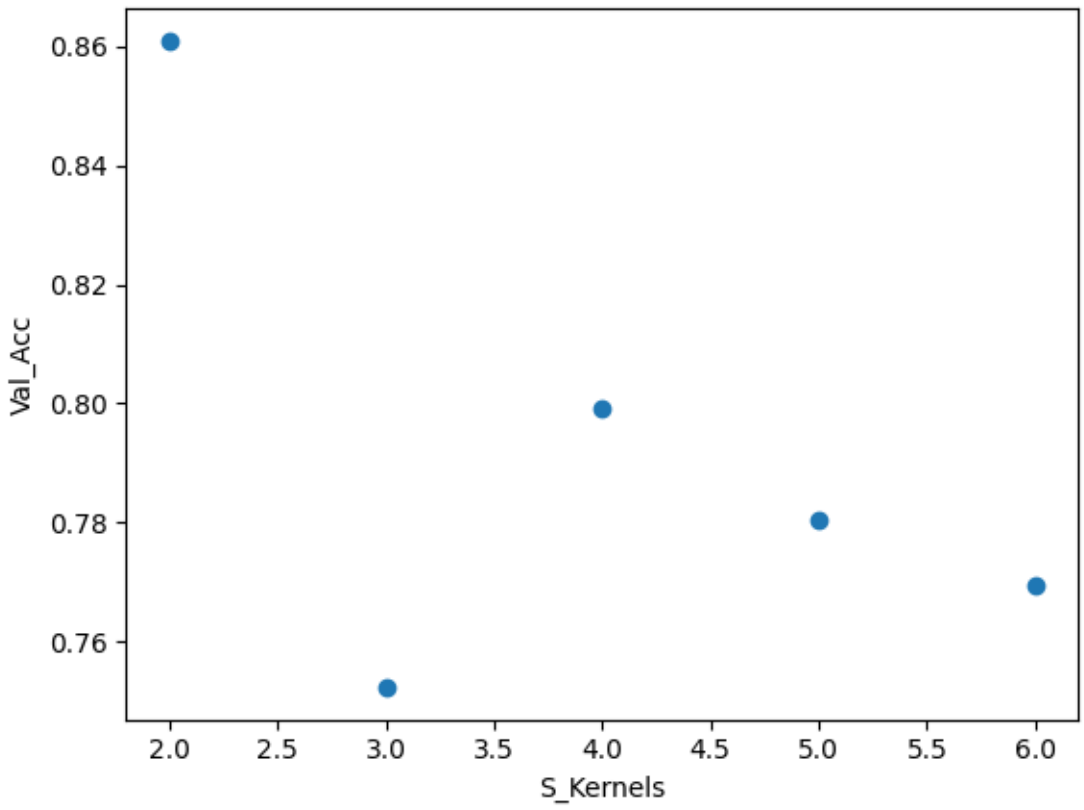
[15]:
```
# CNN Simples:
# 1. Camada convolucional com função de ativação
# 2. Camada de pooling
# 3. Camada de saída softmax
# Analisar Acurácia x N_Kernels e Acurácia x Tam_Kernels


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)
```