Luiz Felipe Nagatani Rabello

Relatório LAMIA - Módulo 15

Seção 5

Convolução

Iniciando a Seção 5, o instrutor mostra algumas formas de se entender a convolução nas CNNs, iniciando com a perspectiva da convolução nas imagens, a que a convolução é um modificador que pode ser aplicado a uma imagem, sendo possível mudar a operação executada mudando o operador convolucional.

A convolução é realizada em uma imagem ao transformar a imagem em uma matriz, e multiplicar repartições da matriz pela matriz utilizada como operador convolucional.

É explicado também o padding, prática para manter o mesmo tamanho de matriz na entrada e na saída da convolução, para isto é adicionada uma matriz imaginária nula ao redor da matriz de entrada, permitindo que o filtro da convolução acesse os valores da matriz sem que altere as dimensões da mesma.

Foi apresentado o Full padding, onde se aumenta o tamanho da matriz imaginária para deixar a matriz de saída maior que a matriz de entrada, porém esta prática não é comum atualmente.

O instrutor aborda alguns conceitos mostrados no módulo 14, porém de forma mais profunda, como a diferença entre imagens RGBs e imagens preto e branco, onde as imagens RGBs tem 3 canais por conta das cores, enquanto as outras possuem apenas um canal.

Pooling

Explica o pooling de forma mais detalhada, mostrando o pooling como uma forma de reduzir o tamanho da imagem, um poolsize de 2, divide o tamanho por 2, e um pool size de tamanho 4 divide o tamanho da imagem por 4.

O pooling pode ser dividido entre max e average pooling, onde o max pooling utiliza o valor máximo da partição da matriz e o passa para a matriz de saída, enquanto o average utiliza a média aritmética da repartição para a saída.

Reduzir o tamanho da imagem utilizando o pooling é benéfico por reduzir a quantidade de data na entrada, permitindo uma melhor execução da CNN.

O pooling é aplicado após a convolução para remediar o aumento da dimensionalidade causada pela convolução.

Aplicação

O instrutor mostra como aplicar os conhecimentos passados na seção 5, começando pelo carregamento do dataset Fashion MNIST, um dataset com diversos tipos de roupas e calçados, no módulo 14 foi abordado o dataset MNIST, o que foi desconsiderado no módulo atual por ser muito fácil e comum em treinamentos de CNNs, porém o fashion MNIST ainda é composto por imagens preto e branco 28x28, assim como o MNIST normal.

Outro dataset apresentado é o CIFAR-10, onde se tem várias fotos de animais, porém são fotos 32x32 com canais RGBs, o que o transforma em 32x32x3.

MNIST Fashion

Para a CNN do MNIST fashion, o instrutor optou por utilizar a API funcional do keras ao invés da sequencial, com 3 convoluções em 3x3, dividindo as dimensões da matriz por 2 a cada camada.

Por opção também tentei realizar o fit do modelo apresentado utilizando sequential em vez do método apresentado no curso, o que se provou muito mais lento, e de acordo com o mencionado pelo instrutor, se provou muito mais restrito de se configurar.

A execução do fit sequencial levou 5 minutos no Colab, enquanto o da API com 2 minutos de execução estava finalizado

Para fins de experimentação, foi criado como código original um bloco onde se obtém uma entrada de um número inteiro N, realizando predições do índice N do dataset, incrementando até achar uma combinação para superior, calça e calçado.

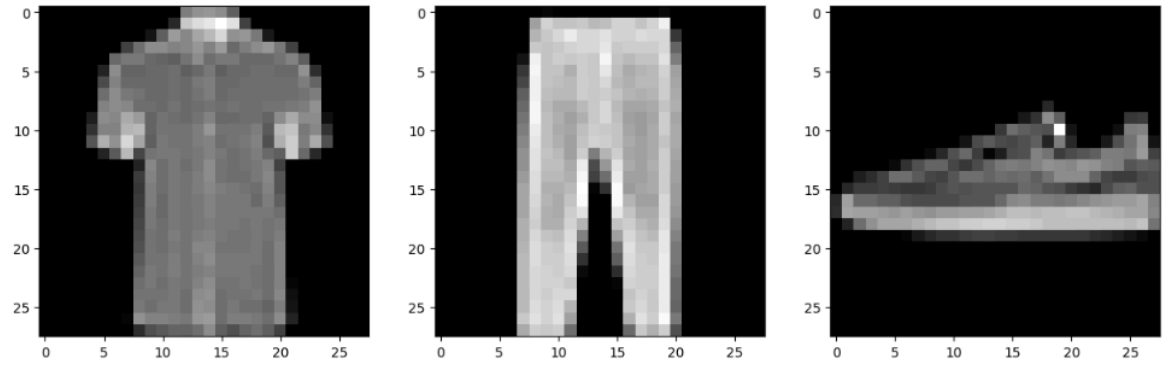
```
entrada=int(input())
cima = False
baixo = False
pe = False
while cima == False or baixo == False or pe == False:
    img = X_test[entrada]
    prev = model.predict(img.reshape(1,28,28,1))
    prev_roupas = np.argmax(prev)
    if roupas[prev_roupas] in Superior and cima == False:
      cima = True


      img1 = (array_to_img(img))
    elif roupas[prev_roupas] in Inferior and baixo == False:
      baixo = True

      img2 =(array_to_img(img))
    elif roupas[prev_roupas] in Calçado and pe == False:
      pe = True

      img3= (array_to_img(img))
    entrada += 1
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
axs[0].imshow(img1, cmap='gray')
axs[1].imshow(img2, cmap='gray')
axs[2].imshow(img3, cmap='gray')
```

```
579
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 23ms/step
<matplotlib.image.AxesImage at 0x7b70ddc7bdc0>
```



exemplo de saída para a entrada “579”

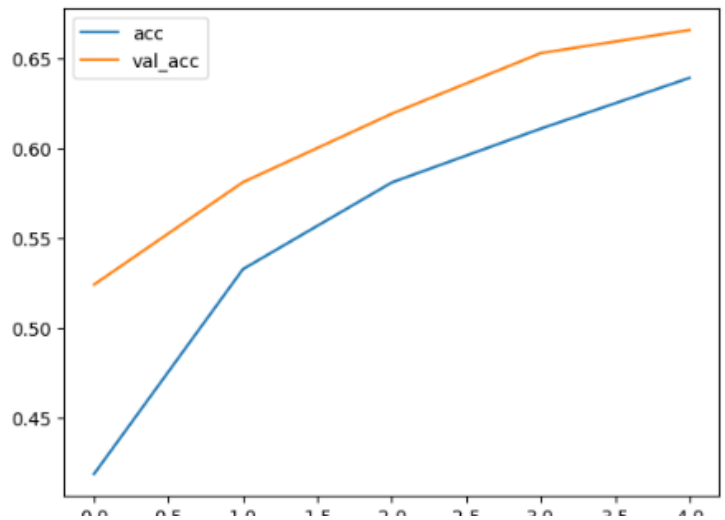
CIFAR-10

A arquitetura da rede apresentada pelo instrutor é semelhante a do exemplo passado, isto se dá pois o input recebe o formato de x_train[0], portanto mesmo recebendo 3 canais de cores ao invés de 1, não é necessário mudar o jeito de escrever a estrutura.

Utilizando a mesma rede mostrada no vídeo os resultados foram bem melhores, não ocorrendo erros de treino na precisão. Porém o último vídeo da seção 5 é sobre tuning deste modelo, então será abordado mais tarde

```
plt.plot(r.history['accuracy'], label='acc')
plt.plot(r.history['val_accuracy'], label='val_acc')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7d93fea966e0>
```



Augmentation

O instrutor explica o conceito de data augmentation abordado no card 14 de forma mais profunda, explicando como criar dados novos sem passar pelo processo de geração ou scraping de dados. Para fazer data augmentation basta deslocar a imagem, redimensioná-la ou distorcê-la, visto que o modelo deve procurar as características isoladas da imagem, não o contexto inteiro.

Melhoria

Para melhorar o CIFAR-10 o instrutor adicionou camadas de convolução antes de cada pooling para melhorar a precisão, sendo capaz de identificar e combinar características de forma mais eficiente.

Seção 6

Processamento de linguagem natural

Para trabalhar com linguagem natural e categorizá-la é necessário o embedding, pois caso fosse utilizada uma abordagem já apresentada antes como o onehot encoding, seria necessário um vetor binário muito extenso, por conta da quantidade de palavras existentes. No embedding se transforma as palavras em números e depois posicionando-as em vetores, mantendo palavras semelhantes como canil e canico em distâncias euclidianas menores utilizando o gradiente ao realizar o fit do modelo.

O instrutor utilizou o tokenizer do keras para receber as strings e criar índices para cada palavra.

Seção 7

Convolução aprofundada

Nesta seção o instrutor apresenta como a convolução funciona, sem associá-la a redes neurais utilizando exemplos como efeitos em músicas ou imagens. Portanto na convolução a imagem é transformada de forma que ressalte suas características a fim de torná-las mais perceptíveis para serem utilizadas no treinamento.

Seção 8

Convolução 3D

A convolução 3D é semelhante a convolução 2D, porém resulta numa entrada (X x Y x Z x C) sendo comprimento, altura, profundidade e canais respectivamente.

Seção 9

São apresentadas algumas arquiteturas mais comuns de CNNs, como a LeNet, que são várias camadas de convolução seguidas por camadas densas inteiramente conectadas que recebem uma imagem e retornam um vetor. porém para fins de treinamento não supervisionado, é possível inverter e criar uma rede que receber um vetor e resulta numa imagem.

Seção 10

Loss functions

Nesta seção o instrutor apresenta funções de perda utilizadas nas redes neurais, mostrando a dedução das mesmas e a aplicação de cada uma, tendo como exemplo a Mean squared error, função para realizar regressões, identificando e penalizando grandes erros, forçando o modelo a realizar predições mais precisas.

Em seguida são mostrados o binary e o categorical cross entropy, onde cada um cumpre sua função como função de perda para uma determinada situação, onde o binary é utilizado para classificar entre duas categorias(0 ou 1) e o categorical para classificar por K categorias sendo K>2.

Seção 11

Gradient Descent

O gradient descent é um algoritmo feito para otimizar o aprendizado da máquina, ajustando os parâmetros conforme necessidade e para minimizar funções de custo. Para isto o gradient descent realiza várias iterações, se aproximando do gradiente de uma função até que o valor convirja para o mínimo.

Uma variação do gradient descent é o stochastic gradient descent, onde será realizado o gradient descent em um subgrupo do dataset, para que o mesmo represente o dataset inteiro e forneça um valor aproximado, economizando tempo por não ter que processar todo o dataset.

Momentum

Momentum é uma técnica utilizada em gradient descent para eliminar adversidades encontradas ao se deparar com mínimos locais e erros lineares, para isto cada iteração da GD é efetuada multiplicada por um coeficiente obtido na iteração anterior, a fim de acelerar o processo de forma que a aproximação ainda seja fiel ao resultado real.

Taxa de aprendizado variável

Técnica utilizada para acelerar o processo de treinamento, evitando um processo longo em uma época que resultaria em um aumento insignificante na precisão, a taxa de aprendizado vai se reduzindo ao longo das iterações.

Taxa adaptativa

Existem vários algoritmos que não utilizam uma função fixa para controlar a taxa de aprendizado, um deles é o AdaGrad que utiliza os parâmetros obtidos nos gradientes para modificar a taxa de aprendizado.

Adam

Por último, é apresentado outro algoritmo de taxa adaptativa chamado ‘Adam’, podendo ser resumido em um “RMSprop com momentum”, para isto ele utiliza a taxa de aprendizado dividida pela raiz quadrada dos gradientes

Experimentação

Como teste, tentei aplicar a arquitetura VGG no CIFAR-100, que resultou em uma precisão de aproximadamente 65% obtendo o melhor resultado com 0.2 de dropout, apresentando melhora para 71% com 0.15 de dropout, porém aumentando o percentual de erro, sendo difícil determinar se a mudança era benéfica ou não

```
from keras.preprocessing.image import array_to_img
import matplotlib.pyplot as plt
test=(X_train[3365])

print(np.argmax(model.predict(test.reshape(1,32,32,3)), axis=1))
img=array_to_img(test)
plt.imshow(img)
```

```
1/1 [==============================] - 0s 18ms/step
[42]
<matplotlib.image.AxesImage at 0x7dc63fdd2260>
```