PÓS GRADUAÇÃO EM CIÊNCIA DE DADOS - FUNDAMENTOS DE INTELIGÊNCIA ARTIFICIAL

Trabalho final
Questão 2: Segmentação Semântica

NOMES - MATRÍCULAS:
Silvio Cesar de Santana - 2017103163
Marcel Tinoco Ribeiro - 2020101920
Ruan Braz de Araujo - 2019101232
Luis Claudio Simões Baptista - 2015100108

UNet

É uma rede neural convolucional utilizada em segmentação semantica.
Possui uma arquitetura em forma de U.
Essa arquitetura, geralmente, tem uma boa perforance para problemas de segmentação semantica e é utilizadas em diversas áreas, como mapeamento de locais e na área médica.
O bloco de encoder é responsável pela aplicação de filtros nas camadas de convolução para tratar e redimensionar a imagem a fim de torná-la o mais compacta possível.
O bloco de decoder transforma essa imagem de dimensão reduzida e aplica filtros para atingir a dimensão original. O percorrimento dos pixels na imagem funciona como codigos em uma matriz 3x3.

UNetCompiled

```
def UNetCompiled(input_size=(128, 128, 3), n_filters=32, n_classes=3):

  inputs = Input(input_size)
```


```
conv9 = Conv2D(n_filters,
               3,
               activation='relu',
               padding='same',
               kernel_initializer='he_normal')(ublock9)

conv10 = Conv2D(n_classes, 1, padding='same')(conv9)

# Define the model
model = tf.keras.Model(inputs=inputs, outputs=conv10)
```

Nas camadas encoder

cblock1 - camada de input que recebe os parametros definidos inicialmente, e aplicados os filtros para redução
cblock2 - camada de encoder que recebe o resultado da redução feita na primeira camada e aplicada novo filtro(64) para redução
cblock3 - camada de encoder que recebe o resultado da redução feita na camada anterior e aplicada novo filtro(128) para redução
cblock4 - camada de encoder que recebe o resultado da redução feita na camada anterior e aplicada novo filtro(256) para redução
cblock5 - camada de encoder que recebe o resultado da redução feita na camada anterior e aplicada novo filtro(512) para redução.
Essa é a ultima camada de encoder, e será referencia para a primeira camada de decoder.
Cada camada se conecta com a anterior e o valor do filtro vai aumentando, para reduzir dimensão da imagem.
Max_pooling 2x2 reduz em aproximadamente 75%

Na camada decoder

ublock6 - camada de decoder que recebe o resultado da ultima camada de encoder e aplicada novo filtro(256) para aumento.
ublock7 - recebe o resultado da camada anterior e aplicada novo filtro(128) para aumento.
ublock8 - recebe o resultado da camada anterior e aplicada novo filtro(64) para aumento.
ublock9 - recebe o resultado da camada anterior e aplicada novo filtro(32) para aumento.
Essa é a ultima camada, com aplicação dos filtros, tem a dimensão original da imagem.
Cada camada se conecta com a anterior e o valor do filtro vai diminuindo, para aumentar dimensão da imagem, até a original.

Outras camadas

conv9 - camada convolucional com o valor dos filtros iniciais com referencia da ultima camada decoder, com parametro de ativação.
kernel=3 (3 linhas e 3 colunas)
padding=same (percorre todos os pixels)
conv10 - camada de convolução, com referencia da camada anterior
n_classes=3(padrao RGB)

Criação do modelo, com a camada de entrada e de saída

Detalhamento de funções

load_tiff_image

TIFF ou TIF, Tagged Image File Format. Ele é capaz de descrever dados de imagem de dois níveis,
tons de cinza, cores de paleta e cores em vários espaços de cores. Ele suporta esquemas de compactação
com e sem perdas para escolher entre espaço e tempo para aplicativos que usam o formato. O formato não
depende da máquina e está livre de limites como processador, sistema operacional ou sistemas de arquivos.
O arquivo começa com um cabeçalho de arquivo de imagem de 8 bytes que aponta diretamente para um
arquivo de imagem (IFD). Um IFD contém informações sobre a imagem, bem como ponteiros para os
dados reais da imagem

```
def load_tiff_image (image):
  '''try:
    im = misc.imread(ref)
    return im
  except:
    print ("Can't open image. Check image again:" + ref)'''
  print (image)
  gdal_header = gdal.Open(image)
  img = gdal_header.ReadAsArray()
  return img
```

LoadData

Carrega as imagens dos arquivos indicados nos caminhos salvos em train_images e test_images
Pasta de treino
Uma imagem original .tif, resolução 2565 altura x 1919 largura 3 cores(RGB)

e uma imagem de mascara correspondente a original resolução 2565 altura x 1919 largura 5 cores

Pasta de teste
Uma imagem original .tif, resolução 2558 altura x 2818 largura 3 cores(RGB)
e uma imagem de mascara correspondente a original resolução 2558 altura x 2818 largura 5 cores

```python
def LoadData (path1, path2):

    image_dataset = os.listdir(path1)
    mask_dataset = os.listdir(path2)

    orig_img = []
    mask_img = []
    for file in image_dataset:
        orig_img.append(file)
    for file in mask_dataset:
        mask_img.append(file)

    orig_img.sort()
    mask_img.sort()

    return orig_img, mask_img
```

EncoderMiniBlock

Bloco de convolução seguido do max pooling para redução dimensão, codificando e comprimindo a imagem.

DecoderMiniBlock

Bloco de convolução seguido do max pooling para aumento dimensão, decodificando e expandindo a imagem.

ARQUIVOS

input_size recebe a configuração inicial das imagens que é altura(128px) e largura(128px), n_filters a quantidade de filtros(32) que vai ser aplicado para redimensionamento da imagem. n_classes são os padrões das cores (RGB) (3cores).

normalization

É importante para evitar e corrigir problemas de estabilidade dos dados. Redimensiona a imagem, para a visualização.

weighted_categorical_crossentropy

Define uma função de perda ponderada para lidar com desequilíbrios de classes.

Test

Função para testar o modelo na imagem de teste.

compute_metrics

Calcula métricas como acurácia e pontuações F1.

getWeights

Pesos

Padrões das cores
>white,blue,ciano,green,yellow

+ Código + Texto

```
def getWeights(path):

  img_I = Image.open(path).getdata()

  total = 0
  white=0
  blue=0
  ciano = 0
  green=0
  yellow=0
```

Cada pixel tem um codigo de cor(RGB), para cada imagem sáo percorridos os pixels e verificada a cor(que são iniciadas =0), e incrementado 1 de acordo com a cor encontrada.

```
  for pixel in list(img_I):
    if pixel == (255,255,255):
        white += 1
    if pixel == (0,255,255):
        ciano += 1
    if pixel == (0,0,255):
        blue += 1
    if pixel == (0,255,0):
        green += 1
    if pixel == (255,255,0):
        yellow += 1

  total = img_I.size[0]*img_I.size[1]
  #print(total)
  return [(white/total),(blue/total),(ciano/total),(green/total),(yellow/total)]
```

PlotPred

Exibe as imagens de acordo com a referencia do treinamento e se a imagem é a mesma da previsão.

Train_model

Iniciando o treinamento

patience = 10, significa que, se até 10 epochs o valor loss não melhore o treinamento é interrompido
best são os melhores valores de loss e accuracy

```
def Train_model(net, patches_train, patches_tr_lb_h, patches_val, patches_val_lb_h, ba
  print('Start training.. ')
  patience = 10
  wait = 0
  best = [[0,0]]
```

Inicia os valores de perda com peso em (0)zero

```
 for epoch in range(epochs):
   loss_tr = np.zeros((1 , 2))
   loss_val = np.zeros((1 , 2))
```

Treinando a rede

Obtendo os valores de perda e acuracia

Salvando os valores de loss e acuracia do modelo

```
# Training the network per batch
for  batch in range(n_batchs_tr):
  x_train_b = patches_train[batch * batch_size : (batch + 1) * batch_size , : , : , :]
  y_train_h_b = patches_tr_lb_h[batch * batch_size : (batch + 1) * batch_size , :, :, :]
  loss_tr = loss_tr + net.train_on_batch(x_train_b , y_train_h_b)

# Training loss
loss_tr = loss_tr/n_batchs_tr
print("%d [Training loss: %f , Train acc.: %.2f%%]" %(epoch , loss_tr[0 , 0], 100*loss_tr[0 , 1]

# saving model
net.save('/content/drive/My Drive/dataset_trabalho/hw2_puc/unet.h5')
```

Calculando a quantidade dos dados para validação

Executando a validação

Calculando e exibindo valores validação perda e acurácia

```
# Computing the number of batchs
n_batchs_val = patches_val.shape[0]//batch_size

# Evaluating the model in the validation set
for  batch in range(n_batchs_val):
  x_val_b = patches_val[batch * batch_size : (batch + 1) * batch_size , : , : , :]
  y_val_h_b = patches_val_lb_h[batch * batch_size : (batch + 1) * batch_size , :, :, :]
  loss_val = loss_val + net.test_on_batch(x_val_b , y_val_h_b)

# validation loss
loss_val = loss_val/n_batchs_val
print("%d [Validation loss: %f , Validation acc.: %.2f%%]" %(epoch , loss_val[0 , 0], 10
```

Como wait inicial, foi declarada como 0(zero), então agora fica valendo 1. O valor da val loss vai sendo calculada e verificada: se for maior do que a melhor obtida, continua executando até obter resultado menor, retornando wait=0 e reiniciando no proximo com valor wait=1, assim incrementando 1 até chegar a 10. Como patience=10, quando wait=10 , a execução é interrompida.

```
42      if early_stop:
43        wait += 1
44        print(loss_val[0])
45        if loss_val[0][1] > best[0][1]:
```

Execução e avaliação:

O modelo é compilado e treinado com um conjunto de dados de imagens e suas respectivas máscaras. As imagens são normalizadas e divididas em patches para eficiência. Após o treinamento, o modelo é avaliado em um conjunto de validação.

Tratamento de imagem para CNN - UNET:

As imagens são normalizadas e transformadas em patches. Isso ajuda a rede a focar em partes específicas da imagem, facilitando a aprendizagem e a segmentação.

Recomendações de ajustes e melhorias:

Aumentar Dados: Usar técnicas como rotação, flip e zoom para aumentar o conjunto de dados e melhorar a generalização.

Ajuste Fino de Hiperparâmetros: Usar diferentes taxas de aprendizado, otimizadores e números de filtros nas camadas convolucionais.

Regularização: Adicionar regularizações como Dropout ou L2 para prevenir overfitting.

Outras técnicas como complemento:

Transferência de Aprendizado: Usar uma rede pré-treinada em um conjunto de dados grande e ajustar para uma tarefa específica.

Data Augmentation: Aumentar a variedade de dados com transformações aleatórias.

Ajuste de parâmetros:

Learning Rate: Podemos experimentar diferentes taxas de aprendizado ou usar um agendador de taxa de aprendizado para ajustar dinamicamente durante o treinamento.

BatchNormalization: Pode ser ajustado ao alterar o momento ou adicionando-o após diferentes camadas.