Segmentação Semântica de Cenas Urbanas com Cityscapes: Uma Análise Comparativa de Modelos de Aprendizagem Profunda

Disciplina: Introdução à Aprendizagem Profunda
Professor: Tsang Ing Ren e George Darmiton

Camila Vieira

Centro de Informática UFPE
Universidade Federal de Pernambuco

INTRODUÇÃO

- Compreensão Visual: O principal desafio da visão computacional
  - Identificação e classificação de objetos, entendimento das relações espaciais, inferência de contexto e a interpretação do cenário.
  - Teste de Turing.
- Complexidade da estrutura de dados
  - Informações detalhadas sobre os elementos presentes.
  - Inferências sobre qualquer aspecto necessário.
- Segmentação Semântica: Um elemento fundamental
  - Atribuir rótulos semânticos a cada pixel da imagem.
  - Detecção e localização de objetos; Reconhecimento de atributos; Conhecimento de contexto; Raciocínio espacial; Rastreamento e detecção de movimento.

INTRODUÇÃO

- Segmentação Semântica: Uma interpretação mais precisa e detalhada do conteúdo visual.
    - Investigar técnicas e modelos para aprimorar o desempenho.
    - Cityscapes, um conjunto de dados abrangente contendo cenas urbanas.
    - Análise comparativa das arquiteturas (U-Net, SegNet, FCN, DeepLab e PSPNet).
- Avanços em aplicações práticas.
    - Veículos autônomos; planejamento urbano inteligente; monitoramento ambiental; análises de segurança; realidade aumentada e virtual.
    - Desenvolvimento de tecnologias mais inteligentes e seguras para as cidades do futuro.

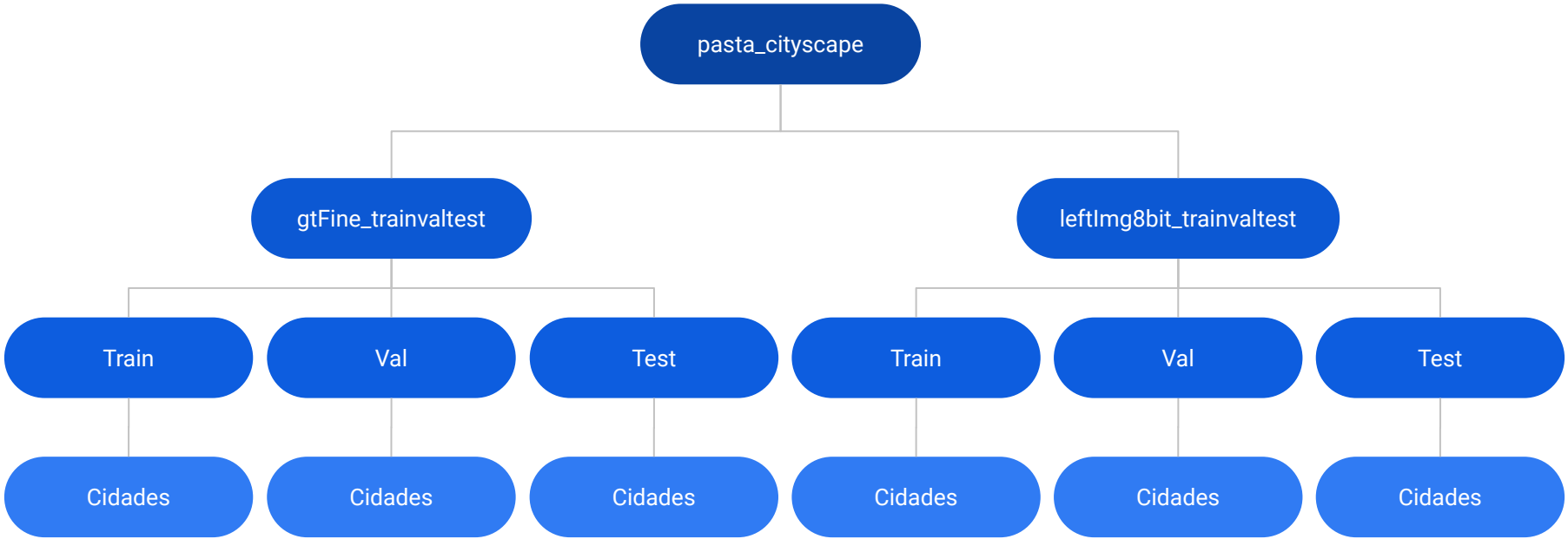
DATASET

- Cityscapes: Criado em 2016 na Alemanha.
  - Conjunto abrangente de cenas urbanas de 50 cidades grandes, ao longo de vários meses, variando condições.
- Dados realistas e desafiadores utilizados em projetos de pesquisa e competições
- Total de 5.000 imagens de alta resolução divididas em treinamento, validação e teste, (divisão representativa e aleatória).
  - Cada imagem é acompanhada de uma máscara de segmentação.
  - Anotações detalhadas e de alta qualidade.
  - Política de rotulamento
- Outras Categorias e Arquivos
- Proibição para uso comercial e são destinados apenas para fins de pesquisa.
- Conjunto realista e diversificado de cenas urbanas para uma análise comparativa sólida.

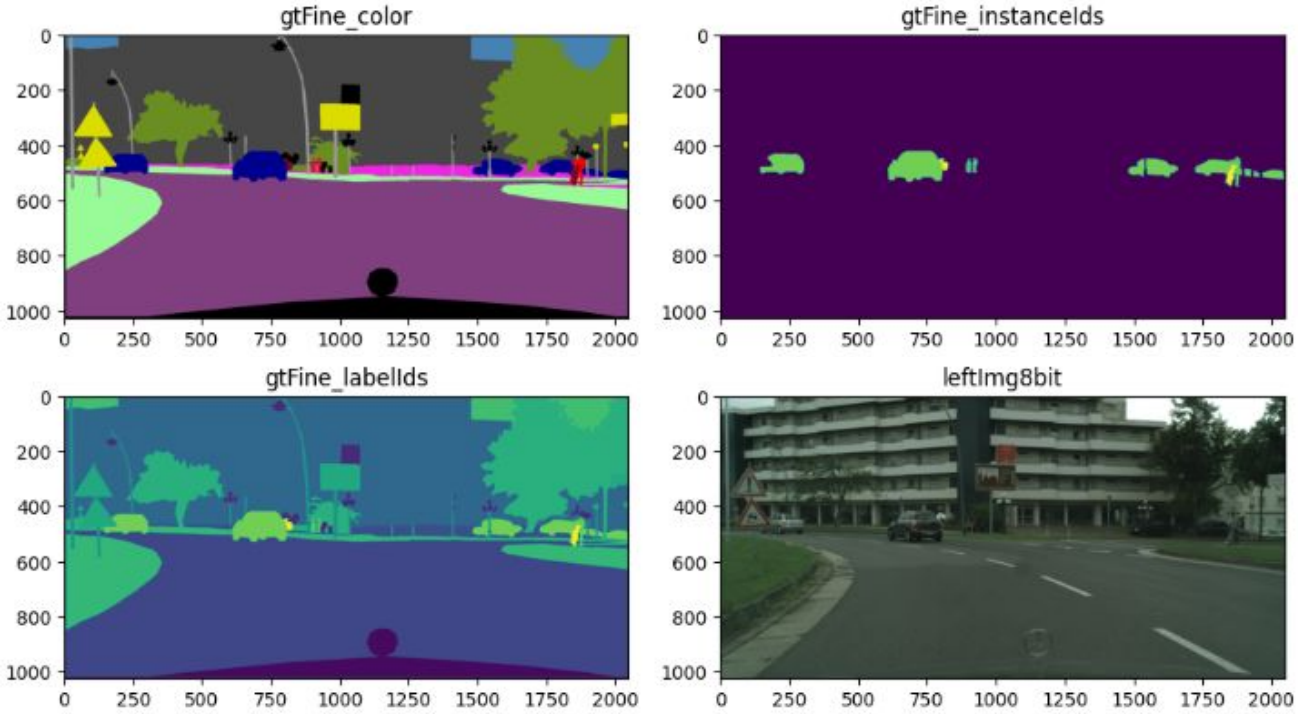
DATASET


Kiel
Rostock
Hamburg
Szczecin
Bremen
Hannover
Berlin
Potsdam
Braunschweig
Leipzig
Dresden
Deutschland
Köln
Frankfurt am Main
Praha
Česká rep
Czech Re
Nürnberg
Mannheim
Stuttgart
Strasbourg
München
Salzburg
Österreich
Basel
Zürich
Innsbruck
Schweiz
Suisse
Liechtens
Map Data ©2018 Google, ORION-ME

EXPLORANDO DIRETÓRIOS



DATASET

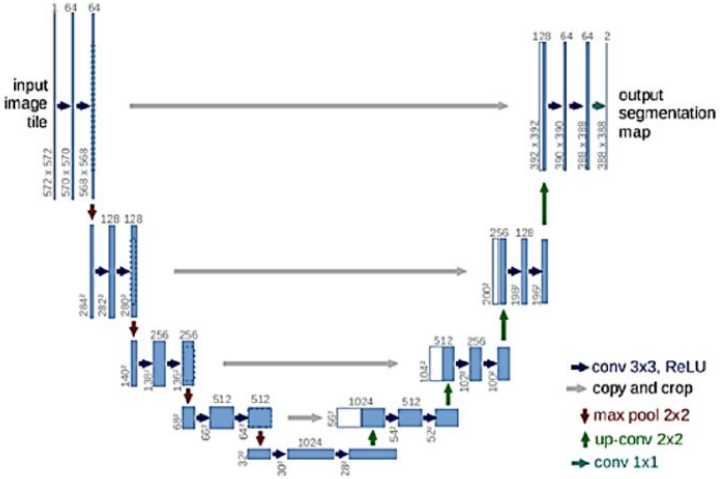

gtFine_color
gtFine_instanceIds
gtFine_labelIds
leftImg8bit
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750
2000

REDES E ARQUITETURAS

- Popularidade e eficácia comprovada em tarefas de segmentação semântica.
- Avaliar o desempenho e a capacidade de generalização de cada modelo.
- Compreender limitações e identificar os pontos fortes de cada arquitetura.
- Foram escolhidas:
    - U-Net
    - SegNet
    - FCN
    - DeepLab
    - PSPNet

U-Net

- Arquitetura encoder-decoder com conexões residuais
- Estrutura em formato de "U"
- Camadas de convolução
- Camadas de pooling (normalmente max-pooling)
- Conexões skip
- Função de ativação (ReLU)
- Função de perda (entropia cruzada)

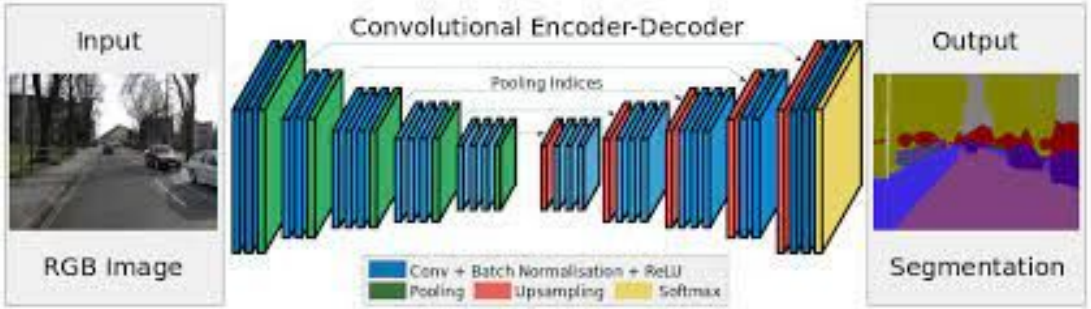


U-Net

```python
def conv_block(x, filters, kernel_size=3):
    x = Conv2D(filters, kernel_size, activation='relu', padding='same')(x)
    x = Conv2D(filters, kernel_size, activation='relu', padding='same')(x)
    return x

def decoder_block(x, skip_connection, filters, kernel_size=3):
    x = UpSampling2D((2, 2))(x)
    x = concatenate([x, skip_connection], axis=-1)
    x = conv_block(x, filters, kernel_size)
    return x

def unet(input_shape, n_classes):
    inputs = Input(input_shape)

    # Encoder
    encoder_blocks = []
    x = inputs
    for filters in [64, 128, 256, 512]:
        x = conv_block(x, filters)
        encoder_blocks.append(x)
        x = MaxPooling2D(pool_size=(2, 2))(x)

    # Bottleneck
    x = conv_block(x, 1024)

    # Decoder
    for filters, skip_connection in zip([512, 256, 128, 64], encoder_blocks[::-1]):
        x = decoder_block(x, skip_connection, filters)

    outputs = Conv2D(n_classes, 1, activation='softmax')(x)
    model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

SegNet

- Arquitetura encoder-decoder com conexões residuais
- Especificamente para segmentação semântica
- Equilíbrio entre desempenho e eficiência computacional
- Redução de Resolução Espacial
- Camadas de Up-Sampling
- Função de ativação (ReLU)
- Função de perda (entropia cruzada)

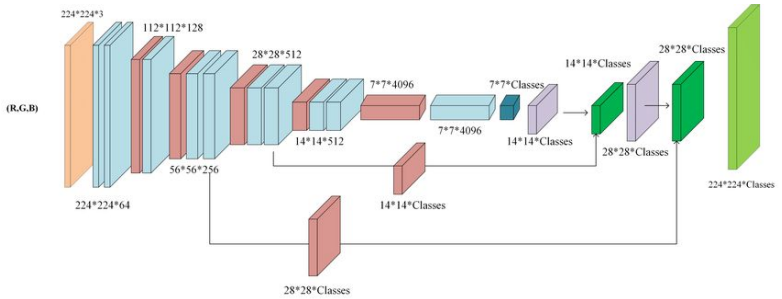


SegNet

```python
def create_segnet(input_shape, n_classes):
    inputs = Input(shape=input_shape)

    conv1 = Conv2D(64, (3, 3), padding='same', activation='relu')(inputs)
    bn1 = BatchNormalization()(conv1)
    pool1, mask1 = MaxPoolingWithArgmax2D((2, 2))(bn1)

    conv2 = Conv2D(128, (3, 3), padding='same', activation='relu')(pool1)
    bn2 = BatchNormalization()(conv2)
    pool2, mask2 = MaxPoolingWithArgmax2D((2, 2))(bn2)

    conv3 = Conv2D(256, (3, 3), padding='same', activation='relu')(pool2)
    bn3 = BatchNormalization()(conv3)
    pool3, mask3 = MaxPoolingWithArgmax2D((2, 2))(bn3)

    up1 = UpSampling2D((2, 2))(pool3)
    conv4 = Conv2D(256, (3, 3), padding='same', activation='relu')(up1)

    up2 = UpSampling2D((2, 2))(conv4)
    conv5 = Conv2D(128, (3, 3), padding='same', activation='relu')(up2)

    up3 = UpSampling2D((2, 2))(conv5)

    unpool1 = MaxUnpooling2D()([up3, mask3])
    unpool2 = MaxUnpooling2D()([unpool1, mask2])
    unpool3 = MaxUnpooling2D()([unpool2, mask1])

    outputs = Conv2D(n_classes, (3, 3), padding='same', activation='softmax')(unpool3)

    model = Model(inputs, outputs)
```

FCN

- Redes neurais convolucionais profundas em modelos para segmentação semântica
- Preservação da resolução espacial, informações contextuais em várias escalas.
- Camadas Convolucionais Completas (sem camadas densas)
- Convoluções 1x1
- Camadas de Up-Sampling
- Fusão de Camadas de Diferentes Resoluções
- Função de ativação (ReLU)
- Função de perda (entropia cruzada)



FCN

```python
def create_fcn(input_shape):
  inputs = Input(shape=input_shape)

  conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
  conv1 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv1)
  maxpool1 = MaxPooling2D((2, 2), strides=(2, 2))(conv1)

  conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(maxpool1)
  conv2 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv2)
  maxpool2 = MaxPooling2D((2, 2), strides=(2, 2))(conv2)

  conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(maxpool2)
  conv3 = Conv2D(256, (3, 3), activation='relu', padding='same')(conv3)
  maxpool3 = MaxPooling2D((2, 2), strides=(2, 2))(conv3)

  conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(maxpool3)
  conv4 = Conv2D(512, (3, 3), activation='relu', padding='same')(conv4)

  up1 = UpSampling2D((2, 2))(conv4)
  concat1 = concatenate([conv3, up1], axis=-1)
  conv5 = Conv2D(256, (3, 3), activation='relu', padding='same')(concat1)

  up2 = UpSampling2D((2, 2))(conv5)
  concat2 = concatenate([conv2, up2], axis=-1)
  conv6 = Conv2D(128, (3, 3), activation='relu', padding='same')(concat2)

  up3 = UpSampling2D((2, 2))(conv6)
  concat3 = concatenate([conv1, up3], axis=-1)
  outputs = Conv2D(n_classes, (1, 1), activation='softmax', padding='same')(concat3)

  model = Model(inputs, outputs)
```

DeepLab

- Dilatações nas camadas convolucionais para aumentar o campo receptivo
- Atrous (Dilated) Convolution
- ASPP (Atrous Spatial Pyramid Pooling)
- Pooling Espacial Através de Convoluções Atrous
- Função de ativação (ReLU)
- Função de perda (entropia cruzada)

DeepLab

```python
def atrous_spatial_pyramid_pooling(inputs):
  dilations = [1, 6, 12, 18]

  conv1x1_1 = Conv2D(256, (1, 1), padding='same', activation='relu')(inputs)
  conv1x1_2 = Conv2D(256, (1, 1), dilation_rate=(dilations[0], dilations[0]), padding='same', activation='relu')(inputs)
  conv1x1_3 = Conv2D(256, (1, 1), dilation_rate=(dilations[1], dilations[1]), padding='same', activation='relu')(inputs)
  conv1x1_4 = Conv2D(256, (1, 1), dilation_rate=(dilations[2], dilations[2]), padding='same', activation='relu')(inputs)
  conv1x1_5 = Conv2D(256, (1, 1), dilation_rate=(dilations[3], dilations[3]), padding='same', activation='relu')(inputs)

  concatenated = concatenate([conv1x1_1, conv1x1_2, conv1x1_3, conv1x1_4, conv1x1_5], axis=-1)
  return concatenated

def create_deeplab(input_shape):
  backbone = MobileNetV2(input_shape=input_shape, include_top=False, weights='imagenet')

  aspp = atrous_spatial_pyramid_pooling(backbone.output)
  upsample = Conv2DTranspose(256, (3, 3), strides=(2, 2), padding='same', activation='relu')(aspp)
  outputs = Conv2D(n_classes, (1, 1), padding='same', activation='softmax')(upsample)

  model = models.Model(backbone.input, outputs)
  return model
```

PSPNet

- A arquitetura PSPNet (Pyramid Scene Parsing Network)
- Pirâmides de pooling para capturar informações contextuais em diferentes escalas
- Camada PSP (Pyramid Pooling Module)
- Backbone
- Camadas Convolutivas
- Função de ativação (ReLU)
- Função de perda (entropia cruzada)

PSPNet

```python
def pyramid_pooling_module(input_tensor, pool_sizes=[1, 2, 3, 6]):
  concat_layers = [input_tensor]

  for size in pool_sizes:
    pooled = AveragePooling2D(pool_size=(input_tensor.shape[1] // size, input_tensor.shape[2] // size))(input_tensor)
    conv = Conv2D(256, (1, 1), padding='same')(pooled)
    upsampled = UpSampling2D(size=(input_tensor.shape[1] // size, input_tensor.shape[2] // size))(conv)
    concat_layers.append(upsampled)

  return concatenate(concat_layers, axis=-1)

def create_pspnet(input_shape):
  inputs = Input(shape=input_shape)

  backbone = ResNet50(include_top=False, weights='imagenet', input_tensor=inputs)

  encoder_output = backbone.get_layer('conv4_block6_out').output

  psp = pyramid_pooling_module(backbone.output)

  x = Conv2D(512, (3, 3), padding='same', activation='relu')(psp)
  x = BatchNormalization()(x)
  x = Conv2D(256, (1, 1), padding='same', activation='relu')(x)
  x = BatchNormalization()(x)

  outputs = Conv2D(n_classes, (1, 1), activation='softmax', padding='same')(x)

  model = Model(inputs=backbone.input, outputs=outputs)
```

MÉTRICAS DE AVALIAÇÃO

- Identificar os pontos fortes e fracos de cada arquitetura e aprender como aprimorar o sistema
- Precision: mede a capacidade do modelo de classificar corretamente os exemplos positivos.
- Pixel Accuracy: avaliar o desempenho geral da segmentação.
- Intersection over Union (IoU): medir a sobreposição entre a máscara de segmentação predita e a máscara de verdade absoluta em cada classe.
- Mean IoU (mIoU): para avaliar o desempenho geral dos modelos.
- IoU por Classe: permite uma análise detalhada do desempenho dos modelos em cada categoria.
- Análise Qualitativa: exemplos de segmentações geradas pelos modelos.
- A combinação de métricas quantitativas e qualitativas proporciona uma avaliação completa e detalhada do desempenho dos modelos.

$$Precision = \frac{TP}{(TP + FP)} \qquad PixelAccuracy = \frac{TP + TN}{(TP + TN + FP + FN)} \qquad IoU = \frac{TP}{(TP + FP + FN)}$$

MÉTRICAS DE AVALIAÇÃO

```python
def eval_model(model):
    test_loss, test_accuracy = model.evaluate(X_test, test_y_cat)
    predictions = model.predict(X_test)

    true_positive = (predictions * test_y_cat).sum()
    false_positive = predictions.sum() - true_positive
    precision = true_positive / (true_positive + false_positive)

    correct_pixels = (predictions == test_y_cat).sum()
    total_pixels = predictions.size
    pixel_accuracy = correct_pixels / total_pixels

    intersection = (predictions * test_y_cat).sum(axis=(1, 2, 3))
    union = (predictions + test_y_cat - (predictions * test_y_cat)).sum(axis=(1, 2, 3))
    iou = (intersection / union).mean()

    class_iou = intersection / union
    mIoU = class_iou.mean()

    class_iou_dict = {}
    print("IoU por Classe:")
    for i in range(n_classes):
        class_iou_dict[f'Classe {i}'] = class_iou[:, i]
        print(f"Classe {i}: {class_iou[:, i]}")

    print(f'Test Loss: {test_loss}, Test Accuracy: {test_accuracy}')
    print(f'Precision: {precision}, Pixel Accuracy: {pixel_accuracy}')
    print(f'Intersection over Union (IoU): {iou}')
    print(f'Mean IoU (mIoU): {mIoU}')

    return class_iou_dict
```

MÉTRICAS DE AVALIAÇÃO

```
41  def seg_sample(model):
42    predictions = model.predict(X_test)
43    fig, axs = plt.subplots(4, 3, figsize=(10, 10))
44
45    predictions_resized = np.argmax(predictions, axis=-1)
46    test_y_resized = np.argmax(test_y_cat, axis=-1)
47
48    for i in range(4):
49      random_num = random.randint(0, len(predictions) - 1)
50      axs[i, 0].set_title('Original Image')
51      axs[i, 1].set_title('True Mask')
52      axs[i, 2].set_title('Predicted Mask')
53
54      axs[i, 0].imshow(X_test[random_num])
55      axs[i, 1].imshow(test_y_resized[random_num])
56      axs[i, 2].imshow(predictions_resized[random_num])
57
58    plt.tight_layout()
59    plt.show()
```

CONCLUSÃO

- Poder Computacional
  - Limitações da GPU
    - Interrupção do treinamento
  - Estouro da RAM
- Melhora individual de cada modelo
  - Otimização de Hiperparâmetros
    - Validação Cruzada

Obrigada!


Centro de
Informática
UFPE
VIRTUS IMPAVIDA
Universidade
Federal
de Pernambuco