Trabajo Deep Learning Articulo Científico

Javier Ochoa

Se escogió el artículo científico:

Road Extraction by Deep Residual U-Net, *que describe la Arquitectura ResUnit*

La segmentación semántica es una tarea importante en visión por computadora. Es el proceso de clasificar cada píxel de una imagen en una etiqueta de clase. La segmentación semántica ayuda a la computadora a comprender el entorno en el que opera. En este artículo que se escogió, recoge este análisis donde se implementa sobre una red neuronal profunda de unidad residual (ResUNet),es una red neuronal totalmente convolucional diseñada para obtener un alto rendimiento con menos parámetros. Es una mejora sobre la arquitectura de U-Net. ResUNet aprovecha tanto la arquitectura de unidades como el aprendizaje residual profundo.

ResUnet se refiere a Unet Residual Profunda. Es una arquitectura codificador-decodificador desarrollada por Zhengxin Zhang según artículo de trabajo a desarrollar esta vez para el curso y de elección entre los 3 paper, para la segmentación semántica. Inicialmente se utilizó para la extracción de carreteras a partir de imágenes aéreas de alta resolución en el campo del análisis de imágenes de teledetección. Posteriormente, fue adoptado por investigadores para muchas otras aplicaciones.

ResUNet es una mejora sobre la arquitectura de unidades existente U-Net, el autor no describe dentro de su anexos, aprovechando tanto la arquitectura de unidades como el aprendizaje residual profundo para armar la estructura ResUnet.

Aquí les dejo un link por si alguno desea profundizar en la arquitectura inicial de U-Net que se basó en un artículo científico “U-Net_Convolutional Networks for Biomedical Image Segmentation”, de donde se empezó a trabajar la segmentación y siguieron las siguientes arquitecturas mejoradas como la del artículo de análisis en este momento.

Material de apoyo U-Net:

https://arxiv.org/abs/1505.04597 (Descargar PDF articulo cientifico)

https://www.youtube.com/watch?v=waIPUsecaaQ

https://www.geeksforgeeks.org/u-net-architecture-explained/

Resnet:

Ventajas

- El uso de bloques residuales ayuda a construir una red más profunda sin preocuparse por el problema de la desaparición del gradiente o la explosión de gradientes. También ayuda a facilitar el entrenamiento de la red.
- Las conexiones de salto en ResUnet ayudan a un mejor flujo de información entre diferentes capas, lo que ayuda a un mejor flujo de gradientes durante el entrenamiento (propagación hacia atrás).

Arquitectura General de ResUnet

ResUnet es una red de codificación y una red de decodificación y un puente que conecta ambas redes, como una Unet. Aplica batch de normalización (BN) y utiliza dos convoluciones de 3 x 3 (Cov2d), donde a cada una le sigue una función de activación ReLU. En el caso de **ResUnet**, estas capas se sustituyen por un bloque residual preactivado.

Codificador

El codificador toma la imagen de entrada y la pasa a través de diferentes bloques que lo que ayuda a la red a aprender una representación abstracta. El codificador consta de tres bloques de codificador, que se construyen utilizando el bloque residual preactivado. La salida de cada bloque codificador actúa como una conexión de salto para el bloque decodificador correspondiente.

Para reducir las dimensiones espaciales (alto y ancho) de los mapas de características, la primera capa de convolución de 3 × 3 utiliza un paso de 2 en el segundo y tercer bloque codificador. Un valor de stride = 2 (cantidad de pixel del kernel a desplazar o zancada) reduce las dimensiones espaciales a la mitad, es decir, de 256 a 128.

Puente (Bloque Bridge)

El puente también consta de un bloque residual preactivado con un valor de stride = 2.

Decodificador

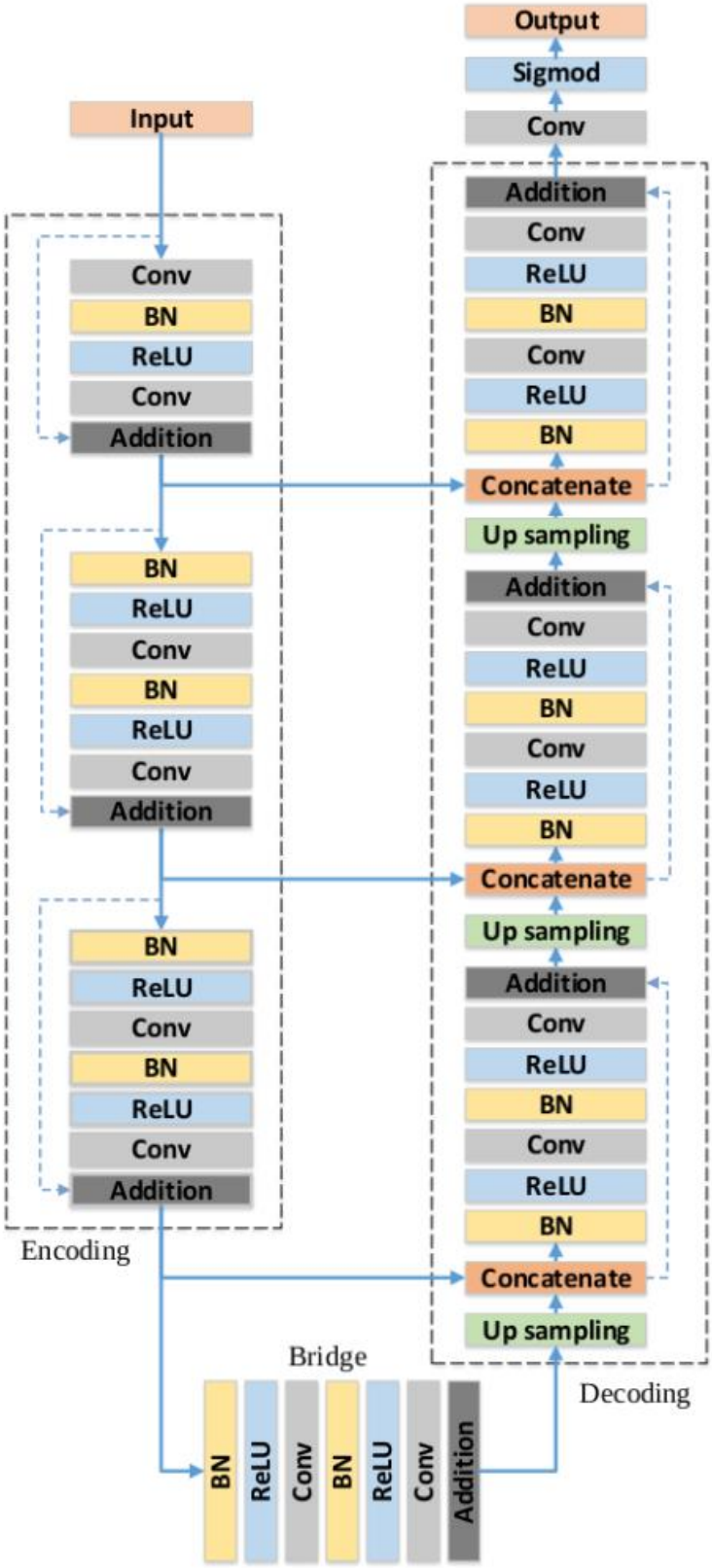
El decodificador toma el mapa de características del puente y las conexiones de salto de diferentes bloques codificadores y aprende una mejor representación semántica, que se utiliza para generar una máscara de segmentación.

El decodificador consta de tres bloques decodificadores y, después de cada bloque, las dimensiones espaciales del mapa de características se duplican y el número de canales de características se reduce.

Cada bloque decodificador comienza con un muestreo ascendente de 2×2, que duplica las dimensiones espaciales de los mapas de características. A continuación, estos mapas de características se concatenan con la conexión de salto adecuada desde el bloque codificador. Estas conexiones de salto ayudan a los bloques decodificadores a obtener la función aprendida por la red del codificador. Después de esto, los mapas de características de la operación de concatenación pasan a través de un bloque residual preactivado.

La salida del último decodificador pasa por una convolución 1×1 con activación sigmoidea. La función de activación sigmoidea proporciona la máscara de segmentación que representa la clasificación por píxeles

Figura2:


Output
Sigmod
Conv
Input
Addition
Conv
ReLU
BN
Conv
ReLU
BN
Concatenate
Up sampling
Addition
Conv
ReLU
BN
Conv
ReLU
BN
Concatenate
Up sampling
Addition
Conv
ReLU
BN
Conv
ReLU
BN
Concatenate
Up sampling
Addition
Conv
ReLU
BN
Conv
ReLU
BN
Conv
ReLU
BN
Addition
Encoding
Bridge
BN
ReLU
Conv
BN
ReLU
Conv
Addition
Decoding

Ahora vamos a construir nuestra arquitectura de unidad ResUNet

Podemos ver en la Figura 2 en este artículo de ResUNet donde esta el diagrama completo de la arquitectura por bloques y lo vamos a usar para construir la arquitectura en Keras usando librerías de TensorFlow

Principalmente consta de tres bloques codificadores, tres bloques decodificadores y un puente que conecta tanto el codificador como el decodificador.

El bloque residual preactivado es el bloque de construcción principal para la arquitectura de la unidad ResUNet . Consiste en dos bloques de convolución, donde cada bloque de convolución consta de una normalización por lotes, una función ReLU y una capa convolucional. A continuación, se encuentra el mapeo de identidad que conecta la entrada y la salida del bloque convolucional.

TABLE I
THE NETWORK STRUCTURE OF RESUNET.

	Unit level	Conv layer	Filter	Stride	Output size
Input					$224 \times 224 \times 3$
Encoding	Level 1	Conv 1	$3 \times 3/64$	1	$224 \times 224 \times 64$
		Conv 2	$3 \times 3/64$	1	$224 \times 224 \times 64$
	Level 2	Conv 3	$3 \times 3/128$	2	$112 \times 112 \times 128$
		Conv 4	$3 \times 3/128$	1	$112 \times 112 \times 128$
	Level 3	Conv 5	$3 \times 3/256$	2	$56 \times 56 \times 256$
		Conv 6	$3 \times 3/256$	1	$56 \times 56 \times 256$
Bridge	Level 4	Conv 7	$3 \times 3/512$	2	$28 \times 28 \times 512$
		Conv 8	$3 \times 3/512$	1	$28 \times 28 \times 512$
Decoding	Level 5	Conv 9	$3 \times 3/256$	1	$56 \times 56 \times 256$
		Conv 10	$3 \times 3/256$	1	$56 \times 56 \times 256$
	Level 6	Conv 11	$3 \times 3/128$	1	$112 \times 112 \times 128$
		Conv 12	$3 \times 3/128$	1	$112 \times 112 \times 128$
	Level 7	Conv 13	$3 \times 3/64$	1	$224 \times 224 \times 64$
		Conv 14	$3 \times 3/64$	1	$224 \times 224 \times 64$
Output		Conv 15	1×1	1	$224 \times 224 \times 1$

Código estructura en keras

```
# Carga librerias

from tensorflow.keras.layers import Conv2D, BatchNormalization,
Activation, MaxPool2D, UpSampling2D, Concatenate, Input
from tensorflow.keras.models import Model

def Normalizacion_Relu(inputs):
     # Aplica Normalizacion & Funcion ReLU
     x = BatchNormalization()(inputs)
     x = Activation("relu")(x)
     return x
```

```
# Se define una funcion para identificar todo proceso de lo nuevo
de la arquitectura ResUnet que maneja bloques residales
# que maneja comunicacion entre capas para mejorar el flujo de
gradientes a través de la red durante el entrenamiento.

def bloque_residual (inputs, num_filters, strides=1):

     # Parte 1=> Capa Convolucional
     capa1 = Normalizacion_Relu(inputs)
     # toma valor de strides que se envia parametro a la funcion
llamada
     capa1 = Conv2D(num_filters, kernel_size=(3, 3),
padding="same", strides=strides)(capa1)
     capa1 = Normalizacion_Relu(capa1)
     # toma valro stride = 1
     capa1 = Conv2D(num_filters, kernel_size=(3, 3),
padding="same", strides=1)(capa1)

     # Conexiones directas o Mapeo de Identidad lo nuevo por
Resunet

     capa2 = Conv2D(num_filters, kernel_size=(1, 1),
padding="same", strides=strides)(inputs)
     salida_residual = capa1 + capa2
     return salida_residual

def bloque_decoder(inputs, skip_features, num_filters):

    x=UpSampling2D((2,2))(inputs)
    x=Concatenate()([x, skip_features])
    x=bloque_residual(x, num_filters, strides=1)
    return x

def arquitectura_resunet(input_shape):

    # arquitectura Resinet

    inputs = Input(input_shape)

    # **Encoder etapa1**

    etapa1 = Conv2D(filters=64, kernel_size=(3, 3), padding="same",
strides=1)(inputs)
    etapa1 = Normalizacion_Relu(etapa1)
    etapa1 = Conv2D(filters=64, kernel_size=(3, 3), padding="same",
strides=1)(etapa1)

    # Aplica la estructura Addition suma convolucion kernel (1,1)
de input original + salida de econder1
```

```
    # tener en cuenta que usa 1x1 para igualar las dimensiones de
los canales entre la entrada y la salida de un bloque residual
    # reducir la profundidad (número de canales) de los mapas de
características menos cant de cálculos de opeaciones, mejorando la
eficiencia del modelo
    # sin perder mucha información relevante

    salida = Conv2D(filters=64, kernel_size=(1, 1),
padding="same")(inputs)
    print("num ope salida",salida)
    # aplica Addition segun estrcutura del articulo

    salida1 = etapa1 + salida

    # Encoder etapa 2 y 3

    salida2 =bloque_residual(salida1, 128, strides=2)
    salida3 =bloque_residual(salida2, 256, strides=2)

    # Bridge

    bridge =bloque_residual(salida3, 512, strides=2)

    # Decoder 1, 2, 3 de abajo hacia arriba

    salida_dec = bloque_decoder(bridge, salida3, 256)
    salida_dec = bloque_decoder(salida_dec, salida2, 128)
    salida_dec = bloque_decoder(salida_dec, salida1, 64)

    # Classificador en la salida aplica convolucion y sigmoidal a
la salida del decodificar

    outputs = Conv2D(1, 1, padding="same",
activation="sigmoid")(salida_dec)

    # Creamos nuestra arquitectura de red: como se necesita definir
como fluyen los datos a través del modelo
    # donde tenemos operaciones que no son secuenciales donde
algunas capas se suma con sus entradas ya no
    # usaremos secuencial sino uno para modleos complejos
importaremos API funcional (Model)

    model = Model(inputs, outputs, name="Arquitectura_Resinet")
    return model

#tomaremos de entrada segun el modelo  224 pixels de altura , 224
de ancho y  3 de profundidad RGB, ademas considerar
# que este modelo La elección de 224x224x3 como forma de entrada
permite reutilizar este modelo a futuro en investigacion
```

```
# preentrenado en otras tareas mediante técnicas de transferencia
de aprendizaje, modificando algunas capas de salida.
# llamda incial al modelo

shape=(224, 224, 3)

# llamamos al modelo
model_arquitectura_resunet = arquitectura_resunet(shape)

# mostramos detalle parametros del modelo
model_arquitectura_resunet.summary()
```