MINI PROYECTO N° 2

Ávila Rendon Alejandra, Osuna Piñero Carlos Álvaro

Departamento de Automática y Electrónica, Redes Neuronales y Deep Learning
Universidad Autónoma de Occidente
Santiago de Cali, Valle del Cauca, Colombia.
alejandra.avila@uao.edu.co, carlos.osuna@uao.edu.co
2146936, 2226436

Resumen – El presente documento contiene cada uno de los procedimientos y/o pasos seguidos por medio de estudio, aplicación, entramiento y validación de una red neuronal capaz de segmentar semánticamente una imagen pertenecientes a un DataSet de elaboración propia por estudiantes aplicando la arquitectura unet . Este mini proyecto está basado en los conocimientos adquiridos por diferentes medios apoyados en plataformas externas como Google Colab, Scale AI, LabelImg y librerías de redes como Tensorflow.

Palabras claves – Multicapa, Red convolucional, Segmentación semantica, DataSet, Unet, Mascara, Entrenamiento, Imagen, ScaleAI

INTRODUCCIÓN

Las redes neuronales artificiales son un modelo computacional que permite simular el comportamiento del cerebro humano, es decir, dotar a las máquinas de la capacidad de aprender de una manera similar a como lo hace nuestro cerebro. Una red neuronal artificial está formada por neuronas artificiales, que son unidades o nodos que reciben información del exterior o de otras neuronas, de manera similar a los impulsos nerviosos que reciben las neuronas del cerebro humano, las procesan y generan un valor de salida que alimenta a otras neuronas de la red o son la salida hacia el exterior de la red. [1]

Una neurona artificial está formada por un conjunto de entradas que son enlaces o interconexiones por donde reciben información del exterior de la red o de otras neuronas y cada una de ellas ponderadas por un peso que determina la importancia de la información, además de esto, la conforman también un conjunto de funciones tales como de propagación, activación y transferencia. Por último, la salida de la neurona refleja el enlace o la interconexión por donde se entrega el resultado al exterior.

Para lograr la realización de este mini proyecto es necesario poner en práctica los conocimientos investigativos adquiridos y practicarlos en el diseño, entrenamiento y validación de la red neuronal que permita la segmentación de una imagen a través de una arquitectura de tipo Unet que dado un dataset de elaboración propia que consta de 200 imágenes de carros y/o objetos en las imágenes que se le pasen a la red. Para lograr el objetivo es necesario primero obtener la data del dataset, pre procesarla para unificar criterios de tamaño, y aplicarle data argumentación con el fin de evitar el sobre entrenamiento, al ser un dataset pequeño, luego esta data se debe dividir en data de entrenamiento, validación y prueba para luego utilizar arquitecturas como la Unet, para este caso, solo se copió la arquitectura de dicho modelo más no se realizó transfer learning, obteniendo así luego del entrenamiento, una predicción en cada imagen sobre la segmentación de 5 elementos que se apreciaban en la foto, y resaltarlo con otro color.

La Segmentación Semántica es un conjunto de técnicas que permiten crear regiones dentro de una imagen y atribuir significado semántico a cada una de ellas. La imagen de un paisaje puede ser subdividida en árboles, ríos, nubes, y cada una de ellas coloreada de un color uniforme. A cada pixel se le atribuye el color correspondiente al tipo de objeto de que se trate.[2]

MARCO TEORICO

Las RNPs presentan gran variedad estructural en cuanto a sus formas y tamaños dependiendo de su aplicación. Sus arquitecturas varían en cuanto a número y tipo de capas, así como también, cantidad de conexiones entre capas. Hay numerosos estudios en los que se proponen arquitecturas de redes, o mejoras de otras ya estudiadas, para lograr mejores exactitudes de desempeño y eficiencia en las tareas de IA para las cuales fueron diseñadas. [3][4]

Multicapa: Es el tipo de red neuronal más extendido. Consta de una capa de entrada que recibe valores del exterior, una serie de capas intermedias u ocultas que van procesando la información y una capa de salida que entrega los resultados de la red neurona.

Red Convolucional: Son similares a las redes neuronales multicapa, pero la diferencia radica en que en estas las neuronas de cada capa no se interconectan con todas las de la capa siguiente, si no con un subconjunto de estas, y no se interconectan entre las neuronas de la misma capa.

DataSet: Representa un conjunto completo de datos, incluyendo las tablas que contienen, ordenan y restringen los datos, así como las relaciones entre las tablas.

Segmentación de imagen: La segmentación semántica es un algoritmo de deep learning que asocia una etiqueta o categoría a cada píxel presente en una imagen. Se utiliza para reconocer un conjunto de píxeles que conforman distintas categorías.

Mascara: Utilizada para obtener los parámetros iniciales del crecimiento de regiones, además de como condición de confinamiento del mismo, además, es usada también con el fin de obtener la semilla óptima que hace al algoritmo reproducible.

Unet: U-Net es una red neuronal convolucional que se desarrolló para la segmentación de imágenes biomédicas en el Departamento de Ciencias de la Computación de la Universidad de Friburgo .La red se basa en la red totalmente convolucionaly su arquitectura se modificó y amplió para trabajar con menos imágenes de entrenamiento y generar segmentaciones más precisas. La segmentación de una imagen de 512 × 512 toma menos de un segundo en una GPU moderna

Entrenamiento: Ajustar cada uno de los pesos de las entradas de todas las neuronas que forman parte de la red neuronal, para que las respuestas de la capa de salida se ajusten lo más posible a los datos que conocemos.

Encoder: significa convertir datos a un formato requerido.

Decoder: significa convertir un mensaje codificado en un lenguaje inteligible.

Imagen: Representación visual de un objeto, una persona, un animal o cualquier otra cosa plausible de ser captada a través de diferentes técnicas como ser la pintura, el diseño, la fotografía y el video, entre otras.

Neurona: Encargada de procesar la data recibida a partir de su peso sináptico y bias, serie de algoritmos que buscan relaciones en un conjunto de datos.

DESCRIPCIÓN DEL PROBLEMA

El problema a atacar para este proyecto es relacionado con la segmentación semántica de imágenes, en este se desea segmentar los distintos elementos en un carro, los cuales son: ***ventanas, chasis, luces, cauchos y fondo*** así como muestra la figura 1, la intención es pasar una foto y recibir la segmentación de cada uno de sus elementos, esto como complemento al problema que se puede solventar de conducción utilizando el dataset de camvid. El dataset es de elaboración propia y a continuación se explicará el proceso que se llevó a cabo para su construcción

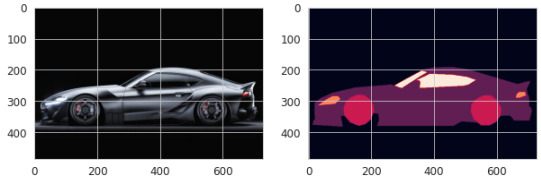

Figura 1 - Segmentación Semántica de Carros. Fuente: Elaboración Propia.

PLANTEAMIENTO DE LA SOLUCIÓN

Para encontrar la solución del problema descrito anteriormente, lo primero a atacar fue el dataset:

A. Segmentación de Objetos del Data Set

Una vez recopiladas todas las imágenes de carros que conforman el dataset, es necesario crear las máscaras o segmentaciones de los objetos en cada una de ellas, para ello con la ayuda de ScaleAI se utilizó se pintó cada imagen y posteriormente este generó la máscara correspondiente, como se muestra en la figura 2.

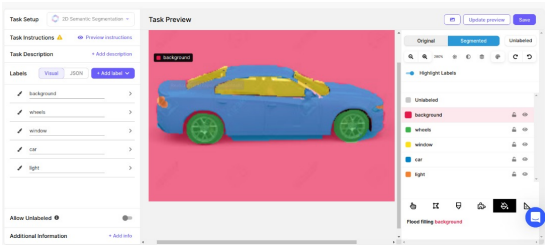

Figura 2 - Segmentación Semántica en Scale AI. Fuente: Elaboración Propia.

Posteriormente al momento de exportar el dataset, este generó dos carpetas y un archivo JSON, la primera carpeta contiene las imágenes originales, la segunda las máscaras y el archivo JSON indica por cada imagen las ubicaciones de cada elemento en la imagen, tal como se muestra en la figura 3.

Figura 3 - Polígonos de cada segmentación de una foto. Fuente: Elaboración Propia

B. Preprocesado de Datos

Una vez se tiene el dataset correspondiente, ahora resta la codificación que permita utilizar dichos datos y entrenar el modelo, para ello antes lo primero a realizar en el notebook adjunto a dicho documento fue un generador de datos, o en inglés *DataGenerator*, que consiste en una clase generadora que a partir de distintos métodos genera el dato a trabajar en el modelo, en este caso retorna una tupla de valores, el primer valor es la foto del carro original y el segundo valor la máscara del mismo, tal como se presenta en la figura 4.

Figura 4 - DataGenerator de la Solución Planteada. Fuente: Elaboración propia.

Se cargaron todos los archivos y se dividieron en conjunto de prueba,

entrenamiento y validación siguiente una proporción 70-20-10.

Así mismo el dataset original es realmente pequeño, comparado con la cantidad de imágenes que suelen tener los datasets más grandes es por ello que se aplicó la técnica de data augmentation, en donde a partir de las imágenes de entrenamiento, se crearán otras que tengan diferentes perspectivas a las originales, como zoom, rotación, entre otros.

C. IMPLEMENTACIÓN DEL MODELO

Una vez generada los datos a procesar, se comenzó con el desarrollo de la arquitectura solucionar dicho problema, donde en esta caso se optó por utilizar la arquitectura en U, específicamente Unet, que cuenta con un encoder que reduce la imagen a su mínima expresión y luego el decoder que convierte el resultado a algo entendible para nosotros. En la figura 5 se puede apreciar la arquitectura del modelo de Unet, que en vez de elegir la alternativa de aplicar Transfer Learning para solventar el problema se optó por replicar la arquitectura del mismo y entrenar de cero, con el propósito de lograr un mejor aprendizaje y además modificar ciertas capas para solucionar mejor el problema, el cual se presenta en la figura 6.

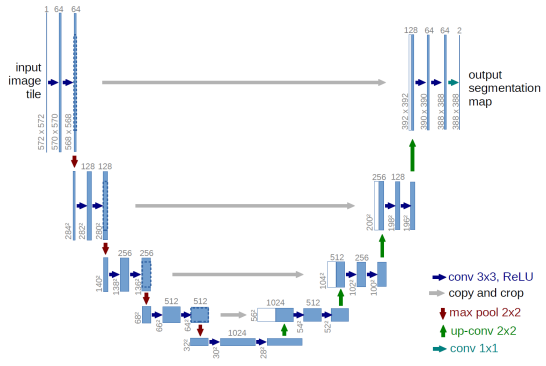

Figura 5 - Arquitectura Unet.

```
def unet(pretrained_weights=None, input_size=(256, 256, 1)):
    inputs = Input(input_size)
    conv1 = Conv2D(64, 3, activation = None, padding = 'same', kernel_initializer = 'he_nor
    conv1 = BatchNormalization()(conv1)
    conv1 = LeakyReLU(alpha=leakyrelu_alpha)(conv1)
    #conv1 = Dropout(dr_rate)(conv1) ###
    conv1 = Conv2D(64, 3, activation = None, padding = 'same', kernel_initializer = 'he_nor
    conv1 = BatchNormalization()(conv1)
    conv1 = LeakyReLU(alpha=leakyrelu_alpha)(conv1)
    #conv1 = Dropout(dr_rate)(conv1) ###
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
    #pool1 = Dropout(dr_rate)(pool1) ###

    conv2 = Conv2D(128, 3, activation = None, padding = 'same', kernel_initializer = 'he_no
    conv2 = BatchNormalization()(conv2)
    conv2 = LeakyReLU(alpha=leakyrelu_alpha)(conv2)
    #conv2 = Dropout(dr_rate)(conv2)###
    conv2 = Conv2D(128, 3, activation = None, padding = 'same', kernel_initializer = 'he_no
    conv2 = BatchNormalization()(conv2)
    conv2 = LeakyReLU(alpha=leakyrelu_alpha)(conv2)
    #conv2 = Dropout(dr_rate)(conv2)###
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

    conv3 = Conv2D(256, 3, activation = None, padding = 'same', kernel_initializer = 'he_no
    conv3 = BatchNormalization()(conv3)
    conv3 = LeakyReLU(alpha=leakyrelu_alpha)(conv3)
    #conv3 = Dropout(dr_rate)(conv3) ###
    conv3 = Conv2D(256, 3, activation = None, padding = 'same', kernel_initializer = 'he_no
    conv3 = BatchNormalization()(conv3)
    conv3 = LeakyReLU(alpha=leakyrelu_alpha)(conv3)
    #conv3 = Dropout(dr_rate)(conv3) ###
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)

    conv4 = Conv2D(512, 3, activation = None, padding = 'same', kernel_initializer = 'he_no
    conv4 = BatchNormalization()(conv4)
    conv4 = LeakyReLU(alpha=leakyrelu_alpha)(conv4)
    #conv4 = Dropout(dr_rate)(conv4) ###
    conv4 = Conv2D(512, 3, activation = None, padding = 'same', kernel_initializer = 'he_no
```

Figura 6 - Código de la Arquitectura Implementada. Fuente: Elaboración propia.

Posteriormente se llamó a la función que genera el modelo, además de agregar métricas como el checkpoint para obtener el mejor modelo a lo largo del entrenamiento, así mismo como optimizadores que en este caso se utilizó el de Adam para finalmente comenzar con el entrenamiento con un total de 100 épocas, que duró alrededor de 3 horas tal como se presenta en la figura 7 los resultados del entrenamiento.

```
history = model.fit_generator(X_train, epochs=100, callbacks=[model_checkpoint],
                              validation_data=X_valid)

Epoch 1/100
10/10 [==============================] - ETA: 0s - loss: 4.7183 - accuracy: 0.1403
Epoch 1: loss improved from inf to 4.71826, saving model to ejercicio.hdf5
10/10 [==============================] - 44s 3s/step - loss: 4.7183 - accuracy: 0.1403 - va
Epoch 2/100
10/10 [==============================] - ETA: 0s - loss: 4.8712 - accuracy: 0.1720
Epoch 2: loss did not improve from 4.71826
10/10 [==============================] - 30s 3s/step - loss: 4.8712 - accuracy: 0.1720 - va
Epoch 3/100
10/10 [==============================] - ETA: 0s - loss: 5.1085 - accuracy: 0.1672
Epoch 3: loss did not improve from 4.71826
10/10 [==============================] - 29s 3s/step - loss: 5.1085 - accuracy: 0.1672 - va
Epoch 4/100
10/10 [==============================] - ETA: 0s - loss: 5.2081 - accuracy: 0.1382
Epoch 4: loss did not improve from 4.71826
10/10 [==============================] - 30s 3s/step - loss: 5.2081 - accuracy: 0.1382 - va
Epoch 5/100
10/10 [==============================] - ETA: 0s - loss: 5.2582 - accuracy: 0.1104
Epoch 5: loss did not improve from 4.71826
10/10 [==============================] - 30s 3s/step - loss: 5.2582 - accuracy: 0.1104 - va
Epoch 6/100
10/10 [==============================] - ETA: 0s - loss: 5.2877 - accuracy: 0.1064
Epoch 6: loss did not improve from 4.71826
10/10 [==============================] - 29s 3s/step - loss: 5.2877 - accuracy: 0.1064 - va
Epoch 7/100
10/10 [==============================] - ETA: 0s - loss: 5.3635 - accuracy: 0.1010
Epoch 7: loss did not improve from 4.71826
10/10 [==============================] - 29s 3s/step - loss: 5.3635 - accuracy: 0.1010 - va
Epoch 8/100
10/10 [==============================] - ETA: 0s - loss: 5.2841 - accuracy: 0.1145
Epoch 8: loss did not improve from 4.71826
10/10 [==============================] - 31s 3s/step - loss: 5.2841 - accuracy: 0.1145 - va
Epoch 9/100
10/10 [==============================] - ETA: 0s - loss: 5.3264 - accuracy: 0.1071
Epoch 9: loss did not improve from 4.71826
10/10 [==============================] - 30s 3s/step - loss: 5.3264 - accuracy: 0.1071 - va
Epoch 10/100
10/10 [==============================] - ETA: 0s - loss: 5.3606 - accuracy: 0.1091
Epoch 10: loss did not improve from 4.71826
```

Figura 7 - Entrenamiento del Modelo. Fuente: Elaboración Propia.

D. RESULTADOS

Posterior al entrenamiento del modelo se obtuvieron las gráficas de la pérdida para ver el desempeño del mismo, que se presenta en la figura 8.

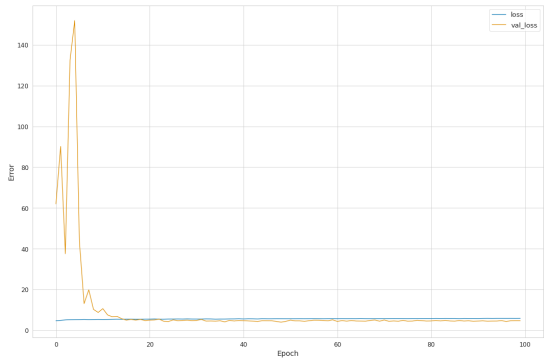

Figura 8 - Pérdida del Modelo Entrenado. Fuente: Elaboración propia.

Por último, el conjunto de prueba se pasó por el modelo y se guardaron las imágenes, con el propósito de ver los resultados de la segmentación, tal como se ve en la figura 9.

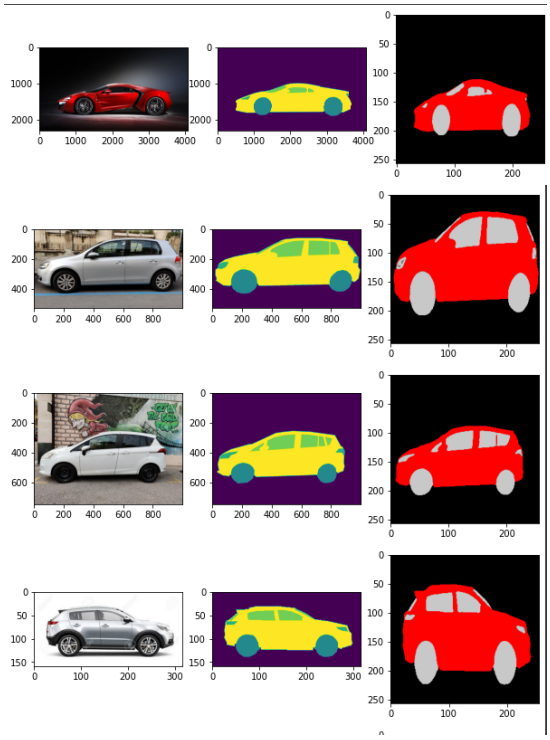

Figura 9 - Resultados de Segmentación. Fuente: Elaboración Propia.

CONCLUSIONES

Luego de llevar a cabo el presente trabajo, el cual sin duda alguna se ha aprendido bastante sobre redes neuronales convolucionales para la segmentación semántica, se puede concluir que:

El primer paso, y que sirve de base fundamental para la construcción del modelo de redes neuronales, es construir un dataset que contenga la mayor cantidad posible de imágenes, así como aplicar diferentes tćnicas de preprocesamiento que ayudarán al entrenamiento y posterior efectividad de la red, como un resize, así como aplicar data augmentation a las que formen parte del data set de entrenamiento con el objetivo de evitar problemas como el sobre entrenamiento, y ayudar a la red a trabajar con los objetos en muchas perspectivas diferentes que le permitan identificar las características propias e innatas de cada uno de ellos, y no colores, formas o fondos.

Posteriormente no hace falta reinventar la rueda para lograr resolver el problema planteado, las redes convolucionales son redes que consumen muchísimos recursos de cómputo, y que tardan horas o días en entrenarse, además que requieren de datasets enormes para empezar a ser efectivas, es por ello que es muy útil en estos casos utilizar arquitecturas que ya tienen modelos y pesos sinápticos entrenados y ajustados a las necesidades de dicha red, para este caso no se utilizó pero es importante tomar en cuenta que se tienen estas opciones presentes al momento de codificar, pero esto siempre será el primer paso, ya que se tiene que refinar el modelo al problema asociado, cambiar pesos sinápticos, congelar y descongelar capas, entre otros.

Por último, se puede concluir que diseñar y construir un modelo y arquitectura de redes neuronales convolucionales que apliquen segmentación semántica no es tarea sencilla. Requiere de mucha investigación de arquitecturas y modelos, de horas recolectando imágenes y generando las máscaras del mismo, ensayos, pruebas y errores. Todo esto es el trabajo de un desarrollador de IA/ML, cada parte del proceso es importante y se complementan entre sí.

REFERENCIAS BIBLIOGRAFICAS

[1]https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf

[2]https://lamaquinaoraculo.com/computacion/u-net-y-segmentacion-semantica/

[3]http://apcam.org.mx/wp-content/uploads/2019/02/PONENCIA-8-UACH.pdf

[4]http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf