Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3084 - Data Science - Ing. Lynette Garcia Perez

Resultados Parciales y Visualizaciones estáticas

CGIAR Eyes on the Ground Challenge

Cristian Fernando Laynez 201281,
Juan Angel Carrera, 20593,
Sara Maria Paguaga 20634,
Guillermo Santos Barrios 191517

Guatemala, 30 de octubre de 2023

Investigación de Algoritmos

SVM (*Support Vector Machine*):

Las Máquinas de vector soporte son modelos supervisados que se destacan al manejar múltiples variables continuas o categóricas (Sanghvi, 2022). La idea es generar hiperplanos que puedan separar las distintas clases en las que se desean clasificar los datos. Es importante seleccionar una función de kernel correctamente para poder obtener los resultados deseados. Entre las funciones más utilizadas están: kernel lineal, kernel gaussiano y kernel polinomial.

K-Nearest Neighbor

Este algoritmo es de los más simples. Puede utilizarse para clasificar imágenes poco complejas. Si el conjunto de datos está lo suficientemente balanceado, también podrá clasificar conjuntos más complejos. El objetivo es obtener la clase para cada uno de los ejemplos deseados. Esto lo hace evaluando las distancias a las demás clases (distancia Euclidiana o distancia Manhattan). Se selecciona la clase que tiene más vecinos cercanos (Sanghvi, 2022).

CNN (Convolutional Neuronal Networks):

Son redes diseñadas especialmente para identificar imágenes. Capas importantes de este tipo de redes son las capas de convolución y las capas de pooling. Las capas de convolución permiten identificar bordes en una imagen por medio de filtros y las capas de pooling reducen el tamaño de las matrices con los valores más representativos de las imágenes, de esta forma abstraen información y se reduce el costo computacional.

(Mishra, 2020)

Arquitecturas de CNNs para detección de imágenes:

- ResNet:
  Es común integrar más capas en una red neuronal profunda, para disminuir la tasa de error. Sin embargo, añadir más capas no siempre garantiza mejores resultados y puede presentar el problema de desvanecimiento y explosión de gradientes.

  La arquitectura ResNet permite que los gradientes fluyan de una mejor forma a través de la red al momento de hacer backpropagation, permitiendo entrenar redes mucho más profundas. Esta arquitectura propone una técnica llamada skip connections, la cual consiste en conectar las activaciones de una capa con otras capas saltándose algunas capas intermedias. Forma un bloque residual y las redes se forman apilando los bloques residuales.

  (“Residual Networks ResNet Deep Learning,” 2020)

- LeNet-5:

  LeNet-5 es una de las primeras redes neuronales convolucionales diseñadas específicamente para reconocimiento de dígitos escritos a mano. Fue introducida por Yann LeCun en 1998 y es reconocida por establecer la base para las redes convolucionales que la sucedieron.

Arquitectura:
La arquitectura LeNet-5 consta de 7 capas en total, entre las cuales hay capas de convolución, capas de pooling (subsampling) y capas completamente conectadas.

Función de Activación:
LeNet-5 utiliza la función sigmoide tangente hiperbólica como función de activación.
A pesar de que la arquitectura LeNet-5 es bastante simple en comparación con las redes modernas, fue pionera y demostró la eficacia de las redes neuronales convolucionales para tareas de visión por computadora. A lo largo de los años, las CNNs han evolucionado y se han vuelto más complejas y profundas, pero LeNet-5 sigue siendo una referencia importante en la historia de las redes neuronales.
(Gradient-based learning applied to document recognition, 1998)

RNN (*Recurrent Neuronal Networks*):

Es un tipo de red neuronal que utiliza datos secuenciales, es utilizada para casos con datos ordinales, es decir, que pueden ser ordenados según una escala o para problemas temporales. Este tipo de red neuronal se caracteriza por su capacidad de tomar *inputs* previos para influenciar *inputs* y *outputs* actuales y tener una "memoria", lo cual permite dar predicciones más precisas de información reciente. (*IBM*, 2023)

La estructura de este tipo de red es simple y tiene pocos parámetros lo cual representa una ventaja en términos de memoria y costo computacional así como un efecto positivo en tiempo de entrenamiento. Sin embargo, con este tipo de red se presenta el problema de explosión y desvanecimiento del gradiente cuando se entrena con secuencias muy largas lo cual genera inestabilidad numérica. ("How Do You Choose between RNN and LSTM for Natural Language Processing Tasks?," 2023)

LSTM (Long Short Term Memory):

Es un tipo de red neuronal recurrente, este modelo permite retener información por un largo periodo de tiempo. Además, se usa para predecir y clasificar en base a datos de una serie de tiempo. Está diseñada precisamente para manejar datos secuenciales y es capaz de aprender de dependencias a largo plazo. ("Deep Learning Introduction to Long Short Term Memory," 2019)

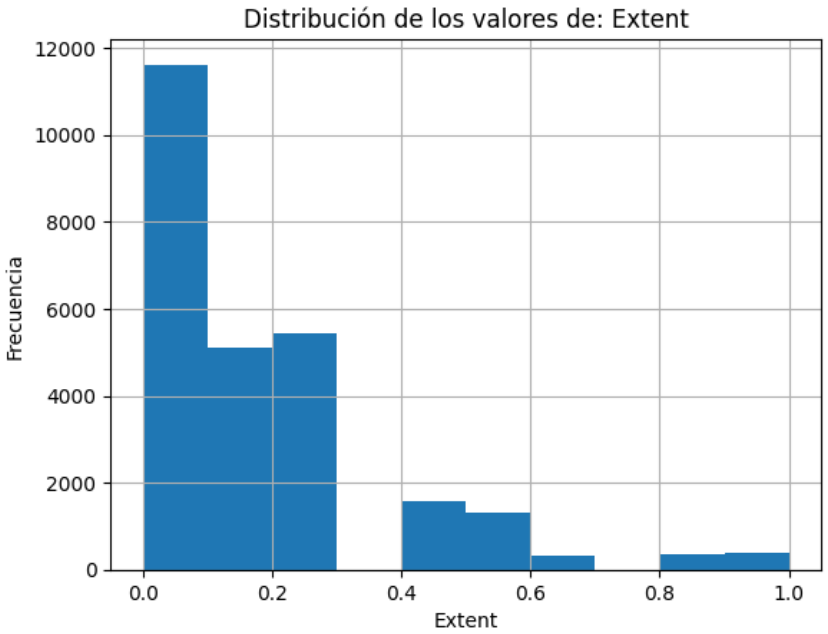
A comparación de la red neuronal recurrente este tipo de red evita el problema de explosión y desvanecimiento del gradiente y soporta entrenar con secuencias más largas. Por otra parte es menos propensa a sobre ajustarse en comparación a una RNN. No obstante, este tipo de red puede tener un mayor tiempo computacional. ("How Do You Choose between RNN and LSTM for Natural Language Processing Tasks?," 2023)

Tanto una Red Neuronal Recurrente (RNN) como una red de tipo LSTM pueden ser de ayuda para la predicción de la extensión del área infectada y establecer la extensión de daño en un cultivo, dado que son datos secuenciales en una serie de tiempo.

Modelos utilizados

Para resolver este problema se hicieron modelos de regresión y clasificación. Los modelos de clasificación se utilizaron la columna *extent* como las etiquetas objetivo. En total se separaron 11 etiquetas, cada valor representa un incremento de 10%. De esta forma se tienen todas las posibles opciones para el valor de *extent*.

Hay que mencionar que los datos están desbalanceados, la etiqueta con valor 0 tiene la mayoría de las imágenes. Esto se verá reflejado en los resultados de los modelos.



1. Red Neuronal Convolucional: Se implementó una *CNN* simple con aumentación de datos. La arquitectura de la red también incluye una implementación de *dropout* para evitar el overfitting.

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 180, 180, 3)	0
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_1 (Conv2D)	(None, 90, 90, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_2 (Conv2D)	(None, 45, 45, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 64)	0
dropout (Dropout)	(None, 22, 22, 64)	0
flatten (Flatten)	(None, 30976)	0
dense (Dense)	(None, 128)	3965056
dense_1 (Dense)	(None, 1)	129

Total params: 3988769 (15.22 MB)
Trainable params: 3988769 (15.22 MB)
Non-trainable params: 0 (0.00 Byte)

Se entrenó al modelo con 4 épocas, los resultados de este entrenamiento se muestran en la siguiente imágen.

Estos resultados no se pueden utilizar en la práctica dado que el *accuracy* tiene un valor bajo (arriba de 0.42 en el mejor de los casos). Asimismo se puede observar que el modelo empieza a sobre ajustarse a partir de la segunda época.

2. Red Neuronal Convolucional (Arquitectura ResNet): Se implementó una Red Neuronal Convolucional con arquitectura ResNet con el propósito de evitar el problema de desvanecimiento y explosión del gradiente dado que la arquitectura permite las conexiones residuales que permite que las señales de entrada fluyan a través de capas más profundas y tener una mayor precisión.

```
Epoch 1/5
483/483 [==============================] - 14s 29ms/step - loss: 1.0985 - accuracy: 0.5927 - val_loss: 1.9492 - val_accuracy: 0.3863
Epoch 2/5
483/483 [==============================] - 14s 28ms/step - loss: 0.8428 - accuracy: 0.6855 - val_loss: 2.0644 - val_accuracy: 0.3915
Epoch 3/5
483/483 [==============================] - 13s 27ms/step - loss: 0.5967 - accuracy: 0.7818 - val_loss: 2.8124 - val_accuracy: 0.3803
Epoch 4/5
483/483 [==============================] - 14s 28ms/step - loss: 0.4174 - accuracy: 0.8521 - val_loss: 3.5245 - val_accuracy: 0.3879
Epoch 4: early stopping
```

Se obtuvo un buen porcentaje de precisión en el conjunto de entrenamiento pero no un buen porcentaje de precisión para el conjunto de validación.

Se muestra un claro sobreajuste para los modelos.

3. Red Neuronal Fully Connected. Se implementó una arquitectura simple de una NN con dos capas Lineales, siendo la capa oculta con ReLu. Se entrenó el modelo durante 1000 épocas. El input de la red son el groth state y el Damage de la planta, el output era una regresión lineal de 0 a 100 para la extensión del daño

```python
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(10, 64)  # Capa oculta
        self.fc2 = nn.Linear(64, 1)   # Capa de salida

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

```
Epoch [1/1000], Training Loss: 592.3436, Validation Loss: 304.4400
Epoch [101/1000], Training Loss: 121.9725, Validation Loss: 180.8509
Epoch [201/1000], Training Loss: 139.8675, Validation Loss: 180.6762
Epoch [301/1000], Training Loss: 203.3805, Validation Loss: 180.9379
Epoch [401/1000], Training Loss: 149.9240, Validation Loss: 181.6129
Epoch [501/1000], Training Loss: 260.3688, Validation Loss: 181.1832
Epoch [601/1000], Training Loss: 209.8331, Validation Loss: 180.8877
Epoch [701/1000], Training Loss: 301.4784, Validation Loss: 181.0817
Epoch [801/1000], Training Loss: 177.9344, Validation Loss: 181.8973
Epoch [901/1000], Training Loss: 127.5555, Validation Loss: 181.2428
```

4. Red Neuronal Fully Connected. Se implementó una arquitectura simple de una NN con dos capas Lineales, siendo la capa oculta con ReLu. Se entrenó el modelo durante 1000 épocas. El input de la red son el groth state y el Damage de la planta, el output es una label objetivo del damage extend

```
Epoch [1/1000], Training Loss: 1.1935, Validation Loss: 0.9764
Epoch [101/1000], Training Loss: 1.1496, Validation Loss: 0.9473
Epoch [201/1000], Training Loss: 0.9797, Validation Loss: 0.9507
Epoch [301/1000], Training Loss: 0.8369, Validation Loss: 0.9528
Epoch [401/1000], Training Loss: 0.8397, Validation Loss: 0.9562
Epoch [501/1000], Training Loss: 1.0711, Validation Loss: 0.9604
Epoch [601/1000], Training Loss: 0.9531, Validation Loss: 0.9631
Epoch [701/1000], Training Loss: 1.0963, Validation Loss: 0.9653
Epoch [801/1000], Training Loss: 1.0105, Validation Loss: 0.9664
Epoch [901/1000], Training Loss: 0.9771, Validation Loss: 0.9695
```

-

Discusión:
En base a los resultados obtenidos se pudo observar que en los modelos generados se obtuvo un sobreajuste. Sin embargo en el modelo de regresión se obtuvo un alto valor para el error cuadrático medio, se puede apreciar que en la etapa de entrenamiento se perdió mucha data en las primeras dos épocas, mientras que en la validación hubo pérdida mínima, posiblemente fue la cantidad de épocas utilizadas que fue muy poco.

Un factor que afectó mucho el análisis y la clasificación fue que todas las imágenes a analizar son de diferentes tamaños, fueron tomadas con diferentes ángulos, tienen diferente resolución y esto hace que sea más difícil de clasificar para los modelos implementados y ver el estado de las plantas por medio de las imágenes.

Conclusiones:

- Las imágenes que se tienen y el desbalance del dataset provoca que no se pueden generalizar los datos.
- Para lograr una mejor clasificación se debería hacer un ajuste a los datos para una mejor clasificación.
- Un modelo clasificatorio con 10 etiquetas no fue eficiente para generalizar la información.
- La decisión de la competencia de prohibir el uso de la columna “tipo de daño” perjudicó los resultados.

Enlace de github: https://github.com/MGonza20/Proyecto2_DataScience
Enlace de Google docs:
https://docs.google.com/document/d/1u1W-2XdlDAissXbhWRaVHuF6NISpnGYQdlqDJMCxmrI/edit?usp=sharing
Enlace de presentación de canva:
https://www.canva.com/design/DAFyweIAVFE/_4ASZwn2MY6q9R3B6yAWHQ/edit

Referencias:

Mishra, M. (2020, Augosto 26). Convolutional Neural Networks, Explained - Towards Data Science. Recuperado de:

https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

Sanghvi, K. (2023, Abril 21). Image Classification Techniques - Analytics Vidhya - Medium. *Medium*. Recuperado de: https://medium.com/analytics-vidhya/image-classification-techniques-83fd87011cac#:~:text=The%20algorithms%20include%20linear%20regression,%2C%20and%20k%2Dnearest%20neighbor.

Residual Networks ResNet Deep Learning. (2020, Junio 3). Recuperado de:

https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

What are Recurrent Neural Networks? | IBM. (2023). Ibm.com. Recuperado de: https://www.ibm.com/topics/recurrent-neural-networks

Deep Learning Introduction to Long Short Term Memory. (2019, Enero 16). Recuperado de:

https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/

How do you choose between RNN and LSTM for natural language processing tasks? (2023). Recuperado de:

https://www.linkedin.com/advice/0/how-do-you-choose-between-rnn-lstm-natural-language