Tecnológico de Monterrey

Reconocimiento del clima mediante redes neuronales convolucionales

Alumno: Rafael Hinojosa López

Matrícula: A01705777

TC3002B.301 Desarrollo de aplicaciones avanzadas de ciencias computacionales

Profesores:

Benjamín Valdés Aguirre

Pedro Óscar Pérez Murueta

Manuel Casillas

02 de junio del 2023

Campus Querétaro

Problema

En el mundo existen muchos fenómenos del clima: soleado, nublado, lluvioso, neblina, tormentoso, entre otros. La necesidad para identificar a cuál clima pertenece una imagen se encuentra en áreas como la agricultura, el monitoreo del clima, el estado actual de las vialidades en las ciudades, las condiciones para realizar eventos al aire libre y más.

Ante este contexto, es necesario generar una herramienta que, al leer una imagen, la clasifique correctamente en el clima que representa y así tomar las decisiones pertinentes según sea el caso de aplicación. Para lograrlo, se crearon tres arquitecturas para obtener el modelo más preciso de clasificación de imágenes del clima, como se mostrará a continuación.

Set de datos

Utilizamos un set de datos llamado *Weather Phenomenon Database (WEAPD)* que contiene 6862 imágenes de tipo *.jpg*. Estas imágenes están clasificadas en 11 grupos, cada uno representando un tipo de clima: *dew, fogsmog, Frost, glaze, hail, lightning, rain, rainbow, rime, sandstorm* y *snow.* Estas imágenes fueron utilizadas en el trabajo de investigación *Classification of Weather Phenomenon From Images by Using Deep Convolutional Neural Network*, realizado por Xiao, Zhang, Shen, Wu y Zhang en el 2021, para el volumen 8 de la revista científica *AGU Earth and Space Science*.

Cabe mencionar que las fotos no tienen un tamaño uniforme, cada una tiene tamaños diferentes, por lo que se realizó un preprocesado de datos, así como la separación de estos para entrenar, validar y probar los modelos.

Preprocesado y separación de los datos

Los datos pasaron por tres procesos para ser utilizadas: reajuste de tamaño, separación y aumentación. Primero, se reajustó el tamaño de todas las imágenes a 76px x 76px con la librería *OpenCV* para que su tamaño coincidiera con el *target_size* especificado en el recolector de imágenes para entrenamiento, validación y pruebas. Este atributo pide el tamaño de las imágenes a utilizar y es por esta razón que es importante reajustarlas antes de entrenar a los modelos, para que haya consistencia con el tamaño de las imágenes y no haya errores.

Seguido de esto, dividimos el set de datos en tres grupos: entrenamiento (*train*), validación (*validation*) y pruebas (*test*). El 70% de las imágenes fueron separadas para entrenamiento, el 10% para validación y el 20% restante para pruebas. Esto permitirá tener un amplio set de datos para entrenar al modelo y mientras se entrena, hacer validaciones con una pequeña parte de las imágenes para poder realizar ajustes a los hiper parámetros del modelo antes de continuar a la fase de pruebas.

Finalmente, se le modificaron ciertas características a las imágenes, como se explicará a continuación.

Modelos

Se implementó un modelo con 3 diferentes arquitecturas. La primera es una red neuronal convolucional que es entrenado con imágenes sin modificar, la segunda, es una red neuronal convolucional entrenada con imágenes aumentadas y finalmente se utilizó VGG16, una red neuronal convolucional con 16 capas de profundidad con pesos ya asignados a sus neuronas, lo que nos permitirá obtener mayor precisión en la predicción de las clases de las imágenes.

Simple CNN

El modelo simple utiliza el modelo presentado, con los parámetros mostrados para ingresar las imágenes, entrenar el modelo y obtener la precisión con base al set de pruebas. Este modelo es la base de los siguientes dos.

Simple CNN

Train generator

```
train_datagen = ImageDataGenerator(rescale=1./255)
train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size = (WIDTH, HEIGHT),
    batch_size = 8,
    class_mode ='categorical',
)
```

```
Found 4809 images belonging to 11 classes.
```

Se utilizaron 4809 imágenes para el set de entrenamiento, y se utilizó un *batch_size* de 8.

Create model

```
model = models.Sequential()

model.add(layers.Conv2D(10, (3, 3), activation="relu", input_shape = (WIDTH,HEIGHT,3)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
# As sigmoid is used for binary classification, softmax is for multiclass purposes
model.add(layers.Dense(11, activation='softmax'))

model.summary()

model.compile(
    loss='categorical_crossentropy',
    optimizer='Adam',
    metrics=['acc']
)
```

Este modelo fue la base de todos los modelos. Se utiliza 1 capa convolucional 2D con 10 matrices y un *kernel* de 3 x 3. Seguido de este, se implementa una capa *Flatten* en la que se tiene un arreglo 1D

con la información obtenida de la capa convolucional anterior. Finalmente, se cuentan con 2 capas densas: una con 64 neuronas, y la capa de salida con 11 neuronas, representando las 11 clases contempladas con una función de activación *softmax*. Se optó por esta función en vez de la de *sigmoid* porque *sigmoid* es utilizada para clasificación binaria y *softmax*, para multiclasificación (más de 2 clases).

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 74, 74, 10)        280
_________________________________________________________________
flatten (Flatten)            (None, 54760)             0
_________________________________________________________________
dense (Dense)                (None, 64)                3504704
_________________________________________________________________
dense_1 (Dense)              (None, 11)                715
=================================================================
Total params: 3,505,699
Trainable params: 3,505,699
Non-trainable params: 0
_________________________________________________________________
```

Finalmente, el modelo se entrenó con 15 épocas y 80 pasos por época, siendo que cada paso consiste en leer un *batch*, de tamaño 8 (8 imágenes por batch y 80 pasos por época).

```
history = model.fit_generator(
    train_generator,
    steps_per_epoch = 80,
    epochs = 15
)
```

Data Augmentation

Para el aumento de datos, creamos imágenes aumentadas con base en las que ya se tenían guardadas. Se rotaron, cortaron (con *shear_range*), acercaron (con *zoom_range*), voltearon horizontalmente (con *horizontal_flip*), desplazaron horizonal y verticalmente (con *width_shift_range* y *height_shift_range*, respectivamente) para obtener un rango mayor de imágenes y trabajar con una variedad de datos mayor.

Estas son las modificaciones realizadas para el aumento de datos:

```
train_aug_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.3,
    zoom_range = 0.3,
    horizontal_flip = True
)
```

El modelo se conservó igual que el *Simple CNN*, pero con un *data generator* que contiene imágenes aumentadas.

Transfer Learning

Basado en la investigación sobre detección de tumores en el cerebro, de Pokharel, Rakjarnikar, Karn y Shrestha (2021), utilizamos VGG16, el cual es una arquitectura utilizada en el concurso ImageNet y cuyos pesos ya han sido entrenados, por lo que se obtiene ventaja de ellos para con el modelo actual. El utilizar esta arquitectura implica que estamos empleando *Transfer Learning* y la aplicamos al modelo actual.

```
vgg_conv_base= VGG16(
    weights='imagenet',
    include_top = False,
    input_shape = (WIDTH,HEIGHT,3)
)
```

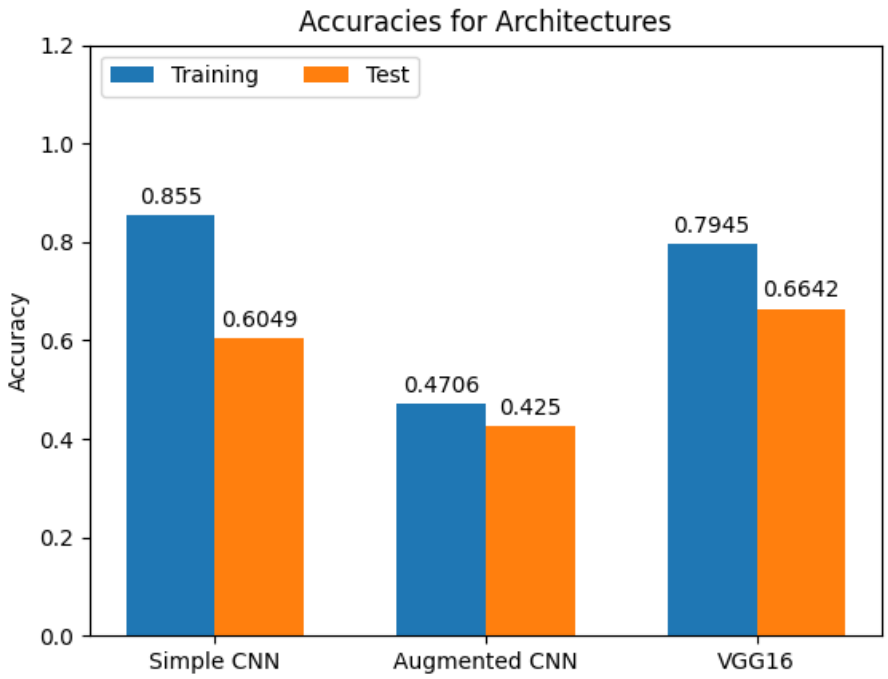
```
model_vgg = models.Sequential()

model_vgg.add(vgg_conv_base)
model_vgg.add(layers.Flatten())
model_vgg.add(layers.Dense(64,activation='relu'))
model_vgg.add(layers.Dense(11, activation='softmax'))

vgg_conv_base.trainable = False

model_vgg.summary()

model_vgg.compile(
    loss='categorical_crossentropy',
    optimizer='Adam',
    metrics=['acc']
)
```

En contraste con el *Simple CNN,* este modelo implementó la base ya hecha de VGG16 y se declaró no entrenable, pues queríamos los pesos que ya traía, no modificarlos.

Análisis de resultados

Comparación de precisión

Basándonos en la precisión de los 3 modelos, tanto en entrenamiento como en pruebas, obtenemos la siguiente gráfica:



Comparación de la precisión en entrenamiento y pruebas de los 3 modelos

Podemos observar que el modelo *Simple CNN* tiene *overfitting*, por lo que se buscó mejorarlo con un set de imágenes aumentadas. Como se observa, el set de datos aumentados tiene una diferencia del 5%, por lo que lo que no hay *overfitting*; sin embargo, si tiene *underfitting* por su baja precisión del 42%. Por otra parte, el modelo que utiliza la arquitectura VGG16, tiene una diferencia del 13.02% con el set de entrenamiento y de pruebas, aunque, obtuvo la mejor precisión de los 3, al ser de 66.42%. Podemos apreciar que esta arquitectura con pesos predefinidos incrementó en gran manera la precisión con respecto a sus antecesores.

Matriz de confusión

```
generate_confusion_matrix(model)
```

```
<tf.Tensor: shape=(11, 11), dtype=int32, numpy=
array([[108,   0,   2,   5,   0,   2,   0,   2,   1,   1,   0],
       [  0, 148,   1,   2,   1,   9,  10,  14,  10,  36,   5],
       [  1,   0,  28,   4,   6,   1,   1,   0,   2,   0,   3],
       [  2,   0,  11,  51,   9,   1,  11,   0,   4,   0,   4],
       [  5,   2,   7,   8,  70,  14,   9,   5,   4,   0,   2],
       [  0,   0,   0,   0,   0,  13,   0,   0,   0,   0,   0],
       [ 10,   0,  16,  15,  11,   0,  40,   2,   4,   1,   7],
       [  0,   1,   0,   0,   0,   2,   0,   9,   0,   1,   0],
       [  8,   7,  14,  23,   4,  28,   9,   9, 183,   4,  20],
       [  2,   6,   0,   0,   1,   0,   0,   4,   0,  95,   0],
       [  3,   5,  16,  19,  16,   5,  25,   1,  24,   0,  82]])>
```

Matriz de confusión de Simple CNN

```
generate_confusion_matrix(model_aug)
```

```
<tf.Tensor: shape=(11, 11), dtype=int32, numpy=
array([[ 76,   0,   0,   2,   0,   2,   0,   0,   0,   0,   0],
       [  1, 112,   0,   0,   0,   7,   4,  11,   1,  35,   1],
       [ 16,   0,  15,  11,   7,   1,   2,   0,   1,   0,   1],
       [  5,   0,  30,  27,  11,   2,   6,   0,   7,   0,   1],
       [  3,   1,   1,   0,  22,   4,   2,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   3,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  1,   0,   0,   0,   0,   1,   0,   3,   0,   0,   0],
       [ 12,  51,  16,  36,  31,  53,  27,  28, 193,  32,  60],
       [  4,   5,   0,   0,   1,   0,   0,   2,   0,  70,   0],
       [ 21,   0,  33,  51,  46,   2,  64,   2,  30,   1,  60]])>
```

Matriz de confusión de Augmented Model

```
generate_confusion_matrix(model_vgg)
```

```
<tf.Tensor: shape=(11, 11), dtype=int32, numpy=
array([[ 90,   0,   4,   6,   3,   1,   1,   4,   1,   2,   0],
       [  4, 135,   2,   2,   0,   4,   6,   1,   7,  24,   6],
       [ 12,   0,  35,   8,   6,   0,   0,   0,   2,   1,   3],
       [ 11,   1,  25,  64,   8,   0,   1,   0,  22,   0,  10],
       [  8,   0,   6,   5,  88,   3,   5,   3,   0,   0,   3],
       [  3,   5,   4,   1,   1,  56,   2,   3,   1,   3,   1],
       [  0,   1,   1,   1,   0,   0,  61,   0,   0,   2,   6],
       [  3,   4,   0,   0,   1,   0,   2,  33,   2,   2,   1],
       [  1,   8,  15,  30,   4,   6,   5,   2, 183,   7,  18],
       [  2,   8,   1,   1,   1,   4,   3,   0,   3,  90,   2],
       [  5,   7,   2,   9,   6,   1,  19,   0,  11,   7,  73]])>
```

Matriz de confusión de VGG16

Las 3 matrices de confusión tienen un punto en común que sobre sale de los demás y es la clase *rime* (antepenúltima lista). Esta clase es la que contiene más imágenes de las 11, con 1160 (la siguiente en el orden tiene 851). Por esta y otras condiciones, se podría considerar analizar a fondo por qué esta clase falla mucho y eliminarla para observar la precisión de los modelos sin ella. Así como esta, la clase *snow* (última lista) tiene casi todas sus imágenes predichas como todas las otras clases.

Oportunidades de mejora

La siguiente es una lista con cómo se podría mejorar el modelo actual:

1. Como se observó, se tiene *overfitting* en el modelo que utiliza *VGG16*, por lo que se pueden agregar 2 capas de *Dropout* entre las capas *Dense* para ignorar aleatoriamente un porcentaje de las neuronas y evitar que el modelo memorice resultados durante el entrenamiento.
2. Aumentar el tamaño de las imágenes de 76px a 224px, tal como lo hicieron Xiao, Zhang, et al. (2021), para que VGG16 mejore su predicción en las imágenes. Esto se debe a que VGG16 contiene capas de *pooling*, que producen imágenes aún más pequeñas que las originales representando características de las imágenes, por lo que al tener menos pixeles en las imágenes se pueden perder detalles que se preservarían con imágenes más grandes. Por otro lado, el aumento del tamaño permitirá que se pierda menos información en la aumentación de las imágenes, permitiendo que estas conserven ciertos rasgos importantes del clima que representan.
3. Activar como *True* el atributo *shuffle* en los *data generators* para que las imágenes se lean en orden aleatorio y evitemos que le modelo aprenda a clasificar una imagen basándose en el orden de aparición de estas y se enfoque en los patrones que representa.

4. Aumentar el *batch_size* de 8 a 32, basándonos en la investigación de Xiao, Zhang, et al. (2021). Esto permitirá que la red neuronal se actualice cada 32 imágenes procesadas, en vez de cada 8.

Referencias

Xiao, H., Zhang, F., Shen, Z., Wu, K., & Zhang, J. (2021). Classification of weather phenomenon from images by using deep convolutional neural network. *Earth and Space Science*, 8, e2020EA001604. https://doi.org/10.1029/2020EA001604

Pokharel, M., Rajkarnikar, L., Karn, G., Shrestha, S. (2021) Detection of Brain Tumor using VGG-16 Architecture. International *Journal of Advanced Engineering, 4.* https://ictaes.org/wp-content/uploads/2021/09/IJAE-Vol.04-No.02/IJAE-V4No2-7.pdf?ckattempt=1