Informe Final

Fundamentos de Deep Learning

Estudiante:

Carlos Alfredo Pinto Hernández

Profesor:

Raúl Ramos Pollán

Universidad de Antioquia

Facultad de Ingeniería

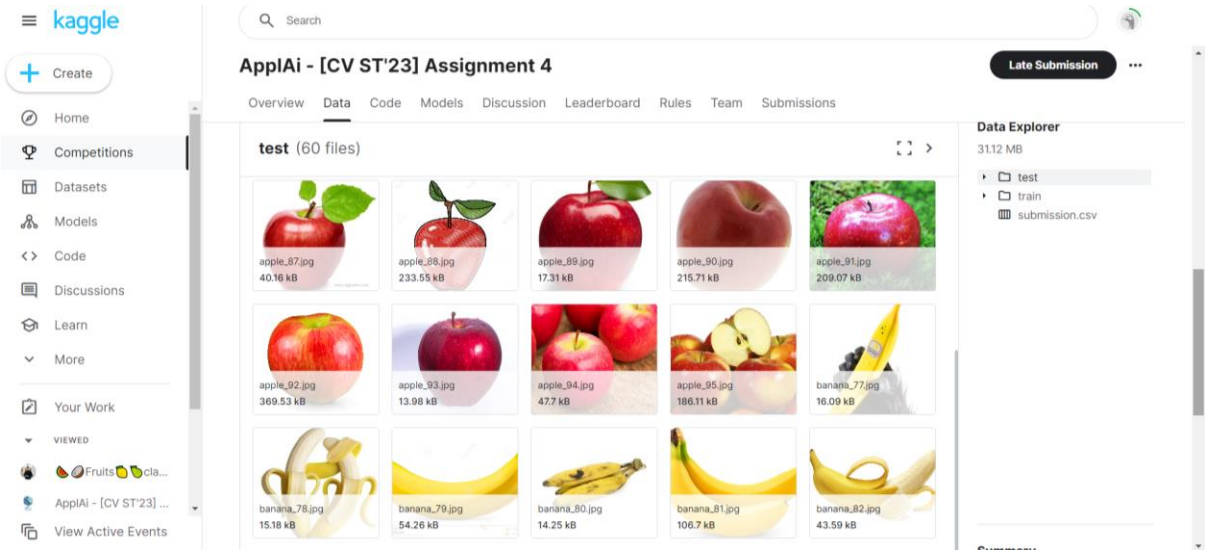
Departamento de Ingeniería de Sistemas

2024

Contexto de aplicación.

Las redes neuronales convolucionales (CNN), las cuales han supuesto una revolución en el sector del reconocimiento de imágenes, ya que, a diferencia de las redes neuronales convencionales y otros algoritmos de clasificación de imágenes, usan un procesamiento relativamente pequeño.

La aplicación pretende, a través de técnica de detección de objetos con algoritmos de Deep learning, realizar una clasificación de imágenes de frutas utilizadas en la competición de kaggle “ApplAi - [CV ST'23] Assignment 4”.



Dataset: tipo de datos, tamaño (número de datos y tamaño en disco), distribución de las clases

El dataset se obtiene de la competición de kaggle denominada “ApplAi - [CV ST'23] Assignment 4”, cuenta con 300 imágenes (240 de entrenamiento y 60 de prueba), el tamaño de los archivos es de 31.12MB en formatos .jpg y .xml

1. Descripción de la estructura de los notebooks

Para este trabajo de reconocimiento de unas imágenes de frutas en tres categorías: manzanas, bananos y naranjas. Se contaba con 270 imágenes de frutas simples y 30 imágenes de frutas compuestas, se definió eliminar estas últimas. Se utilizaron dos notebooks:

1.1 **Extracción de características:** A través de este notebook y con ayuda de la libería OpenCV (librería con funciones de procesamiento de imágenes y video) se obtuvo el arreglo vector de las imágenes previamente estandarizadas tanto en colores como en tamaño para hacer el procesamiento. Estos generaban

uno arreglo de los elementos de los colores en cada uno de los pixeles de la imagen y otro arreglo con la categoría a la que pertenecía. Con el algoritmo se generaba en otra carpeta estos vectores que sería utilizados en la aplicación de los modelos.

```
# Recorrer todas las clases
for clase in clases:
    path = os.path.join(rutaProyecto, 'BASE_DATOS', clase)

    # Cargar las imágenes de la clase
    listaImagenes = os.listdir(path)
    # Recorrer todas las imágenes de la clase
    for imagen in listaImagenes:
        # Leer cada imagen
        imgOriginal = cv.imread(os.path.join(path, imagen), 1)
        if imgOriginal is None or imgOriginal.shape == (0, 0):
            print("Error!: " + os.path.join(path, imagen))
        else:
            imgFiltered = cv.medianBlur(imgOriginal, 3)
            imgResize = cv.resize(imgFiltered, (128, 128), interpolation=cv.INTER_AREA)
            vectorImagenes.append(imgResize)
            vectorSalida.append(clases_dict[clase])
```

1.2 **Aplicación de modelos**: en este notebook se toman los dos vectores resultantes del notebook anterior y se procesaba.

Cargar Vector Imágenes y vector de Salidas

```
X = np.load(rutaPath + '/vectorImagenesCNN.npy')
print(X.shape)

y = np.load(rutaPath + '/vectorYCNN.npy')
print(y.shape)

```

Luego de normalizar los arreglos de las X y categorizar las salidas en un vector de 3 posiciones.

Salidad Categórica - Generar Array y

```
y=np.array(y)
y_categorical = to_categorical(y, num_classes=3)

## 0: Apple
## 1: Banana
## 2: Orange
```

Posterior se hacía la distribución de las imágenes para el entrenamiento y la prueba en una relación 70%/30%

Selección del Train/Test

```
# 70% Train y 30% Test
x_train, x_test, y_train, y_test = train_test_split(X_norm, y_categorical, test_size=0.3,random_state=109)
print(x_train.shape)
print(y_test.shape)
```

```
(192, 128, 128, 3)
(83, 3)
```

Luego se aplicó un modelo tipo red neuronal convolucional (CNN, por sus siglas en inglés) y otro modelo de red neuronal profunda.

2. Descripción de la solución

Se hizo el análisis de dos modelos: se crea una red neuronal de convulsión con 9 capas descritas así:

- Input Layer: La capa de entrada con forma (128, 128, 3), que corresponde a una imagen RGB de 128x128 píxeles
- Conv2D (3x3): Una capa convolucional con 30 filtros de tamaño 3x3 y función de activación ReLU.
- MaxPooling2D (2x2): Una capa de max-pooling con un tamaño de ventana de 2x2.
- Conv2D (5x5): Una capa convolucional con 30 filtros de tamaño 5x5 y función de activación ReLU.
- MaxPooling2D (2x2): Una capa convolucional con 30 filtros de tamaño 5x5 y función de activación ReLU.
- Flatten: Aplana las salidas de las capas anteriores para convertirlas en un vector de una dimensión
- Dense (60 neuronas): Una capa completamente conectada con 60 neuronas y función de activación ReLU
- Dropout (0.2): Una capa de dropout con una tasa de 0.2 para evitar el sobreajuste
- Dense (num_classes neuronas, capa de salida): La capa de salida con un número de neuronas igual a num_classes y función de activación softmax para la clasificación.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0

 conv2d (Conv2D)             (None, 126, 126, 30)      840

 pooling (MaxPooling2D)      (None, 63, 63, 30)        0

 conv2d_1 (Conv2D)           (None, 59, 59, 30)        22530

 pooling2 (MaxPooling2D)     (None, 29, 29, 30)        0

 flatten (Flatten)           (None, 25230)             0

 dense (Dense)               (None, 60)                1513860

 dropout (Dropout)           (None, 60)                0

 output_1 (Dense)            (None, 3)                 183

=================================================================
Total params: 1537413 (5.86 MB)
Trainable params: 1537413 (5.86 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

El otro modelo que se tomó fue ResNet152 el cual es una red neuronal profunda y poderosa que se beneficia de las conexiones residuales para entrenar eficientemente redes extremadamente profundas y lograr un alto rendimiento en tareas de visión por computadora. Con este se quería probar otro tipo de modelos para verificar las métricas obtenidas.

3. Descripción de iteraciones realizadas y resultados

En el desarrollo del modelo de realizaron múltiples iteraciones alcanzando para el primer modelo un accuracy de 0.9156 y para el segundo modelo 0.9036

Primer modelo

```
3/3 [==============================] - 0s 47ms/step - loss: 0.0037 - accuracy: 1.0000 - val_loss: 0.2291 - val_accuracy: 0.9036
Epoch 30/50
3/3 [==============================] - 0s 46ms/step - loss: 0.0042 - accuracy: 1.0000 - val_loss: 0.2266 - val_accuracy: 0.9277
Epoch 31/50
3/3 [==============================] - 0s 47ms/step - loss: 0.0029 - accuracy: 1.0000 - val_loss: 0.2281 - val_accuracy: 0.9277
Epoch 32/50
3/3 [==============================] - 0s 48ms/step - loss: 0.0037 - accuracy: 1.0000 - val_loss: 0.2257 - val_accuracy: 0.9277
Epoch 33/50
3/3 [==============================] - 0s 48ms/step - loss: 0.0028 - accuracy: 1.0000 - val_loss: 0.2291 - val_accuracy: 0.9036
Epoch 34/50
3/3 [==============================] - 0s 67ms/step - loss: 0.0053 - accuracy: 1.0000 - val_loss: 0.2253 - val_accuracy: 0.9398
Epoch 35/50
3/3 [==============================] - 0s 45ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.2335 - val_accuracy: 0.9157
Epoch 36/50
3/3 [==============================] - 0s 47ms/step - loss: 0.0025 - accuracy: 1.0000 - val_loss: 0.2380 - val_accuracy: 0.9157
Epoch 37/50
3/3 [==============================] - 0s 46ms/step - loss: 0.0028 - accuracy: 1.0000 - val_loss: 0.2335 - val_accuracy: 0.9277
Epoch 38/50
3/3 [==============================] - 0s 46ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.2494 - val_accuracy: 0.9157
Epoch 39/50
3/3 [==============================] - 0s 46ms/step - loss: 0.0033 - accuracy: 1.0000 - val_loss: 0.2834 - val_accuracy: 0.9036
Epoch 40/50
3/3 [==============================] - 0s 46ms/step - loss: 0.0053 - accuracy: 1.0000 - val_loss: 0.2550 - val_accuracy: 0.9157
Epoch 41/50
3/3 [==============================] - 0s 48ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.2621 - val_accuracy: 0.9277
Epoch 42/50
3/3 [==============================] - 0s 45ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.2698 - val_accuracy: 0.9277
Epoch 43/50
3/3 [==============================] - 0s 66ms/step - loss: 0.0030 - accuracy: 1.0000 - val_loss: 0.2517 - val_accuracy: 0.9157
Epoch 44/50
3/3 [==============================] - 0s 44ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.2670 - val_accuracy: 0.9157
Epoch 45/50
3/3 [==============================] - 0s 48ms/step - loss: 0.0031 - accuracy: 1.0000 - val_loss: 0.2738 - val_accuracy: 0.9157
Epoch 46/50
3/3 [==============================] - 0s 46ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.2543 - val_accuracy: 0.9277
Epoch 47/50
3/3 [==============================] - 0s 46ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.2403 - val_accuracy: 0.9277
Epoch 48/50
3/3 [==============================] - 0s 66ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.2373 - val_accuracy: 0.9398
Epoch 49/50
3/3 [==============================] - 0s 50ms/step - loss: 0.0036 - accuracy: 1.0000 - val_loss: 0.2430 - val_accuracy: 0.9277
Epoch 50/50
3/3 [==============================] - 0s 43ms/step - loss: 9.2452e-04 - accuracy: 1.0000 - val_loss: 0.2605 - val_accuracy: 0.9157
3/3 [==============================] - 0s 10ms/step - loss: 0.2605 - accuracy: 0.9157
{'loss': 0.26047244668006897, 'accuracy': 0.9156626462936401}
```

Segundo modelo

```
Epoch 980/1000
12/12 [==============================] - 2s 203ms/step - loss: 0.4152 - accuracy: 0.8438 - val_loss: 0.3070 - val_accuracy: 0.9036
Epoch 981/1000
12/12 [==============================] - 2s 201ms/step - loss: 0.4357 - accuracy: 0.8490 - val_loss: 0.3050 - val_accuracy: 0.8916
Epoch 982/1000
12/12 [==============================] - 2s 203ms/step - loss: 0.4638 - accuracy: 0.8125 - val_loss: 0.3083 - val_accuracy: 0.8916
Epoch 983/1000
12/12 [==============================] - 2s 199ms/step - loss: 0.4198 - accuracy: 0.8438 - val_loss: 0.3058 - val_accuracy: 0.8916
Epoch 984/1000
12/12 [==============================] - 2s 210ms/step - loss: 0.3963 - accuracy: 0.8646 - val_loss: 0.3134 - val_accuracy: 0.9036
Epoch 985/1000
12/12 [==============================] - 2s 205ms/step - loss: 0.3860 - accuracy: 0.8490 - val_loss: 0.3144 - val_accuracy: 0.8916
Epoch 986/1000
12/12 [==============================] - 2s 204ms/step - loss: 0.3173 - accuracy: 0.9167 - val_loss: 0.3208 - val_accuracy: 0.9036
Epoch 987/1000
12/12 [==============================] - 2s 200ms/step - loss: 0.3995 - accuracy: 0.8750 - val_loss: 0.3192 - val_accuracy: 0.9036
Epoch 988/1000
12/12 [==============================] - 2s 199ms/step - loss: 0.4258 - accuracy: 0.8542 - val_loss: 0.3158 - val_accuracy: 0.9036
Epoch 989/1000
12/12 [==============================] - 2s 206ms/step - loss: 0.4020 - accuracy: 0.8542 - val_loss: 0.3152 - val_accuracy: 0.9036
Epoch 990/1000
12/12 [==============================] - 3s 211ms/step - loss: 0.5086 - accuracy: 0.7708 - val_loss: 0.3164 - val_accuracy: 0.9036
Epoch 991/1000
12/12 [==============================] - 2s 205ms/step - loss: 0.4251 - accuracy: 0.8125 - val_loss: 0.3174 - val_accuracy: 0.8916
Epoch 992/1000
12/12 [==============================] - 2s 203ms/step - loss: 0.3789 - accuracy: 0.8490 - val_loss: 0.3191 - val_accuracy: 0.8916
Epoch 993/1000
12/12 [==============================] - 2s 199ms/step - loss: 0.4391 - accuracy: 0.8229 - val_loss: 0.3156 - val_accuracy: 0.8916
Epoch 994/1000
12/12 [==============================] - 2s 203ms/step - loss: 0.3753 - accuracy: 0.8646 - val_loss: 0.3163 - val_accuracy: 0.8916
Epoch 995/1000
12/12 [==============================] - 2s 209ms/step - loss: 0.3737 - accuracy: 0.8646 - val_loss: 0.3147 - val_accuracy: 0.9036
Epoch 996/1000
12/12 [==============================] - 2s 208ms/step - loss: 0.4115 - accuracy: 0.8802 - val_loss: 0.3083 - val_accuracy: 0.9036
Epoch 997/1000
12/12 [==============================] - 2s 199ms/step - loss: 0.3979 - accuracy: 0.8385 - val_loss: 0.3087 - val_accuracy: 0.9036
Epoch 998/1000
12/12 [==============================] - 2s 204ms/step - loss: 0.4287 - accuracy: 0.8750 - val_loss: 0.3079 - val_accuracy: 0.9036
Epoch 999/1000
12/12 [==============================] - 2s 199ms/step - loss: 0.4005 - accuracy: 0.8385 - val_loss: 0.3069 - val_accuracy: 0.9036
Epoch 1000/1000
12/12 [==============================] - 2s 204ms/step - loss: 0.3765 - accuracy: 0.8802 - val_loss: 0.3068 - val_accuracy: 0.9036
Model: "sequential"
```