Tarea: Arquitectura Red Neuronal Convolucional

Serapio Hernández Alexis Arturo
Facultad de Ingeniería
UNAM
Ciudad de México, México
alexis.serapio@ingenieria.unam.

García López Erik
Facultad de Ingeniería
UNAM
Ciudad de México, México
erikpumas999@gmail.com

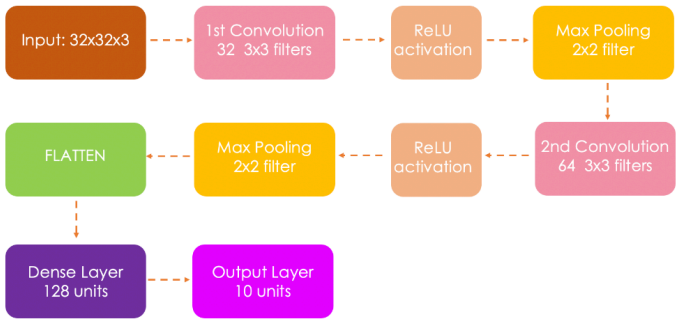
Reyes Herrera Rogelio
Facultad de Ingeniería
UNAM
Ciudad de México, México
masterfive5of@gmail.com

I. DESARROLLO

Traer un código de Matlab o Python de una arquitectura de red neuronal convolucional. No es necesario programarlo, pero si se explicará en clase.

1. *Que especifique el número de capas, y en cada capa los filtros usados en cada capa y todos los hiperpárametros que utiliza. Por ejemplo si hay dropout o no.*

La arquitectura de la red es la siguiente:



- La entrada del modelo es un tensor de 32x32x3, respectivamente, para la anchura, la altura y los canales.
- Tendremos dos capas convolucionales. La primera capa aplica 32 filtros de tamaño 3x3 cada uno y una función de activación ReLU. Y la segunda aplica 64 filtros de tamaño 3x3
- La primera capa de agrupación aplicará una agrupación máxima de 2x2
- La segunda capa de agrupación también aplicará una agrupación máxima de 2x2
- La capa totalmente conectada tendrá 128 unidades y una función de activación ReLU
- Por último, la salida serán 10 unidades correspondientes a las 10 clases, y la función de activación es una softmax para generar las distribuciones de probabilidad.

La implementación queda de la siguiente forma:

En primer lugar, definimos el modelo utilizando la función Secuencial() y cada capa se añade al modelo con la función add() .

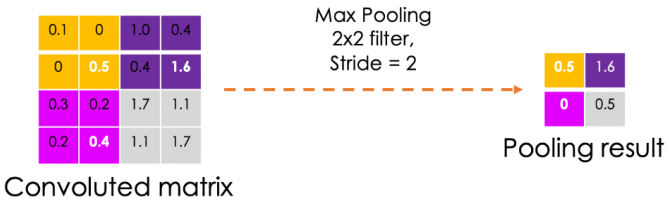
```python
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Variables
INPUT_SHAPE = (32, 32, 3)
FILTER1_SIZE = 32
FILTER2_SIZE = 64
FILTER_SHAPE = (3, 3)
POOL_SHAPE = (2, 2)
FULLY_CONNECT_NUM = 128
NUM_CLASSES = len(class_names)

# Model architecture implementation
model = Sequential()
model.add(Conv2D(FILTER1_SIZE, FILTER_SHAPE, activation='relu', input_shape=INPUT_SHAPE))
model.add(MaxPooling2D(POOL_SHAPE))
model.add(Conv2D(FILTER2_SIZE, FILTER_SHAPE, activation='relu'))
model.add(MaxPooling2D(POOL_SHAPE))
model.add(Flatten())
model.add(Dense(FULLY_CONNECT_NUM, activation='relu'))
model.add(Dense(NUM_CLASSES, activation='softmax'))
```

Tras aplicar la función summary() al modelo, obtenemos un resumen completo de la arquitectura del modelo con información sobre cada capa, su tipo, la forma de salida y el número total de parámetros entrenables.:

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPooling  (None, 15, 15, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 2304)              0

 dense_1 (Dense)             (None, 128)               295040

 dense_2 (Dense)             (None, 10)                1290

=================================================================
Total params: 315,722
Trainable params: 315,722
Non-trainable params: 0
```

Se aplica una función de activación ReLU después de cada operación de convolución. Esta función ayuda a la red a aprender relaciones no lineales entre las características de la imagen, lo que la hace más robusta para identificar distintos patrones. También ayuda a mitigar los problemas de gradiente evanescente.

Se aplica max pooling:



2.*Especificar la función de Aumentado de datos y que hiperparámetros puede utilizar.*

La red que tomamos no utiliza aumentado de datos pero se pueden usar los siguientes con ayuda de ImageDataGenerator:

- **rotation_range**: Rotación aleatoria de hasta 20 grados.
- **width_shift_range** y **height_shift_range**: Desplazamientos aleatorios del 20% del ancho/alto.
- **shear_range**: Cizallamiento aleatorio del 20%.
- **zoom_range**: Zoom aleatorio del 20%.
- **horizontal_flip**: Volteo horizontal aleatorio.

Quedando de la siguiente forma:

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Configuración de aumento de datos
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Ejemplo: cargar imágenes desde un directorio y aplicar aumentación
train_data = datagen.flow_from_directory(
    'ruta/del/dataset/entrenamiento',  # Directorio con las imágenes de entrenamiento
    target_size=(128, 128),
    batch_size=32,
    class_mode='categorical'
)
```

3.- *Código donde especifica cuando es la parte del entrenamiento*

El entrenamiento se hace con las funciones compile() y fit(), que toman los siguientes parámetros :

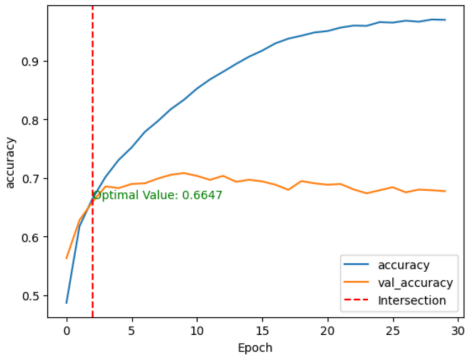
- El Optimizador se encarga de actualizar los pesos y sesgos del modelo. En nuestro caso, utilizamos el optimizador Adam.
- La función de pérdida se utiliza para medir los errores de clasificación, y nosotros utilizamos la Crosentropía().
- Por último, la métrica se utiliza para medir el rendimiento del modelo, y la exactitud, la precisión y el recuerdo se mostrarán en nuestro caso práctico.

```python
from tensorflow.keras.metrics import Precision, Recall

BATCH_SIZE = 32
EPOCHS = 30

METRICS = metrics=['accuracy',
                   Precision(name='precision'),
                   Recall(name='recall')]

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics = METRICS)

# Train the model
training_history = model.fit(train_images, train_labels,
                    epochs=EPOCHS, batch_size=BATCH_SIZE,
                    validation_data=(test_images, test_labels))
```

4.- *Especificar cuando ya se prueba el código que utiliza*

- La evaluación del modelo quedó de la siguiente manera:

```python
import matplotlib.pyplot as plt

def show_performance_curve(training_result, metric, metric_label):

    train_perf = training_result.history[str(metric)]
    validation_perf = training_result.history['val_'+str(metric)]
    intersection_idx = np.argwhere(np.isclose(train_perf,
                                              validation_perf, atol=1e-2)).flatten()[0]
    intersection_value = train_perf[intersection_idx]

    plt.plot(train_perf, label=metric_label)
    plt.plot(validation_perf, label = 'val_'+str(metric))
    plt.axvline(x=intersection_idx, color='r', linestyle='--', label='Intersection')

    plt.annotate(f'Optimal Value: {intersection_value:.4f}',
             xy=(intersection_idx, intersection_value),
             xycoords='data',
             fontsize=10,
             color='green')

    plt.xlabel('Epoch')
    plt.ylabel(metric_label)
    plt.legend(loc='lower right')
```



References

[1] Tutorial de Redes Neuronales Convolucionales (CNN) con TensorFlow. (2024, 11 septiembre). DataCamp. Recuperado 13 de noviembre de 2024, de https://www.datacamp.com/es/tutorial/cnn-tensorflow-python