Autores: Antonio Gálvez, Pablo Millán y José Antonio Laserna

Sistemas Inteligentes para la Gestión en la Empresa

Práctica 2: Deep Learning para clasificación

UNIVERSIDAD DE GRANADA

Índice

Fundamentos teóricos

Red neuronal

Una red neuronal consta de capas de nodos o neuronas artificiales: una capa de entrada, una o varias capas ocultas y una capa de salida. Cada nodo se conecta a los demás y tiene su propia ponderación y umbrales asociados. Si la salida de cualquier nodo individual está por encima del valor umbral especificado, ese nodo se activa y envía datos a la siguiente capa de la red. De lo contrario, no se pasa ningún dato a la siguiente capa de la red.

Las redes neuronales se basan en datos de entrenamiento para aprender y mejorar su precisión con el tiempo

Capa completamente conectada

Una capa de neuronas completamente conectadas es una capa de neuronas en las que cada una de ellas tiene conexiones (con sus respectivos pesos asociados) con todas las neuronas de la capa anterior y con las capa siguiente.

Capa convolucional

Una capa convolucional es una capa de neuronas en las que se aplica una convolución a los datos entrantes. Estos datos deben estar en forma matricial para aplicar esta convolución. Una convolución es una operación que se le aplica a un elemento de una matriz en la que influyen los elementos vecinos del elemento objeto de la operación.

Tasa de aprendizaje

Parámetro de la red que determina el grado en que varían los pesos de la red en cada época.

Kernel

El kernel define el tamaño de la convolución. En el caso de la práctica se está utilizando un kernel de tamaño 3, lo que significa que a la hora de hacer la convolución de un elemento se tendrá en cuenta una matriz de 3x3 vecinos para el elemento.

Descripción de las técnicas empleadas

Padding

Consiste en, a la hora de realizar una convolución, añadir elementos a los valores vecinos ficticios a los elementos extremos de la matriz si estos no cuentan con suficientes vecinos, de forma que estos también puedan verse afectados de por la convolución.

Pooling

Es una operación que reduce la dimensionalidad de los datos reduciendo el número de entradas para la siguiente capa de la red neuronal. En nuestro caso de utiliza un pooling de 2x2, lo que implica que se dividiría la matriz de datos en submatrices de tamaño 2x2 y, de esas submatrices se toma un único elemento.

Batch Normalization

Se utiliza para normalizar la salida (y la entrada) de los datos entre capas de la red neuronal.

Dropout

Otra técnica para evitar el sobre aprendizaje de la red. Consiste en desconectar de manera aleatoria neuronas de la red, evitando su ajuste durante una época.

Weight decay

Técnica que penaliza los pesos grandes. Hay distintas formas de aplicación de esta técnica.

Fine Tunning y ajuste de hiperparámetros

Esta técnica consiste en la modificación de los parámetros que condicionan el aprendizaje de la red neuronal con el objetivo de encontrar la mejor combinación de estos para un problema concreto. Los procesos de Fine Tunning permite automatizar este proceso creando flujos automáticos de variaciones de hiperparámetros, permitiendo conocer al usuario la mejor combinación.

Data Augmentation

Proceso que consiste en introducir variaciones al conjunto de datos de entrenamiento (en nuestro caso, desplazamiento de la imagen, variación cromática, variación de gamma, rotaciones) con el objetivo de lograr una mayor capacidad de generalización del modelo.

```python
class MiCNN_definitivo_v3(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)

        self.conv2 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)

        self.conv3 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(256)

        self.conv4 = nn.Conv2d(256, 320, kernel_size=3, padding=1)
        self.bn4 = nn.BatchNorm2d(320)

        self.conv5 = nn.Conv2d(320, 256, kernel_size=3, padding=1)
        self.bn5 = nn.BatchNorm2d(256)

        self.conv6 = nn.Conv2d(256, 128, kernel_size=3, padding=1)
        self.bn6 = nn.BatchNorm2d(128)

        self.pooling = nn.MaxPool2d(2, 2)
        self.relu = nn.LeakyReLU(0.1)
        self.dropout = nn.Dropout(0.33955714678659366)
        self.flatten = nn.Flatten()
        self.linear = nn.Linear(128 * 3 * 3, 128)
        self.output = nn.Linear(128, 20)  # 20 clases

    def forward(self, x):
        x = self.pooling(self.relu(self.bn1(self.conv1(x))))
        x = self.pooling(self.relu(self.bn2(self.conv2(x))))
        x = self.pooling(self.relu(self.bn3(self.conv3(x))))
        x = self.pooling(self.relu(self.bn4(self.conv4(x))))
        x = self.pooling(self.relu(self.bn5(self.conv5(x))))
        x = self.pooling(self.relu(self.bn6(self.conv6(x))))

        x = self.flatten(x)
        x = self.dropout(x)
        x = self.relu(self.linear(x))
        x = self.output(x)
        return x
```

Solución propia

Mediante la aplicación de estas técnicas, el modelo que hemos decidido implementar (especialmente enfocado al dataset de las 20 clases por una cuestión de tiempo de entrenamiento) se puede observar en la imagen anterior.

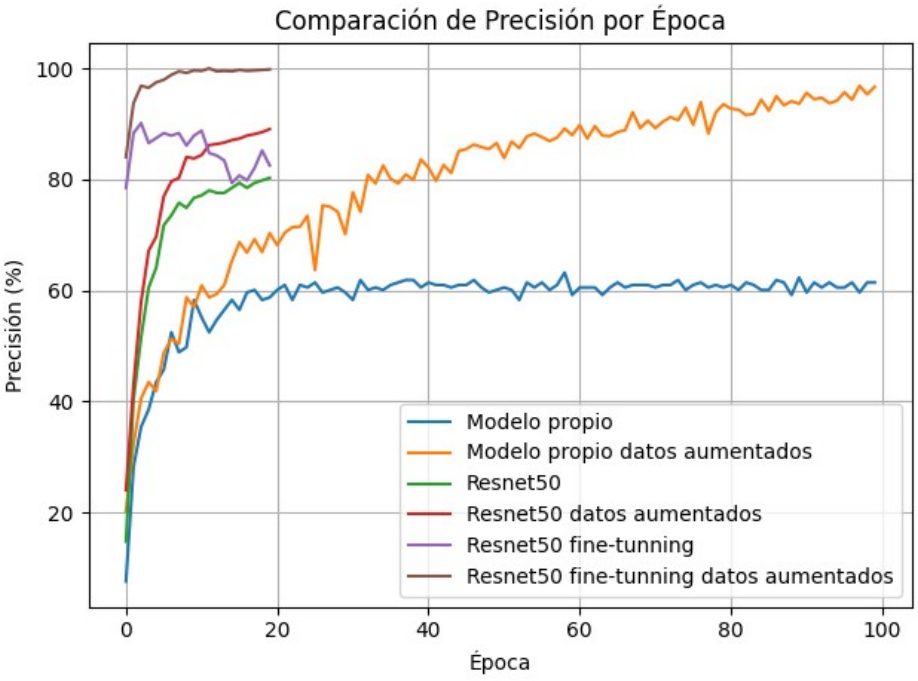
Está conformado por 6 capas convolucionales, 1 capa completamente conectada y una capa de salida de 20 neuronas que corresponden al número de clases existentes. El valor de dropout utilizado se ha obtenido del proceso de Fine-Tuning con Optuna.

Además se emplean las distintas técnicas mencionadas (pooling, dropout, batch normalization).

Como optimizador se ha empleado AdamW para incluir la técnica de weight decay y con una tasa de aprendizaje ajustada mediante fine-tunning

Discusión de resultados

Durante el proceso de implementación de el modelo propio de clasificación, aunque las técnicas de ajuste han significado la diferencia entre un modelo aceptable y un muy buen modelo, hay que señalar que el factor determinante es la construcción de una topología de red adecuada. Para el experimento con **20 clases** se tienen los siguientes resultados:

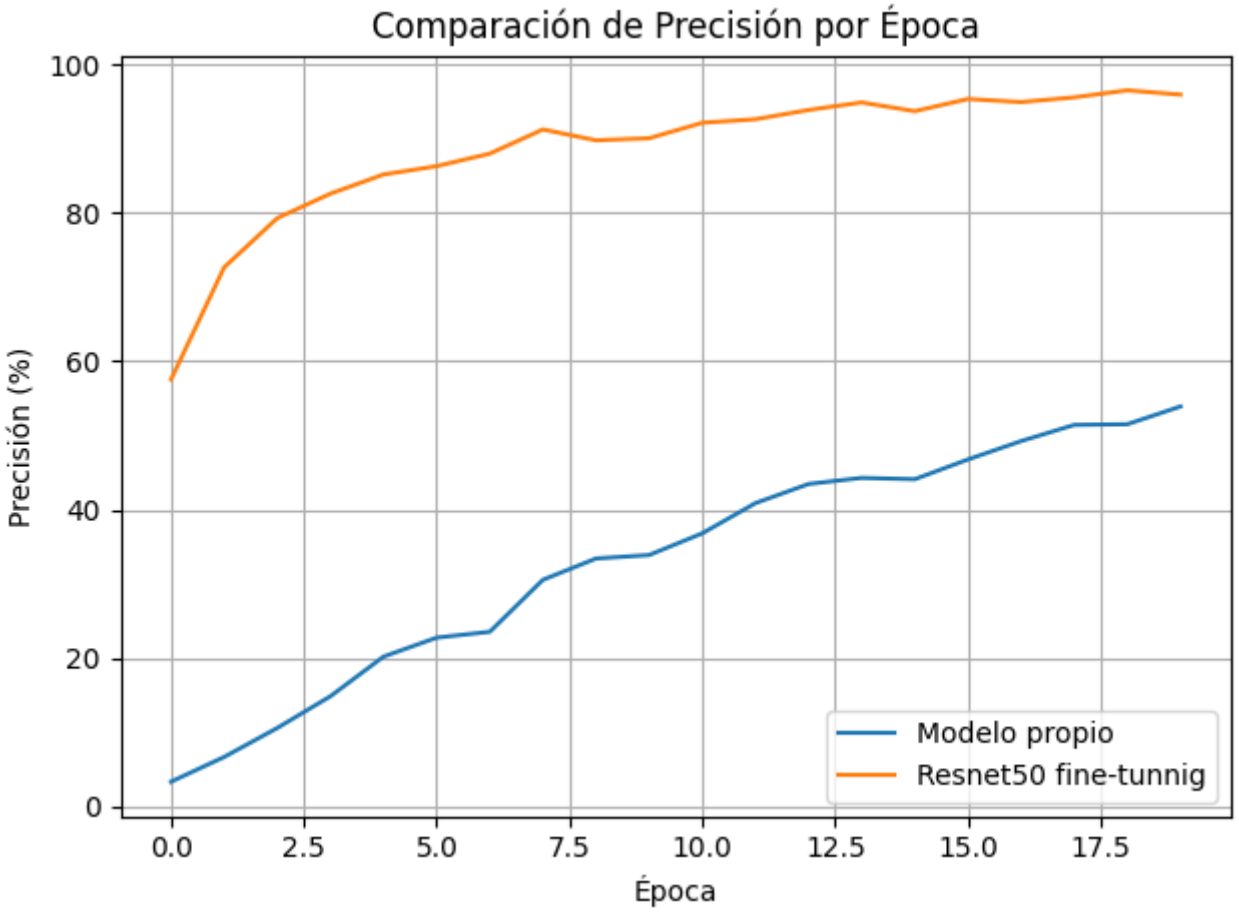


Los resultados comparativos entre nuestro modelo final y un modelo ya existente (en nuestro caso **Resnet50**) indican que una de las técnicas que mejor impacto tienen a la hora de conseguir mejorar los resultados es el data augmentation.

Aunque nuestro modelo alcanza unos buenos resultados en comparación con Resnet50, hay que destacar que el tiempo de entrenamiento es significativamente mayor, no solo por el número de épocas necesarias, sino también por el tiempo por época.

Modelo	Dataset	Tiempo
Resnet50	Datos 20 clases	2 min 57 seg
Resnet50	Datos 20 clases aumentados	5 min 01 seg
Resnet50 fine-tunning	Datos 20 clases	1 min 46 seg
Resnet50 fine-tunning	Datos 20 clases aumentados	3 min 38 seg
Modelo propio	Datos 20 clases	7 min 19 seg
Modelo propio	Datos 20 clases aumentados	16 min 5 seg

Para el problema de las 200 clases, los resultados son similares:



Aquí por desgracia no hemos podido realizar un entrenamiento del modelo propio tan exhaustivo debido a los tiempo de entrenamiento necesarios. Aquí la diferencia de calidad entre el modelo Resnet50 y el propio es más notable, sobretodo en el número de épocas necesarias para ajustar el modelo.

Aunque con el mismo número de épocas nuestro modelo tarda menos en entrenarse, Resnet50 alcanza mejores resultados con muchas menos épocas:

Modelo	Dataset	Tiempo
Modelo propio	Datos 200 clases aumentados	36 minutos
Resnet50	Datos 200 clases aumentados	42 minutos

Conclusiones

Podemos concluir que los modelos basados en redes neuronales convolucionales son efectivos a la hora de su aplicación en este tipo de problemas de clasificación de imágenes. Sin embargo, el proceso de construir el mejor modelo posible no es un problema trivial como se muestra en nuestros experimentos. Tanto es así que a lo largo de los años de desarrollo de estas técnicas se han construido herramientas para ayudar a automatizar este proceso.

Nuestros resultados indican que con unos conocimientos intermedios sobre redes neuronales se pueden construir modelos que tengan muy buenos resultados en este problema. Sin embargo, la implementación de modelos excepcionales requiere de un gran proceso de optimización, de conocimientos profundos sobre la técnica y le problema a resolver, y de gran cantidad de pruebas experimentales.

Destacar también la importancia de disponer de máquinas lo suficientemente potentes para poder disminuir el número de experimentos.

Bibliografía

- Material teórico de la asignatura.
- Material teórico de la asignatura de Inteligencia Computacional del MUII.
- https://www.youtube.com/watch?v=NvC035A4LUw
- https://github.com/omaratef3221/pytorch_tutorials/blob/main/Ex_2_Image_Classification.ipynb
- https://docs.pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- https://docs.pytorch.org/docs/stable/index.html