Proyecto de Reconocimiento de emociones faciales

Integrantes:

Abel A. Cruz Suárez C-511

Daniel Reynel Dominguez C-511

Jose Carlos Piñeira C-511

Resumen

Hoy en día el uso de la inteligencia artificial para tareas de clasificación se ha vuelto algo viral. Esto se debe a que una máquina con todo su poder de cómputo es más perspicaz a la hora de detectar patrones casi irreconocibles para los humanos.

La tarea de este proyecto trata sobre reconocer diferentes emociones a partir de un dataset formado por imágenes faciales. Este trabajo explora el área del aprendizaje supervisado dentro del área de machine learning. Con el se pondrá en práctica el uso de redes neuronales y de visión por computadora para resolver el problema en cuestión, siendo capaz de detectar y aprender ciertos patrones regulares de las caras para posteriormente dar una estimación precisa de que emoción se está detectando.

Redes Neuronales

Como hemos tratado en el curso sobre machine learning las redes neuronales representan una herramienta muy eficaz ante un problema de clasificación. La presentación de su idea a priori parece algo sencilla. Una capa encargada de recibir una entrada en forma de tensor, una serie de capas que ejecutan ciertas una operación algebraica sobre una entrada, que no es más que la salida de una capa anterior y finalmente un capa encargada del proceso de clasificación (en este caso por el problema que se trata, no necesariamente es para clasificar). Este proceso que hemos descrito es conocido como **forward**, donde el resultado de las activaciones de las **neuronas** es propagado hacia adelante, en la capa final con una función de error determinada se determina cuán distante está el resultado de nuestra red de la respuesta original y mediante la técnica de **back-propagation** se ajustan los pesos de la red neuronal. Esta última fase se realiza mediante el algoritmo de optimización llamado gradiente descendiente, donde un peso w_i se actualiza de la siguiente manera $w_i = w_i - \alpha \frac{\partial E}{w_i}$. La idea detrás del gradiente descendiente es que el cálculo de las derivadas otorga un vector indicando la dirección de máximo de la función y si avanzamos una cantidad α en dirección contraria convergemos al mínimo de la función, lo cuál es nuestro objetivo, ya que se quiere minimizar la función E. Este proceso **forward & Back-propagation** se puede repetir innumerables ocasiones para alcanzar un resultado conveniente. Por supuesto aquí entran en cuestión otros factores como el **overfitting** o **underfitting** que provocan que entrenar mucho sobre los mismos datos o tener pocos datos se convierta en un problema.

Dataset

El dataset en cuestión es **Fer2013**. Los datos consisten en imágenes de rostros, posando o no, en escala de grises de 48x48 píxeles. Cada La imagen en **FER-2013** está etiquetada como una de las siete emociones: feliz, triste, enojado, asustado, sorprendido, disgustado y neutral, siendo la felicidad la emoción más prevalente. El conjunto de datos **FER-2013** se creó recopilando los

resultados de una búsqueda de imágenes en Google de cada emoción y sinónimos de las emociones.

Modelo basado en redes neuronales convolucionales

Para nuestro problema hacemos uso de las **Redes neuronales convolucionales** o **Convolutional neural networks (CNN)**. La principal diferencia entre las DNN y las CNN es que las primeras aprenden patrones globales, mientras que las segundas patrones locales que son reconocidos en pequeñas ventanas de dos dimensiones. El proceso de convolución puede ser visto como si deslizáramos una ventana de mxm sobre una determinada imagen de NxN; la ventana que deslizamos tiene un tamaño mucho menor y su idea es ser la de un filtro que busca determinado patrón sobre la imagen donde quiera que se encuentre.

Esta característica clave, de reconocer patrones locales, le otorga a las CNN dos propiedades interesantes:

- Los patrones que aprenden son invariantes a la traducción. Después de aprender un cierto patrón en la esquina inferior derecha de una imagen, un **CNN** puede reconocerlo en cualquier lugar: por ejemplo, en la esquina superior izquierda. Una red neuronal densa tendría que aprender el patrón de nuevo si apareciera en una nueva ubicación. Esto hace que las **CNN** sean eficientes al procesar imágenes porque el mundo visual es fundamentalmente de traslación invariante. O sea que no importa si un objeto es trasladado, rotado, reducido o ampliado su tamaño, etc; siempre se puede reconocer el mismo objeto, pues se aprenden patrones locales de él y se reconocen donde quiera que se muevan. Esto permite que se necesitan menos muestras de entrenamiento para aprender representaciones que tienen poder de generalización.
- Pueden aprender jerarquías espaciales de patrones. Una primera capa de convolución aprenderá pequeños patrones locales como bordes, un la segunda capa de convolución aprenderá patrones más grandes hechos de las características de las primeras capas, y así sucesivamente. Esto permite que las **CNN** aprendan de manera eficiente conceptos visuales cada vez más complejos y abstractos, lo que resulta de extrema importancia pues el mundo observable es espacialmente jerárquico. O sea, cuando los seres vivos reconocemos un objeto es porque inicialmente reconocimos determinado patrón de líneas, luego su forma global y finalmente lo que representa.

Las redes convolucionales operan sobre un tensor de 3 dimensiones, esto es **largo** x **ancho** x **profundidad**. Esta profundidad en el caso de las imágenes en color es de valor 3 por emplear el patrón RGB. Para nuestro caso tratamos las imágenes en escala de grises por lo que la profundidad es 1. El proceso de convolución de forma práctica sería de la siguiente manera: Se designa la cantidad de filtros a aplicar a una imagen (ej: 32). La salida de esta capa de convolución no es más que un mapa de características, o sea son imágenes con el resultado de aplicar los filtros; para nuestro ejemplo de 32 supongamos que tenemos una imagen de 100 x 100 x 1, luego de aplicar los filtro tendremos una salida formada por 32 imágenes de 100 x 100 que lo mismo que 100 x 100 x 32, donde la profundidad ahora representa la cantidad de filtros aplicados. Los filtros codifican aspectos específicos de los datos de entrada: por ejemplo, a un alto nivel, un solo filtro podría codificar el concepto “presencia de un rostro en la entrada”.

Una convolución funciona deslizando estas ventanas (filtros) de tamaño 3×3 o 5×5 sobre el mapa de características de entrada $3D$, deteniéndose en cada ubicación posible, para extraer el parche $3D$ de las características circundantes. Cada filtro $3D$ luego se transforma (a través de un producto tensorial con la misma matriz de pesos aprendida, llamada núcleo de convolución) en un vector de una dimensión de la forma (**filtros**,). Todos estos vectores se vuelven a ensamblar

espacialmente en un mapa de salida **3D** de la forma ($alto$, $ancho$, $filtros$). Cada ubicación espacial en el mapa de características de salida corresponde a la misma ubicación en el mapa de características de entrada (por ejemplo, la esquina inferior derecha de la
la salida contiene información sobre la esquina inferior derecha de la entrada).

Maxpooling

En una CNN, antes de las primeras capas **MaxPooling2D**, el mapa de características es de 48×48, pero la operación lo reduce a la mitad a 13×13. Ese es el papel de una capa **Maxpooling**: reducir agresivamente los mapas de características.
Maxpooling consiste en extraer ventanas de los mapas de características de entrada
y devolver el valor máximo de cada canal. Es conceptualmente similar a convolución, excepto que en lugar de transformar los parches locales a través de un transformación lineal (el kernel de convolución), se transforman a través de una operación de tensor máximo codificada. Esta operación normalmente se realiza con ventanas de 2x2, lo que permite reducir a la mitad los mapas de características.

En resumen, la razón para usar la reducción de resolución es reducir el número de coeficientes del mapa de características a procesar, así como inducir jerarquías de filtros espaciales haciendo que las sucesivas capas de convolución miren ventanas cada vez más grandes.

Batch normalization, dropout y data augmentation

La normalización de batches es una técnica para estandarizar las entradas a una red, aplicada a las activaciones de una capa anterior o entradas directamente. La normalización por batches acelera el entrenamiento, en algunos casos reduciendo a la mitad las épocas o mejor, y proporciona cierta regularización, lo que reduce el error de generalización. La idea es normalizar los valores de entrada a las capas convolucionales. De forma tal que estén centradas en $\mu = 0$ y $\sigma = 1$.

El **dropout** es una de las más efectivas y más utilizadas técnicas de regularización para redes neuronales. **Dropout**, aplicado a una capa, consiste en descartar aleatoriamente (establecer en cero) un número de características de salida de la capa durante el entrenamiento. La idea central es que introducir ruido en los valores de salida de una capa puede romper patrones casuales que no son significativos, que la red comenzará a memorizar si no hay ruido.

La técnica de **data augmentation** se intoduce para evitar el overfitting, pues permite aplicar diferentes operaciones a las imágenes con el objetivo de ampliar el tamaño del dataset. Entre estas operaciones encontramos rotación, reescalado, extensión, etc.

Arquitectura

La arquitectura que se propone es la de una capa convolucional, seguida de una capa de normalización, otra de maxpooling y finalmente aplicar dropout. Esta arquitectura es apilada 4 veces más. En la fase de convolución se aplican 64 filtros de 5x5, en la segunda 128 de igual tamaño en las tercera y cuarta 512 de tamaño 3x3. La salida de estas fases de convolución es pasada a una red neuronal encargada de realizar la clasificación para ello se pasa por una capa de aplanamiento donde el mapa final de características se convierte en un vector, luego se pasa a una capa oculta con 256 "neuronas", su salida a otra capa oculta de 512 "neuronas" y finalmente esta salida se convierte en la entrada de una capa con solo 7 salidas acorde a las clases que debemos clasificar.

A continuación se muestra el código de la arquitectura:

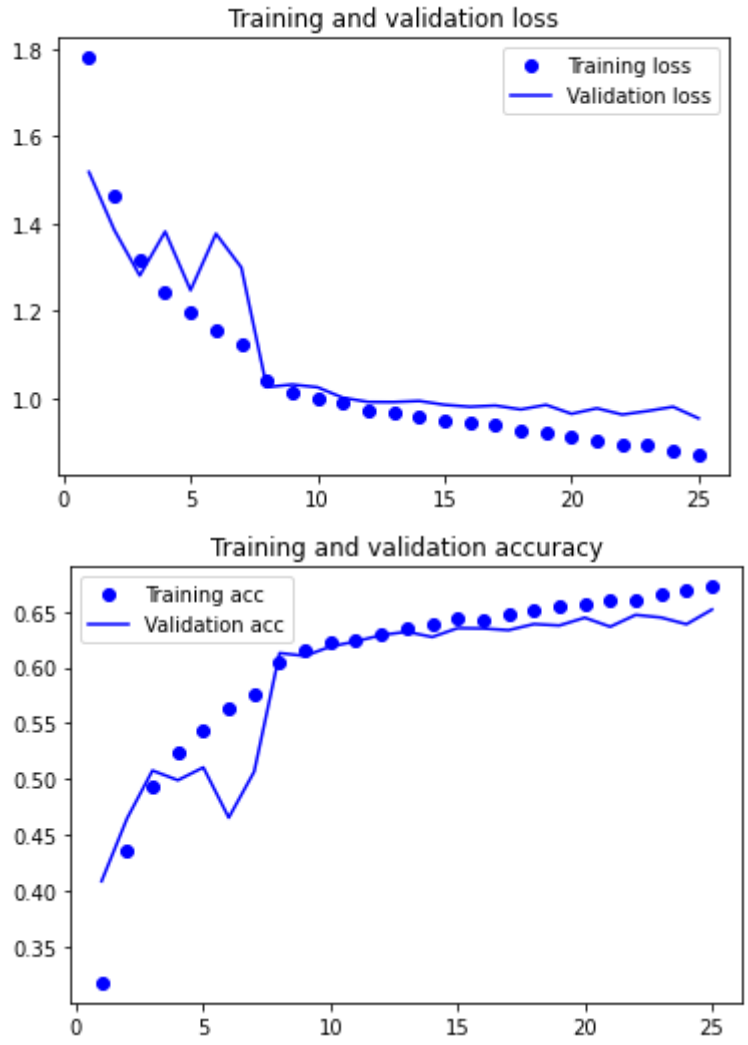
```
model = Sequential()
# 1 - conv
model.add(Conv2D(64,(5,5), padding='same', input_shape=(48,48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
# 2 - conv
model.add(Conv2D(128,(5,5), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
# 3 - conv
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
# 4 - conv
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(7,activation='softmax'))

opt=Adam(learning_rate=0.0005)
model.compile(optimizer=opt ,loss='categorical_crossentropy', metrics=
['accuracy'])
model.summary()
```

Como función de activación de las fases convolucionales se usa $Relu$. Esta función (rectified linear unit) es destinada a poner a cero los valores negativos y mantener los positivos. Como función de optimización se emplea $Adam$ que constituye una alternativa a emplear gradiente stocástico descendiente y puede verse como una combinación de las técnicas de optimización $RMSProp$ y $AdaGrad$. Para la función de pérdida se emplea $categorical\ crossentropy$ que permite otorgar valores probabilistas a la de salida de la red. Se entrena por 25 épocas con batches de tamaño 32

Resultados

Para la arquitectura anterior se obtuvo en la fase de entrenamiento un $loss = 0.8718$ y $presición : 0.6723\%$ y $val\_loss : 0.9547$ - $val\_presición : 65.18\%$. En la fase de prueba se obtuvieron los siguientes resultados: $loss = 0.95416259765625$ y $presición = 65.338534116745\%$. Como se puede apreciar la validación en entrenamiento no está muy distante de la fase testeo.

En la siguiente gráficas se aprecia el resultado del entrenamiento:



Como se puede apreciar cerca de la época 15 ya comienzan a distanciarse las curvas de entrenamiento y validación por lo que se aprecia algo de overfitting.

Empleando otra arquitectura con cuatro fases convolucionales de 32, 64, 128 y 128 filtros respectivamente, sin normalización, ni dropout, con una sola capa oculta de 512 "neuronas", imágenes de 150x150, tamaño de batch de 32 y las mismas funciones de pérdida y optimización se obtuvieron los siguientes resultados:

En la fase de entrenamiento $loss = 0.4153$ y $presición : 85.37\%$ y $val\_loss : 2.5991$ - $val\_presición : 42.38\%$. En la fase de prueba se obtuvieron los siguientes resultados: $loss = 2.1458539962768555$ y $presición = 50.93340873718262\%$. Como se puede apreciar en entrenamiento se alcanza una precisión de aproximadamente 85% muy distinta al 42.38% de precisión en la validación esto indica a todas luces que se está cometiendo mucho overfitting, aprendiendo mucho las características de las imágenes de entrenamiento. Esto se comprueba con los valores que se alcanzan en la fase de testeo: aproximadamente 51% de precisión.

Las gráficas se muestran a continuación: