LAB2: Video Classification

Adrián Rubio Pintado y Miguel Ángel Martínez Pay
May 14, 2022

I. Introducción

En esta práctica se cubre el problema de clasificación de vídeo. El problema se abordará probabilísticamente para N categorías de acciones dadas previamente. Se usarán dos algoritmos base: método aleatorio y fijo, junto a una tercera propuesta basada en CNN's y transfer learning. Para el entrenamiento de este último se proporcionarán datos etiquetados de acuerdo al problema. Finalmente se hará un estudio de las curvas de aprendizaje del clasificador CNN de acuerdo al distinto número de vídeos a clasificar, para evaluar su rendimiento en un problema real.

II. Método e Implementación

Las tareas de las sesiones 1 y 2 se han fusionado, por tener una clara relación con la temática a estudiar.

A. Estructura del código

El código principal donde se muestran todos los resultados y se llama a todas las librerías .py se encuentra en los notebooks **`LAB2-Session1/Session_1.ipynb`** y **`LAB2-Session2/Session_2.ipynb`**. Los notebooks se han ejecutado en Google Colab [1] Cada carpeta contiene los apartados pedidos en cada sesión. Dentro de cada una de ellas hay una carpeta `code` con las librerías locales usadas. Se puede ver una descripción de la funcionalidad de los ficheros más relevantes en la Tabla I.

Fichero	Descripción/Funcionalidad
`LAB2-Session1/Session_1.ipynb`	Tareas Sesión 1
`LAB2-Session2/Session_2.ipynb`	Tareas Sesión 2
data.py	Crea frames con los vídeos proporcionados
model_utils.py	Limpieza y parseo de ficheros
plot_train_cnnlog.py	Gráfica las curvas de validación y loss
processor.py	Convierte imágenes en arrays
random_fixed_mode	Implementación de algoritmos Sesión 1
train_cnn.py	Implementación de clasificador CNN
validate_cnn.py	Validación de ejemplos concretos

TABLE I: Estructura del código

B. Clasificador Random Mode

Algoritmo baseline proporcionado de clasificación. Como su propio nombre indica, este método se basa en seleccionar de manera aleatoria la clase a la que pertenece cada frame pasado. Dicha aleatoriedad utiliza la distribución uniforme, por lo que la probabilidad de acertar la clase de un vídeo pasado es de $\frac{1}{N}$, siendo N el número de clases. No es realmente un clasificador como tal, solo se utiliza para comparaciones con otros métodos de clasificación. Se encuentra ubicado en `random_vs_fixedmode.py`. No se detalla su implementación por ser obvia.

C. Clasificador Fixed Mode

Algoritmo baseline proporcionado de clasificación . Haciendo justicia a su nombre, elige siempre la misma clase a la que pertenece cada vídeo a la hora de hacer predicciones. Por lo que la probabilidad de acierto o no de una clase depende ahora de **la densidad de clases dentro del dataset pasado.** Se encuentra ubicado en `random_vs_fixedmode.py`. La clase que siempre predice se le pasa como argumento. De nuevo, se obvian detalles de implementación.

D. Clasificador CNN frame por frame

Este clasificador de vídeo hace uso de redes convolucionales para su cometido. Se usa como modelo pre-entrenado una red utilizada en clasificación de imágenes, dada que las tarea es similar(los frames del vídeo son imágenes). El modelo pre-entrenado es InceptionV3 [4].

Con dicho modelo, se remplazan las capas densas más altas por otras nuevas a las que se hace un fine tuning ligero, es decir, se entrenan con pocas épocas. De este modo se adapta ligeramente los nuevos pesos a la nueva tarea. Esto es debido a que las capas más altas suelen aprender características de más alto nivel que las inferiores, y se suelen corresponder a la tarea específica a resolver. A continuación, se re-entrena las capas densas medias y altas con todos los datos de entrenamiento con los que dispongamos para obtener el modelo final deseado.

Para implementar dicho modelo, se ha utilizado la librería Keras [2], que permite crear y modificar redes neuronales de manera sencilla. Dicha implementación se encuentra en el fichero `train_cnn.py`. La implementación concreta del modelo se ubica en la función `get_model()` y se puede ver en la Figura 1.

```
def get_model (nb_classes, weights= 'imagenet'):
    base_model = InceptionV3(weights=weights,
                 include_top=False)
    x = base_model. output
    x = GlobalAveragePooling2D()(x)
    x = Dense (1024 , activation='relu')(x)
    predictions = Dense(nb_classes,activation='softmax')(x)
    model = Model(base_model.input, predictions,
            name="InceptionV3-finetune")
    return model
```

Fig. 1: Arquitecura del Clasificador CNN

Como se ha podido apreciar, al modelo pre-entranado InceptionV3 se le ha quitado la capa densa final(`include_top=False`)). Luego se le ha añadido otra capa densa de 1024 unidades(tras un average pooling) y finalmente una capa softmax con tantas salidas como número de clases deseemos clasificar. La salida de la softmax es la probabilidad

con la que el modelo predice cada una de las clases. La clase predicha por el modelo será la que mayor probabilidad obtenga.

Se puede evaluar el modelo con los ficheros `validate_cnn.py`, en el que se puede ver la salida softmax del modelo para algunos frames aleatorios, y `plot_train_cnn.py` con el que se puede apreciar la evolución temporar de las curvas de validación y loss durante el entrenamiento en función del número de épocas transcurrido.

Para entrenar un modelo basta con llamar a la función `execute()` con los respectivos hiperparámetros.

III. Datos

Para esta práctica se usa el dataset UCF101 [3], un dataset dirigido al reconocimiento de acciones de vídeos realistas de Youtube. UCF101 proporciona una gran diversidad a la hora de mostrar las acciones, gracias a la gran variedad de movimientos en la cámara, aparición de objetos en la escena y pose, punto de vista, iluminación, etc. El dataset consta de 101 categorías de acciones, de los cuales se toma para las distintas pruebas solo N= 5,10,15 y 20 acciones distintas.

Se puede ver en la Tabla II la cantidad de datos para train y validación totales disponibles para los distintos números de categorías elegidos .

Num. Clases	Total	Train	Validation
5	13399	9490	3909
10	25303	18452	6851
15	41264	30134	11130
20	58475	42253	16222

TABLE II: Distribución de los frames por número de clases elegido

IV. Metodología Experimental

En esta sección hablaremos sobre la metodología que se ha llevado a cabo para realizar las diferentes tareas que se han pedido en la práctica y que se implementan por orden en los notebooks.

A. Métricas

La métrica usada para evaluar y comparar los clasificadores es la precisión o accuracy, es decir, la proporción de muestras que han sido clasificadas correctamente sobre el total de las mismas. Además, en las redes neuronales que han sido entrenadas se muestra la evolución de su función de pérdida en el proceso de entrenamiento, tanto en el conjunto de entrenamiento como en el de validación. Esta función de perdida es la entropía cruzada categórica (adaptación de la entropía cruzada binaria para soportar más de 2 clases).

B. Tareas

1) Random vs. Fixed Mode: Para esta tarea se pide comparar el rendimiento de ambos algoritmos, según la clase fijada para el algoritmo Fixed Mode. Para esta tarea usamos solo N=5 clases distintas: `ApplyEyeMakeup`, `ApplyLipstick`, `Archery`, `BabyCrawling`, `BalanceBeam`. La ejecución de random mode tiene una componente de aletoriedad, por ello ejecutaremos N = 1000 ejecuciones y calcularemos la media muestral. De este modo veremos accuracy medio para las 5 clases del clasificador aleatorio. Dado que Fixed Mode depende de la densidad de los ejemplos, no hay componente de aletoriedad y no necesitamos calcular ninguna media. Se ha calculado la función auxiliar sencilla `execute_radom_vs_fixed(n_executions)` en el notebook para ello.

V. Resultados y Análisis

A. Random vs. Fixed Mode

Se puede ver la comparación entre el clasificador aleatorio y el fijo en la Tabla III.

Clase Fijada	Acc. Random(Medio)	Acc. Fixed
ApplyEyeMakeup	20.01%	22.52%
ApplyLipstick	19.91%	17.70%
Archery	20.09%	22.52%
BabyCrawling	20.00%	20.50%
BalanceBeam	20.04%	16.77%

TABLE III: Accuracy Random Medio(1000 ejecuciones) vs. Fixed Mode

En este caso, la desviación para las N=1000 ejecuciones del cálculo de accuracy para el random mode es de 0.02, por lo que se puede considerar la media como un estimador esperado estable del accuracy. Como se aprecia, el modo aleatorio es mucho más estable, y como es de esperar acierta el 20% de las clases al haber 5 clases($p = \frac{1}{5} = 0.20$).

Se obrserva cómo el accuracy del modo fijo para cada clase es simplemente su densidad de ejemplos. El dataset no está perfectamente balanceado en número de ejemplos, y por tanto en aquellos clases que disponen de una mayor representatividad, la percisión del clasificador fijo en ellos es ligeramente mayor que en el clasificador aleatorio.

B. Clasificador Random vs. Fixed vs. CNN

Se compara el clasificador CNN entrenado para 5 clases con el clasificador aleatorio y fijo. Se ve la evolución de la precisión en función del número de épocas de entrenamiento en la Figura 2. Se puede ver cómo para úna sola época de entrenamiento ya superamos al clasificador aleatorio y al fijo(ambos con precisión de 20% constante ya que no requieren de entrenamiento). Su precisión pasa del 80% a estabilizarse por encima del 90% en 14 épocas. Se puede afirmar que es con diferencia el mejor clasificador. De hecho es el único clasificador útil como tal, ya que un accuracy del 20% es un claro indicativo de que estamos antes unos pésimos modelos. En contra el entrenamiento del clasificador CNN requiere de mucho más tiempo y poder de computación. Recordamos que lo hemos entranado en Google Colab con GPU's.

C. Clasificador CNN para distinto número de clases

Se procede a compartar el rendimiento del clasificador CNN de acuerdo a distintos número de clases. Concretamente con N= 5 ,10,15 y 20 clases. Para generar las gráficas

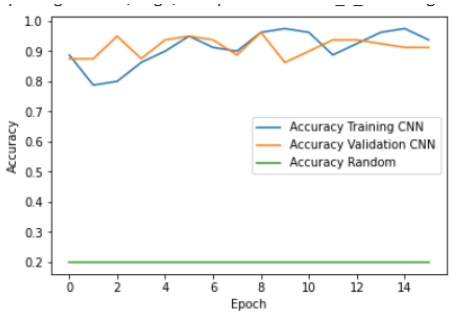
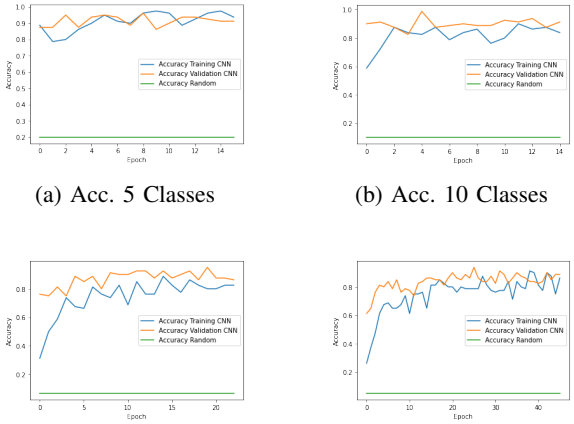


Fig. 2: Accuracy de los modelos según el número de épocas de entrenamiento

de dicho apartado se ha utilizado el código ubicado en el fichero `plot_train_cnnlog.py`. Utilizando accuracy como métrica, se puede ver la comparativa en la Figura 3. Cuanto mayor es el número de clases, el accuracy inicial disminuye, y la mejora en función el número de épocas es mayor. También el número de épocas necesarias para entrenar es mayor a medida que aumentamos el número de clases. Esto es debido a que para la misma arquitectura, el problema es más complicado, y por tanto tarda más en aprender a discriminar entre clases.



(a) Acc. 5 Classes (b) Acc. 10 Classes

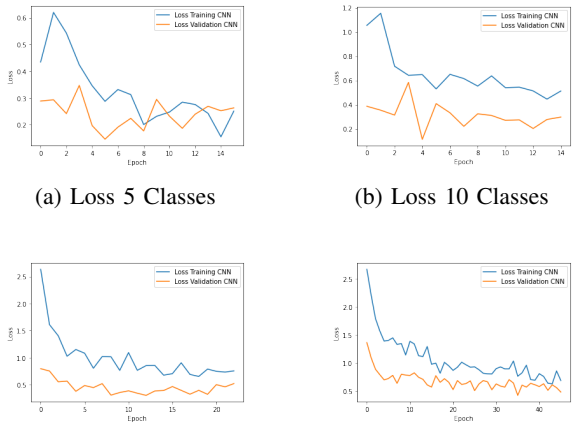
(c) Acc. 15 Classes (d) Acc. 20 Classes

Fig. 3: Accuracy para N=5,10,15 y 20 Clases

Las curvas de loss o pérdida se pueden ver en la Figura 4. De nuevo observamos el mismo comportamiento que con las curvas de precisión pero a la inversa: a medida que aumentamos el número de clases la pérdida inicial es mayor. Para el mismo número de épocas la pérdida es mayor. Todo ello a causa de una mayor compejidad en el problema.

Es decir, las métricas empeoran ligeramente a medida que le añadimos clases (complejidad) al problema.

Otro detalle importante a destacar observando la Figura 4 es el hecho de que la curva de loss en validación está por debajo de la de training(y viceversa en accuracy). Además dicha pérdida se estabiliza al final de cada época, lo que significa que el modelo ha aprendido a generalizar suficientemente bien y no hay underfitting. Sin embargo, la distancia final entre ambas curvas sigue siendo realtivamente grande,

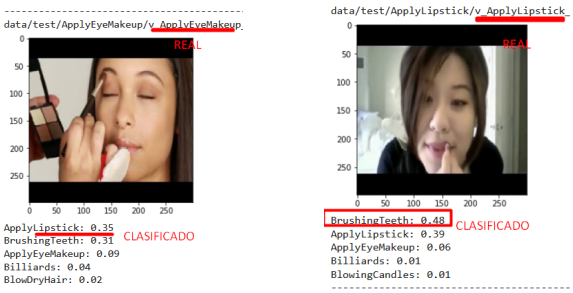


(a) Loss 5 Classes (b) Loss 10 Classes

(c) Loss 15 Classes (d) Loss 20 Classes

Fig. 4: Loss para N=5,10,15 y 20 Clases

lo que es síntoma de que un número de ejemplos más grande en el dataset harían que el modelo mejorara a generalizar aun más, mejorando por tanto sus métricas.



(a) Error en Clasificación nº 1 (b) Error en Clasificación nº 2

Fig. 5: Errores de clasificación del modelo CNN con 20 clases

Esto último se puede evidenciar con algunos ejemplos de acciones parecidas en la vida real que el modelo confunde con facilidad, por ejemplo, como podemos ver en la Figura 5. En el caso (a), se ve como confunde la acción de aplicar contorno para los ojos (35% de probabilidad) con el de pintarse los labios (9% de probabilidad). Si se suma en este caso concreto las dos probabilidades más altas, con la segunda opción: cepillarse los dientes, siendo las dos erróneas, suman un 66%. En defensa del modelo se puede afirmar que las 3 acciones más probables suponen mover un objeto con forma de bolígrafo y moverlo en círculos en la cara de una persona. En el caso (b) podemos ver otro ejemplo de un caso similar, aunque ahora la opción correcta tiene un porcentaje de probabilidad mucho más alto.

Estos problemas podrían corregirse muy probablemente, tal y como se ha mencionado, disponiendo de un número mayor de ejemplos para que el modelo aprenda a discriminar mejor ejemplos tan parecidos.

VI. Conclusiones

Hemos visto como los clasificadores deep learning basados en CNN's mediante el análisis frame by frame son muy eficaces en la tarea de clasificación de vídeo. Incluso cuando disponemos de un tamaño de dataset bastante limitado, como es el caso de la práctica. Sin embargo, gracias a técnicas como el transfer learning, podemos adaptar modelos ya existentes para otras tareas similares a la nuestra concreta. La técnica seguida ha consistido en un primer paso de fine tuning a la capa densa sustituida del modelo pre-entranado InceptionV3, seguido de un fine tuning con todos los datos de entrenamiento a las capas medias y altas del nuevo modelo.

Los resultados obtenidos son bastante satisfactorios, con un accuracy de en torno al 90%, con los que aunque son mejorables dichas métricas, no hemos caído en el underfitting ni el overfitting. Cabe destacar que a medida que aumentamos el número de clases la complejidad de problema aumenta como es de esperar, y por tanto el tiempo de entrenamiento, el cual no es despreciable, ya que estamos entrenando los modelos con GPU's. Además se ha observado que ante tareas muy parecidas, como pintarse los labios o lavarse los dientes, el modelo encuentra dificultades al clasificar, que se podrían resolver si se estuviera disponible un dataset con un número mayor de ejemplos.

VII. Referencias

References

[1] Tiago Carneiro, Raul Victor Medeiros Da NóBrega, Thiago Nepomuceno, Gui-Bin Bian, Victor Hugo C. De Albuquerque, and Pedro Pedrosa Rebouças Filho. Performance analysis of google colaboratory as a tool for accelerating deep learning applications. *IEEE Access*, 6:61677–61685, 2018.

[2] Francois Chollet et al. Keras, 2015.

[3] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild, 2012.

[4] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

Appendix

Registro de tiempo A continuación se expone una estimación del tiempo dedicado a cada parte del trabajo:

- Comprensión del código: 45 minutos.
- Preparación de los experimentos: 2 horas (donde se incluye la preparación o modificación de ficheros y la organización de los datos a utilizar).
- Ejecución de los experimentos: 2 horas y 30 minutos.
- Interpretación de los resultados: 30 minutos.
- Elaboración del informe: 4 horas.