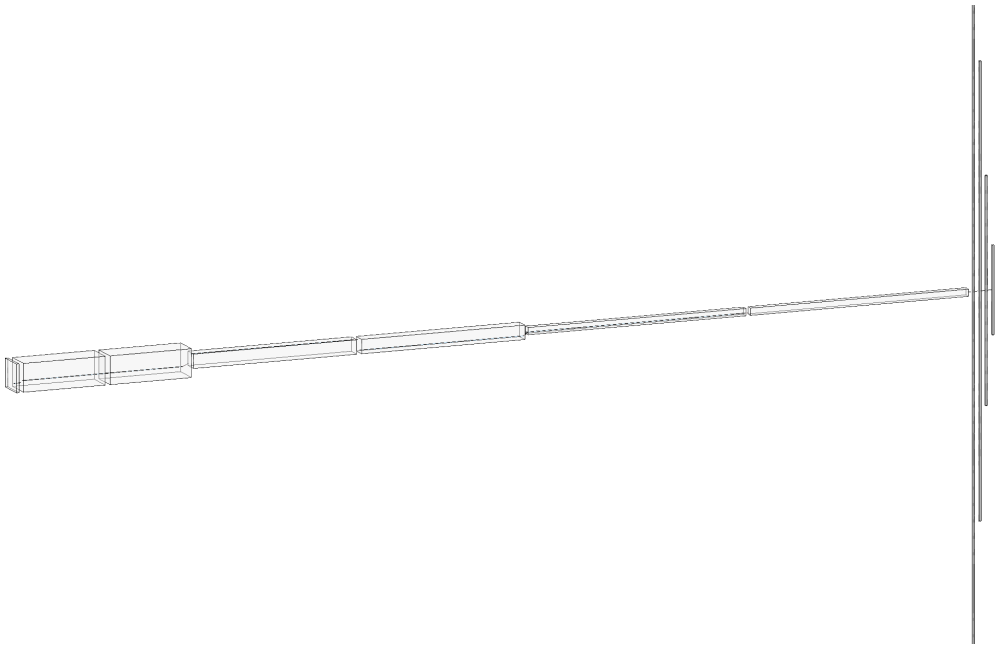
PRÁCTICA II

Pablo Chantada Saborido & José Romero Conde



1. Introducción y aspectos generales

El problema a resolver es clasificación en 100 clases de un conjunto de datos de 50000 ejemplos, de los cuales 40000 no estan etiquetados y de los 10000 restantes hemos dedicado 1000 a validación. Además las clases son muy parecidas entre si (hombre y niño o bicicleta y motocicleta) y los ejemplos de cada clase no tienen porque ser parecidos entre sí tampoco (puentes, castillos o sillas pueden tienen alta variación dentro de ellas mismas). Es por esto que decimos que el problema es difícil. Como se sugiere en AlexNet [4], la profundidad en una RNA puede permitir mejor capacidad de representación, es por esto que nosotros, después de plantear un baseline con una red "pequeña" decidimos hacerla más grande, y es esta la que consta en el código entreagado.

Sobre esto, a priori pensaríamos que sería un camino de rosas el simplemente hacer un modelo más grande, pero nos encontramos conque, en general, la red grande necesitaba más épocas para conseguir los mismos resultados que la pequeña en entrenamiento; por contrapartida, la pequeña sobreentrenama más mientras que la grande solía ser bastante robusta. Es por esto que sospechamos fuertemente que los resultados aquí presentes son inconclusos en cuanto a la capacidad que tienen los modelos definidos de aprender. Nosotros decidimos tomar esta decisión (en vez de aferrarnos al modelo más pequeño y rápido de entrenar) porque aunque con el modelo pequeño conseguíamos buenos resultados, sabíamos que era consecuencia del sobreentrenamiento.

Antes de prodecer con el anális de cada ejercicio nos gustaría justificar algunas decisiones generales que hemos tomado en cuanto al entrenamiento y definición de los modelos. Como optimizador hemos usado *AdamW* [5], y para enfrentarnos al problema de la explosión del gradiente en el entrenamiento de las redes contrastivas hemos tenido que usar *Gradient Clipping* [6], esta última decisión por desgracia no fue aventurada con antelación sino que fue *debuggeando* el código, que la red no entrenaba más de un epoch sin incluirlo, fue al usar `print(gradients.magnitude)` que nos dimos cuenta. Como hay pocos ejemplos etiquetados y por tanto era relativamente sencillo caer en sobreentrenamiento decidimos optar por tres medidas de regularización de forma transversal a lo largo de la práctica:

1. Penalización de la norma euclídea de los coeficientes de los filtros convolucionales y de las capas densas. Sabemos que si forzamos a nuestros coeficientes a ser pequeños (o en este caso, equivalente a tener un *prior* de que $\theta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, más adelante exploraremos otros priores).

2. Dropout [9] para las capas densas que constituyen a los clasificadores. A lo largo de la práctica se ha usado siguiendo distintas probabilidades de dropout y siguendo esquemas fijos o dinámicos (distintas capas de la misma red tienen distinta probailidad). También incluimos Dropout entre cada bloque convolucional

3. Aumento de datos leve y estático. Al principio de todos los modelos aquí propuestos existe un aumento de datos, lo cual dificulta aún más el problema pero nos garantiza que no sobreentrenará (o no lo hará tanto). Después de probar con distintas opciones hemos llegado a la siguiente configuración:

   - Volteo horizontal. Aunque en dominios como OCR no es aplicable, después de examinar las clases de CIFAR100, hemos visto que todas ellas pueden ser volteadas sin consecuencia. Produciendo así mayor variedad y simultaneamente preparándonos para posibles ejemplos volteados en test
   - Rotación aleatoria con probabilidad 0.2
   - Zoom aleatorio con probabilidad 0.2
   - Translación aleatoria pequeña
   - Leve emborronado gausiano, esto nos hace ser robustos a características de baja frecuencia, que en CIFAR100 están presentes en, sobretodo, frutas y flores ya que pueden ser fácilmente discriminables sólo con información de baja frecuencia.

2. Entrena un modelo, creado sobre TensorFlow, haciendo uso únicamente de las instancias etiquetadas de entrenamiento. Dicho modelo debe de tener al menos cuatro capas densas y/o convolucionales.

2.1. ¿Qué red has escogido? ¿Por qué? ¿Cómo la has entrenado?

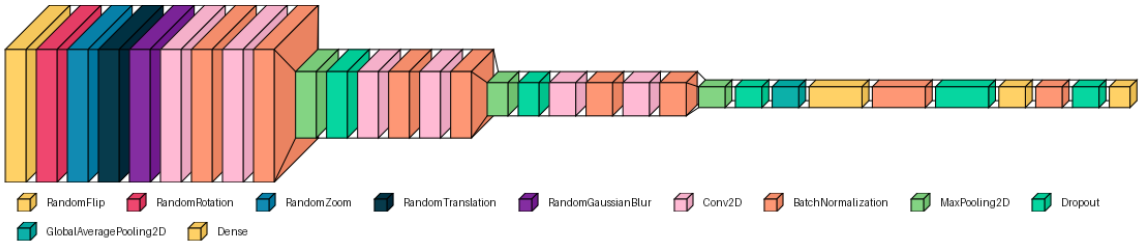


Figura 1: Diseño de la Red Convolucional

Los datos han sido divididos según las especificaciones del enunciado (40.000 muestras no etiquetadas, 10.000 etiquetadas). Estas últimas, a su vez, han sido divididas en entrenamiento (9.000) y validación (1.000). Hemos seleccionado una red convolucional compuesta por tres bloques convolucionales (6 capas convolucionales en total) y 2 capas densas de clasificación. Implementamos diferentes técnicas de regularización para intentar evitar el sobreentrenamiento:

- **BatchNormalization** después de cada capa convolucional para estabilizar el aprendizaje.
- **Regularización L2** ($\lambda = 0{,}003$) para penalizar pesos de gran magnitud.
- **Dropout** con probabilidades entre 0.15 y 0.3 en distintas capas.
- **Data augmentation** para aumentar artificialmente la cantidad de datos, aunque no tengan la diferencia real del conjunto original.

Para la reducción de dimensionalidad utilizamos capas de **MaxPooling** tras cada bloque convolucional. Antes de las capas de clasificación aplicamos **GlobalAveragePooling** que reduce drásticamente el número de parámetros comparado con el enfoque clásico de Flatten. Como función de activación utilizamos **ReLU** en todas las capas intermedias al ser una función de activación referente al tratar con redes neuronales, con **Softmax** en la capa de salida para obtener probabilidades de clase. Para la optimización, implementamos:

- **Learning Rate Scheduler** con decaimiento coseno para reducir progresivamente la tasa de aprendizaje.
- Optimizador **AdamW** que incorpora una regularización de decaimiento de pesos.
- **EarlyStopping** con paciencia 4 para detener el entrenamiento.

El entrenamiento se realizó con un tamaño de batch de 128 durante un máximo de 100 épocas, aunque el proceso se detuvo prematuramente debido al early stopping.

2.2. ¿Cuál es el rendimiento del modelo en entrenamiento? ¿Y en prueba?

El modelo alcanza una precisión de aproximadamente 40 % en el conjunto de entrenamiento y 30 % en el conjunto de prueba. Como se observa en la Figura 2, existe una clara distancia entre las curvas de entrenamiento y validación que comienza a ampliarse considerablemente después de la época 20, indicando sobreentrenamiento. La pérdida en el conjunto de entrenamiento continúa disminuyendo hasta aproximadamente 3.0, mientras que la pérdida de validación se estabiliza alrededor de 3.7, confirmando que el modelo memoriza patrones del conjunto de entrenamiento, pero no se traspasa este aprendizaje bien a datos nuevos. Considerando que estamos trabajando con 90 muestras por clase para entrenamiento (tras separar el conjunto de validación), un rendimiento del ≈30 % no es extremadamente malo. En el ejercicio siguiente (Autoaprendizaje), se intentará abordar este problema añadiendo las muestras no etiquetadas al modelo.

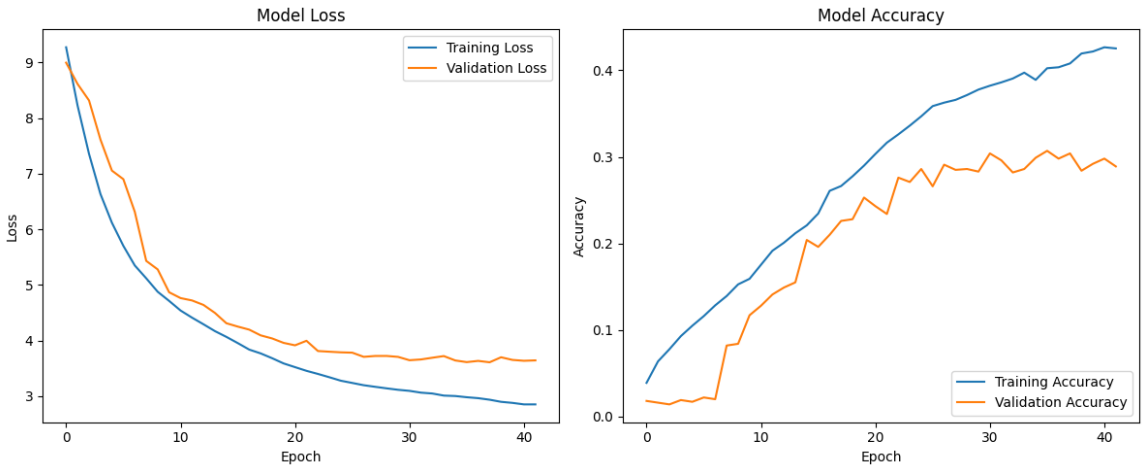


Figura 2: Resultado de Entrenamiento con muestras Etiquetadas

2.3. ¿Qué conclusiones sacas de los resultados detallados en el punto anterior?

Los resultados obtenidos nos muestran las limitaciones a las que nos enfrentamos, dada la complejidad del problema y las restricciones planteadas:

1. **Insuficiencia de datos etiquetados**: Con solo 90 ejemplos por clase para entrenamiento (tras separar el conjunto de validación), el modelo no puede aprender las características representativas de cada clase.

2. **Sobreentrenamiento**: A pesar de implementar múltiples técnicas de regularización (L2, dropout, data augmentation, batch normalization), el modelo sigue mostrando un claro sobreentrenamiento. Esto sugiere que el modelo inevitablemente va a presentar un sobreajuste ya sea por la dificultad del conjunto de datos o por la falta de los mismos.

Estos resultados nos indican que un modelo convolucional básico no tiene la capacidad de superar este problema. Para ello, necesitamos explorar otras técnicas como el autoaprendizaje para aumentar .[a]rtificialmente.[el] conjunto de datos etiquetados.

3. Entrena el mismo modelo, incorporando las instancias no etiquetadas de entrenamiento mediante la técnica de auto-aprendizaje. Opcionalmente, se ponderará cada instancia de entrada en función de su calidad (o certeza).

3.1. ¿Qué parámetros has definido para el entrenamiento?

Para los nuevos parámetros hemos modificado levemente los hiperparámetros del modelo para adaptarlo al autoaprendizaje:

- **Learning Rate** reducido para facilitar un proceso de fine-tuning más estable
- **Dropout** levemente más alto para prevenir el sobreajuste al incorporar nuevas muestras
- **Regularización L2** levemente más alta para controlar mejor la complejidad del modelo cuando se agregan nuevas muestras
- **Ponderación:** las instancias iniciales (etiquetadas) tienen una ponderación de 2, mientras que las nuevas instancias tienen una ponderación asignada según su nivel de certeza (valor de probabilidad)

Los parámetros específicos del proceso de autoentrenamiento se seleccionaron tras realizar múltiples experimentos con diferentes valores:

- **Umbral de confianza (threshold):** probamos diferentes valores entre 0.6 y 0.9
- **Número de iteraciones:** exploramos valores entre 3 y 8 iteraciones
- **Ponderación de muestras:** asignamos pesos proporcionales a la confianza de la predicción

3.2. ¿Cuál es el rendimiento del modelo en entrenamiento? ¿Y en prueba?

Los resultados obtenidos para las diferentes configuraciones de autoentrenamiento son:

- **Configuración 1 (threshold=0.6, iteraciones=4):**
  - Precisión en entrenamiento: 60.39 %
  - Pérdida en entrenamiento: 3.3332
  - Precisión en validación: 26.00 %
  - Pérdida en validación: 3.9404
  - Precisión en prueba: 29.31 %
  - Muestras utilizadas: 19.757
  - Mejora respecto al modelo base: -0.21 %
- **Configuración 2 (threshold=0.85, iteraciones=4):**
  - Precisión en entrenamiento: 63.05 %
  - Pérdida en entrenamiento: 3.4490
  - Precisión en validación: 31.80 %
  - Pérdida en validación: 3.7595
  - Precisión en prueba: 34.11 %
  - Muestras utilizadas: 15.296
  - Mejora respecto al modelo base: +3.31 %

.48

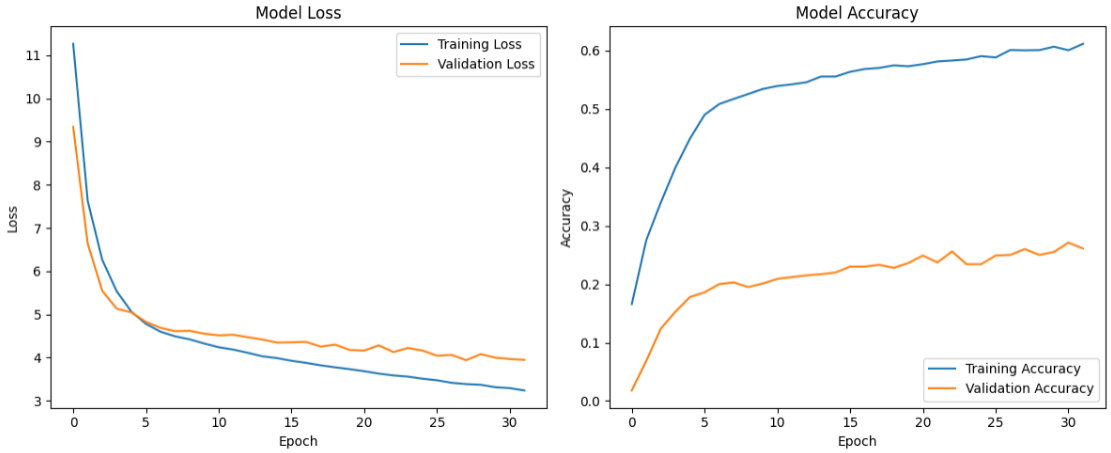


Figura 3: Autoentrenamiento con threshold=0.6

.48

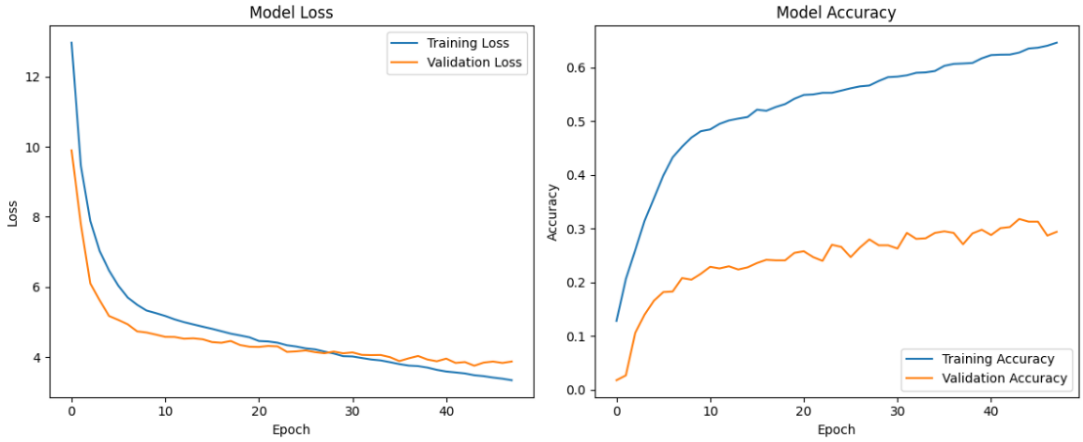


Figura 4: Autoentrenamiento con threshold=0.85

Figura 5: Comparación de modelos con diferentes umbrales de confianza (4 iteraciones)

3.3. ¿Se mejoran los resultados obtenidos en el Ejercicio 1?

Observamos una mejora significativa únicamente con el umbral más alto (0.85), que aumenta la precisión en prueba en un 3.31 % respecto al modelo base. Sin embargo, al incrementar el umbral a 0.9, el modelo no incorpora suficientes muestras nuevas para mejorar significativamente.

Con el umbral más bajo (0.6), no solo no se produce mejora, sino que se observa un ligero deterioro del rendimiento (-0.21 %). Esto sugiere que muchas de las muestras incorporadas tienen etiquetas incorrectas que confunden al modelo.

En cuanto al número de iteraciones, encontramos que 4 es el valor óptimo. Con más iteraciones, el modelo tiende a sobreentrenar y su rendimiento en prueba se deteriora notablemente, probablemente por la acumulación de errores en las pseudo-etiquetas.

Por último, cabe señalar que se observa un aumento en el sobreajuste del modelo con autoaprendizaje, especialmente con umbrales bajos. Esto probablemente se debe a que el modelo, debido a su rendimiento limitado, tiende a asignar pseudo-etiquetas a instancias que son muy similares a las que

ya conoce bien, reforzando patrones que ya había aprendido en lugar de aprender nuevas instancias.

3.4. ¿Qué conclusiones sacas de los resultados detallados en los puntos anteriores?

Las técnicas de autoaprendizaje pueden mejorar el rendimiento de un clasificador cuando se aplican correctamente, pero presentan limitaciones importantes:

1. **Calidad del modelo base:** El éxito del autoaprendizaje depende crucialmente de la calidad del clasificador inicial. Con una precisión base de aproximadamente 30 %, nuestro modelo asigna etiquetas incorrectas a una proporción significativa de las muestras no etiquetadas.

2. **Umbral de confianza crítico:** Un umbral bajo (0.6) incorpora demasiadas muestras incorrectamente etiquetadas, mientras que un umbral demasiado alto (¿0.85) no incorpora suficientes muestras para mejorar significativamente el rendimiento.

3. **Cantidad vs. calidad:** Observamos que el modelo con umbral 0.85 incorpora menos muestras (15.296 vs 19.757) pero logra mejor rendimiento, confirmando que la calidad de las etiquetas generadas por el clasificador es más importante que la cantidad.

4. **Límite práctico:** El número de muestras añadidas (5.000-10.000 de 40.000 disponibles) indica que el modelo solo puede etiquetar con alta confianza una pequeña fracción del conjunto no etiquetado, lo que limita el potencial del autoaprendizaje en nuestras condiciones.

5. **Iteraciones controladas:** Más iteraciones no siempre son mejores; el error puede propagarse y amplificarse en cada ciclo de pseudo-etiquetado. Comprobamos que a partir de las 6-8 iteraciones el modelo comienza a perder rendimiento rápidamente.

En conclusión, el autoaprendizaje muestra potencial incluso con un clasificador medio como base (mejora de 3.31 %), pero requiere una cuidadosa calibración de parámetros para equilibrar la incorporación de nuevas muestras con el riesgo de introducir etiquetas incorrectas. Con un clasificador base más preciso, podríamos esperar mejoras más significativas ya que podría asignar pseudo-etiquetas correctas a una mayor proporción de las muestras no etiquetadas.

4. Entrena un modelo de aprendizaje semisupervisado de tipo autoencoder en dos pasos (primero el autoencoder, después el clasificador). La arquitectura del encoder debe ser exactamente la misma que la definida en los Ejercicios 1 y 2, a excepción del último bloque de capas.

4.1. ¿Cuál es la arquitectura del modelo? ¿Y sus hiperparámetros?

La arquitectura del autoencoder fue una decisión con pocos grados de libertad poque debía usar la arquitectura anterior como encoder y ser especular hacia delante. En concreto, sobre el "encoder" que nos venía dado añadimos:

- Una capa convolucional de 256 filtros seguidos de un BatchNorm [3] y UpSampling.
- Un bloque de dos capas convolucionales de 192. Entre ellas BatchNorm y al final UpSampling
- Otro bloque como el anterior pero las convoluciones de 96 filtros.
- Para el segundo de los pasos se usó un clasificador idéntico al presentado en el ejercicio 1

Figura 6: Arquitectura del AutoEncoder

Todas las convoluciones usan filtros 3×3 como fue sugerido por la arquitectura VGG [8]. En cuanto a los hiperparámetros:

- Tasa de aprendizaje de 0.01 para el autoencoder y 0.05 para el clasificador.
- Regularición L_2 con $\lambda = 0{,}0005$ para ambos.
- Dropout con probabilidad 0.05 para el autoencoder y 0.1 para el clasificador.
- Tamaño de batch de 512 para el autoencoder y 4096 para el clasificador.
- 15 épocas para el autoencoder y 50 para el clasificador.

Sobre estas decisiones comentar dos cosas. En primer lugar no hemos podido hacer la exploración del espacio de hiperparámetros que nos hubiese gustado con, por ejemplo, un gridsearch. Por tanto estamos seguros de haber elegido hiperparámetros subóptimos, pero por cuestiones de cómputo no pudo ser de otra forma. La segunda consideración también está relacionada con la capacidad de cómputo, las épocas. Si cuando acaba la ejecución vemos que la perdida estaba bajando a un cierto ritmo, o en el caso del clasificador, la precisión subiendo... sabemos que con más épocas hubiese ofrecido mejores resultados. Sobre eso estamos seguros por el siguiente argumento: si una red pequeña ofrece buenos resultados en reconstrucción pero malos en clasificación, una red grande que ofrezca mejores resultados en clasificación no puede ofrecer peores resultados en reconstrucción. O por lo

menos eso en teoría, en la práctica tenemos que lidiar con las dificultades del entrenamiento de las redes profundas. Para que una red más profunda aprenda necesita más cambios en sus parámetros pues existe una relación más intrincada entre ellos mismos, las entradas y las salidas.

Figura 7: A la izquierda, reconstrucción con un autoencoder pequeño, a la derecha con el grande actual (15 épocas).

Entonces la pregunta es ¿Porqué nos aferramos al modelo grande si da peores resultados en el autoencoder? Porque creemos fuertemente que con suficientes épocas (y seguramente consideraciones adicionales de entrenamiento que se nos escapan) aprendería por lo menos tan bien como el otro, pero con mucha seguridad mejor.

4.2. ¿Cuál es el rendimiento del modelo en entrenamiento? ¿Y en prueba?

El rendimiento en entrenamiento es positivo en el sentido de que no para de mejorar y no hemos observado en ninguna ejecución *plateau*. No obstante como hemos sugerido antes, no hubo épocas suficientes y el rendimiento del clasificador en el conjunto de datos de prueba es peor que un uno por ciento, es decir peor que aleatorio. Ciertamente viendo la Figura 7 no podríamos esperar mucho de esas representaciones.

4.3. ¿Se mejoran los resultados obtenidos en los Ejercicios 1 y 2?

No, en absoluto. Lo cierto esque esos dos ejercicios eran directos mientras que este se esforzaba en primer lugar en intentar resolver un problema más grande, en concreto, aprender la distribución de $\mathcal{X}$. Como vimos en [1]:

> Vapnik's principle: When trying to solve some problem, one should not solve a more difficult problem as an intermediate step.

Y efectivamente podemos comprobar que con recursos de tiempo limitado, ser más directo ofrece mejores resultados

4.4. ¿Qué conclusiones sacas de los resultados detallados en los puntos anteriores?

Que si sólo quisiesemos reconstruir imágenes y tuviesemos poco timepo quizás el autoencoder pequeño sería mejor opción. Pero si tuviesemos que usar un modelo para producción en un contexto real, nosotros apostaríamos por el grande, aunque probablemente no se pueda entrenar con nuestros portátiles en una tarde, por poner un ejemplo.

5. Entrena un modelo de aprendizaje semisupervisado de tipo autoencoder en un paso (autoencoder y clasificador al mismo tiempo). La arquitectura del autoencoder será la misma que la definida en el Ejercicio 3, y la combinación de encoder y clasificador será igual a la arquitectura definida en el Ejercicio 1.

5.1. ¿Cuál es la arquitectura del modelo? ¿Y sus hiperparámetros?

La arquitectura no puede ser otra que la del ejercicio anterior, no obstante si vale la pena mencionar que en el primero como cada una de las partes era muy éstandar, lo hicimos usando `tf.keras.models.Sequential([])` pero para este ejercicio tuvimos que usar la opción *funcional*, por lo demás son idénticos (deben serlo). En cuanto a hiperparámetros:

- Tasa de aprendizaje de 0.0035
- Peso extra al decoder, esto es:

$$\mathcal{L}_{AutoencoderSSL} = (1-\alpha)\,\mathcal{L}_{\text{clasificación}} + (1+\alpha)\,\mathcal{L}_{\text{reconstrucción}}$$

  donde $\alpha = 0{,}5$. Esta decisión se basó en la creencia de que con buenas representaciones será sencillo clasificar, además de que si no le damos más peso al AutoEncoder, el clasificador sobreentrenará muy seguramente el conjunto de datos de entrenamiento, consiguiendo buenas métricas pero sin conseguir buenas representaciones (que es lo que realemente nos importa para generalizar).
- Regularición L_2 con $\lambda = 0{,}00005$
- Dropout con probabilidad 0.05
- Tamaño de batch de 512

De nuevo, la elección de hiperparámetros fue (desgraciadamente) poco informada en el sentido de que es probable que existan mejores combinaciones de hiperparámetros pero por lo menos creemos que nuestra decisión fue razonable. Y efectivamente podemos ver que entrenar entrena

5.2. ¿Cuál es el rendimiento del modelo en entrenamiento? ¿Y en prueba?

Después de 16 épocas (claramente muy poco y por tanto muy poco concluyentes serán los resultados) obtuvimos una precisión en el conjunto de datos de entrenamiento de un 2.75 % y en test 2.49 % lo cual es significativamente mejor que el ejercicio anterior y en menos épocas.

5.3. ¿Se mejoran los resultados obtenidos en los ejercicios anteriores?

Con respecto a los dos primeros no, por el motivo comentado. En cambio con el ejercicio anterior sí, y aunque tienen la misma arquitectura no es una sorpresa que ocurra porque ahora los cambios en las capas de representación y clasificación están sincronizados y ambas se modifican bajo la misma señal de error [7]; podríamos pensar que "hablan entre sí". Adicionalmente en el ejercicio anterior podría ocurrir que los ejemplos supervisados sobreescriban las representaciones obtenidas con los ejemplos no supervisados, lo cierto esque no es ese el caso en este ejercicio. Una comparativa un poco pseudocientífica puede obtenerse comparando las recontrucciones de las figuras 7 y 8.

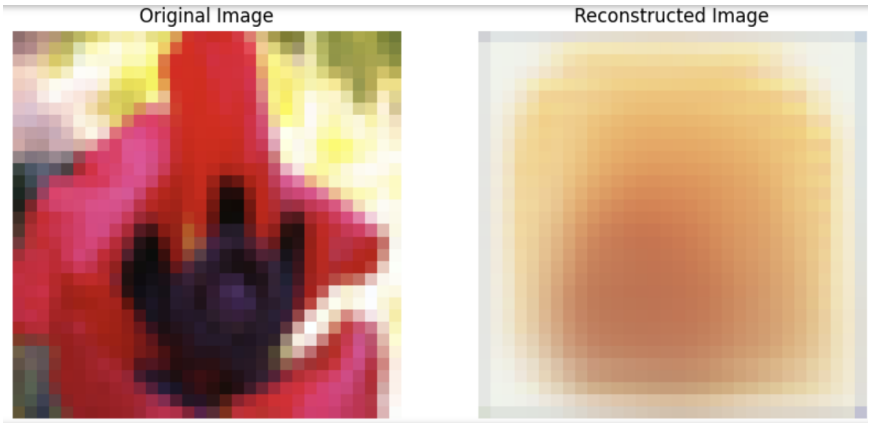


Figura 8: Reconstrucción para el ejercicio 4 después de 16 épocas

5.4. ¿Qué conclusiones sacas de los resultados detallados en los puntos anteriores?

Por lo obtenido experimentalmente, apoyado del razonamiento, concluimos que es mejor esta segunda forma. Además, esta aproximación soportaría, por ejemplo, que `x_unlabeled` fuese todo imagenet (o cualquier otro conjunto de datos grande como este propio CIFAR100) y el `x_train` aquello que se quisiese resolver si es que es muy general. O si por el contrario uno quisiese discriminar abejas de belutinas, podría incluir efectivamente ejemplos etiquetados en `x_train` mientras que en `x_unlabeled` podría tener incluso insectos cualquiera. Sabemos [10] que las representaciones se pueden entrenar con ejemplos de una distribución $\mathcal{X}_1$ para aprender a discriminar ejemplos de $\mathcal{X}_2$, sobretodo si $\mathcal{X}_1 \supset \mathcal{X}_2$. Lo cual es cierto en nuestro caso ya que los datos etiquetados sin duda pertenecen a la misma distribucion que los no etiquetados, y el procedimiento de este ejercicio permite actualizar los pesos de una pasada teniendo en cuenta a ambos. No obstante, los resultados en si son malos, pero de nuevo, confiamos que con más épocas eventualmente consiguiese mejores resultados.

6. Repite el mismo entrenamiento de los Ejercicios 1-4, pero eliminando las instancias no etiquetadas más atípicas con respecto a los datos etiquetados. Se cumplirán los siguientes puntos: (a) La arquitectura de la red de clasificación en una clase será la misma a la utilizada en el clasificador del Ejercicio 1, a excepción de la capa de salida. (b) Utiliza la técnica explicada en el Notebook 5, usando un valor de $v = 0{,}9$

6.1. ¿Se mejoran los resultados con respecto a los anteriores ejercicios? ¿Qué conclusiones sacas de estos resultados?

Después de ejecutar el modelo de detección de anomalías, este clasifica al 89.8 % de los ejemplos como típicos y el 10.2 % como atípicos. Si inspeccionamos las imágenes consideradas como atípicas, vemos entre ellas dos tipos, aquellas que tienen un fondo blanco (en vez de un fondo natural) y aquellas que no tienen forma y son un *blob*, podemos ver un ejemplo en la figura 9. Lo que observamos en general es un empeoramiento en la calidad de generalización esto puede estar debido a que precisamente son los átipicos los que empujan y definen el borde de las regiones en las que se tesela el espacio de píxeles. Al quedarse con los típicos, el problema se vuelve más sencillo (aunque hay menos datos) y por tanto es esperable la observada empeora en generalización.

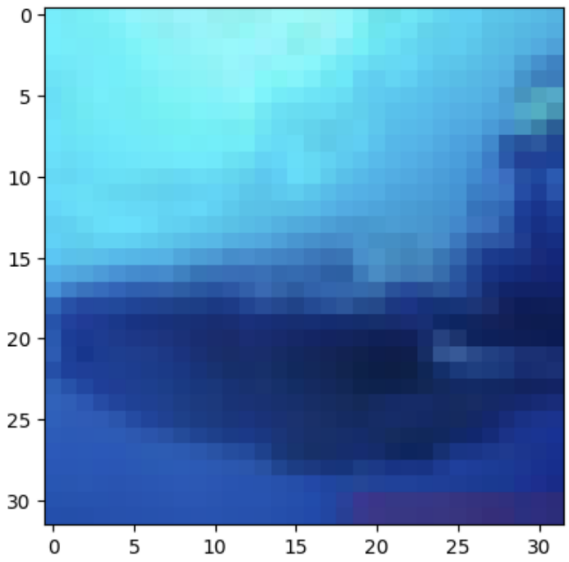


Figura 9: Imagen detectada como atípico por el detector de atípicos. Nuestra hipótesis es que un filtro de gabor en las primeras capas haya dado una respuesta inusualmente alta y por eso atípico.

7. Repite los Ejercicios 3-5 cambiando el autencoder por la técnica definida en el apartado "Hay vida más allá del autoencoder" del Notebook 4. Contesta a las preguntas de dichos ejercicios. Se cumplirán los siguientes puntos:

7.1. La arquitectura de la red será igual a la parte encoder del autencoder definido en los ejercicios anteriores.

7.2. El modelo debe entrenar correctamente.

Sobre el entrenamiento de la aproximación contrastiva hemos experimentado las dos siguientes dificultades:

1. Como en [2] se recomienda el uso de grandes *BatchSize* para el mejor rendimiento del aprendizaje contrastivo, y simultaneamente es este el modelo más pesado computacionalmente de la práctica (multiplicaciones de matrices enormes para conseguir la matriz de similaridad $\mathcal{M}$) ocurre que: si entrenamos con *BatchSize* pequeños sabemos de primera mano que nos espera mala fortuna pero si entrenamos con *BatchSize* grandes, eventualmente ocurrió que las GPUs de nuestros portatiles dejaban de funcionar dando error. Es por esto que entrenar este modelo fue complejo.

2. La arquitectura grande dificulta aún más todo esto, porque a priori un ejemplo x si lo aumentamos una vez (x_{aum1}) y luego otra (x_{aum2}) y luego normalizalizamos para que $||x_{aumi}||_2 = 1$ para $i \in \{1, 2\}$ no cabría esperar, a priori que $\langle x_{aum1}, x_{aum2} \rangle = 1$, que es exactamente lo que le pedimos, y precisamente cabría esperar que muy buenas fuesen las representaciones para conseguirlo. Es por esto que de nuevo hemos observado terribles matrices de similaridad como puede verse un ejemplo en 10.

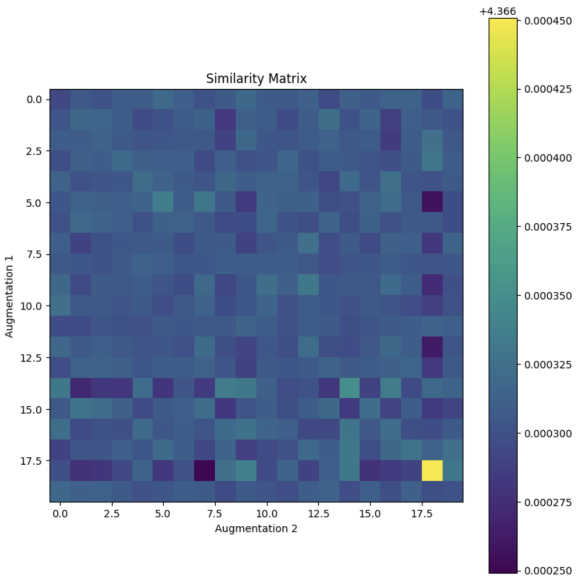


Figura 10: Matriz de similaridad

8. Consideraciones

Debido a las limitaciones de recursos computacionales y restricciones de tiempo, no fue posible ejecutar la práctica con la profundidad y exhaustividad que hubiéramos deseado. Para complementar este trabajo y demostrar la procedencia de nuestros resultados, se han incluido varios archivos adicionales en la carpeta de entrega. Entre ellos se encuentran el archivo principal de ejecución "main.ipynb", que contiene el flujo de trabajo completo, además de dos notebooks específicos para el ejercicio 2 y un notebook adicional que muestra resultados parciales de la ejecución. Estos materiales complementarios sirven como evidencia de que los resultados presentados no fueron extraídos de manera arbitraria, sino que son producto de implementaciones reales, aunque por las limitaciones mencionadas no se pudieron explorar todas las configuraciones ni ejecutar todos los experimentos planificados originalmente.

Referencias

[1] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, Cambridge, MA, 2006.

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119, pages 1597–1607. PMLR, 2020.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS 2012)*, pages 1097–1105, 2012.

[5] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2018.

[6] Tomas Mikolov. *Statistical Language Models based on Neural Networks*. PhD thesis, Brno University of Technology, Brno, Czech Republic, 2012.

[7] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by backpropagating errors. *Nature*, 323(6088):533–536, 1986.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[10] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.