

Memoria del proyecto del semestre: Entrenamiento de modelos de MLP, CNN y GAN usando técnicas modernas de ML.

Christian Mollière

Departamento de Sistemas Informáticos y Computación

Professor: Francisco Casacuberta Nolla, Subject: Redes Neuronales Artificiales

7 de febrero de 2020

Resumen

En la siguiente memoria se examinarán los resultados del proyecto semestral de entrenamiento de un MLP en el MNIST, así como de un CNN en el CIFAR-10. Se examinan diferentes arquitecturas de modelos y métodos de entrenamiento y se presenta el mejor resultado mostrando su historial de entrenamiento y su rendimiento final. Además, como proyecto adicional, traté de sintetizar imágenes de perros utilizando una colección de modelos de GAN entrenados en un conjunto de datos autocompilados.

Índice

1. Entrenamiento de MLP en MNIST	3
1.1. Arquitectura modelo	3
1.2. Cuadrícula de hiperparámetros	3
1.3. Resultados	3
2. Entrenamiento de CNN en CIFAR-10	4
2.1. Arquitectura modelo	4
2.2. Encontrar el modelo correcto	5
2.3. Resultados	5
3. Entrenamiento de Redes Generativas Adversariales	7
3.1. Generando un conjunto de datos	7
3.2. Entrenamiento de GANs convolucionales profundos	7
3.3. Resultados del entrenamiento de DCGAN	9
3.4. Entrenamiento de Wasserstein GANs	9
3.5. Resultados del entrenamiento de WGAN	10
4. Conclusión	16
A. Conjuntos de datos	17
B. Resúmenes de modelos	20
B.1. MLP MNIST classifier model summary	20
B.2. CNN CIFAR10 classifier model summary	21
B.3. DCGAN model summary	23
B.4. WGAN model summary	25

1. Entrenamiento de MLP en MNIST

Entrenar un MLP en MNIST es un ejercicio clásico para entrar en el campo del aprendizaje de las máquinas. El conjunto de datos contiene 60000 imágenes en escala de grises de 28x28 que representan dígitos escritos a mano del 0 al 9. A continuación se discuten las técnicas y parámetros utilizados, lo que lleva a un resultado final de 6.28 % de error en la prueba. Por lo tanto, sólo 63 de las 10000 muestras de datos de prueba dadas son mal interpretadas.

1.1. Arquitectura modelo

La red MLP utilizada utiliza una secuencia de capas fully connected que emplean una función de activación ReLU. Además, se aplica una activación softmax a la capa de clasificación final, como es habitual cuando se clasifican muestras de datos. Por último, se utilizan técnicas como la batch normalization, noise regularization, dropout y data augmentation para mejorar aún más el rendimiento de las redes.

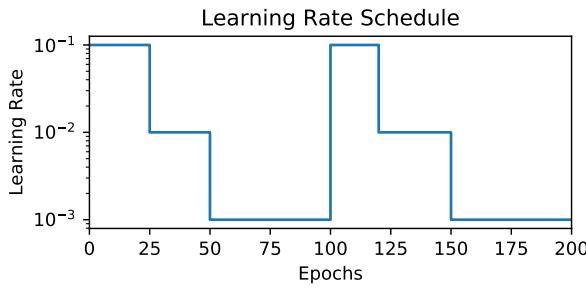


Figura 1: Ritmo utilizado para el entrenamiento del modelo MLP.

Tamaños de las capas Una simple pila de tres capas ocultas (1024,1024,1024) y una capa de salida (10) lograron inmediatamente el rendimiento deseado. El resumen completo del modelo puede verse en la sección B.1.

Regularización Para evitar que el modelo se sobreajuste a los datos de entrenamiento se aplicó el ruido gaussiano, la deserción y la regularización de L2. Batch normalization se utilizó para disminuir el cambio de co-varianza de las características y acelerar el entrenamiento.

Tasa de aprendizaje Una función de tasa de aprendizaje escalonado (Ver Figura 1) proporcionó el ritmo de aprendizaje en la época actual, disminuyendo la velocidad de aprendizaje de 0.1 a 0.001 a medida que el entrenamiento avanza. Después de la

época 100 la tasa de aprendizaje se eleva de nuevo para ayudar potencialmente al optimizador a escapar de los mínimos locales.

Data Augmentation El preprocesamiento de imágenes se utilizó para ampliar los datos de entrenamiento introduciendo un cambio de posición de hasta 10 % y un cambio de rotación de hasta 5° al conjunto de datos. Como la clasificación de los dígitos es sensible a su orientación, no se empleó ningún desplazamiento horizontal o vertical.

1.2. Cuadrícula de hiperparámetros

Se realizó una búsqueda en cuadrícula de los siguientes hiperparámetros, que condujo al resultado obtenido.

- Gaussian Noise $\sigma \in \{0.1, 0.2, 0.3, 0.4\}$
- Dropout Rate $d \in \{0.3, 0.5\}$
- Batch Size = 128
- Activation ReLU & SeLU
- L2 Regularization $\lambda \in \{0, 0.01\}$

Los modelos fueron entrenados para 250 épocas sin detenerse antes, utilizando el descenso de gradiente estocástico y una categórica pérdida de entropía cruzada.

1.3. Resultados

El mejor modelo obtenido de la búsqueda de la cuadrícula alcanzó las siguientes métricas como se muestra en la Tabla 1. Se entrenó usando una tasa de abandono de 0.3, sin regularización de L2, activación de ReLU y capas de ruido gaussiano con $\sigma = 0.1$.

Train Loss	Train Acc	Test Loss	Test Acc
0.7619	0.9862	0.1490	0.9932

Tabla 1: El mejor clasificador del MNIST MLP encontrado.

Si se observa la precisión del modelo y la pérdida durante el entrenamiento (ver Figura 2), se puede ver claramente el repunte de la tasa de aprendizaje en la época 100. Llevando al modelo a una caída temporal de la precisión antes de aumentar el rendimiento un poco más. Además, la menor precisión de entrenamiento y la mayor pérdida de entrenamiento (en comparación con las métricas de prueba) indican que se está produciendo poco o ningún ajuste excesivo y que el modelo se está generalizando bien. Esto se debe probablemente a las técnicas de regularización utilizadas, como se discute en la Sección 1.

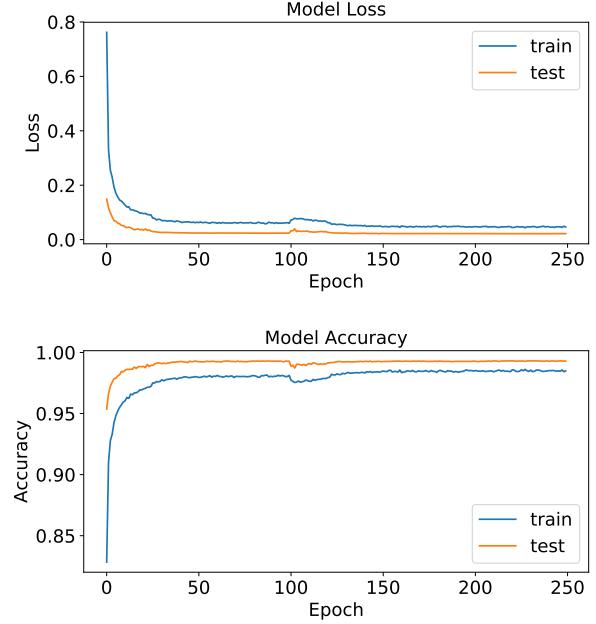


Figura 2: Historia del entrenamiento del clasificador del MNIST.

2. Entrenamiento de CNN en CIFAR-10

Como segunda tarea, se entrenó a una red neural convolutiva (CNN) para clasificar las imágenes del conjunto de datos de imágenes CIFAR-10, que contiene 60000 muestras de diferentes objetos como imágenes en color de 32x32 RGB. El mejor modelo encontrado logró un error de prueba de 9,71

2.1. Arquitectura modelo

El modelo de arquitectura empleado se basa en el Modelo A de VGG introducido en [1]. Utiliza un conjunto de capas convolucionales y de agrupamiento seguidas de un bloque de capas totalmente conectadas. Dado que este modelo sólo necesita distinguir entre 10 clases en vez de 100 como en el caso del VGG, el tamaño de las capas fully connected se reduce de 4096 a 256. El resumen completo del modelo puede verse en la Sección B.2 así como en la última fila de la tabla 3.

Regularización Este modelo utiliza las mismas técnicas de regularización que el modelo MLP en la sección 1. La regularización de L2 y dropout fueron completamente omitidas en este modelo, ya que no aumentó el rendimiento en varios prototipos. Sin embargo, investigaciones recientes sugieren que dropout puede de hecho combinarse con éxito con batch normalization [2].

Tasa de aprendizaje Para lograr una alta precisión del modelo por encima del 90 %, se combinó un programa de aprendizaje cíclico con un descenso de gradiente estocástico, como se propone en [?]. El programa opera entre tasas de aprendizaje de 0.001 a 0.3 y se muestra en la Figura 3.

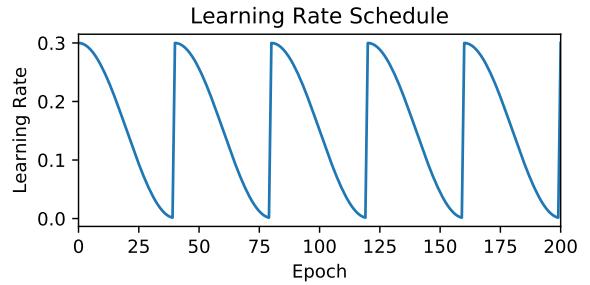


Figura 3: Programa utilizado para el entrenamiento del modelo de la CNN.

Data Augmentation Se generaron datos adicionales de entrenamiento usando la técnica de mixup como se introdujo en [3]. Usando mixup, el conjun-

to de datos de entrenamiento se duplicó en tamaño, produciendo 100000 muestras individuales para entrenar. Además, se empleó el preprocesamiento de imagen aplicando escalado (hasta 20 %), rotación (hasta 20°), cambio de posición (hasta 20 %), volteo horizontal y normalización de datos en función de las características.

2.2. Encontrar el modelo correcto

Dado que la formación de un CNN es mucho más costosa en tiempo que una MLP normal, no se utilizó ninguna búsqueda de cuadrículas para determinar el mejor conjunto de hiperparámetros modelo. En su lugar, se entrenó un conjunto escaso de modelos seleccionados manualmente para lograr el resultado final. Los diferentes modelos entrenados se pueden ver en la tabla 3 que muestra el mejor resultado en la última fila. La principal diferencia entre los modelos está en la variación de los tamaños de las capas densas y los programas de aprendizaje.

2.3. Resultados

El mejor modelo logró las siguientes métricas como se muestra en la Tabla 2.

Train Loss	Train Acc	Test Loss	Test Acc
0.5292	0.9550	0.4274	0.9005

Tabla 2: El mejor clasificador CIFAR-10 encontrado.

Al mirar la historia de la formación del modelo final (ver Figura 4) el programa de aprendizaje cíclico es claramente visible en las métricas de pérdida y precisión. Al reimpulsar el ritmo de aprendizaje cada

40 épocas, se inhibe la meseta del entrenamiento y el modelo puede escapar a los mínimos locales. Parece que el período del ciclo es un poco corto porque el modelo no siempre se estabiliza antes de cada reimpulso. El mayor efecto del programa de aprendizaje cíclico se observa en el conjunto de datos de entrenamiento que contiene muestras de mixup. La cifra de precisión muestra un ligero sobreajuste del modelo hacia el final del entrenamiento. Sin embargo, estas últimas épocas son necesarias para lograr un error de prueba inferior al 10 %.

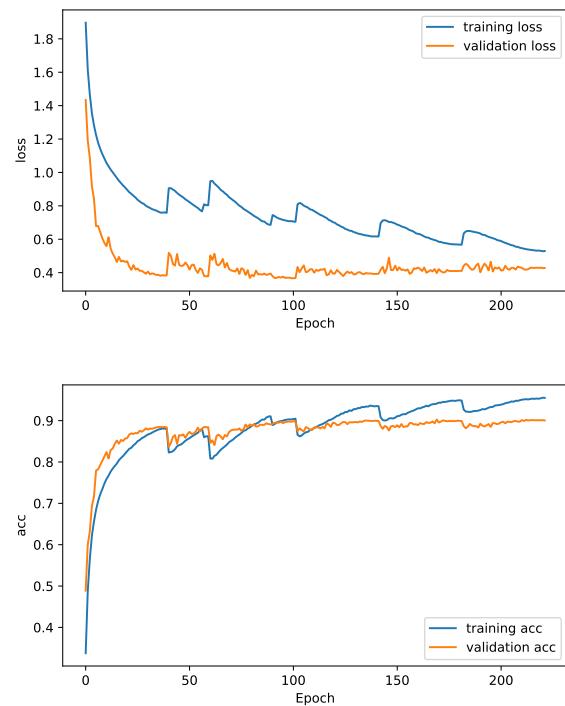


Figura 4: Historia del entrenamiento del clasificador CIFAR-10.

Model Description	LR Schedule	Mixup	Batch Size	Validation Accuracy
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 512	stepped linear 0.1 to 0.001	+50 %	64	87.9 %
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 512	single reboost 0.25 to 0.001	+50 %	32	85.85 %
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 512	single reboost 0.25 to 0.001	+100 %	64	89.04 %
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 512, Dense 256	cyclic 0.3 to 0.001 period 40 epochs	+100 %	64	89.72 %
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 1024, Dense 512	cyclic 0.3 to 0.001 period 40 epochs	+100 %	64	88.65 %
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 256, Dense 128	cyclic 0.3 to 0.001 period 40 epochs	+100 %	64	90.29 %

Tabla 3: Diferentes modelos entrenados en CIFAR-10.

3. Entrenamiento de Redes Generativas Adversariales

Para obtener una visión más profunda de los modelos generativos elegí entrenar diferentes arquitecturas de GAN para sintetizar imágenes de un conjunto de datos autocompilados. Los resultados se discuten a continuación.

3.1. Generando un conjunto de datos

La mayor parte de la educación de ML gira en torno a conjuntos de datos académicos cuidadosamente compilados para entrenar y comparar diferentes algoritmos. Dado que el rendimiento objetivo de los GAN's sigue siendo una medida difícil de evaluar [4], decidí compilar mi propio conjunto de datos de imágenes, ya que de todas formas no podría comparar el rendimiento directamente con los modelos existentes.

Conjunto de datos iniciales Como primer paso se eligió un tema, en este caso el objetivo era generar imágenes de perros. Por lo tanto, se obtuvo una lista completa de 531 razas de perros de la lista de razas que se puede ver en la Sección [5]. En el siguiente paso se construyó un web scraper para descargar sistemáticamente los primeros 100 resultados de una búsqueda en Google Image para cada nombre de raza de perro de la lista obtenida (ver Sección C). Todas las imágenes fueron recortadas a 64x64 píxeles, las imágenes demasiado pequeñas fueron eliminadas del conjunto de datos. El código fuente del web scraper construido está disponible en [6]. Esto produjo un conjunto de datos inicial D_0 que contenía 38148 muestras de imágenes RGB de 64x64 que presumiblemente mostraban perros. Las muestras de D_0 se pueden ver en la Figura 16 en la Sección A.

Conjuntos de datos filtrados Como se puede ver en la Figura 16, el conjunto de datos inicial contiene una cierta cantidad de muestras que son de estilo cómico o contienen un fondo mayormente blanco. Como el objetivo era sintetizar imágenes que parezcan tan reales como sea posible, el conjunto de datos fue filtrado para excluir todas las imágenes que contengan más del 10 % de píxeles sin color que sean más brillantes que el 80 %. Esto produjo un conjunto de datos filtrados D_1 que contenía 30094 muestras. Las muestras de D_1 se pueden ver en la Figura 17 en la Sección A. Finalmente, en la Sección 2 se entrenó un clasificador de imágenes para

obtener un tercer conjunto de datos D_2 incluyendo todas las muestras de D_1 que fueron clasificadas con éxito como imágenes de perros. Esto produjo un conjunto de datos considerablemente reducido D_2 de sólo 4169 muestras. La fuerte reducción de las muestras podría deberse a la gran variedad de perspectivas, antecedentes y razas de perros diferentes, que podrían no ser suficientemente similares a las muestras contenidas en el conjunto de datos de CIFAR-10. Las muestras de D_2 pueden verse en la Figura 18 en la Sección A.

Preprocesamiento durante el entrenamiento

El conjunto de datos de la imagen se reajusta de $[0 - 255]$ a $[-1, 1]$ para el entrenamiento de todos los modelos de GAN.

3.2. Entrenamiento de GANs convolucionales profundos

En general, la idea de una GAN es entrenar a dos modelos adversarios que intentan superarse mutuamente. El primer modelo se llama el generador G , que toma muestras de un espacio latente z y genera una imagen falsa. El segundo modelo, llamado discriminador D , entonces ingresará imágenes reales del conjunto de datos e imágenes falsas del generador y aprenderá a distinguirlas. Si los dos modelos encuentran un equilibrio durante el entrenamiento, el generador converge gradualmente en la muestra de la distribución real aproximada del conjunto de datos [7]. Este concepto se resume en la figura 5.

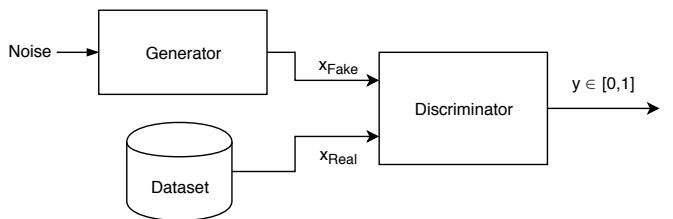


Figura 5: Visión general de la arquitectura de una GAN.

Arquitectura modelo La implementación tanto del generador como del discriminador se basan en

la propuesta DCGAN de [8]. Sus principales características en comparación con las tradicionales GAN de la CNN son:

- No hay capas fully connected en la arquitectura más profunda
- Activación de ReLU y Batch-Norm en todo el modelo de generador
- Activación de Leaky ReLU en el modelo discriminador
- Capas de pooling son reemplazadas por convoluciones de paso 2

El generador utiliza capas convolucionales transpuestas con un paso de 2 para aumentar el tamaño de los mapas de entrada de ruido remodelados hasta el tamaño final de 64x64x3. Además, se aplica dropout de 0.2 en las dos primeras convoluciones transpuestas. La capa final del generador utiliza una activación Tanh para permitir sólo una salida del modelo de $[-1, 1]$, que es coherente con el conjunto de datos reescalado. Su arquitectura modelo se muestra en la Figura 6.

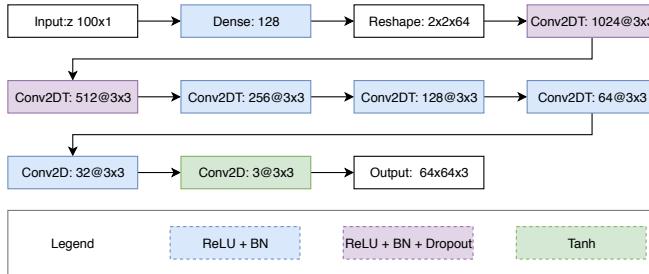


Figura 6: Arquitectura modelo del generador.

El discriminador utiliza convoluciones de paso con un tamaño de 2 para evitar la acumulación de capas. Por lo tanto, su imagen de entrada se reduce de 64x64x3 a 4x4x512 y luego pasa a una capa densa final que emplea la activación sigmoide para la clasificación. Su modelo de arquitectura se muestra en la Figura 7.

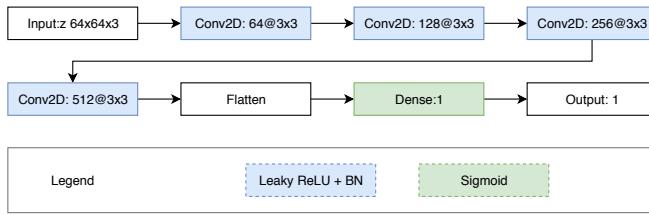


Figura 7: Arquitectura modelo del discriminador.

Tanto el modelo combinado como el discriminador se entrena usando una función de pérdida de entropía binaria. Durante el entrenamiento del modelo combinado los pesos del discriminador se congelan para asegurar que sólo el generador se entrena durante este paso. Los resúmenes completos del modelo tanto del discriminador como del generador pueden verse en la Sección B.3.

Optimizador Como se propuso en [8] se utilizó un optimizador Adam con una tasa de aprendizaje inicial de $\rho = 0.0002$ y un impulso de $\beta = 0.5$.

Suavizar la etiqueta Se podría lograr un rendimiento adicional de entrenamiento empleando el suavizado de etiquetas tanto para las etiquetas reales como para las falsas como se discute en [9], lo que llevaría a etiquetas reales de $y_{\text{real}} \in [0.7, 1]$ y etiquetas falsas de $y_{\text{fake}} \in [0, 0.3]$. Esto evita que el discriminador se confíe demasiado y asegurará gradientes estables para períodos de entrenamiento más largos.

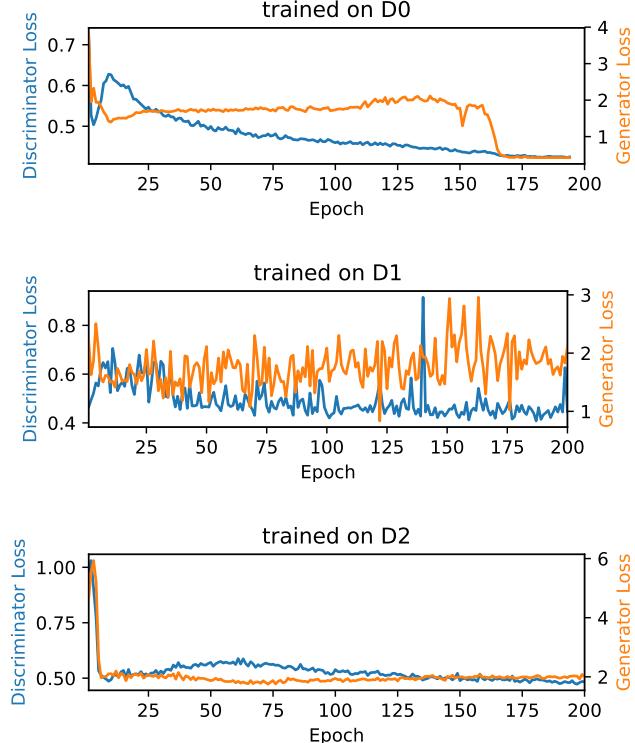


Figura 8: (Top) Loss en D_0
(Middle) Loss en D_1
(Bottom) Loss en D_2

3.3. Resultados del entrenamiento de DCGAN

Los siguientes resultados se obtuvieron después de entrenar el modelo DCGAN definido en la Sección 3.2 usando los tres conjuntos de datos D_0 , D_1 y D_2 para las 200 épocas. El número de iteraciones de entrenamiento por época difiere ya que cada conjunto de datos tiene una cantidad diferente de muestras y el modelo fue entrenado usando un batch size fijo de 128.

La Figura 8 muestra el rendimiento del modelo en los conjuntos de datos D_0 a D_2 respectivamente. Dado que los dos modelos son adversarios que tratan de aumentar la pérdida del otro, la métrica de la pérdida en general no puede ser utilizada para inferir la convergencia del modelo [4]. Sin embargo, el entrenamiento en los tres conjuntos de datos parece ser estable, ya que ninguna de las pérdidas del modelo cae a cero, lo que indicaría un modo de fallo común del entrenamiento de GAN. Para determinar la convergencia de un modelo es útil visualizar la salida de los generadores mediante el muestreo de un número de muestras a intervalos fijos de iteraciones.

La Figura 12 muestra muestras del modelo DCGAN entrenado en el conjunto de datos D_0 . Al principio aprende bastante despacio, logrando siluetas en forma de perro a partir de iteraciones de 2.5k. Parece tener problemas para entrenar simultáneamente en las imágenes de exterior y en las de estudio, que tienen un fondo blanco, mostrando algunas muestras de ruido blanco en la salida.

La Figura 13 muestra muestras del modelo DCGAN entrenado en el conjunto de datos D_1 . Eliminar muchas de las imágenes de estudio y los cómics de D_0 parece ayudar al modelo a entrenar más rápido y a conseguir formas más complejas. Dado que la mayoría de las muestras en el conjunto de datos de entrada tienen un fondo verde hierba, esto también está fuertemente representado en la distribución de datos aprendidos.

La Figura 14 muestra muestras del modelo DCGAN entrenado en el conjunto de datos D_2 . Este conjunto de datos parece funcionar mejor para el entrenamiento de la DCGAN, aprendiendo más rápido que los dos conjuntos de datos anteriores y logran-

do una representación más detallada de diferentes perros y poses. Aún así, incluso este modelo de DCGAN mejor encontrado no generará imágenes que engañen a un observador humano. Algunas partes de las imágenes están claramente relacionadas con los perros, pero las imágenes producidas son demasiado ruidosas y a menudo distorsionadas.



Figura 9: Colapso del modo de DCGAN en D_1 después de 25k iteraciones.

Colapso del modo El entrenamiento de estos modelos por más tiempo siempre resultaba en el colapso del modo (ver Figura 9), lo que significa que la distribución de los datos aprendidos se colapsa en un espacio pequeño. Por lo tanto, el modelo sólo está muestreando imágenes similares con poca variedad. Incluso una mayor capacitación de las iteraciones de 35k+ hizo que los modelos fueran inestables, lo que les permitió volver a generar sólo salidas ruidosas. Batch sizes más pequeños llevan en general a un colapso más rápido del modo.

3.4. Entrenamiento de Wasserstein GANs

El concepto de la GAN de Wasserstein fue propuesto inicialmente en 2017 [10]. En lugar de una entropía cruzada binaria, maximiza una aproximación de la función de distancia Earth-Mover, que los autores denominan distancia de Wasserstein d_W (ver Ecuación ??). El discriminador se modifica para ser más bien una crítica que califica las imágenes de entrada por su similitud con el conjunto de datos. Los autores afirman que esta implementación de GAN es me-

nos sensible a la arquitectura del modelo y muestra una cantidad considerablemente reducida de colapso de modo. Además, la distancia de Wasserstein puede utilizarse para rastrear la convergencia del modelo, en contraste con la métrica de pérdidas oscilantes de la implementación tradicional de GAN. La función de pérdida correspondiente C se describe en la ecuación 1 promediando la etiqueta $y \in [-1, 1]$ veces la, según la predicción del crítico, puntuación $\hat{y} \in \mathbb{R}$.

$$C = \frac{1}{n} \sum_{i=1}^n \hat{y}_i \cdot y_i \quad (1)$$

Durante el entrenamiento del modelo crítico las imágenes reales se etiquetan $y_r = -1$ y las imágenes generadas se etiquetan $y_f = 1$. Por lo tanto, al minimizar la función de pérdida C , las imágenes reales alcanzarán una alta puntuación no negativa, mientras que las imágenes generadas deben ser puntuadas con una alta puntuación negativa. Esto resultará en una distribución divergente de las puntuaciones. La distancia de Wasserstein (ver Ecuación 2) puede calcularse entonces utilizando el modelo crítico D , el modelo generador G , muestras del conjunto de datos \mathbf{x} y entradas de ruido muestreadas para el generador \mathbf{z} . El optimizador intentará maximizar d_W .

$$d_W = -[C(D(\mathbf{x}), \mathbf{y}_r) + C(D(G(\mathbf{z})), \mathbf{y}_f)] \quad (2)$$

El entrenamiento del generador se emplea entonces usando una etiqueta volteada $y_f = -1$ para que las imágenes generadas apunten a puntuaciones dentro del área de la imagen real de la distribución. El optimizador del generador minimiza C_{gen} (ver Ecuación 3).

$$C_{\text{gen}} = C(D(G(\mathbf{z})), \mathbf{y}_f) \quad (3)$$

Arquitectura modelo La arquitectura del modelo es esencialmente la misma que se describe en la Sección 3.2. Sólo se hacen unas pocas modificaciones para transformar el discriminador en el modelo crítico. Estas modificaciones son:

- Cambiando la activación del discriminador final de Sigmoide a Lineal
- Implementando la función de pérdida de Wasserstein (ver Ecuación 1)
- Limitando los pesos de los críticos a $\pm c$
- Entrenar los pasos críticos de n_c más a menudo que el generador por paso de entrenamiento

Un resumen completo del modelo crítico y generador puede verse en la Sección B.4.

Optimizador En lugar del optimizador Adam se utiliza un optimizador RMSprop con una tasa de aprendizaje inicial de $\rho = 5e-5$ como se propone en [10], porque el optimizador Adam hace que el proceso de entrenamiento del modelo WGAN se vuelva inestable.

3.5. Resultados del entrenamiento de WGAN

El entrenamiento del modelo WGAN fue ejecutado usando el hiperparámetro recomendado propuesto en [10]. Estos son:

- Tasa de aprendizaje inicial $\rho = 5e-5$
- El recorte de peso del crítico para $c = \pm 0.01$
- Entrenamiento crítico por entrenamiento de generador $n_c = 5$
- Batch size = 64

El modelo WGAN sólo fue entrenado en el conjunto de datos D_2 , ya que los resultados del entrenamiento de DCGAN mostraron que los modelos se desempeñan mejor en este conjunto de datos (ver Sección 3.3).

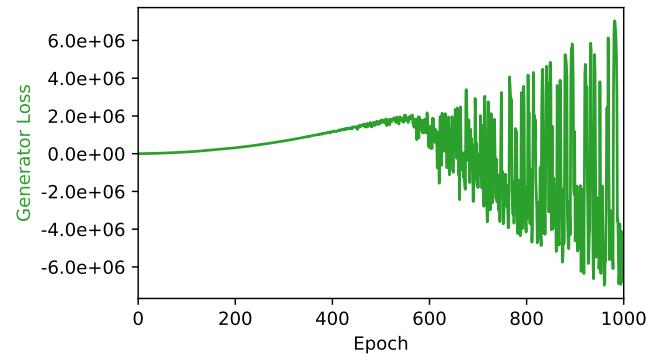
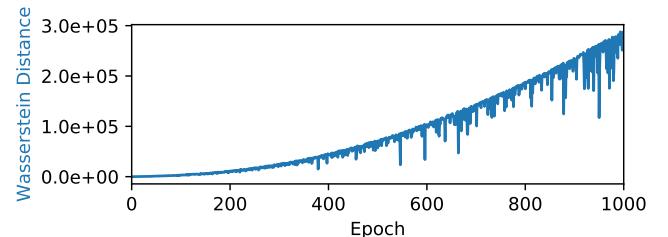


Figura 10: (Top) d_W de WGAN en D_2
(Bottom) C_{gen} de WGAN en D_2

La Figura 10 muestra el historial de entrenamiento del modelo WGAN en el conjunto de datos D_2 . El modelo fue entrenado durante 1000 épocas, logrando casi 65k iteraciones de entrenamiento del generador. La distancia de Wasserstein d_W aumenta gradualmente durante el entrenamiento. Esto indica que el crítico es progresivamente capaz de distinguir entre las muestras reales y las generadas. Idealmente, un muy buen generador debería producir un valor de pérdida negativo ya que su objetivo es su minimización. Sin embargo, una pérdida creciente del generador, como en este caso, no significa necesariamente que el generador no esté aprendiendo, sino que el modelo crítico es muy superior en la detección de las muestras generadas. Lamentablemente ninguno de los modelos probados de Wasserstein convergió a distancia, lo que indica un estado de entrenamiento óptimo. En cambio, el entrenamiento comenzó a ser inestable justo en el momento en que el generador está comenzando a engañar al modelo crítico en la época de 550 en adelante.

En general, las muestras generadas del generador de WGAN (ver Figura 15) parecen contener menos ruido y características más detalladas en comparación con los resultados de los DCGAN. El modelo de WGAN parece centrarse más en los detalles, mientras que no aprende la forma general de un perro. Por lo tanto, vemos diferentes patrones de pelo y

características singulares como patas, cabezas o colas desde la época 200 y en adelante. Como se vio en la época 300, el modelo parece estar distraído por los fondos de la imagen, aprendiendo a generar representaciones precisas de la hierba y los prados. Finalmente, a partir de la época 450 el modelo es capaz de generar caras de perro pero no es capaz de encontrar una representación completa de un objeto con forma de perro. A partir de ahí el modelo comienza a ser cada vez más inestable, deteriorando la calidad de la imagen y la información aprendida. Esto se puede ver en la Figura 11 que muestra las muestras generadas a partir de la época 650.



Figura 11: Salida del modelo WGAN en la época 650.



Figura 12: DCGAN trained on dataset D_0 . (Top) 1.25k iterations, (Upper Middle) 2.5k iterations, (Lower Middle) 5k iterations, (Bottom) 6.5k iterations.



Figura 13: DCGAN trained on dataset D_1 . (Top) 1.25k iterations, (Upper Middle) 2.5k iterations, (Lower Middle) 5k iterations, (Bottom) 6.5k iterations.



Figura 14: DCGAN trained on dataset D_2 . (Top) 1.25k iterations, (Upper Middle) 2.5k iterations, (Lower Middle) 5k iterations, (Bottom) 6.5k iterations.



Figura 15: WGAN trained on dataset D_2 . (Top) 6.5k iterations, epoch 100; (Upper Middle) 13k iterations, epoch 200; (Lower Middle) 20k iterations, epoch 300; (Bottom) 30k iterations, epoch 450.

4. Conclusión

Los modelos entrenados en la Sección 1 y la Sección 2 lograron los puntos de referencia de rendimiento requeridos de una clasificación MNIST mejor que el 99.2 % y una clasificación CIFAR-10 mejor que el 10 % utilizando técnicas como el recocido de la tasa de aprendizaje, el reinicio de la tasa de aprendizaje y el aumento de datos.

La recopilación y compilación de un conjunto de datos propios (ver Sección 3.1) puede ser un proceso tedioso y requiere mucho cuidado y precaución. El simple hecho de utilizar muchos datos no funcionó tan bien como el de utilizar menos datos mejor elegidos. Por lo tanto, construir el conjunto de datos correcto para un proyecto de aprendizaje de máquinas es uno de los pasos más importantes del proceso.

El entrenamiento de los modelos DCGAN (ver Sección 3.2) y WGAN (ver Sección 3.4) no produjo un modelo generativo que sea capaz de producir imágenes foto-realistas de perros, pero los modelos son capaces de inferir ciertos rasgos de la especie objetivo. Características como la silueta, partes de los rostros, piernas y diferentes pieles son parcialmente aprendidas y pueden ser generadas. Un siguiente paso prometedor para mejorar aún más el rendimiento del modelo podría ser el uso de la Normalización del Peso Espectral (ver [11]) para aumentar la estabilidad durante períodos más largos de entrenamiento o la arquitectura mejorada de WGAN como se describe en [12], que evita el recorte de los pesos del crítico. Dado que el entrenamiento era en general muy costoso desde el punto de vista computacional, incluso usando la GPU gratuita de Google Colab, la investigación en estos modelos fue un proceso bastante lento, limitando mi capacidad de buscar completamente el hiperparámetro y el espacio del modelo.

Por último, parece que entrenar los algoritmos de ML en datos del mundo real es mucho más difícil que simplemente aplicarlos a un conjunto de datos bien conocido. Aunque no pude alcanzar plenamente mi objetivo de generar imágenes foto-realistas de los perros, este es un resultado útil para comprender la aplicabilidad real de estos algoritmos en problemas fuera del ámbito académico.

El código fuente de todas las implementaciones de modelos se puede encontrar en [13].

A. Conjuntos de datos

A continuación se muestra un conjunto de muestras de cada conjunto de datos.



Figura 16: Muestras del conjunto de datos D_0



Figura 17: Muestras del conjunto de datos D_1



Figura 18: Muestras del conjunto de datos D_2

B. Resúmenes de modelos

B.1. MLP MNIST classifier model summary

Model: "MNIST MLP model"

Layer (type)	Output Shape	Param #
reshape_1 (Reshape)	(None, 784)	0
gaussian_noise_1 (GaussianNoise)	(None, 784)	0
dense_1 (Dense)	(None, 1024)	803840
gaussian_noise_2 (GaussianNoise)	(None, 1024)	0
activation_1 (Activation)	(None, 1024)	0
batch_normalization_1 (Batch Normalization)	(None, 1024)	4096
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 1024)	1049600
gaussian_noise_3 (GaussianNoise)	(None, 1024)	0
activation_2 (Activation)	(None, 1024)	0
batch_normalization_2 (Batch Normalization)	(None, 1024)	4096
dropout_2 (Dropout)	(None, 1024)	0
dense_3 (Dense)	(None, 1024)	1049600
gaussian_noise_4 (GaussianNoise)	(None, 1024)	0
activation_3 (Activation)	(None, 1024)	0
batch_normalization_3 (Batch Normalization)	(None, 1024)	4096
dropout_3 (Dropout)	(None, 1024)	0
dense_4 (Dense)	(None, 10)	10250
activation_4 (Activation)	(None, 10)	0

Total params: 2,925,578

Trainable params: 2,919,434

Non-trainable params: 6,144

B.2. CNN CIFAR10 classifier model summary

Model: "CIFAR-10 CNN model"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 64)	1792
gaussian_noise_1 (GaussianNoise)	(None, 32, 32, 64)	0
activation_1 (Activation)	(None, 32, 32, 64)	0
batch_normalization_1 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73856
gaussian_noise_2 (GaussianNoise)	(None, 16, 16, 128)	0
activation_2 (Activation)	(None, 16, 16, 128)	0
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 128)	0
conv2d_3 (Conv2D)	(None, 8, 8, 256)	295168
gaussian_noise_3 (GaussianNoise)	(None, 8, 8, 256)	0
activation_3 (Activation)	(None, 8, 8, 256)	0
batch_normalization_3 (BatchNormalization)	(None, 8, 8, 256)	1024
conv2d_4 (Conv2D)	(None, 8, 8, 256)	590080
gaussian_noise_4 (GaussianNoise)	(None, 8, 8, 256)	0
activation_4 (Activation)	(None, 8, 8, 256)	0
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 256)	1024
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 256)	0
conv2d_5 (Conv2D)	(None, 4, 4, 512)	1180160
gaussian_noise_5 (GaussianNoise)	(None, 4, 4, 512)	0
activation_5 (Activation)	(None, 4, 4, 512)	0

batch_normalization_5 (Batch	(None, 4, 4, 512)	2048
conv2d_6 (Conv2D)	(None, 4, 4, 512)	2359808
gaussian_noise_6 (GaussianNo	(None, 4, 4, 512)	0
activation_6 (Activation)	(None, 4, 4, 512)	0
batch_normalization_6 (Batch	(None, 4, 4, 512)	2048
max_pooling2d_4 (MaxPooling2	(None, 2, 2, 512)	0
flatten_1 (Flatten)	(None, 2048)	0
dense_1 (Dense)	(None, 256)	524544
gaussian_noise_7 (GaussianNo	(None, 256)	0
activation_7 (Activation)	(None, 256)	0
batch_normalization_7 (Batch	(None, 256)	1024
dense_2 (Dense)	(None, 256)	65792
gaussian_noise_8 (GaussianNo	(None, 256)	0
activation_8 (Activation)	(None, 256)	0
batch_normalization_8 (Batch	(None, 256)	1024
dense_3 (Dense)	(None, 128)	32896
gaussian_noise_9 (GaussianNo	(None, 128)	0
activation_9 (Activation)	(None, 128)	0
batch_normalization_9 (Batch	(None, 128)	512
dense_4 (Dense)	(None, 10)	1290
activation_10 (Activation)	(None, 10)	0

Total params: 5,134,858

Trainable params: 5,130,122

Non-trainable params: 4,736

B.3. DCGAN model summary

Model: "DCGAN Discriminator"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 32, 32, 64)	1792
leaky_relu_1 (LeakyReLU)	(None, 32, 32, 64)	0
batch_normalization_8 (Batch Normalization)	(None, 32, 32, 64)	256
conv2d_4 (Conv2D)	(None, 16, 16, 128)	73856
leaky_relu_2 (LeakyReLU)	(None, 16, 16, 128)	0
batch_normalization_9 (Batch Normalization)	(None, 16, 16, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 256)	295168
leaky_relu_3 (LeakyReLU)	(None, 8, 8, 256)	0
batch_normalization_10 (Batch Normalization)	(None, 8, 8, 256)	1024
conv2d_6 (Conv2D)	(None, 4, 4, 512)	1180160
leaky_relu_4 (LeakyReLU)	(None, 4, 4, 512)	0
batch_normalization_11 (Batch Normalization)	(None, 4, 4, 512)	2048
flatten_1 (Flatten)	(None, 8192)	0
dense_2 (Dense)	(None, 1)	8193

Total params: 3,124,098

Trainable params: 1,561,089

Non-trainable params: 1,563,009

Model: "DCGAN Generator"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 256)	25856
reshape_1 (Reshape)	(None, 2, 2, 64)	0
batch_normalization_1 (Batch Normalization)	(None, 2, 2, 64)	256
conv2d_transpose_1 (Conv2DTranspose)	(None, 4, 4, 1024)	590848

activation_1 (Activation)	(None, 4, 4, 1024)	0
batch_normalization_2 (Batch)	(None, 4, 4, 1024)	4096
dropout_1 (Dropout)	(None, 4, 4, 1024)	0
conv2d_transpose_2 (Conv2DTr)	(None, 8, 8, 512)	4719104
activation_2 (Activation)	(None, 8, 8, 512)	0
batch_normalization_3 (Batch)	(None, 8, 8, 512)	2048
dropout_2 (Dropout)	(None, 8, 8, 512)	0
conv2d_transpose_3 (Conv2DTr)	(None, 16, 16, 256)	1179904
activation_3 (Activation)	(None, 16, 16, 256)	0
batch_normalization_4 (Batch)	(None, 16, 16, 256)	1024
conv2d_transpose_4 (Conv2DTr)	(None, 32, 32, 128)	295040
activation_4 (Activation)	(None, 32, 32, 128)	0
batch_normalization_5 (Batch)	(None, 32, 32, 128)	512
conv2d_transpose_5 (Conv2DTr)	(None, 64, 64, 64)	73792
activation_5 (Activation)	(None, 64, 64, 64)	0
batch_normalization_6 (Batch)	(None, 64, 64, 64)	256
conv2d_1 (Conv2D)	(None, 64, 64, 32)	18464
activation_6 (Activation)	(None, 64, 64, 32)	0
batch_normalization_7 (Batch)	(None, 64, 64, 32)	128
conv2d_2 (Conv2D)	(None, 64, 64, 3)	867
activation_7 (Activation)	(None, 64, 64, 3)	0

Total params: 6,912,195

Trainable params: 6,908,035

Non-trainable params: 4,160

B.4. WGAN model summary

Model: "WGAN Critic Model"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 32, 32, 64)	1792
leaky_re_lu_1 (LeakyReLU)	(None, 32, 32, 64)	0
batch_normalization_8 (Batch Normalization)	(None, 32, 32, 64)	256
conv2d_4 (Conv2D)	(None, 16, 16, 128)	73856
leaky_re_lu_2 (LeakyReLU)	(None, 16, 16, 128)	0
batch_normalization_9 (Batch Normalization)	(None, 16, 16, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 256)	295168
leaky_re_lu_3 (LeakyReLU)	(None, 8, 8, 256)	0
batch_normalization_10 (Batch Normalization)	(None, 8, 8, 256)	1024
conv2d_6 (Conv2D)	(None, 4, 4, 512)	1180160
leaky_re_lu_4 (LeakyReLU)	(None, 4, 4, 512)	0
batch_normalization_11 (Batch Normalization)	(None, 4, 4, 512)	2048
flatten_1 (Flatten)	(None, 8192)	0
dense_2 (Dense)	(None, 1)	8193

Total params: 3,124,098

Trainable params: 1,561,089

Non-trainable params: 1,563,009

Model: "WGAN Generator Model"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 256)	25856
activation_1 (Activation)	(None, 256)	0
reshape_1 (Reshape)	(None, 2, 2, 64)	0
batch_normalization_1 (Batch Normalization)	(None, 2, 2, 64)	256

conv2d_transpose_1 (Conv2DTr	(None, 4, 4, 1024)	590848
activation_2 (Activation)	(None, 4, 4, 1024)	0
batch_normalization_2 (Batch	(None, 4, 4, 1024)	4096
dropout_1 (Dropout)	(None, 4, 4, 1024)	0
conv2d_transpose_2 (Conv2DTr	(None, 8, 8, 512)	4719104
activation_3 (Activation)	(None, 8, 8, 512)	0
batch_normalization_3 (Batch	(None, 8, 8, 512)	2048
dropout_2 (Dropout)	(None, 8, 8, 512)	0
conv2d_transpose_3 (Conv2DTr	(None, 16, 16, 256)	1179904
activation_4 (Activation)	(None, 16, 16, 256)	0
batch_normalization_4 (Batch	(None, 16, 16, 256)	1024
conv2d_transpose_4 (Conv2DTr	(None, 32, 32, 128)	295040
activation_5 (Activation)	(None, 32, 32, 128)	0
batch_normalization_5 (Batch	(None, 32, 32, 128)	512
conv2d_transpose_5 (Conv2DTr	(None, 64, 64, 64)	73792
activation_6 (Activation)	(None, 64, 64, 64)	0
batch_normalization_6 (Batch	(None, 64, 64, 64)	256
conv2d_1 (Conv2D)	(None, 64, 64, 32)	18464
activation_7 (Activation)	(None, 64, 64, 32)	0
batch_normalization_7 (Batch	(None, 64, 64, 32)	128
conv2d_2 (Conv2D)	(None, 64, 64, 3)	867
activation_8 (Activation)	(None, 64, 64, 3)	0

Total params: 6,912,195

Trainable params: 6,908,035

Non-trainable params: 4,160

C. Lista de razas de perros

Affenpinscher dog	Basque Shepherd Dog
Afghan Hound dog	Basset Artsien Normand dog
Afghan Shepherd dog	Basset Bleu de Gascogne dog
Aidi dog	Basset Fauve de Bretagne dog
Airedale Terrier dog	Basset Hound dog
Akbash dog	Bavarian Mountain Hound dog
Akita dog	Beagle dog
Alano Espaol dog	Beagle-Harrier dog
Alaskan husky dog	Bearded Collie dog
Alaskan Klee Kai dog	Beauceron dog
Alaskan Malamute dog	Bedlington Terrier dog
Alaunt dog	Belgian Shepherd Dog (Groenendael)
Alopekis dog	Belgian Shepherd Dog (Laekenois)
Alpine Dachsbracke dog	Belgian Shepherd Dog (Malinois)
Alpine Mastiff dog	Belgian Shepherd Dog (Tervuren)
Alpine Spaniel dog	Bergamasco Shepherd dog
American Akita dog	Berger Blanc Suisse dog
American Bulldog	Berger Picard dog
American Bully dog	Bernese Mountain Dog
American Cocker Spaniel dog	Bichon Fris dog
American English Coonhound dog	Billy dog
American Eskimo Dog	Black and Tan Coonhound dog
American Foxhound dog	Black and Tan Virginia Foxhound dog
American Hairless Terrier dog	Black Norwegian Elkhound dog
American Pit Bull Terrier dog	Black Russian Terrier dog
American Staffordshire Terrier dog	Black Mouth Cur dog
American Water Spaniel dog	Bloodhound dog
Anatolian Shepherd Dog	Blue Heeler dog
Andalusian Hound dog	Blue Lacy dog
Anglo-Franais de Petite Vnerie dog	Blue Paul Terrier dog
Appenzeller Sennenhund dog	Blue Picardy Spaniel dog
Argentine Polar Dog	Bluetick Coonhound dog
Ariegeois dog	Boerboel dog
Armant dog	Bohemian Shepherd dog
Armenian Gampr dog	Bolognese dog
Artois Hound dog	Border Collie dog
Australian Cattle Dog	Border Terrier dog
Australian Kelpie dog	Borzoi dog
Australian Shepherd dog	Bosnian Coarse-haired Hound dog
Australian Stumpy Tail Cattle Dog[10]	Boston Terrier dog
Australian Terrier dog	Bouvier des Ardennes dog
Austrian Black and Tan Hound dog	Bouvier des Flandres dog
Austrian Pinscher dog	Boxer dog
Azawakh dog	Boykin Spaniel dog
Bakharwal dog	Bracco Italiano dog
Barbado da Terceira dog	Braque d'Auvergne dog
Barbet dog	Braque de l'Ariège dog
Basenji dog	Braque du Bourbonnais dog

Braque du Puy dog	Cirneco dell 'Etna dog
Braque Francais dog	Clumber Spaniel dog
Braque Saint-Germain dog	Collie , Rough dog
Brazilian Dogo	Collie , Smooth dog
Brazilian Terrier dog	Combai dog
Briard dog	Cordoba Fighting Dog
Briquet Griffon Venden dog	Coton de Tulear dog
Brittany dog	Cretan Hound dog
Broholmer dog	Croatian Sheepdog
Bruno Jura Hound dog	Cumberland Sheepdog
Brussels Griffon dog	Curly-Coated Retriever dog
Bucovina Shepherd Dog	Cursinu dog
Bull and Terrier dog	Czechoslovakian Wolfdog
Bull Terrier dog	Dachshund dog
Bulldog	Dalbo dog
Bullenbeisser dog	Dalmatian dog
Bullmastiff dog	Dandie Dinmont Terrier dog
Bully Kutta dog	Danish-Swedish Farmdog
Burgos Pointer dog	Deutsche Bracke dog
Cairn Terrier dog	Doberman Pinscher dog
Canaan Dog	Dogo Argentino
Canadian Eskimo Dog	Dogo Cubano
Cane Corso dog	Dogue de Bordeaux
Cantabrian Water Dog	Drentse Patrijshond dog
Co da Serra de Aires dog	Drever dog
Co de Castro Laboreiro dog	Dunker dog
Co de Gado Transmontano dog	Dutch Shepherd dog
Co Fila de So Miguel dog	Dutch Smoushond dog
Carolina Dog	East Siberian Laika dog
Carpathian Shepherd Dog	East European Shepherd dog
Catalan Sheepdog	Elo dog
Caucasian Shepherd Dog	English Cocker Spaniel dog
Cavalier King Charles Spaniel dog	English Foxhound dog
Central Asian Shepherd Dog	English Mastiff dog
Cesky Fousek dog	English Pointer dog
Cesky Terrier dog	English Setter dog
Chesapeake Bay Retriever dog	English Shepherd dog
Chien Franais Blanc et Noir dog	English Springer Spaniel dog
Chien Franais Blanc et Orange dog	English Toy Terrier (Black & Tan) dog
Chien Franais Tricolore dog	English Water Spaniel dog
Chien-gris dog	English White Terrier dog
Chihuahua dog	Entlebucher Mountain Dog
Chilean Terrier dog	Estonian Hound dog
Chinese Chongqing Dog	Estrela Mountain Dog
Chinese Crested Dog	Eurasier dog
Chinook dog	Field Spaniel dog
Chippiparai dog	Fila Brasileiro dog
Chiribaya Dog	Finnish Hound dog
Chow Chow dog	Finnish Lapphund dog
Cierny Sery dog	Finnish Spitz dog

Flat-Coated Retriever dog	Hokkaido dog
Fox Terrier , Smooth dog	Hortaya borzaya dog
Fox Terrier , Wire dog	Hovawart dog
French Brittany dog	Huntaway dog
French Bulldog	Hygen Hound dog
French Spaniel dog	Ibizan Hound dog
Gaddi Kutta dog	Icelandic Sheepdog
Galgo Espaol dog	Indian pariah dog
Galician Shepherd Dog	Indian Spitz dog
Garafian Shepherd dog	Irish Red and White Setter dog
Gascon Saintongeois dog	Irish Setter dog
Georgian Shepherd dog	Irish Terrier dog
German Longhaired Pointer dog	Irish Water Spaniel dog
German Pinscher dog	Irish Wolfhound dog
German Roughhaired Pointer dog	Istrian Coarse-haired Hound dog
German Shepherd Dog	Istrian Short-haired Hound dog
German Shorthaired Pointer dog	Italian Greyhound dog
German Spaniel dog	Jack Russell Terrier dog
German Spitz dog	Jagdterrier dog
German Wirehaired Pointer dog	Swedish Elkhound dog
Giant Schnauzer dog	Japanese Chin dog
Glen of Imaal Terrier dog	Japanese Spitz dog
Golden Retriever dog	Japanese Terrier dog
Gordon Setter dog	Jindo dog
Gran Mastn de Bornquen dog	Jonangi dog
Grand Anglo-Franais Blanc et Noir dog	Kaikadi dog
Grand Anglo-Franais Blanc et Orange dog	Kai Ken dog
Grand Anglo-Franais Tricolore dog	Kangal Shepherd Dog
Grand Basset Griffon Venden dog	Kanni dog
Grand Bleu de Gascogne dog	Karakachan dog
Grand Griffon Venden dog	Karelian Bear Dog
Great Dane dog	Karst Shepherd dog
Great Pyrenees dog	Keeshond dog
Greater Swiss Mountain Dog	Kerry Beagle dog
Greek Harehound dog	Kerry Blue Terrier dog
Greek Shepherd dog	King Charles Spaniel dog
Greenland Dog	King Shepherd dog
Greyhound dog	Kintamani dog
Griffon Bleu de Gascogne dog	Kishu Ken dog
Griffon Fauve de Bretagne dog	Komondor dog
Griffon Nivernais dog	Koolie dog
Guatemalan Dogo	Koyun dog
Gull Terrier dog	Kromfohrlnder dog
Hamiltonstvare dog	Kumaon Mastiff dog
Hanover Hound dog	Kunming Wolfdog
Hare Indian Dog	Kur dog
Harrier dog	Kuvasz dog
Havanese dog	Kyi-Leo dog
Hawaiian Poi Dog	Labrador Husky dog
Himalayan Sheepdog	Labrador Retriever dog

Lagotto Romagnolo dog	Old Croatian Sighthound dog
Lakeland Terrier dog	Old Danish Pointer dog
Lancashire Heeler dog	Old English Bulldog
Landseer dog	Old English Sheepdog
Lapponian Herder dog	Old English Terrier dog
Lapponian Shepherd dog	Old German Shepherd Dog
Leonberger dog	Old Spanish Pointer dog
Lhasa Apso dog	Old Time Farm Shepherd dog
Lithuanian Hound dog	Olde English Bulldogge
Louisiana Catahoula Leopard Dog	Otterhound dog
Lwchen dog	Pachon Navarro dog
Mackenzie River husky dog	Pandikona dog
Magyar agr dog	Paisley Terrier dog
Mahratta Greyhound dog	Papillon dog
Maltese dog	Parson Russell Terrier dog
Manchester Terrier dog	Pastore della Lessinia e del Lagorai dog
Maremma Sheepdog	Patterdale Terrier dog
Marquesan Dog	Pekingese dog
McNab dog	Perro de Pastor Mallorquin dog
Miniature American Shepherd dog	Perro de Presa Canario dog
Miniature Bull Terrier dog	Perro de Presa Mallorquin dog
Miniature Fox Terrier dog	Peruvian Inca Orchid dog
Miniature Pinscher dog	Petit Basset Griffon Venden dog
Miniature Schnauzer dog	Petit Bleu de Gascogne dog
Miniature Shar Pei dog	Phalne dog
Molossus dog	Pharaoh Hound dog
Molossus of Epirus dog	Phu Quoc Ridgeback dog
Montenegrin Mountain Hound dog	Picardy Spaniel dog
Moscow Watchdog	Plummer Terrier dog
Moscow Water Dog	Plott Hound dog
Mountain Cur dog	Podenco Canario dog
Mucuchies dog	Poitevin dog
Mudhol Hound dog	Polish Greyhound dog
Mudi dog	Polish Hound dog
Mnsterlnder , Large dog	Polish Hunting Dog
Mnsterlnder , Small dog	Polish Lowland Sheepdog
Neapolitan Mastiff dog	Polish Tatra Sheepdog
Nederlandse Kooikerhondje dog	Pomeranian dog
Newfoundland dog	Pont-Audemer Spaniel dog
New Zealand Heading Dog	Poodle dog
Norfolk Spaniel dog	Porcelaine dog
Norfolk Terrier dog	Portuguese Podengo dog
Norrbottenspets dog	Portuguese Pointer dog
North Country Beagle dog	Portuguese Water Dog
Northern Inuit Dog	Posavac Hound dog
Norwegian Buhund dog	Potsdam Greyhound dog
Norwegian Elkhound dog	Prask Krysak dog
Norwegian Lundehund dog	Pudelpointer dog
Norwich Terrier dog	Pug dog
Nova Scotia Duck Tolling Retriever dog	Puli dog

Pumi dog	Shetland Sheepdog
Pungsan dog	Shiba Inu dog
Pyrenean Mastiff dog	Shih Tzu dog
Pyrenean Shepherd dog	Shikoku dog
Rafeiro do Alentejo dog	Shiloh Shepherd dog
Rajapalayam dog	Siberian Husky dog
Rampur Greyhound dog	Silken Windhound dog
Rastreador Brasileiro dog	Silky Terrier dog
Rat Terrier dog	Sinhala Hound dog
Ratonero Bodeguero Andaluz dog	Skye Terrier dog
Ratonero Mallorquin dog	Sloughi dog
Ratonero Murciano de Huerta dog	Slovak Cuvac dog
Ratonero Valenciano dog	Slovakian Wirehaired Pointer dog
Redbone Coonhound dog	Slovensk kopov dog
Rhodesian Ridgeback dog	Smlandsstvare dog
Romanian Mioritic Shepherd Dog	Small Greek Domestic Dog
Romanian Raven Shepherd Dog	Soft-Coated Wheaten Terrier dog
Rottweiler dog	South Russian Ovcharka dog
Russian Spaniel dog	Southern Hound dog
Russian Toy dog	Spanish Mastiff dog
Russian Tracker dog	Spanish Water Dog
Russo-European Laika dog	Spinone Italiano dog
Russell Terrier dog	Sporting Lucas Terrier dog
Saarloos Wolfdog	Stabyhoun dog
Sabueso Espaol dog	Staffordshire Bull Terrier dog
Sabueso fino Colombiano dog	Standard Schnauzer dog
Saint Bernard dog	Stephens Cur dog
Saint John's water dog	Styrian Coarse-haired Hound dog
Saint-Usage Spaniel dog	Sussex Spaniel dog
Sakhalin Husky dog	Swedish Lapphund dog
Salish Wool Dog	Swedish Vallhund dog
Saluki dog	Tahitian Dog
Samoyed dog	Tahltan Bear Dog
Sapsali dog	Taigan dog
arplaninac dog	Taiwan Dog
Schapendoes dog	Talbot Hound dog
Schillerstvare dog	Tamaskan Dog
Schipperke dog	Teddy Roosevelt Terrier dog
Schweizer Laufhund dog	Telomian dog
Schweizerischer Niederlaufhund dog	Tenterfield Terrier dog
Scotch Collie dog	Terceira Mastiff dog
Scottish Deerhound dog	Thai Bangkaew Dog
Scottish Terrier dog	Thai Ridgeback dog
Sealyham Terrier dog	Tibetan Mastiff dog
Segugio Italiano dog	Tibetan Spaniel dog
Seppala Siberian Sleddog	Tibetan Terrier dog
Serbian Hound dog	Tornjak dog
Serbian Tricolour Hound dog	Tosa dog
Seskar Seal Dog	Toy Bulldog
Shar Pei dog	Toy Fox Terrier dog

Toy Manchester Terrier dog	Welsh Corgi , Pembroke dog
Toy Trawler Spaniel dog	Welsh Sheepdog
Transylvanian Hound dog	Welsh Springer Spaniel dog
Treeing Cur dog	Welsh Terrier dog
Treeing Tennessee Brindle dog	West Highland White Terrier dog
Treeing Walker Coonhound dog	West Siberian Laika dog
Trigg Hound dog	Westphalian Dachsbracke dog
Tweed Water Spaniel dog	Wetterhoun dog
Tyrolean Hound dog	Whippet dog
Cimarrn Uruguayo dog	White Shepherd dog
Vanjari Hound dog	Wirehaired Pointing Griffon dog
Villano de Las Encartaciones dog	Wirehaired Vizsla dog
Villanuco de Las Encartaciones dog	Xiasi Dog
Vizsla dog	Xoloitzcuintli dog
Volpino Italiano dog	Yakutian Laika dog
Weimaraner dog	Yorkshire Terrier dog
Welsh Corgi , Cardigan dog	

Referencias

- [1] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014.
- [2] G. Chen, P. Chen, Y. Shi, C.-Y. Hsieh, B. Liao, and S. Zhang, “Rethinking the usage of batch normalization and dropout in the training of deep neural networks,” 2019.
- [3] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, “mixup: Beyond empirical risk minimization,” 2017.
- [4] A. Borji, “Pros and cons of GAN evaluation measures,” *CoRR*, vol. abs/1802.03446, 2018.
- [5] “Wikipedia: List of Dog Breeds.” https://en.wikipedia.org/wiki/List_of_dog_breeds. Accessed: 2020-01-04.
- [6] “Source Code: ImageDatasetCompiler.” <https://github.com/chrismolli/ImageDatasetCompiler>. Accessed: 2020-01-04.
- [7] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” 2014.
- [8] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” 2015.
- [9] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” 2016.
- [10] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan,” 2017.
- [11] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, “Spectral normalization for generative adversarial networks,” *CoRR*, vol. abs/1802.05957, 2018.
- [12] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved training of wasserstein gans,” *CoRR*, vol. abs/1704.00028, 2017.
- [13] “Source Code: MLP, CNN and GAN models for MNIST, CIFAR-10 and custom datasets..” https://github.com/chrismolli/redes_neuronales_semester_project. Accessed: 2020-01-31.