UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

MODELIZACIÓN DE EMPRESAS

Generación de datos sintéticos: Problema de clasificación de imágenes.

Noviembre de 2023

Gonzalo García, Sergio Hernando, Jon Mañueco y Lucas Martín *: platanos_canarias*

Índice

1. Abstract

Here below is our detailed solution to a data analysis problem made up of a classification model using a convolutional neural network. We did this by classifying fruits from images. In the link at the footnote there is an access to our Python notebooks with the implementation and results. The idea of the problem was to train and test the neural network using a full dataset and compare it with the results of said network trained with a very reduced data set treated with data augmentation algorithms: Generative Adversary Networks and image modification procedures from the Albumentations library. After this task was finished, we analysed the results using different metrics such as accuracy, F1-score and sensitivity. After all, we obtained much better results with the full dataset but our data augmented datasets all clasified correctly around 90 % of all the test sets.

Aclaración introductoria

Todos los cuadernos de Python, datasets y cualquier información relevante al trabajo que no esté en este documento se puede encontrar en el siguiente link[1]. La explicación del contenido de los cuadernos y la proveniencia de los conjuntos de datos se puede encontrar en la sección 3.4. Por otro lado, se recomienda la lectura de los cuadernos, ya que consideramos que cierta información sobre las técnicas utilizadas y los detalles más específicos no tenían cabida en este documento, pero no por ello deben ser ignorados. Se recomienda especialmente una lectura intensiva de los cuadernos de clasificación y determinación de los hiperparámetros, pues son los que más literatura presentan.

2. Introducción - Planteamiento del problema

Los problemas de clasificación son un tipo de problema muy habitual del aprendizaje automático. Consisten en diferenciar elementos de un conjunto en función a una variable cualitativa que representa una propiedad que cada objeto cumple de forma distinta. Es decir, clasifica los elementos según su naturaleza, dicho de una forma más coloquial.

Este tipo de problemas utiliza una gran cantidad de datos para poder establecer inferencias en los conjuntos de entrenamiento que extrapolar a los conjuntos de test, por lo que si no se

[1]EL link explícito a la carpeta: https://drive.google.com/drive/folders/1g8wCWtFY4RO-YrCp_dgky-Qwll9t72vB?usp=drive_link

contara con suficientes datos es complicado que el modelo sea eficaz. Precisamente por eso, existen técnicas que dado un conjunto de datos o dataset reducido, que consideramos poco adecuado para clasificar, permiten ampliar los datos iniciales generando más elementos.

Particularizando en el problema propuesto por GMV, contamos con un gran número de frutas divididas en 33 tipos. Aplicando un algoritmo de aprendizaje automático, podemos implementar un clasificador de frutas con ese conjunto de datos. Una vez hecho esto, tomamos una muestra reducida de este conjunto y, utilizando algoritmos y herramientas de data augmentation y data generation, volvemos a configurar el clasificador de frutas y lo comparamos con los resultados obtenidos en primera instancia.

3. Método experimental

3.1. Modelo de clasificación

Para empezar a analizar los datos, decidimos realizar un preprocesamiento inicial de datos para tener las imágenes como información más útil que los .png. Para ello, en el archivo extractData.py convertimos los .png a matrices de pixeles 100x100x3 que representan los píxeles y la codificación RGB de cada uno (así como la comprobación de que los valores son regulares). Incorporamos todas las imágenes a un array de numpy con su etiqueta (el tipo de fruta que les corresponde) en otro array paralelo.

Una vez hecho esto, aplicamos OneHotEncoding para cambiar las etiquetas de cada conjunto por un vector en el que todos los vectores son 0s salvo la posición que se corresponde con la etiqueta. Así no se establecen relaciones de orden entre clases, con lo que se impide la inferencia por orden. También hemos separado los conjuntos de test y de train, y solo accedemos a test una vez entrenado el modelo y solo para evaluar finalmente el modelo. Con esta práctica se previene el *Data Snooping*.

También hemos comprobado que las imágenes estén uniformemente distribuidas por clases, y al estarlo no ha sido necesario realizar ningún proceso de equilibrado de clases ni métricas especiales adaptadas a conjuntos con desequilibrio de clases. Además, como las muestras se tratan de imágenes, todos los valores de las muestras se tratan de píxeles que se encuentran en la misma escala y por lo tanto no ha hecho falta normalizar los datos.

El conjunto de datos, como se explica en el enunciado del problema, no tiene ruido y se tratan de imágenes con fondo blanco en el que cada imagen contiene solo una fruta. Gracias a esto no ha hecho falta más preprocesamiento de los datos

Habiendo preprocesado los datos de esta manera, podemos almacenarlos en un archivo .npz y utilizarlos para muestreo aleatorio y generación de datos.

Para poder controlar la efectividad de nuestra clasificación, procedemos a definir una serie de métricas de éxito basadas en controlar el número de aciertos y fracasos y la forma de los mismos. La descripción más detallada de las métricas que empleamos, la matriz de confusión y nuestra implementación de las mismas se puede consultar en el cuaderno *Modelo_Clasificacion final* que se encuentra en el repositorio cuyo enlace está en el apartado de implementación. En lo relativo al algoritmo de clasificación, empleamos una red neuronal convolucional, muy usada en algoritmos de este tipo.

3.2. Data Augmentation

Data Augmentation o aumento de Datos es un proceso que se lleva a cabo cuando la base de datos es insuficiente o necesitamos un gran número de datos para ser entrenar un modelo, como por ejemplo una red neuronal. En nuestro caso, contábamos con una base de datos de frutas. Como hemos descrito en la sección anterior, hemos tratado los datos de cierta manera, de cada imagen hemos extraído los píxels en una matriz $100 \times 100 \times 3$, es decir cada imagen son 100×100 píxels y cada píxel está descompuesto en 3 colores que representan los colores rojo, verde y azul. Para poder implementar el algoritmo comentado en la primera entrada de la bibliografía, que utiliza funciones de la librería *Albumentations* para modificar de forma aleatoria imágenes del dataset reducido.

Para ello, hemos comenzado realizando una función que genera una máscara para cada fruta. Una máscara es una representación alternativa de una imagen que marca los bits útiles (los de la propia imagen de la fruta) como blanco y el fondo como negro. Esto lo haremos diferenciando los píxeles blancos o casi blancos y poder ayudarnos de ello para luego poder realizar operaciones como la rotación sobre las pocas imágenes que tendremos posteriormente en el dataset.

Una vez hemos procesado las imágenes para obtener sus máscaras, aplicamos una serie de transformaciones aleatorias unas pocas veces a cada imagen del dataset reducido para obtener uno más grande compuesto únicamente por las imágenes modificadas.

Para esto hemos instalado y utilizado la biblioteca *Albumentations*, que nos permite hacer transformaciones geométricas, cambios de color, cambios de enfoque, cambios de brillo y algunas otras aplicaciones. Hemos comenzado realizando diferentes transformaciones, esto lo realizamos creando diferentes pipelines de *Albumentations*. Cada una de las funciones tiene

diferentes parámetros, como la probabilidad con la que se apliquen las transformaciones, el ángulo de giro, la intensidad del difuminado, etc.

Uno de los mayores problemas que nos hemos encontrado es el de mantener el tamaño de la imagen, por ejemplo, hicimos varios intentos de trabajar con perspectivas. Sin embargo, debido a que un cambio de perspectiva produce un cambio en el tamaño de la imagen, y al reescalar se modifica mucho la imagen, no se han usado. Al principio no se consiguió rotar imágenes otro ángulo que no fuera 90°, debido al mismo problema con el tamaño de la imagen. Sin embargo, gracias al estudio de la librería Albumentations y el Github que hay sobre ello, conseguimos mantener los bordes fijos y poder rotarlo un ángulo aleatorio entre un ángulo que hemos definido previamente, manteniendo las propiedades de imagen.

Una vez hecho esto, hemos elaborado 3 funciones que modifican de forma distinta cada imagen. Se toma un número aleatorio para decidir cuál de las funciones se aplica en cada iteración. Las funciones, llamadas *transforms obj*, *transforms bg obj* y *transforms obj 2*, aplican las siguientes transformaciones:

- *transforms obj*: rotacion de 90º, modificación de contraste y de brillo.
- *transforms obj 2*: rotacion aleatoria, modificación de contraste y de brillo, modificación de saturación y simetría horizontal o vertical.
- *transforms bg obj*: rotacion de 90º, fluctuación de color y difuminado.

3.3. Generative Adversarial Networks

El tercer método que se nos proporcionó para la generación de los datos sintéticos fue la utilización de las *GAN's (Generative Adversarial Networks)*. La arquitectura de este modelo se puede observaren la imagen fig. 1:

Donde podemos distinguir los dos elementos que la componen:

- **Generador**: El generador es una red neuronal que recibe como *input* un vector de ruido **z**, generado de manera aleatoria. Este atraviesa la red (cuya estructura describiremos a continuación), y emite un *output*, que recibe el discriminador.

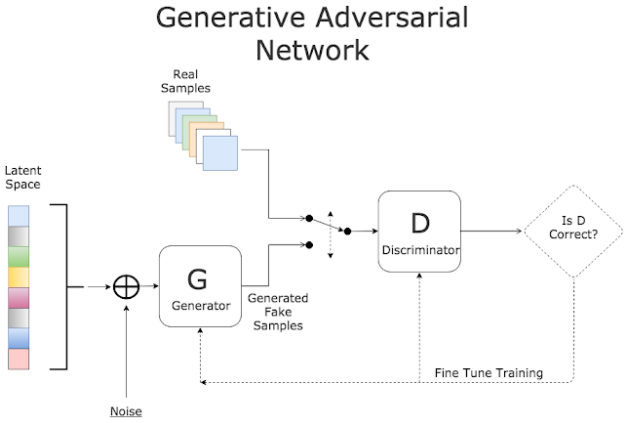

Figura 1: *Arquitectura de una DCGAN.*

- **Discriminador**: El discriminador es otra red neuronal (su estructura también describiremos), cuya función es recibir como *inputs* imágenes tanto reales como falsas (generadas sintéticamente), y devuelve si la imagen dada es verdadera o falsa. Por supuesto, durante este proceso descrito, se van actualizando los pesos y sesgos de las redes neuronales de acuerdo con la función de pérdida y el optimizador del modelo, con el objetivo de que podamos generar imágenes que el discriminador no sea capaz de distinguir. En este punto diremos que hemos sido capaces de generar una imagen de manera sintética.

3.3.1. Arquitecturas del generador y del discriminador

La estructura original del generador del modelo del artículo [Radford et al., 2015], se puede encontrar en la imagen fig. 2:

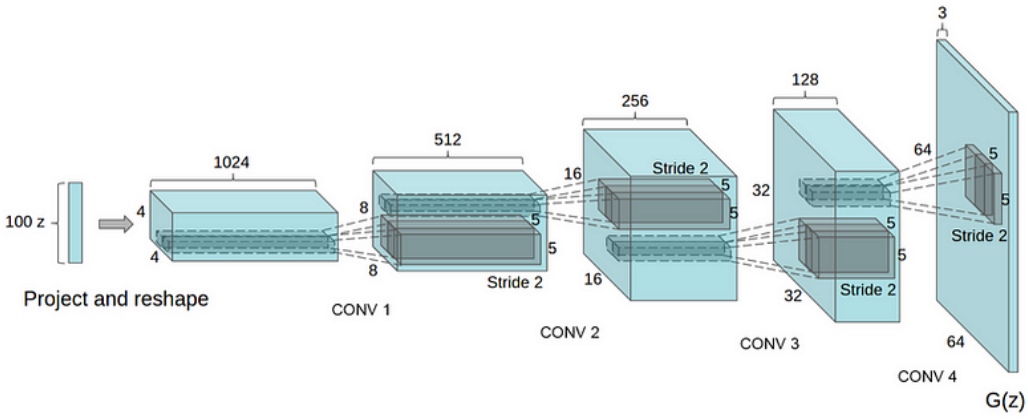

Figura 2: *Estructura original del generador del artículo.*

Donde se puede observar que tiene una estructura formada por un vector de entrada **z** de dimensión 100, una primera capa convolucional de (1024,4,4), una segunda de (512,8,8), una tercera de (256,16,16), una cuarta de (128,32,32), y una salida de (64,64,3); es decir, una imagen de 64×64 píxeles en formato RGB. Sobre esta base, hemos decidido alterar ligeramente la estructura del generador, de manera que nuestra estructura es de la forma:

$$(100\times1) \rightarrow (512\times4\times4) \rightarrow (256\times8\times8) \rightarrow (128\times16\times16) \rightarrow (64\times32\times32) \rightarrow (64\times64\times32) \rightarrow (64\times64\times3)$$

Donde podemos observar que hemos introducido una capa más que la que se indicaba en el paper, pero sin embargo hemos reducido sustancialmente la dimensionalidad en todas las capas (entre dos, particularmente).

Las especificaciones sobre los parámetros se pueden encontrar en el notebook correspondiente a las *GANS*, pues supone un largo desarrollo y carece de interés para esta parte del problema.

Por otro lado, la estructura del discriminador con cinco capas es [2] :

$$(64 \times 64 \times 3) \rightarrow (32 \times 32 \times 32) \rightarrow (16 \times 16 \times 64) \rightarrow (8 \times 8 \times 128) \rightarrow (1 \times 1)$$

Naturalmente, este proceso no se realiza para una única imagen, sino que los parámetros de las redes son recalculados cuando se ha realizado el proceso indicado un cierto número de veces, conocido como *batch_size*.

3.3.2. ¿Por qué este modelo?

Las razones detrás de la elección de este modelo en particular son:

1. **Sustitución de las max-pooling por capas convolucionales**: Esto permite mejorar la precisión del modelo, ya que se evita una reducción en la dimensionalidad, lo que permite evitar una pérdida de información que puede resultar clave en la generación de las imágenes. Además las capas convolucionales permiten una computación del descenso del gradiente más eficiente que las capas *max - pooling*.

2. **La normalización de los batches tanto en el generador como en el discriminador**:Esta elección sirve varios propósitos: en primer lugar mejora la estabilidad de la *GAN*[3]previniendo el colapso de la misma; a su vez, actúa como un regularizador, lo que permite evitar el sobreajuste de los datos tanto del generador como del discriminador; y por último, también mejora el flujo en e descenso del gradiente.

3. **Leaky Relu**: Nuevamente, la utilización de *Leaky Relu* como la función de activación de las capas del generador y del discriminador presenta varias ventajas ya que no solo en un primer lugar evita la desaparición de los gradientes, puesto que permite que

[2]Todas las capas de esta red son también convolucionales, en la siguiente subsubsección explicaremos porqué.

[3]En ocasiones, los generadores de las *GAN* pueden sufrir de un colapso en la generación: el generador se dedica a producir pequeñas modificaciones de las mismas imágenes. Esta es una situación que debemos evitar, en general.

los gradientes sean ligeramente negativos (a diferencia de la Relu habitual), sino que también permite capturar detalles más ricos del *dataset* al presentar un rango de valores posible más grandes. La utilización de esta función de activación es especialmente útil en las capas profundas.

4. **Eficiencia en la reducción de píxeles**: Por último, las imágenes que se nos han proporcionado presentan dimensiones $(100 \times 100 \times 3)$. Utilizar tantos píxeles en la generación de datos supondría un coste computacional que no estamos dispuestos a asumir, ya que creemos que se puede obtener información estadística muy similar generando imágenes de $(64 \times 64 \times 3)$ como las que genera nuestra red para posteriormente realizar un reescalado por interpolación mediante una función integrada en una librería de Python, mucho más eficiente a nivel computacional.

3.4. Implementación

La implementación del modelo ha sido realizada totalmente en Python. Particularmente en tres cuadernos distintos:

- *Modelo_clasificacion*: Este cuaderno contiene el preprocesamiento de los datos, la elección del modelo clasificador, las métricas de error utilizadas y el modelo final elegido. Por último contiene la comparación del rendimiento del modelo en cuatro escenarios diferentes: utilizando el dataset amplio inicial, los dos data set reducidos que ha sido aumentados vía el proceso de *Data Augmentations*, y por último un data set más amplio que ha sido generado por la *GAN*. De esta manera seremos capaces de comprobar la eficacia en los datos generados sintéticamente comparándolos con los datos reales.

- *Modelo_eleccion_hiperparametros*: Este cuaderno está dedicado a la elección mediante optimización del modelo del conjunto de hiperparámetros que utilizaremos posteriormente para entrenar al clasificador en los distintos conjuntos de datos. Debería estar contenido el *Modelo_clasificación*, pero hemo decidido separarlo por altos tiempos de ejecucción que implica este proceso. No es necesario a la hora de mostrar los resultados ya que solo se trata de elegir los mejores hiperparámetros y no recomendamos su ejecución.

- *Modelo_generacion_data_augmentation*: Este es el cuaderno dedicado a la generación de datos sintéticos mediante los procesos de *Data Augmentation* descritos en la sección 3.2. Adquiere como input el dataset completo del que se extrae un dataset reducido y

devuelve un número arbitrario de imágenes que han sido modificadas como se ha des crito anteriormente .

- *Modelo_generacion_gan*: Este es el cuaderno dedicado a la generación de datos sintéticos mediante el entrenamiento de una *Deep Convolutional GAN*, de acuerdo con la arquitectura y las ventajas que presenta este modelo, expuestas en la sección section 3.3. Adquiere como conjunto de entrada un dataset amplio creado mediante el cuaderno anterior, y devuelve un dataset de tamaño arbitrario de acuerdo a como se ha descrito anteriormente. Tampoco recomendamos ejecutar este cuaderno, puesto supone el entrenamiento de 33 redes GAN, lo que implica a su vez varias horas de entrenamiento.

Los datasets que hemos generado y utilizado a lo largo del proyecto son los siguientes:

- *total_data.npz*: Contiene el archivo de imágenes que se nos proporcionó junto con el enunciado del problema, y ya dividido en los conjuntos de entrenamiento y de test que vamos a utilizar.
- *data_dos_transformaciones.npz*: Se trata de un dataset generado a partir de un subconjunto reducido de *total_data* en el cuaderno *Modelo_generacion_data_augmentations* a partir de las transformaciones que implementa el artículo de crear datos sintéticos que se propone en la bibliografía recomendada para el problema.
- *data_tres_transformaciones.npz*: Se trata de un dataset generado a partir de un subconjunto reducido de *total_data* en el cuaderno *Modelo_generacion_data_augmentations* a partir de las transformaciones que implementa el artículo de crear datos sintéticos que se propone en la bibliografía recomendada para el problema más una tercera transformación que se ha implementado estudiando la documentación de la librería Albumentations.
- *data_cuatro_transformaciones.npz*: Se trata de un dataset generado a partir del dataset *data_tres_transformaciones.npz*. La razón por la que la GAN ha sido entrenada con un conjunto de datos tan grande, radica en que este requiere de tantos datos para devolver resultados aceptables.

4. Presentación de resultados

Finalmente con los conjuntos de datos que hemos generado hemos entrenado nuestro modelo con los hiperparámetros finales y hemos evaluado su rendimiento con el conjunto de test. Es

importante mencionar que el conjunto de test fue apartado del conjunto global al principio del experimento y no se ha utilizado para nada más para evitar que este conjunto de test influenciase las decisiones sobre los modelos y no cometer Data Snooping. De esta forma el conjunto de test es una fiel representación del conjunto global de posibles muestras del problema con el que poder evaluar el rendimiento del modelo entrenado con cada uno de los conjuntos de entrenamiento.

A continuación vamos a mostrar la tabla con los resultados finales del modelo entrenado con los destinos conjuntos de entrenamiento:

Conjunto utilizado	Error de test	Accuracy	F1 - Score	Sensitivity	Métrica personalizada
Conjunto 1	0.024357	0.993177	0.992709	0.991694	0.992884
Conjunto 2	1.526816	0.899436	0.878473	0.898398	0.896136
Conjunto 3	0.923319	0.883862	0.868885	0.879858	0.881015
Conjunto 4	1.506544	0.866953	0.845595	0.863690	0.863260

Tabla 1: Comparación de los rendimientos de cada conjunto de datos.

Como podemos observar en la tabla 1, utilizando como conjunto de entrenamiento el conjunto inicial que nos han proporcionado para nuestro modelo (sin la parte de test que hemos apartado antes) obtenemos un resultado final de más del 99 % en *Accuracy*, en *Sensitivity* en *Sensitivity* y en la métrica que hemos creado como métrica general, es decir que obtiene unos resultados casi perfectos en test.

En cuanto al resto de conjuntos, el modelo consigue buenos resultados también aunque no tan perfectos y en algún caso rozando el 90 % en las métricas.

5. Conclusiones

Respecto a la clasificación del conjunto de datos, hemos conseguido obtener un buen modelo que se ajusta correctamente a este problema y consigue generalizar casi a la perfección el mismo. Este modelo de red neuronal convolucional es común y no es muy complejo, ya que podríamos haber utilizado un modelo preentrenado con conjuntos de datos mayores de imágenes y haber entrenado después con nuestro conjunto, pero al obtener tan buenos resultados con el modelo que hemos utilizado no ha hecho falta utilizar un modelo más complejo que este. Esto nos indica también que este problema de clasificación no es difícil en términos del aprendizaje automático ya que es asequible para un modelo con capacidad como el que hemos utilizado. Aún así, hemos tratado de seguir el camino ideal para encontrar

el mejor modelo de clasificación realizando los pasos necesarios de preprocesado, elección del conjunto de hipótesis (modelos), elección de hiperparámetros, etc obteniendo un modelo con muy buenos o casi perfectos resultados para nuestro problema.

Por otro lado la parte de generación de imágenes se trata de un problema menos estudiado y con menos focalización por lo que hemos tenido que probar distintas técnicas y obtener finalmente las que nos han proporcionado mejores resultados. Mientras que en la clasificación es conocido que las redes neuronales convolucionales son muy útiles y ofrecen muy buenos resultados a la hora de clasificar imágenes, en la generación de imágenes a partir de un conjunto reducido no se conocen técnicas definitivas que consigan resultados excelentes en este ámbito, sino que aún esta en desarrollo e investigación.

A la vista de los resultados obtenidos, podemos afirmar que el conjunto que mejor se ha adaptado a nuestro dataset [4] es el segundo, correspondiente a *data_dos_transformaciones.npz*, en el que utilizamos las técnicas de generación recomendadas en la bibliografía. Como los tiempos de ejecución son similares para los dos primeros conjuntos sintéticos (conjuntos 2 y 3), utilizamos la métrica de error del modelo, es el rendimiento el que nos hace inclinarnos por el segundo conjunto de datos. Por otro lado tenemos el conjunto 4, generado mediante las *GAN's* que, aunque tiene un rendimiento muy similar al resto de conjuntos en términos de errores del clasificador, no recomendaríamos utilizar por los costes computacionales que este implica. La generación de los datos superó las 2.5h de tiempo de ejecución[5], y el entrenamiento del clasificador con el conjunto de este modelo también tuvo el tiempo de ejecución más largo (de hecho, se tardó casi lo mismo en entrenar el clasificador con estos datos que con todo el resto de conjuntos juntos). Además, hemos requerido de utilizar un conjunto previamente aumentado para poder entrenar a la red, en caso de no haberlo hecho el rendimiento habría sido mucho peor.

Para terminar, debemos comentar que consideramos que el experimento ha resultado exitoso: hemos sido capaces de crear un clasificador que organice los elementos de los conjuntos de datos de manera claramente eficaz, independiente de si era entrenado por un conjunto de datos sintéticos o real. Por otro lado, hemos creado un conjunto de datos sintéticos cuyas distribuciones de probabilidad creemos están muy cerca de las que se nos proporcionaron en el enunciado, en base a los resultados de pérdida obtenidos. Hemos utilizado varias técnicas de generación de datos, la composición de las mismas y hecho un estudio autocontenido y consistente tanto con las predicciones teóricas, como con los los propios resultados obtenidos.

[4] Y que por lo tanto creemos que ha generado mejor los datos, nótese que es la única métrica que teníamos a nuestra disposición.

[5] Por gran cantidad de datos que se requieren para que devuelva resultados mínimamente aceptables.

Referencias

[Alb,] Documentación de albumentations. https://albumentations.ai/docs/api_reference/full_reference/. Accessed: 2023-11-3.

[Met,] Documentación de métricas de tensorflow. https://www.tensorflow.org/api_docs/python/tf/keras/metrics. Accessed: 2023-11-3.

[Mas,] How to mask an image in opencv python? https://www.tutorialspoint.com/how-to-mask-an-image-in-opencv-python. Accessed: 2023-11-3.

[Bok, 2021] (2021). Getting started with albumentation: Winning deep learning image augmentation technique in pytorch example. https://towardsdatascience.com/getting-started-with-albumentation-winning-deep-learning-image-augmentation-technique Accessed: 2023-11-3.

[Med, 2022] (2022). How to create synthetic dataset for computer vision (object detection). https://medium.com/@alexppppp/how-to-create-synthetic-dataset-for-computer-vision-object-detection-fd8ab2fa5249. Accessed: 2023-11-3.

[Radford et al., 2015] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.