1. Implemente una función para leer las imágenes de Caltech 101 y para retornar una matriz de numpy. Debido a que las imágenes no son homogéneas, se plantean varias dificultades. Proponga un criterio uniforme para resolver estos problemas.

El script principal para esta primera parte es “caltech.py”. Se obtiene la lista de archivos y se hace un análisis del tamaño y formato de las imágenes. En caso de que se fueran a utilizar las imágenes completas, se podría escalar a la resolución máxima. La distribución del dataset se analiza con un boxplot. Para este punto, para limitar el consumo de memoria, se redujeron las imágenes (sin perder la relación de aspecto) y se rellenaron con ceros para que cumplan el mismo tamaño. Las imágenes en escala de grises (en un solo canal) se convirtieron a RGB replicando el contenido del canal. Las imágenes se almacenan en un tensor [Nºimagen x ancho x alto x canal]

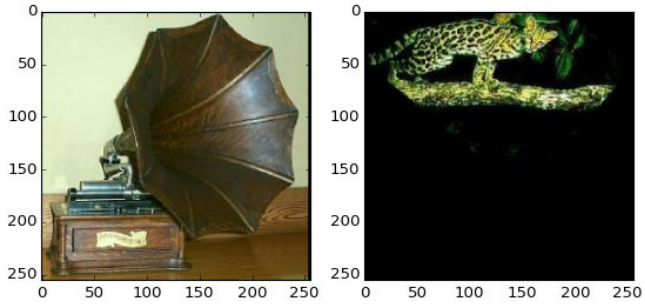


Fig 1: Ejemplo de imágenes cargadas en “images”.

2. Implemente funciones de preprocesamiento para las imágenes de manera de extraer ventanas de tamaño 16x16 en posiciones aleatorias de la imágen, con el valor medio de las imágenes sobre cada canal sea 0 y la varianza 1. Describa el efecto sobre las imágenes procesadas.

La función “getPatches.py” extrae N parches de cada imágen. Se utilizaron las imágenes originales para evitar los bordes negros. Si las imágenes del dataset son heterogéneas, como se espera que sean, las diferencias de tamaño no deberían repercutir en la información contenida en los parches. Por ejemplo, dos objetos iguales pueden estar mapeados en diferentes tamaños o ángulos, aún cuando las imágenes tengan el mismo tamaño.

La función “normalizeImage.py” realiza un z-score en cada canal para obtener una imagen de media 0 y varianza 1. El efecto principal es normalizar los rangos de valores para que las entradas al clasificador obtengan pesos similares. En las imágenes, se puede observar que se hace un balance de color, normalizando el brillo general y el peso de los canales. Si bien es deseable eliminar estos efectos que en general dependen de la luz ambiente y tipo de captura, también hay información del balance de color entre canales que se pierde al normalizar.

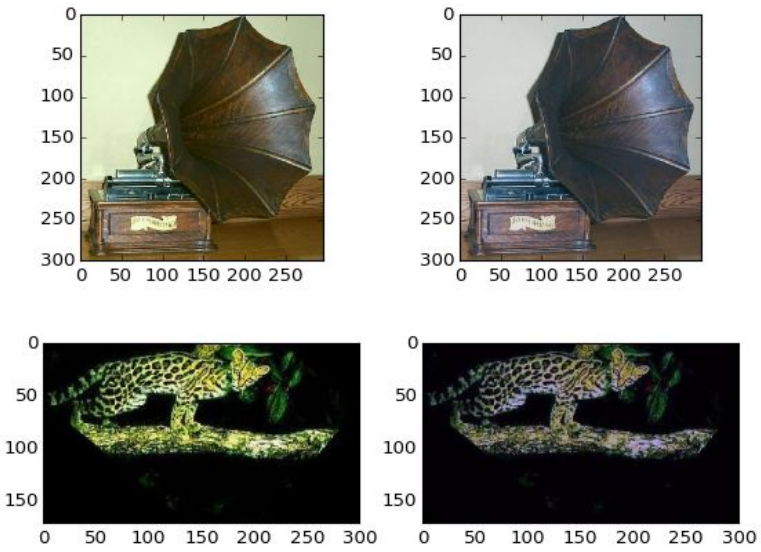


Fig 3: A la izquierda las imágenes originales, a la derecha las imágenes luego de la normalización.

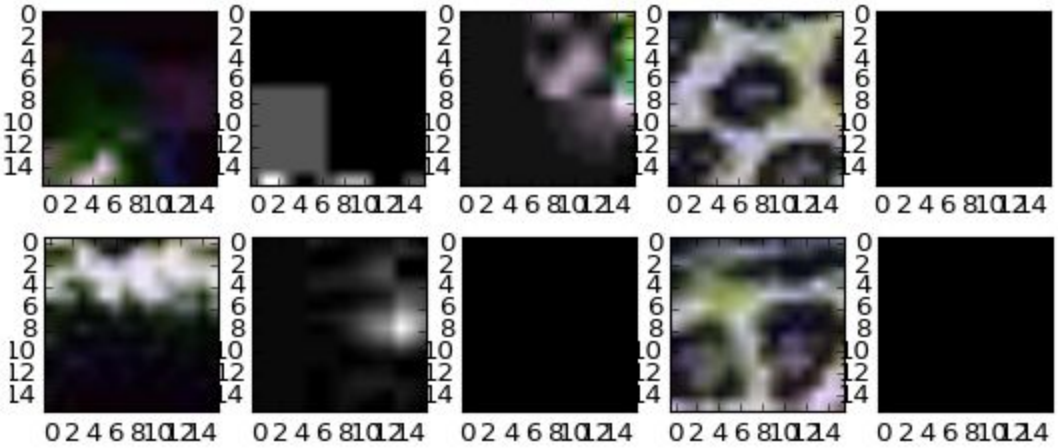


Fig 4: Parches normalizados de 16x16 extraídos de una imagen de ejemplo.

3. Investigue y describa el uso de whitening en datasets de imágenes.

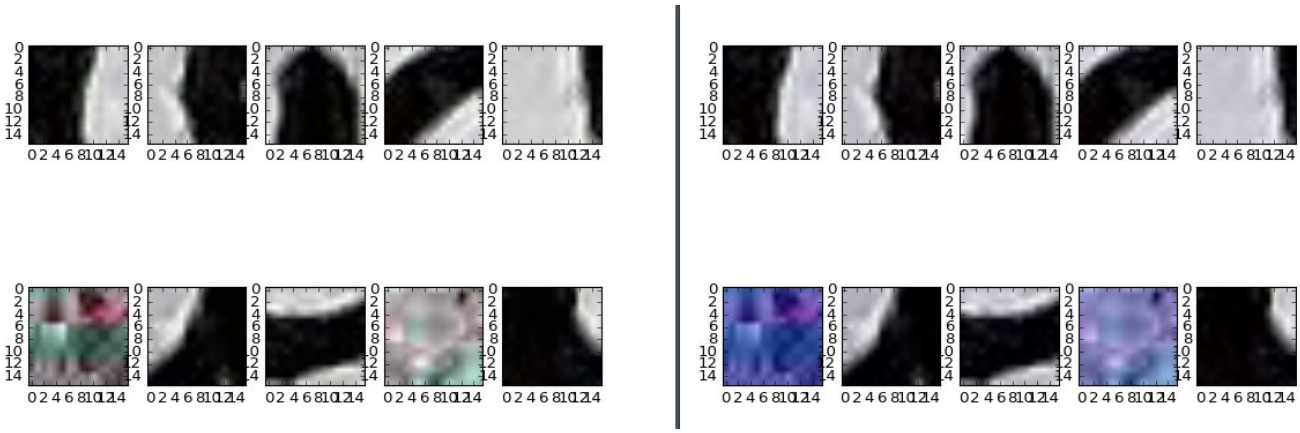
El objetivo del whitening es decorrelacionar variables. Los datos son transformados a un conjunto de variables donde la matriz de covarianza es la identidad (o una matriz diagonal con diferentes varianzas). En cierta forma es similar a la transformación anterior, con la diferencia que la anterior normaliza las varianzas, pero el whitening elimina también las correlaciones entre variables.

Una forma de hacer whithening es efectuar PCA. Esta transformación genera un conjunto de variables decorrelacionadas entre sí. También se puede utilizar este procedimiento para reducir la cantidad de variables maximizando la varianza de los datos.

En el caso de las imágenes, si se ve el conjunto de píxeles como variaels (16x16 en el caso de los parches), el whitening redefine los píxeles de forma que la imagen se vería como ruido blanco al graficarse. El whitening reduce la redundancia (en las imagenes, pixeles contiguos tienen información similar). Dependiendo de la técnica de aprendizaje maquinal, el whitening puede ser necesario o no.

4. Utilice las funciones de los ejercicios anteriores sobre las imágenes de test de las palabras manuscritas del texto de Borges. Describa el efecto sobre las imágenes procesadas.

El script es 'borges.py'. El estandarizado no parece influir demasiado en la representación (que esta escalada para graficar), pero sí es necesario para balancear los pesos de las entradas.

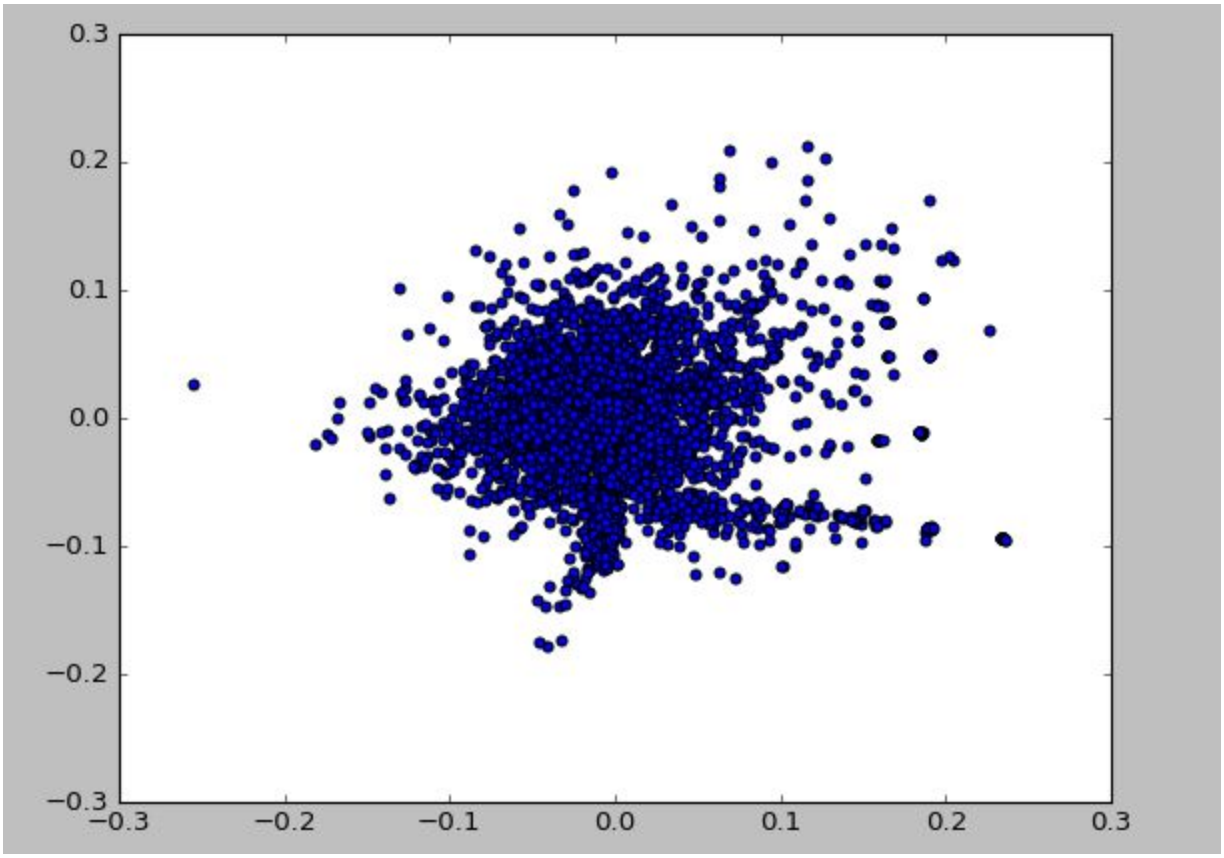


5. Utilizando este dataset con textos de Nietzsche, enumere cada oración y transfórmela a vectores utilizando:
    a. Un vector con la frecuencia de cada palabra utilizando las 1000 palabras más comunes.
    b. Un vector con la frecuencia inversa de cada palabra utilizando las 1000 palabras menos comunes.

En el script 'nietzsche.py' se realiza este punto. Se separaron las oraciones con los signos de puntuación '.?!'. Otros signos de puntuación como ',', ';', '(' se consideraron como "palabras" y también se separaron. Como resultado, se obtuvieron las 1000 palabras más comunes y las 1000 menos comunes. Estas últimas tienen todas solo 1 aparición en el texto. Algunas oraciones son solo el número de página o están mal formateadas, por lo que podría ser necesario un pre-procesamiento adicional.

6. Reduzca la dimensionalidad de los datos generados en 5a y 5b utilizando PCA y grafique los resultados en 2D (puede utilizar matplotlib o gnuplot).

Se puede ver en el scatterplot que las oraciones se distribuyen en una zona homogenea y se pueden diferenciar algunas que se separan más. Por ejemplo, analizando los que se observan en la imagen a la derecha:



Lo que podrían ser títulos de secciones:
oración 2548, X=(0.24,-0.10): 89 =Vanity.
oración 2664, X=(0.24,-0.09): 100 =Shame.
oración 2348, X=(0.23,-0.09): 49 =Well-Wishing.
oración 2477, X=(0.23,-0.09): 71 =Hope.
oración 2063, X=(0.23,-0.09): 19 =Number.

Y en el extremo a la izquierda, posiblemente por la alta frecuencia de comas en una oración corta:
oración 1743, X=(-0.26,0.03): Hand, gait, face, changed?