Red Neuronal Convolucional: Modelo Clasificador de Perros y Gatos

Christopher Gómez, *Ingeniería Mecatrónica (chrisdavid29@gmail.com)*, **Jairo González**, Ingeniería Mecatrónica *(jgonzalez1998@live.com), y* ***Kenneth Harewood****, Ingeniería Mecatrónica (kenneth.w.23.98@gmail.com)* ***Universidad Latina de Panamá****.*

***Abstract*-- El objetivo de este proyecto es clasificar la imagen de entrada como una imagen de perro o de gato. Se analizará la entrada de imagen que es provista al sistema y el resultado predicho se brindará como salida. El algoritmo de aprendizaje automático, red neuronal convolucional, se utiliza para clasificar la imagen.**

El modelo así implementado puede extenderse a un dispositivo móvil o a cualquier sitio web según las necesidades del desarrollador.

I. Introducción

El conjunto de datos Perros vs. Gatos es un conjunto de datos estándar de visión por computador que implica la clasificación de las fotos ya sea que contengan un perro o un gato.

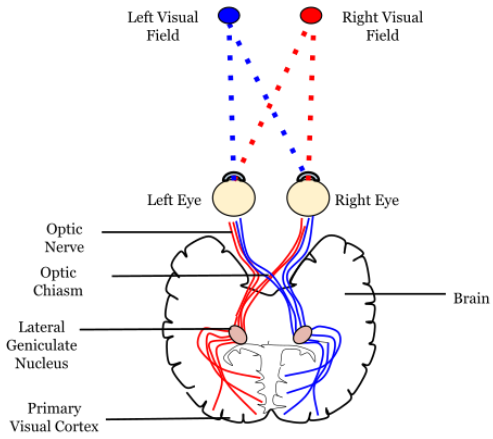
Aunque el problema parece sencillo, sólo se ha abordado de forma efectiva en los últimos años mediante el uso de redes neuronales convolucionales de aprendizaje profundo. Aunque el conjunto de datos se ha resuelto de forma efectiva, puede utilizarse como base para el aprendizaje y la práctica de cómo desarrollar, evaluar y utilizar las redes neuronales convolucionales de aprendizaje profundo para la clasificación de imágenes desde cero.

Esto incluye cómo desarrollar un robusto arnés de prueba para estimar el rendimiento del modelo, cómo explorar las mejoras del modelo y cómo guardar el modelo y posteriormente cargarlo para hacer predicciones sobre nuevos datos.

II. Redes Neuronales Convolucionales (CNNs)

Las CNNs son modelos de aprendizaje profundo adecuados para analizar imágenes visuales. Están fuertemente influenciados por la forma en que nosotros, los humanos, vemos el mundo que nos rodea. Antes de proceder a las CNNs para analizar cómo ven las computadoras, enfoquémonos en cómo los humanos son capaces de hacerlo.

A. ¿Cómo ven los humanos?



Estructura visual del cerebro humano

Cuando vemos un objeto, los receptores de luz en sus ojos envían señales a través del nervio óptico a la corteza visual primaria, donde se procesa la entrada. La corteza visual primaria tiene sentido de lo que el ojo ve.

La estructura jerárquica profundamente compleja de las neuronas y las conexiones en el cerebro juegan un papel importante en este proceso de recordar y etiquetar los objetos.

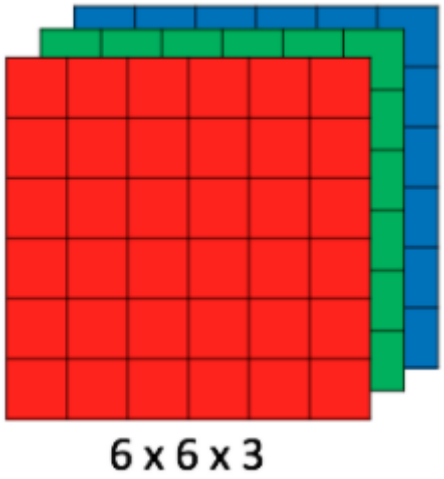
Sin entrar en más detalles, el cerebro humano analiza las imágenes en las capas de creciente complejidad. La primera capa distingue atributos básicos como líneas y curvas. En niveles más altos, el cerebro reconoce que una combinación de bordes y colores es, por ejemplo, un tren o un perro.

Las neuronas corticales individuales responden a los estímulos sólo en una región restringida del campo visual conocida como campo receptivo. Los campos receptivos de las diferentes neuronas se superponen parcialmente de tal manera que cubren todo el campo visual.

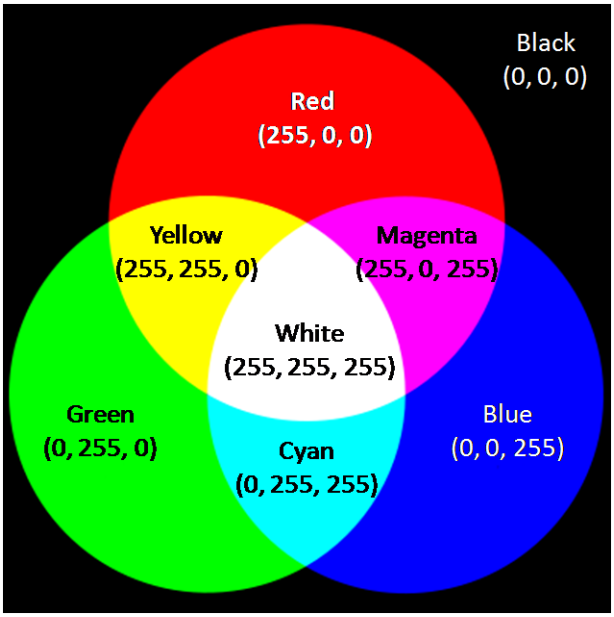
B. ¿Cómo ven las computadoras?

Para una computadora, una imagen es sólo un conjunto de valores. Típicamente es una matriz tridimensional (RGB) de valores de píxeles.

Por ejemplo, una representación de imagen abstracta 6x6 RGB se vería así.



Donde cada píxel tiene un valor específico de rojo, verde y azul que representa el color de un píxel dado.



Pero esto es sólo una entrada al sistema de reconocimiento visual. ¿Cómo puede un ordenador darle sentido? De la misma manera que nosotros podemos hacerlo.

La CNN procesa las imágenes utilizando matrices de pesos llamadas filtros (características) que detectan atributos específicos como bordes verticales, bordes horizontales, etc. Además, a medida que la imagen avanza a través de cada capa, los filtros son capaces de reconocer atributos más complejos. El objetivo último de la CNN es detectar lo que está sucediendo en la escena.

C. Arquitectura de la CNN

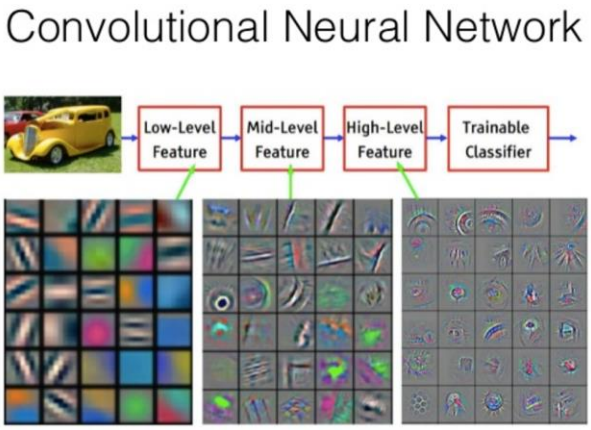
Hemos establecido antes que, de manera similar a los humanos, las computadoras son capaces de analizar imágenes visuales usando sistemas de capas profundas.
Vayamos paso a paso y analicemos cada capa de la **Red Neural Convolucional**.

Entrada

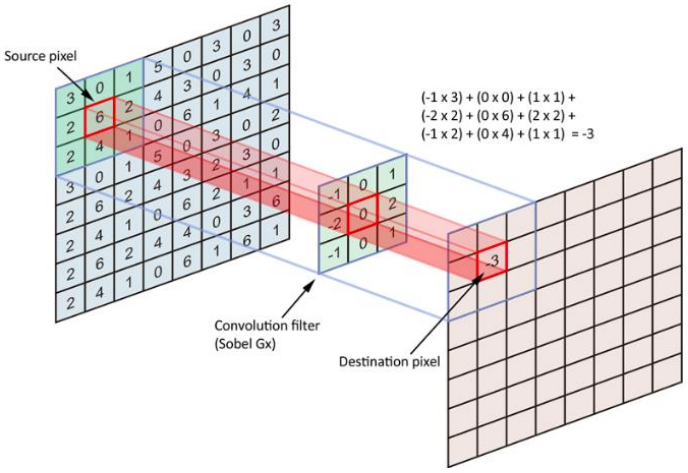
Una matriz de valores de píxeles en forma de [WIDTH, HEIGHT, CHANNELS]. Supongamos que nuestra entrada es [128x128x3].

Convolución

El objetivo final de esta capa es recibir un **mapa de características**. Por lo general, empezamos con un número bajo de filtros para la detección de características de bajo nivel. Cuanto más nos adentramos en la CNN, más filtros (normalmente también son más pequeños) usamos para detectar características de alto nivel.



La detección de características se basa en 'escanear' la entrada con el filtro de un tamaño dado y aplicar cálculos matriciales para derivar un mapa de características.



Echemos un vistazo al siguiente filtro 3x3 con paso 1.

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

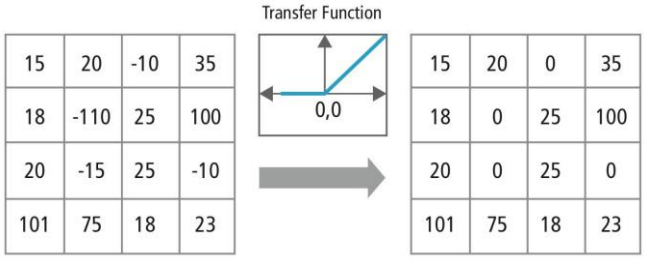
4		

Asumiendo que usamos 12 filtros, terminaremos con una

salida [128x128x12].

Activación

Sin entrar en más detalles, utilizaremos la función de activación ReLU que devuelve 0 por cada valor negativo en la imagen de entrada mientras que devuelve el mismo valor por cada valor positivo.

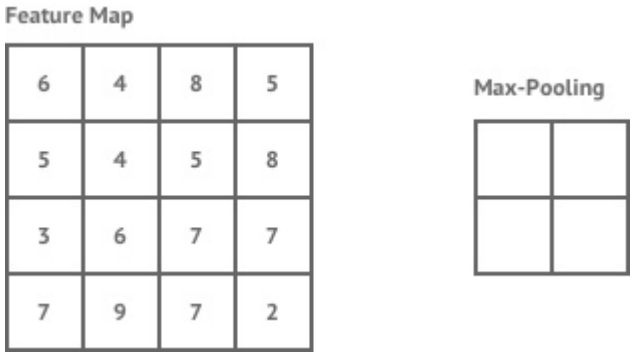


La forma permanece inalterada [128x128x12]

Pooling

El objetivo de esta capa es proporcionar una varianza espacial, lo que significa simplemente que el sistema será capaz de reconocer un objeto como tal, incluso cuando su apariencia varía de alguna manera.

La capa de agrupación realizará una operación de reducción de muestras a lo largo de las dimensiones espaciales (width, height), dando como resultado una salida como [64x64x12] para pooling_size = (2, 2).



(2, 2) pooling

Fully Connected

En una capa completamente conectada, aplanamos la salida de la última capa de convección y conectamos cada nodo de la capa actual con el otro nodo de la siguiente capa.

Las neuronas en una capa totalmente conectada tienen conexiones completas a todas las activaciones de la capa anterior, como se ve en las redes neuronales normales y funcionan de forma similar.

La última capa de nuestra CNN calculará las puntuaciones de probabilidad de la clase, resultando en un volumen de [1x1xNUMBER_OF_CLASSES].

D. Comparativa entre una red neuronal «tradicional» y una CNN

En una red tradicional, los datos de entrada de la capa inicial son las características que analizamos, por ejemplo: ancho, alto, grosor, etc., en una CNN estos datos serán pixeles de una imagen, si es color, serán 3 capas para rojo, verde, azul.

En una red tradicional, para las capas ocultas, elegimos una cantidad de neuronas, en una CNN, tenemos dos tipos, convolución (con un tamaño de kernel y una cantidad de filtros) y subsampling.

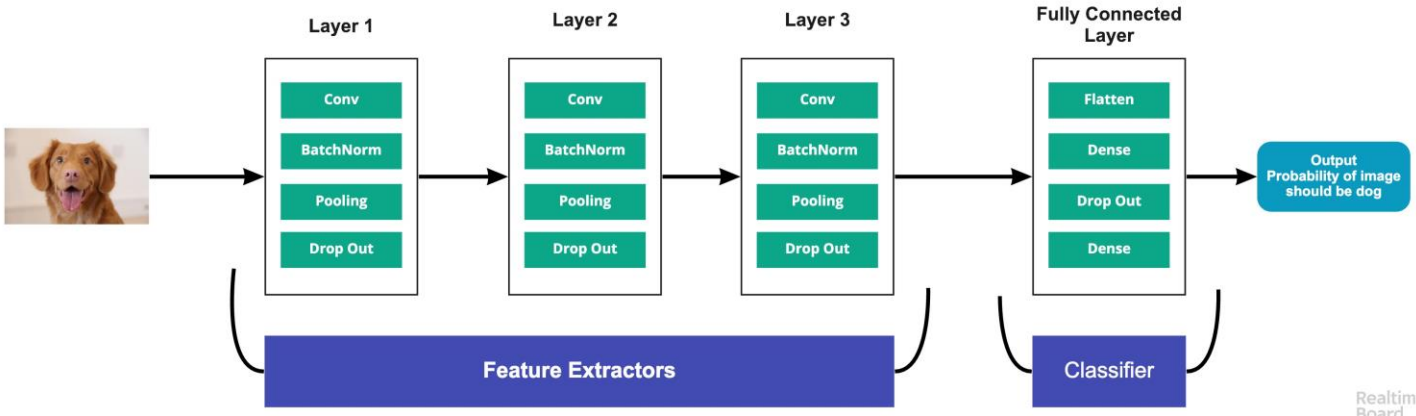
En la capa de salida, una red tradicional tenemos que ingresar la cantidad de neuronas que queremos clasificar, en una CNN, debemos «aplanar» la última convolución con una (ó más) capas de neuronas ocultas «tradicionales» y hacer una salida mediante SoftMax a la capa de salida que clasifica las neuronas.

Algo que tienen en común una red tradicional y una CNN, es que las dos tienen aprendizaje supervisado.

Las interconexiones, en una red tradicional, son entre capas, es decir, todas las neuronas de una capa con la siguiente, en cambio, en una CNN son muchas menos conexiones necesarias, pues realmente los pesos que ajustamos serán los de los filtros/kernels que usamos.

En una red tradicional, el significado de la cantidad de capas ocultas es desconocido, no representa algo en sí mismo, pero en una CNN, las capas ocultas son mapas de detección de características de la imagen y tienen jerarquía: primeras capas detectan líneas, luego curvas y formas cada vez más elaboradas.

En Backpropagation, las redes tradicionales se utilizan para ajustar los pesos de todas las interconexiones de las capas, en cambio, las CNN se utiliza para ajustar los pesos de los kernels.



III. Conjunto de datos y Tareas

Nuestra tarea básica es crear un algoritmo para clasificar si una imagen contiene un perro o un gato. La entrada para esta tarea son imágenes de perros o gatos del conjunto de datos de entrenamiento, mientras que el resultado es la precisión de clasificación del conjunto de datos de prueba.

El conjunto de datos usado es el conjunto de datos de Asirra proporcionado por Microsoft Research. Nuestro kit de entrenamiento contiene 25000 imágenes, incluyendo 12500 imágenes de perros y 12500 imágenes de gatos, mientras que el conjunto de datos de prueba contiene 12500 imágenes. El tamaño de estas imágenes es de alrededor de 350X500 pixeles.

Nuestra tarea de aprendizaje es entrenar un modelo de clasificación para determinar el límite de decisión para el conjunto de datos de formación.

IV. Nuestro Modelo

Lo primero que se hará será importar las librerías necesarias, para iniciar se hará uso de Numpy (que es un paquete de Python que significa “Numerical Python”, es la librería principal para la informática científica, proporciona potentes estructuras de datos, implementando matrices y matrices multidimensionales. Estas estructuras de datos garantizan cálculos eficientes con matrices), Pandas (es un paquete de Python que proporciona estructuras de datos similares a los dataframes de R. Pandas depende de Numpy, la librería que añade un potente tipo matricial a Python. Los principales tipos de datos que pueden representarse con pandas son datos tabulares con columnas de tipo heterogéneo con etiquetas en columnas y filas, y series temporales), Tensorflow (es una librería de código abierto desarrollada por Google cuya finalidad es extender el uso del deep learning a un rango de tareas muy amplio, los modelos matemáticos utilizados en TensorFlow son redes neuronales, que en función de la arquitectura de capas y neuronas que la conforman se podrá modelizar desde un simple modelo de regresión hasta una arquitectura mucho más compleja de machine learning), sklearn(sklearn, llamada también Scikit-Learn, cuenta con algoritmos de clasificación, regresión, clustering y reducción de dimensionalidad, además, presenta la compatibilidad con otras librerías de Python como NumPy, SciPy y matplotlib.) y modelos asociados con Keras (es una biblioteca de redes neuronales de código abierto escrita en Python. Es capaz de ejecutarse sobre TensorFlow, entre otros .Está especialmente diseñada para posibilitar la experimentación en más o menos poco tiempo con redes de aprendizaje profundo, sus fuertes se centran en ser amigable para el usuario, modular y extensible).

```
import numpy as np
import pandas as pd
from keras.preprocessing.image import ImageDataGenerator,
load_img
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import random
import os
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
```

Luego se definen las variables del filtro y se prepara el dataset que será entrenado.

```
FAST_RUN = False
IMAGE_WIDTH=128
IMAGE_HEIGHT=128
IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)
IMAGE_CHANNELS=3

filenames = os.listdir("./input/train")
categories = []
for filename in filenames:
    category = filename.split('.')[0]
    if category == 'dog':
        categories.append(1)
    else:
        categories.append(0)

df = pd.DataFrame({
    'filename': filenames,
    'category': categories
})
```

Luego se prueba haciendo que muestre una imagen aleatoria del mismo, usando la librería matplotlib

```
sample = random.choice(filenames)
image = load_img("./input/train/"+sample)
plt.imshow(image)
```

Después se empezará a construir el modelo, usando las instancias de Keras

```
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu',
input_shape=(IMAGE_WIDTH, IMAGE_HEIGHT,
IMAGE_CHANNELS)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='rmsprop',
metrics=['accuracy'])

model.summary()
```

Como el programa puede ser propenso a tener over fitting, se hace un Early Stopping que detenga el aprendizaje después de 10 epochs y si no se incrementa exactitud en el entrenamiento…

```
earlystop = EarlyStopping(patience=10)

learning_rate_reduction =
ReduceLROnPlateau(monitor='val_acc',
                                patience=2,
                                verbose=1,
                                factor=0.5,
                                min_lr=0.00001)

callbacks = [earlystop, learning_rate_reduction]
```

Ya cuando se crea el código para disminuir overfitting, entonces se procede a ajuste y así generar epochs

```
epochs=3 if FAST_RUN else 50
history = model.fit_generator(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=total_validate//batch_size,
    steps_per_epoch=total_train//batch_size,
    callbacks=callbacks)
```

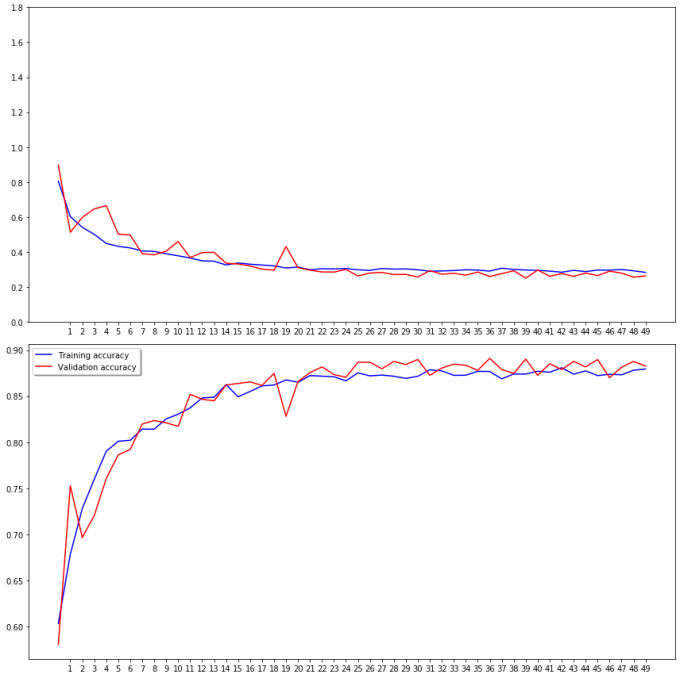
Al terminar el ajuste, se procede a hacer una gráfica para ver mejor como fue el desempeño y se guardo el modelo ya entrenado

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))
ax1.plot(history.history['loss'], color='b', label="Training loss")
ax1.plot(history.history['val_loss'], color='r', label="validation
loss")
ax1.set_xticks(np.arange(1, epochs, 1))
ax1.set_yticks(np.arange(0, 2, 0.2))

ax2.plot(history.history['acc'], color='b', label="Training
accuracy")
ax2.plot(history.history['val_acc'], color='r',label="Validation
accuracy")
ax2.set_xticks(np.arange(1, epochs, 1))
ax1.set_yticks(np.arange(0, 2, 0.2))

legend = plt.legend(loc='best', shadow=True)
plt.tight_layout()
plt.show()

model.save_weights("model.h5")
```



V. Interacción con Web App

En este proyecto se decidió usar la plataforma Anvil para hacer una interfaz entre el código creado y entrenado, Anvil está diseñado para que podamos desarrollar software más rápido. Es una plataforma para construir y alojar aplicaciones web completas escritas íntegramente en Python.

Utilizamos Anvil Uplink para enlazar nuestro código en Jupyter Notebook con Anvil. Anvil Uplink hace que cualquier código de Python se comporte como un módulo de servidor, y nos permite poder llamar funciones desde la aplicación, llamar funciones en la aplicación desde Juypter Notebook, almacenar datos en la aplicación desde Jupyter Notebook entre otros.

Usando este código podemos hacer una conexión entre Anvil y el Jupyter

```
import anvil.server
import anvil.media
from PIL import Image
import PIL
from keras.preprocessing import image

anvil.server.connect("Inserte Key de Anvil")

@anvil.server.callable
def classify_image(file):
    with anvil.media.TempFile(file) as filename:
        img = load_img(filename)
        img = img.resize((128, 128),
resample=PIL.Image.BICUBIC)
        arr = image.img_to_array(img)
        arr = np.expand_dims(arr, axis=0)
        arr /= 255.0

        score = model.predict(arr)
        return ('dog' if score < 0.3 else 'cat' float(score))
```

RECONOCIMIENTOS

Agradecimiento al ingeniero Rangel Alvarado por la guía a lo largo del curso de Inteligencia Artificial y las correcciones en el código de predicción y parámetros de aprendizaje.

REFERENCIAS

[1] Aurlein Gron, “Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems,” O'Reilly Media, *primera edición*, capitulo 13, Marzo 2017.

[2] Aurélien Geron, A series of Jupyter notebooks that walk you through the fundamentals of Machine Learning and Deep Learning in python using Scikit-Learn and TensorFlow, https://github.com/ageron/handson-ml. 2017

[3] NVIDIA course, “Fundamentals of Deep Learning for Computer Visison,” NVIDIA Deep Learning Institute, Enero 2018.

[4] ASIRRA, “Dog vs. Cat” by Microsoft Research, Enero 2013. URL: https://www.kaggle.com/c/dogs-vs-cats/data.

[5] Jason Browniee, “How to Classify Photos of Dogs and Cats (with 97% accuracy),” *Machine Learning Mastery*, October 2019. URL: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/.

[6] K M Yatheendra Pravan, “Project Idea | Cat vs Dog Image Classifier using CNN implemented using Keras,” *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/project-idea-cat-vs-dog-image-classifier-using-cnn-implemented-using-keras/.

[7] Uysim Ty, “Get Start Image Classification,” *Kaggle*, Abril 2019. URL: https://www.kaggle.com/uysimty/get-start-image-classification/notebook.

[8] Chris Varano, “Disentangling Variational Autoencoders for Image Classification,” *Stanford University Course Project Reports: Spring 2017*. URL: http://cs231n.stanford.edu/reports/2017/pdfs/3.pdf.