Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Proyecto final

Visual Question Answering

Integrantes: Joaquín Zepeda
Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla
Ayudante: Danilo Moreira
Santiago de Chile

Índice de Contenidos

Índice de Figuras

Índice de Tablas

1. Introducción

Visual Question Answering o VQA consiste en, dadas una imagen y una pregunta relacionada con la imagen, entregar una respuesta precisa a esta última en lenguaje natural. Si bien el ser humano puede de forma simple responder a preguntas dadas imágenes, construir un sistema computacional que pueda realizar esta tarea es algo muy ambicioso y que todavía esta en desarrollo. El objetivo de este proyecto es diseñar y entrenar un modelo de que resuelva la tarea de VQA. Para esto, se considerará una versión reducida del dataset VQA, que solo contiene preguntas con respuesta múltiple en vez de preguntas abiertas, y es de menor tamaño que el dataset principal. Las imágenes del dataset reducido contienen dibujos abstractos de escenas en vez de imágenes de la realidad. Resolver este problema mezcla múltiples ramas de inteligencia computacional: visión computacional; procesamiento de lenguaje natural; representación de información; razonamiento. Considerando esto, en está tarea se desarrollan e implementan 2 soluciones a este problema, el primero utilizando Bag of Words como módulo de lenguaje y una red VGG19 como módulo de visión y el segundo utilizando una red recurrente LSTM como módulo de lenguaje la misma red VGG19 como módulo de visión.

2. Desarrollo

2.1. Descripción del dataset

El dataset está compuesto por 3 conjuntos: entrenamiento, validación y prueba. Los primeros 2 conjuntos poseen: una imagen, una pregunta, alternativas y la respuesta a la pregunta. Por otro lado el conjunto de prueba solo tiene una imagen y pregunta, no así una respuesta. En la figura 1.a se puede observar un ejemplo de una imagen del dataset de validación, se observa una pregunta simple respecto a la imagen y la respuesta correcta. El conjunto de entrenamiento tiene un tamaño de 60000 y el conjunto de validación tiene un tamaño de 30000, debido a esto los tiempos de entrenamiento son bastante extensos.

(a) Original image (b) Resized image

Figura 1

Las muchas preguntas variadas, pero las respuestas se pueden dividir en 3 grandes tipos: **yes/no**, **number**, **other**.

2.2. Preprocesamiento de los datos

Para trabajar con la red VGG19 se realiza un *resize* a las imágenes para que tengan el tamaño de 224x224 que corresponde al input de esta red. El nuevo tamaño de la imagen se puede observar en la figura 1.b. Además, es necesario realizar un vocabulario con todas las palabras para las preguntas y otro para las respuestas. Estos vocabularios lo que hacen es guardar todas las palabras y asignarles un número con el cual se representen.

2.3. Modelos

Los modelos propuestos tienen la estructura de la figura 2, existen 2 módulos principales: el modulo de extracción de características de la imagen y el modulo de extracción de características para las

preguntas. Luego estos se combinan y para luego ser llevados a un clasificador el cual determina la respuesta.

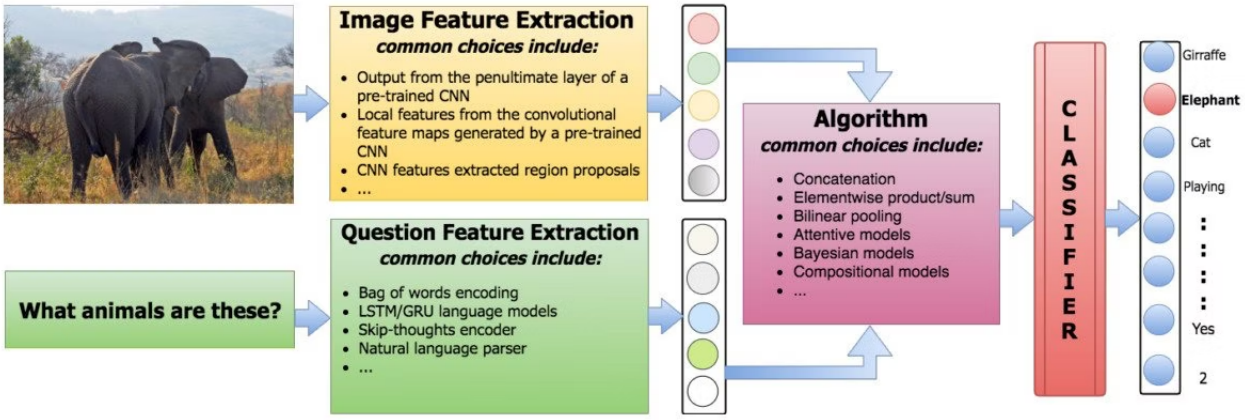


Figura 2: Forma general de las soluciones.

Tabla 1: Estructura de modelos implementados. Las características se unen mediante la **multiplicación** de ambos vectores de características.

Primera implementación		Segunda implementación	
BoW	VGG19	LSTM	VGG19
Bow_features x VGG19_features		LSTM_features x VGG19_features	
Fully connected 1024		Fully connected 1024	
Fully connected 1000		Fully connected 1000	
Fully connected 1000+Softmax		Fully connected 1000+Softmax	

- Primera implementación BoW+VGG19+MLP: Este modelo tiene un modelo de visión Bag of Words, esto lo que hace es codificar las preguntas en un vector de características, esto lo realiza contando el número de repeticiones que tiene cada palabra en el vocabulario previamente creado. Esto es algo relativamente simple en comparación con la segunda implementación que utiliza una red neuronal recurrente LSTM. Las características se unen mediante la **multiplicación** de ambos vectores de características.

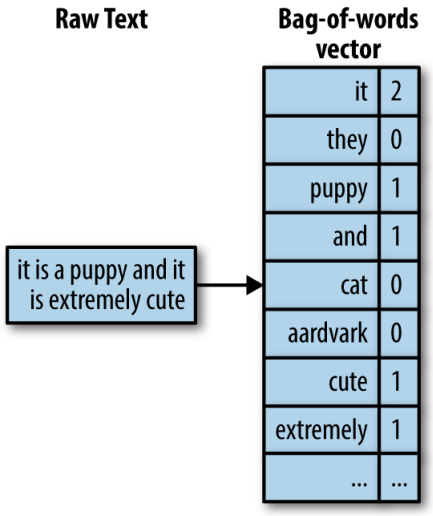


Figura 3: Bag of words encoding.

El modelo se resume de forma muy general en lo siguiente:

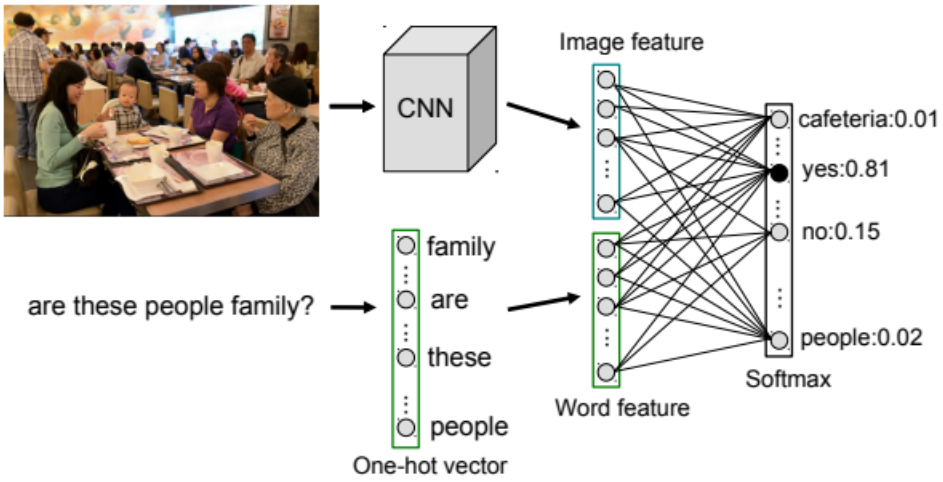


Figura 4: Modelo primera implementación.

- Segunda implementación LSTM+VGG19+MLP: utiliza una red neuronal tipo recurrente LSTM para el modelo de lenguaje y el modelo de visión VGG19, para luego terminar con una red MLP con 1000 neuronas de salida, cada una representa la probabilidad de cada respuesta, esto se muestra en la figura 5. Las características se unen mediante la **multiplicación** de ambos vectores de características.

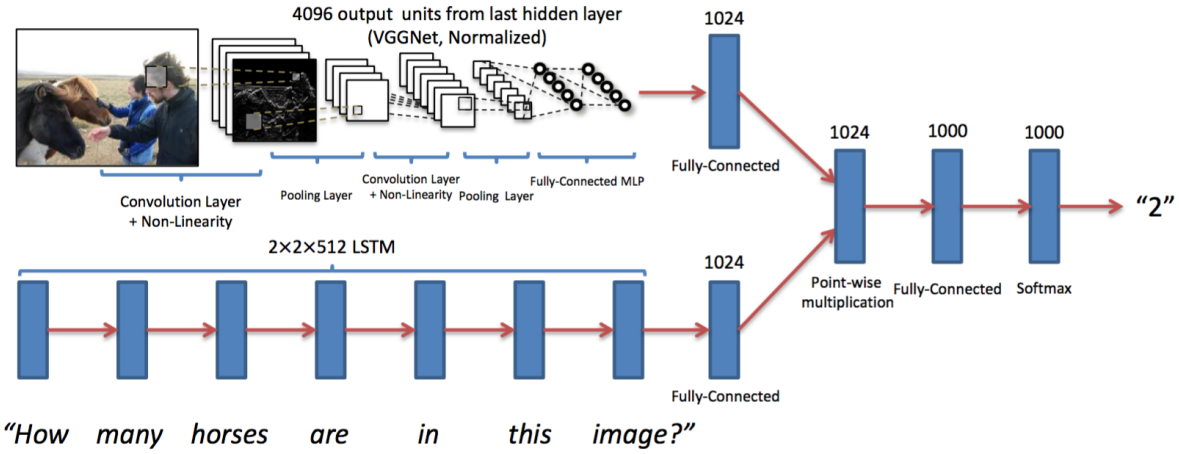


Figura 5: Modelo segunda implementación.

Se generó un vocabulario utilizando una función auxiliar del repositorio [2]. Estos vocabularios lo que hacen es guardar todas las palabras por separado y asignarles un número (o índice) con el cual se representen. Se generan 2, un vocabulario para las preguntas y otro para las respuestas.

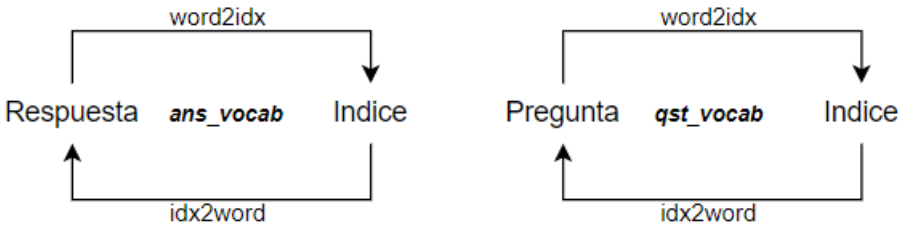


Figura 6: Uso de vocabulario para convertir palabras e índices y viceversa.

El número de respuestas son 24058 pero se realiza un vocabulario utilizando el top 1000 de respuestas más comunes. De esta manera por ejemplo la palabra *yes* se representa con el índice 1, cabe destacar también que existe una respuesta correspondiente a *<unk>*.

yes -> 1

A continuación se muestra un ejemplo de como se codifica una pregunta para llevarla a la red, esta se le realiza un padding a la derecha para que así todas las preguntas tengan el mismo largo y poder introducirlas a la red. Se eligió un largo de 30 pues la pregunta más larga tiene un tamaño de 21.

Is the grass green? -> [2432 4774 2047 2052 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Para ambos modelos, se utiliza la red VGG19 se utilizan los siguientes hiperparámetros:

- Se utiliza Cross Entropy Loss como función de error, esta ya incluye la función Softmax.
- Tasa de aprendizaje de 0.001, se espera realizar experimentos variando esta para ver que tan sensible es con respecto al cambiar este hiperparámetro.
- Optimizador Adam.
- Batch size de 64.

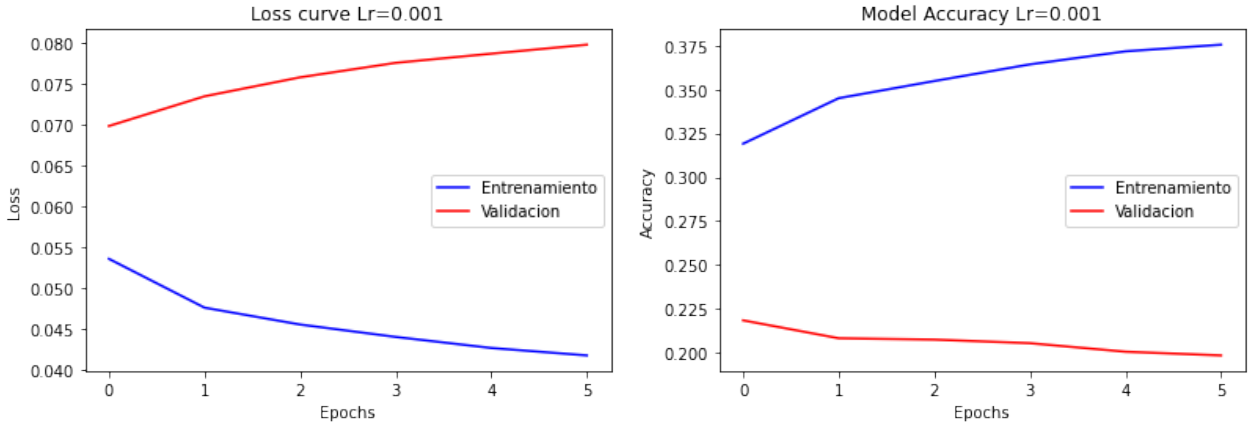
3. Resultados

A continuación se presentan los resultados de las dos implementaciones, además en la tabla 2 se puede observar un resumen de los resultados para los distintos experimentos. Cabe destacar que no se pudo realizar más experimentos con la primera implementación por temas de tiempo.

Tabla 2: Tabla resumen resultados.

	Min Train Loss	Best Train accuracy	Min Val Loss	Best Val accuracy
BoW+VGG19 Lr = 0.001	0.0418	0.3755	0.0698	0.2185
LSTM+VGG19 Lr = 0.001	0.0169	0.6374	0.0261	0.5225
LSTM+VGG19 Lr = 0.01	0.0100	0.1100	0.1840	0.2500
LSTM+VGG19 Lr = 0.1	2.1109	0.0930	2.0062	0.2439

3.1. Resultados primera implementación

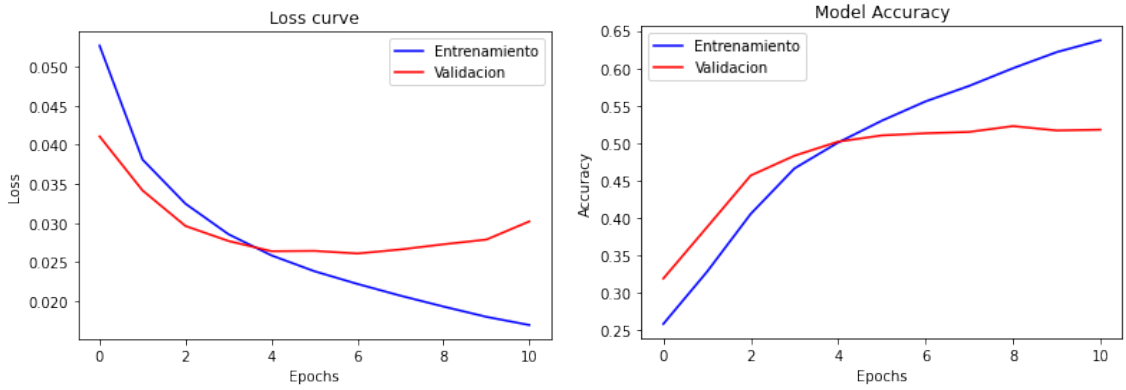


(a) Loss curve

(b) Accuracy curve

Figura 7: Utilizando una tasa de aprendizaje de 0.001.

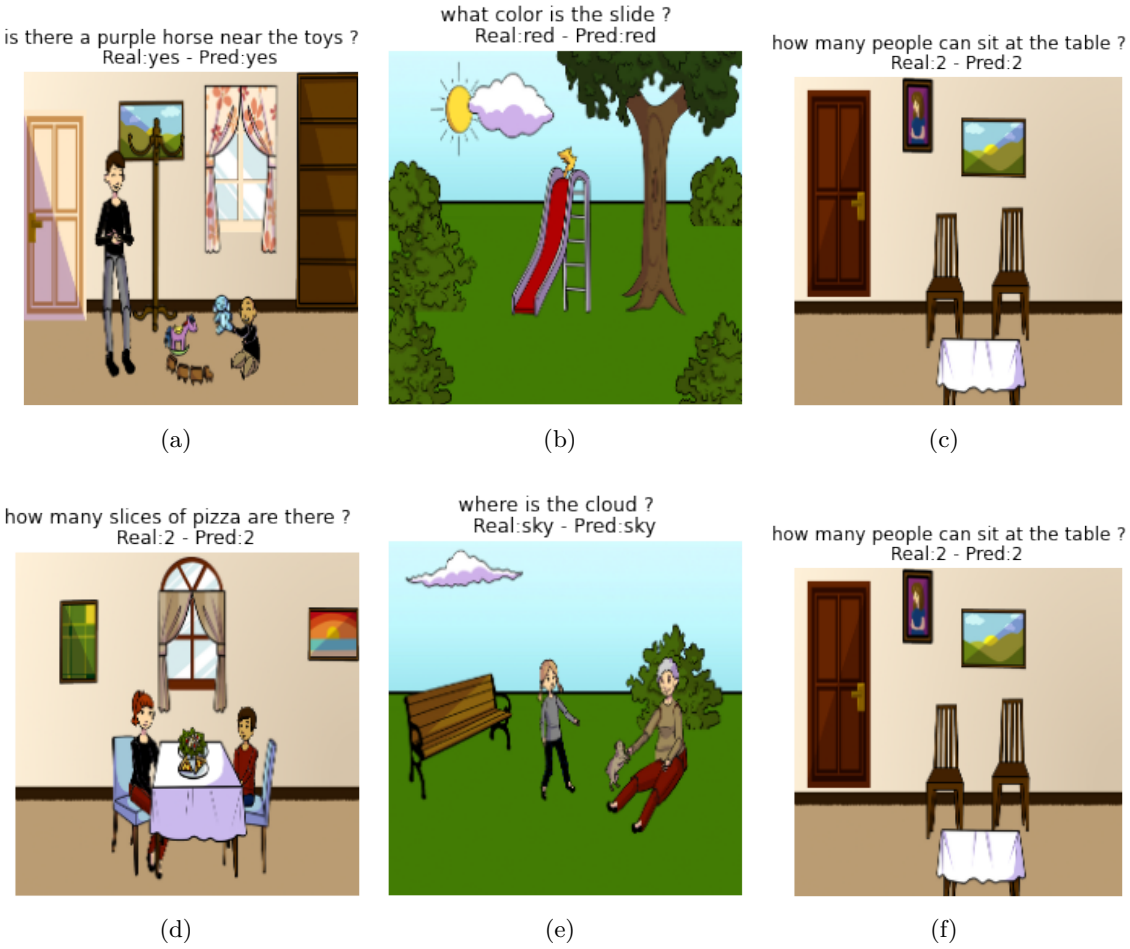
3.2. Resultados segunda implementación



(a) Loss curve (b) Accuracy curve

Figura 8: Utilizando una tasa de aprendizaje de 0.001 como recomendaba el paper.

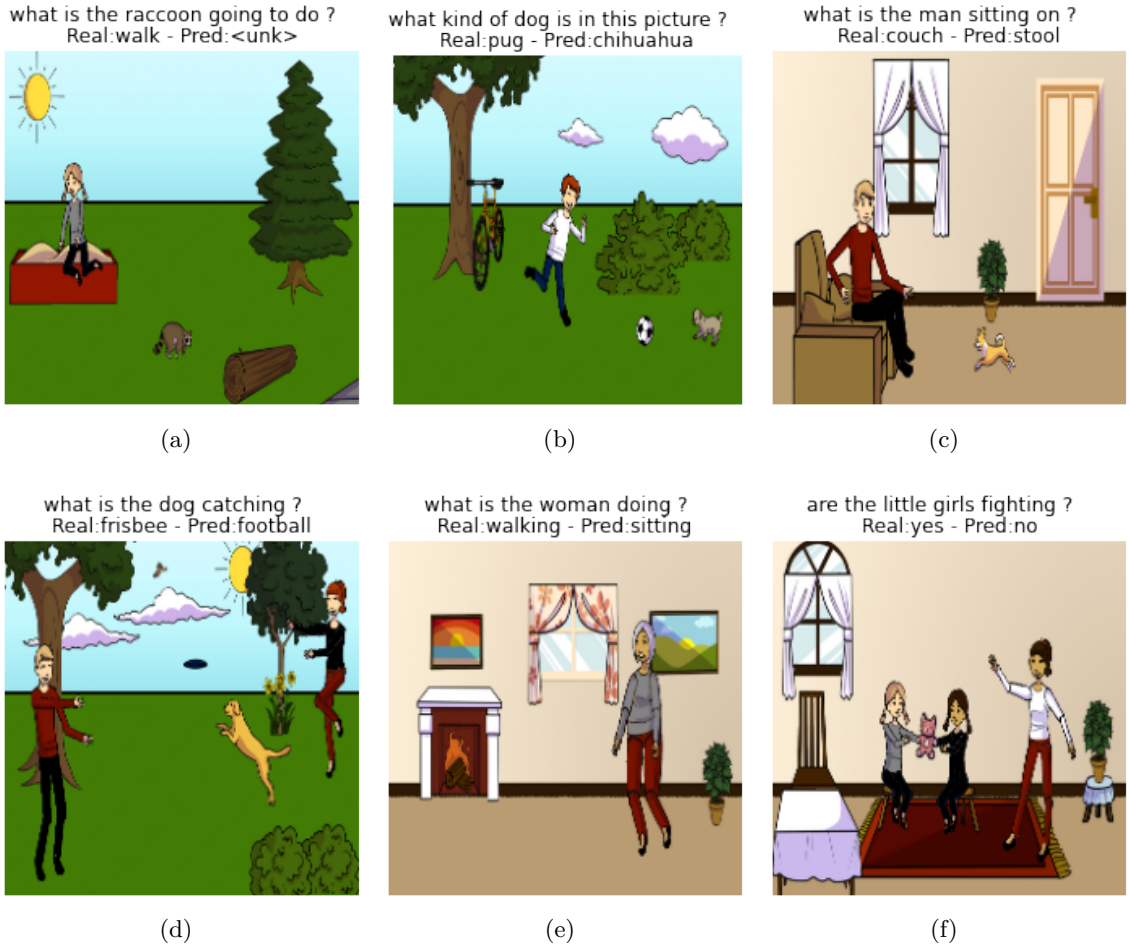
A continuación se visualizan algunas de las predicciones para su posterior análisis.



(a) (b) (c)

(d) (e) (f)

Figura 9: Predicciones correctas.



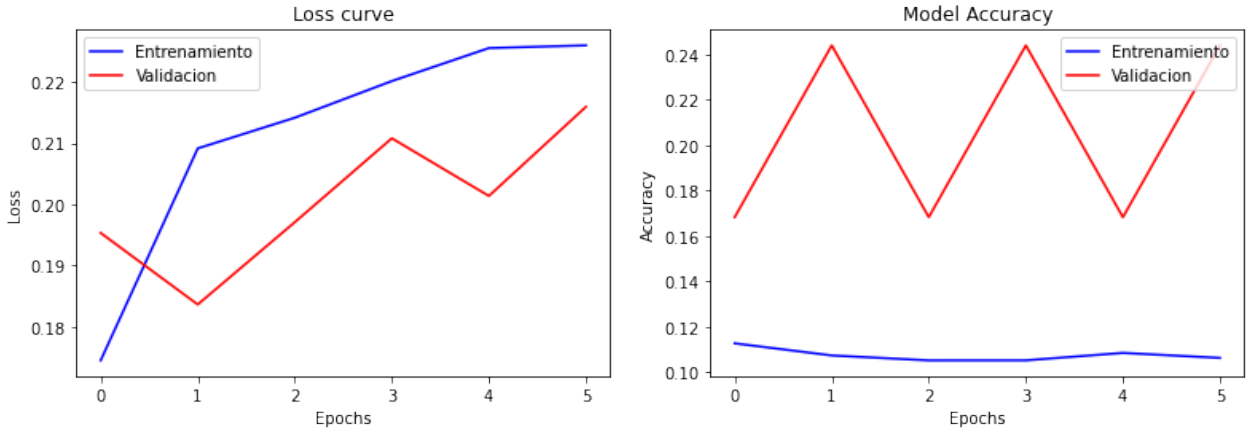
(a) (b) (c)

(d) (e) (f)

Figura 10: Predicciones incorrectas.

3.2.1. Resultados utilizando una tasa de aprendizaje de 0.01

Al utilizar esta tasa de aprendizaje, la red no aprende de forma correcta pues el error no disminuye al aumentar las épocas, por otro lado el accuracy en entrenamiento es prácticamente constante.



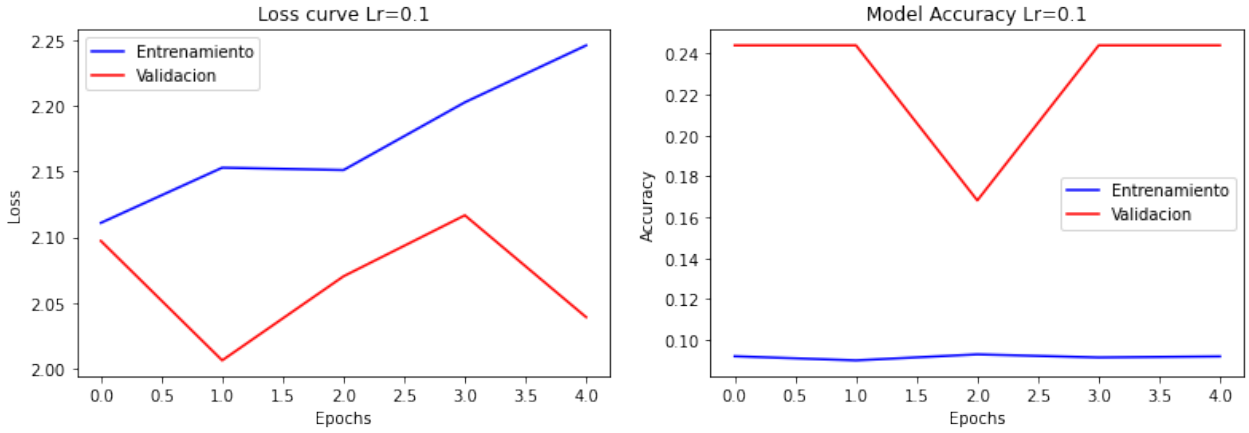
(a) Loss curve (b) Accuracy curve

Figura 11: Utilizando una tasa de aprendizaje de 0.01.

(a) (b) (c)

Figura 12: Extracto predicciones utilizando una tasa de aprendizaje de 0.01.

3.2.2. Resultados utilizando una tasa de aprendizaje de 0.1



(a) Loss curve (b) Accuracy curve

Figura 13: Utilizando una tasa de aprendizaje de 0.1.

(a) (b) (c)

Figura 14: Extracto predicciones utilizando una tasa de aprendizaje de 0.1.

4. Análisis

Los resultados de la primera implementación no son buenos, el accuracy en validación disminuye al aumentar las épocas y el error aumenta. Esto se puede deber a los hiperparámetros elegidos, como la tasa de aprendizaje, el batch size, etc. A pesar de esto, al compararlo con los otros modelos como se ve en la tabla 2, este modelo tiene un menor error en validación que el modelo de la segunda implementación con tasa de aprendizaje 0.1, pero no mejor accuracy en validación. Además, debido a que solo se utilizaron 1024 palabras de las preguntas para realizar el BoW, puede que no se haya generado un vector de características representativo. Los resultados de la segunda implementación fueron los mejores, utilizando una tasa de aprendizaje de 0.001. Este modelo tiene buenos resultados considerando la complejidad de la tarea a realizar, se logra un 52.25 % de accuracy en validación. A pesar de esto, el modelo es bastante sensible al cambiar la tasa de aprendizaje, como se puede observar en la tabla 2, al aumentar la tasa de aprendizaje el accuracy baja considerablemente llegando a un 24.39 % al utilizar una tasa de aprendizaje de 0.1 y teniendo un error creciente al aumentar las épocas, de hecho el accuracy en train se mantiene cercano a cero lo que se debe a un mal entrenamiento. De hecho, al visualizar los resultados en los dos modelos con tasas de aprendizajes de 0.01 y 0.1, los modelos VQA predecían en su gran mayoría **solo la respuesta "yes" sin importar el tipo de pregunta**. Esto se puede observar en las figuras 12 y 14, los cuales corresponden a un extracto de la visualización de los resultados.
Al analizar los resultados visuales del mejor modelo (LSTM+VGG19+MLP), se muestran 2 figuras, la figura 9 muestra las predicciones correctas de este modelo y la figura 10 muestra las predicciones incorrectas de este. En la primera se observa que el modelo puede responder de forma correcta pregunta de los 3 tipos (yes/no, number y other). Por otro lado, al ver algunas de las predicciones incorrectas, podemos observar que el modelo responde algunas de estas preguntas con sentido, por ejemplo en la figura 10.b la respuesta correcta era "pug" la cual corresponde a una raza de perro, pero la respuesta entregada por el modelo fue "chihuahua" la cual es otra raza de perro, si bien esta fue una respuesta incorrecta, estuvo bastante cerca de la solución. De la misma manera, las figuras 10.d y 10.e tienen respuestas con sentido a la pregunta. A pesar de esto, también hay respuestas completamente incorrectas o desconocidas.

5. Conclusión

Las aplicaciones de tener un modelo que resuelva la tarea de VQA son muchas, esta permite automatizar procesos, identificar situaciones abstractas, etc. esto le da importancia a la formación como ingenieros y se presenta como un gran desafió para el curso. Al desarrollar este proyecto, fue posible abarcar y comprender un problema complejo tal como lo es la tarea de VQA, analizar alternativas, implementar y entrenar dos soluciones (modelos), realizar un pequeño estudio de sensibilidad de los hiperparámetros y analizar los resultados de estos, lo cual cumple con el objetivo principal de este proyecto. Se logró resolver el problema en las dos iteraciones propuestas, logrando buenos resultados en la segunda iteración, lo cual era esperando pues la complejidad e inteligencia de la red recurrente LSTM es mucho mayor a la del vector de características provistos por Bag of Words, pero faltó realizar más pruebas para poder realizar un análisis que permita concluir de forma objetivo cual modelo es mejor, de todas maneras el modelo Bag of Words tiene resultados similares a la configuración de la segunda implementación con tasa de aprendizaje 0.1 siendo mucho más simple que esta, por lo que no descarto este modelo, con un vocabulario más grande y tuneando los hiperparámetros se hubieran obtenido mejores resultados. Dicho esto, fue posible comprender un problema mixto, el como mezclar características, sus ventajas y el como implementarlas usando la librería Pytorch. Se recomienda el uso del modulo de lenguaje LSTM y el modulo de visión VGG19 utilizando una tasa de aprendizaje de 0.001 en caso de querer realizar un proyecto similar, pues fue el que presentó mejores resultados respondiendo a las preguntas. La mayor dificultad de esta tarea fue el de hacer calzar las dimensiones de las redes neuronales y formar los vocabularios y diccionarios para el modelo del lenguaje. Por otro lado, la gran cantidad de elementos de entrenamiento provocó que los tiempos de entrenamiento fueran muy grandes, llegando a más de 3 horas en algunos casos, esto complicó el desarrollo del proyecto debido a que se estaba utilizando el entorno de Google Colab, por lo que recomiendo por la dimensión del dataset utilizar un enfoque local para que el entorno no se desconecte.

Referencias

[1] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, "VQA: Visual Question Answering," CoRR, vol. abs/1505.00468, 2015

[2] Repositorio Github Basic VQA, Taebong Moon. Disponible en: https://tryolabs.com/blog/2018/03/01/introduction-to-visual-question-answering

[3] Bolei Zhou, Yuandong Tian, Sainbayar Sukhbaatar, Arthur Szlam and Rob Fergus. Simple Baseline for Visual Question Answering, Massachusetts Institute of Technology.

6. Anexos

```
# -*- coding: utf-8 -*-
"""ProyectoVQAJZ.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1gINRLrOTuTgSNNbMfwAOsZfMWLksb4fE

# Desarrollo proyecto VQA por Joaquin Zepeda

A continuación se presenta la recopilación de los códigos realizados para la tarea, debido a que el entrenamiento podia demorar más de una hora, se realizaron distintos archivos para ejecutarse en paralelo de forma de ser más eficiente, todo esto se recopila en este archivo.

# Desacarga de los datos
"""

# Download datasete
!wget -q -N --show-progress https://s3.amazonaws.com/cvmlp/vqa/abstract_v002/vqa/Questions_Train_abstract_v002.zip
!wget -q -N --show-progress https://s3.amazonaws.com/cvmlp/vqa/abstract_v002/vqa/Questions_Val_abstract_v002.zip
!wget -q -N --show-progress https://s3.amazonaws.com/cvmlp/vqa/abstract_v002/scene_img/scene_img_abstract_v002_train2015.zip
!wget -q -N --show-progress https://s3.amazonaws.com/cvmlp/vqa/abstract_v002/scene_img/scene_img_abstract_v002_val2015.zip
!wget -q -N --show-progress https://s3.amazonaws.com/cvmlp/vqa/abstract_v002/vqa/Annotations_Train_abstract_v002.zip
!wget -q -N --show-progress https://s3.amazonaws.com/cvmlp/vqa/abstract_v002/vqa/Annotations_Val_abstract_v002.zip

!mkdir -p {Questions/train Questions/test Questions/val Images/train Images/test Images/val Annotations/train Annotations/val}

!unzip -q -n Questions_Train_abstract_v002.zip -d Questions
!unzip -q -n Questions_Val_abstract_v002.zip -d Questions
!unzip -q -n scene_img_abstract_v002_train2015.zip -d Images/train
!unzip -q -n scene_img_abstract_v002_val2015.zip -d Images/val
!unzip -q -n Annotations_Train_abstract_v002.zip -d Annotations
!unzip -q -n Annotations_Val_abstract_v002.zip -d Annotations

# Remove unnecessary zip files.

!rm Questions_Train_abstract_v002.zip
!rm Questions_Val_abstract_v002.zip

!rm scene_img_abstract_v002_train2015.zip
!rm scene_img_abstract_v002_val2015.zip
```

```
40
41 !rm Annotations_Train_abstract_v002.zip
42 !rm Annotations_Val_abstract_v002.zip
43 #
    ↪ ##################################################################################
    ↪
44
45 """### Inicializamos las semillas"""
46
47 import torch
48 import numpy as np
49 import random
50
51 SEED = 1234
52
53
54 torch.backends.cudnn.deterministic = True
55 torch.backends.cudnn.benchmark = False
56 torch.manual_seed(SEED)
57 torch.cuda.manual_seed_all(SEED)
58 np.random.seed(SEED)
59 random.seed(SEED)
60
61 """## Flag BOW
62
63 Si BOW es True, se ejecuta todo lo correspondiente a la primera implementación, la cual corresponde
    ↪  a un modulo de lenguaje BOW + un modulo de vision VGG19 + MLP. En caso contrario se
    ↪ ejecuta la segunda implementación la cual cambia el modelo de lenguaje por una LSTM y
    ↪ mantiene lo demás.
64 """
65
66 BOW = True
67
68 """# Codificando las palabras
69
70 Vamos a representar cada palabra con un único número entero que representará su índice.
71 Para eso construiremos un diccionario que mapee palabras a números. La verdad es que son muy
    ↪ similares los vocabularios, hay muy pequeños cambios.
72
73 ## Vocabulario para el BoW
74
75 Se separan los vocabularios para las preguntas y para las respuestas.
76 """
77
78 import re
79 from string import punctuation
80 import os
81 import numpy as np
82 from collections import defaultdict
83 import json
84
```

```
85 def make_bow_qst_vocabulary(input_dir,n_words):
86     SENTENCE_SPLIT_REGEX = re.compile(r'(\W+)')
87     questions_dict = dict()
88     preguntas = []
89     datasets = os.listdir(input_dir)
90     for dataset in ['MultipleChoice_abstract_v002_train2015_questions.json','
        ↪ MultipleChoice_abstract_v002_val2015_questions.json']:
91         with open(input_dir+'/'+dataset) as f:
92             questions = json.load(f)['questions']
93         for question in questions:
94             all_text = question['question'].lower()
95             all_text = ''.join([c for c in all_text if c not in punctuation])
96             words = SENTENCE_SPLIT_REGEX.split(all_text)
97             preguntas.append(all_text)
98             for word in words:
99                 word=word.strip()
100                if word not in questions_dict:
101                    questions_dict[word] = 1
102                if re.search(r"[^\w\s]", word):
103                    continue
104                questions_dict[word] += 1
105    questions_dict = sorted( questions_dict, key= questions_dict.get, reverse=True)
106    top_answers = ['<unk>'] +  questions_dict[:n_words-1] # '-1' is due to '<unk>'
107
108    return top_answers,  questions_dict[:n_words], preguntas
109
110 def make_bow_ans_vocabulary(input_dir,n_words):
111    SENTENCE_SPLIT_REGEX = re.compile(r'(\W+)')
112    ans_dict = dict()
113    datasets = os.listdir(input_dir)
114    answers = []
115    for dataset in ['abstract_v002_train2015_annotations.json','abstract_v002_val2015_annotations.
        ↪ json']:
116        with open(input_dir+'/'+dataset) as f:
117            annotations = json.load(f)['annotations']
118        for annotation in annotations:
119            for answer in annotation['answers']:
120                word = answer['answer'].strip()
121                if re.search(r"[^\w\s]", word):
122                    continue
123                if word not in ans_dict:
124                    ans_dict[word] = 1
125                ans_dict[word] += 1
126
127    ans_dict = sorted( ans_dict, key= ans_dict.get, reverse=True)
128    top_answers = ans_dict[:n_words-1] # '-1' is due to '<unk>'
129
130    return top_answers,  ans_dict[:n_words]
131
132 """## Vocabulario para la LSTM
133
```

```
Se separan nuevamente los vocabularios para las preguntas y respuestas.
"""

import re
from string import punctuation
import os
import numpy as np
from collections import defaultdict

def make_vocab_questions(input_dir):
    """Genera un vocabulario y lo guarda en una lista y diccionario"""
    vocab_set = set()
    SENTENCE_SPLIT_REGEX = re.compile(r'(\W+)')
    question_length = []
    datasets = os.listdir(input_dir)
    preguntas = []
    print('Se eliminan los signos de puntuación: ',punctuation)
    for dataset in ['MultipleChoice_abstract_v002_train2015_questions.json','
     ↪ MultipleChoice_abstract_v002_val2015_questions.json']:
        a = 0
        with open(input_dir+'/'+dataset) as f:
            questions = json.load(f)['questions']
            if a==0:
                print(questions[0]['question'])
            a+=1
        set_question_length = [None]*len(questions)
        for iquestion, question in enumerate(questions):
            all_text = question['question'].lower()
            all_text = ''.join([c for c in all_text if c not in punctuation])

            words = SENTENCE_SPLIT_REGEX.split(all_text)
            preguntas.append(all_text)
            words = [w.strip() for w in words if len(w.strip()) > 0]
            vocab_set.update(words)
            set_question_length[iquestion] = len(words)
        question_length += set_question_length

    vocab_list = list(vocab_set)
    vocab_list.sort()
    # se agregan 2 preguntas especiales para casos desconocidos
    vocab_list.insert(0, '<pad>')
    vocab_list.insert(1, '<unk>')

    print('Estamos haciendo un vocabulario con todas las palabras de todas las preguntas')
    print('El número total de palabras en las preguntas son: %d' % len(vocab_set))
    print('El largo máximo de una pregunta es: %d' % np.max(question_length))

    return vocab_list, preguntas


def make_vocab_answers(input_dir, n_answers):
```

```
    """Vocabulario con el top n_answers, retorna la lista con el top y las respuestas."""
    answers = defaultdict(lambda: 0)
    datasets = os.listdir(input_dir)
    for dataset in ['abstract_v002_train2015_annotations.json','abstract_v002_val2015_annotations.
     ↪ json']:
        with open(input_dir+'/'+dataset) as f:
            annotations = json.load(f)['annotations']
        for annotation in annotations:
            for answer in annotation['answers']:
                word = answer['answer']
                if re.search(r"[^\w\s]", word):
                    continue
                answers[word] += 1

    answers = sorted(answers, key=answers.get, reverse=True)
    assert('<unk>' not in answers)
    top_answers = ['<unk>'] + answers[:n_answers-1] # '-1' is due to '<unk>'

    print('Vocabulario para las respuestas')
    print('Número total de respuestas: %d' % len(answers))
    print('Guarda el %d de respuestas' % n_answers)
    return top_answers, answers

def lookUpBOW(top_words, bow_dict,all_words):
    """Función auxiliar para generar una Look up table"""
    bow_qns = np.zeros((len(all_words), 1024))

    for i in range(len(all_words)):
        w = all_words[i]
        s =  w.split()
        for word in s:
            if word not in top_words:
                idx = 0
            else:
                idx = top_words.index(word)
            # se suma uno por las palabras repetidas
            bow_qns[i][idx] += 1
    return bow_qns

if bow:
    # Se generan los vocabularios y las tablas para ver el funcionameinto.
    top_questionsBOW,  qst_bow_dict, preguntas = make_bow_qst_vocabulary("Questions",1024)
    top_answersBOW,  ans_bow_dict = make_bow_ans_vocabulary("Annotations",1000)

    qst_bow_table = lookUpBOW(top_questionsBOW,  qst_bow_dict, preguntas)
    ans_bow_table = lookUpBOW(top_answersBOW,  ans_bow_dict, top_answersBOW)

"""### Pruebas"""

print("Question",preguntas[0])
print("Question Bow",qst_bow_table[0])
```

```
print("Answer",top_answersBOW[0])
print("Answer Bow",ans_bow_table[0])
print("Indice answer",np.argmax(ans_bow_table[0]))

"""## Funciones auxiliares para LSTM

Se genera un diccionario para acceder a los indices de las palabras y transformar las palabras e indices de forma bidireccional. Cabe destacar que se eliminan los caracteres especiales, solo se consideran las palabras. Algunas de estas funciones se extrajeron del repositorio basic_vqa que se puede encontrar en las referencias del informe.
"""

import re

SENTENCE_SPLIT_REGEX = re.compile(r'(\W+)')

def tokenize(sentence):
    tokens = SENTENCE_SPLIT_REGEX.split(sentence.lower())
    tokens = [t.strip() for t in tokens if len(t.strip()) > 0]
    return tokens
def load_str_list(fname):
    with open(fname) as f:
        lines = f.readlines()
    lines = [l.strip() for l in lines]
    return lines


class VocabDict:

    def ___init___(self, word_list):
        self.word_list = word_list
        self.word2idx_dict = {w:n_w for n_w, w in enumerate(self.word_list)}
        self.vocab_size = len(self.word_list)
        self.unk2idx = self.word2idx_dict['<unk>'] if '<unk>' in self.word2idx_dict else None

    def idx2word(self, n_w):

        return self.word_list[n_w]

    def word2idx(self, w):
        if w in self.word2idx_dict:
            return self.word2idx_dict[w]
        elif self.unk2idx is not None:
            return self.unk2idx
        else:
            raise ValueError('word %s not in dictionary (while dictionary does not contain <unk>)' % w )

    def tokenize_and_index(self, sentence):
        inds = [self.word2idx(w) for w in tokenize(sentence)]
        return
```

```

"""# Dataset personalizados

## Dataset BoW
"""

import json
import torch
import os
from torch.utils.data import Dataset
from skimage import io
import numpy as np
from torchvision import transforms
from PIL import Image

class VQA_BOW(Dataset):
  def __init__(self, questions_path: str, annotations_path: str, images_folder_path: str, transform ,
    ↪ max_qst_length=30, max_num_ans=10):
    super(VQA_BOW, self).__init__()
    self.qn_path = questions_path
    self.an_path = annotations_path
    self.img_path = images_folder_path

    self.transform = transform

    top_questionsBOW,  qst_bow_dict, preguntas = make_bow_qst_vocabulary("Questions",1024)
    top_answersBOW,  ans_bow_dict = make_bow_ans_vocabulary("Annotations",1000)

    self.top_answers = top_answersBOW

    qst_bow_table = lookUpBOW(top_questionsBOW,  qst_bow_dict, preguntas)
    ans_bow_table = lookUpBOW(top_answersBOW,  ans_bow_dict , top_answersBOW)

    self.qst_bow_table = qst_bow_table
    self.ans_bow_table = ans_bow_table

    self.max_qst_length = max_qst_length
    self.max_num_ans = max_num_ans

    self.ann_dataset = json.load(open(self.an_path, 'r'))['annotations']
    qn_json = json.load(open(self.qn_path, 'r'))
    self.qn_dataset = qn_json['questions']
    self.img_prefix = f"{qn_json['data_type']}_{qn_json['data_subtype']}"

  def __len__(self):
    return len(self.qn_dataset)

  def __getitem__(self, item: int):

    qst_bow_table = self.qst_bow_table
    ans_bow_table = self.ans_bow_table
```

```
    max_qst_length = self.max_qst_length
    max_num_ans = self.max_num_ans


    question = self.qn_dataset[item]
    question_str = question['question']

    qst_vec = qst_bow_table[item]

    choices = question['multiple_choices']

    answer = self.ann_dataset[item]['multiple_choice_answer']

    ans2idx = 0
    if answer in self.top_answers:
        #indice de la respuesta
        ans2idx = self.top_answers.index(answer)

    image = Image.open(os.path.join(self.img_path, f"{self.img_prefix}_{question['image_id']:012d}.
      ↪ png")).convert('RGB')
    if self.transform:
      image = self.transform(image)

    return image, question_str, choices, answer, qst_vec, ans2idx

"""## Dataset LSTM"""

import json
import torch
import os
from torch.utils.data import Dataset
from skimage import io
import numpy as np
from torchvision import transforms
from PIL import Image

class VQA_LSTM(Dataset):
  def __init__(self, questions_path: str, annotations_path: str, images_folder_path: str, transform ,
      ↪ max_qst_length=30, max_num_ans=10):
    super(VQA_LSTM, self).__init__()
    self.qn_path = questions_path
    self.an_path = annotations_path
    self.img_path = images_folder_path

    self.transform = transform

    #retorna una lista con un vocabulario de preguntas
    vocab_list, _ = make_vocab_questions('Questions')
    #retorna una lista con el top 1000 respuestas
    top_answers, _ = make_vocab_answers('Annotations',1000)

```

```
380     #Diccionario de preguntas
381     self.qst_vocab = VocabDict(vocab_list)
382     #Diccionario de respuestas
383     self.ans_vocab = VocabDict(top_answers)
384 
385     self.max_qst_length = max_qst_length
386     self.max_num_ans = max_num_ans
387 
388     self.ann_dataset = json.load(open(self.an_path, 'r'))['annotations']
389     qn_json = json.load(open(self.qn_path, 'r'))
390     self.qn_dataset = qn_json['questions']
391     self.img_prefix = f"{qn_json['data_type']}_{qn_json['data_subtype']}"
392 
393   def __len__(self):
394     return len(self.qn_dataset)
395 
396   def __getitem__(self, item: int):
397 
398     qst_vocab = self.qst_vocab
399     ans_vocab = self.ans_vocab
400     max_qst_length = self.max_qst_length
401     max_num_ans = self.max_num_ans
402 
403 
404     question = self.qn_dataset[item]
405     question_str = question['question']
406 
407     qst2idc = np.array([qst_vocab.word2idx('<pad>')] * max_qst_length)  # padded with '<pad>' in
        ↪ 'ans_vocab'
408     qst2idc[:len(tokenize(question['question']))] = [qst_vocab.word2idx(w) for w in tokenize(question['
        ↪ question'])]
409 
410     choices = question['multiple_choices']
411 
412     answer = self.ann_dataset[item]['multiple_choice_answer']
413     ans2idx = ans_vocab.word2idx(answer)
414 
415     image = Image.open(os.path.join(self.img_path, f"{self.img_prefix}_{question['image_id']:012d}.
        ↪ png")).convert('RGB')
416     if self.transform:
417       image = self.transform(image)
418 
419     return image, question_str, choices, answer, qst2idc, ans2idx
420 
421 # trabajo con listas
422 L = ['2', 'yes', 'man on table', 'out on porch', 'cotton', '1', '3', 'blue', 'white', 'brown', 'no', 'yard', '
        ↪ sandals', 'on deck', '4', 'pot', 'red', 'on grass']
423 L.index('on grass'),L[17]
424 
425 import matplotlib.pyplot as plt
426 import numpy as np
```

```
427
428 if not bow:
429     questions_path='Questions'
430     annotation_path='Annotations'
431
432     #generamos los archivos que despues se utilizan para generar el diccionario de preguntas
433     qvocab_list, preguntas = make_vocab_questions(questions_path)
434     #generamos los archivos que despues se utilizan para generar el diccionario de respuestas
435     top_answers, answers = make_vocab_answers(annotation_path,1000)
436
437     # Uso
438     dataset = VQA_LSTM(questions_path='Questions/
        ↪ MultipleChoice_abstract_v002_val2015_questions.json',
439             annotations_path='Annotations/abstract_v002_val2015_annotations.json',
440             images_folder_path='Images/val/', transform=transforms.Compose([
441                                         transforms. ToTensor(),
442                                         transforms.Resize((224,224))
443                                     ]))
444
445     # Tomar muestra aleatoria
446     image, question, choices, answer, qst2idc, anslabel = dataset[50]
447
448     print("Dimension de la imagen:",image.shape)
449     print(question,qst2idc)
450     print("Alternativas:",choices)
451     print("Indice de la solución",anslabel)
452
453
454     # Visualizar
455     img = image.numpy().transpose((1, 2, 0))
456     plt.imshow(img)
457     plt.suptitle(f"{question}")
458     plt.title(f"{answer}")
459     plt.axis('off')
460     plt.show()
461
462 import matplotlib.pyplot as plt
463 import numpy as np
464
465 if bow:
466     # Uso
467     dataset = VQA_BOW(questions_path='Questions/
        ↪ MultipleChoice_abstract_v002_val2015_questions.json',
468             annotations_path='Annotations/abstract_v002_val2015_annotations.json',
469             images_folder_path='Images/val/', transform=transforms.Compose([
470                                         transforms. ToTensor(),
471                                         transforms.Resize((224,224))
472                                     ]))
473
474     # Tomar muestra aleatoria
475     image, question, choices, answer, qst2idc, anslabel = dataset[50]
```

```
476
477     print("Dimension de la imagen:",image.shape)
478     print(question,qst2idc)
479     print("Alternativas:",choices)
480     print("Indice de la solución",anslabel)
481     #print("Solución BOW",top_answersBOW[anslabel])
482
483     # Visualizar
484     img = image.numpy().transpose((1, 2, 0))
485     plt.imshow(img)
486     plt.suptitle(f"{question}")
487     plt.title(f"{answer}")
488     plt.axis('off')
489     plt.show()
490
491 #El indice de yes es el 1, entonces de las mil salidas la neurona con mayor probailidad deberia ser esa
492 top_answers[0:5], len(top_answers)
493
494 """# Generando los dataset
495
496 Se generan los dataset para cada modelo.
497 """
498
499 BATCH_SIZE = 64
500 if bow == True:
501     trainDatasetBOW = VQA_BOW(questions_path='Questions/
      ↪ MultipleChoice_abstract_v002_train2015_questions.json',
502             annotations_path='Annotations/abstract_v002_train2015_annotations.json',
503             images_folder_path='Images/train/', transform=transforms.Compose([
504                                         transforms. ToTensor(),
505                                         transforms.Resize((224,224))
506                                     ]))
507     valDatasetBOW   = VQA_BOW(questions_path='Questions/
      ↪ MultipleChoice_abstract_v002_val2015_questions.json',
508             annotations_path='Annotations/abstract_v002_val2015_annotations.json',
509             images_folder_path='Images/val/', transform=transforms.Compose([
510                                         transforms. ToTensor(),
511                                         transforms.Resize((224,224))
512                                     ]))
513     train_loader = torch.utils.data.DataLoader(trainDataset, batch_size=BATCH_SIZE, shuffle=
      ↪ True, num_workers=2, pin_memory=True)
514     val_loader = torch.utils.data.DataLoader(valDataset, batch_size=BATCH_SIZE, shuffle=True,
      ↪ num_workers=2, pin_memory=True)
515
516 else:
517     trainDataset = VQA_LSTM(questions_path='Questions/
      ↪ MultipleChoice_abstract_v002_train2015_questions.json',
518             annotations_path='Annotations/abstract_v002_train2015_annotations.json',
519             images_folder_path='Images/train/', transform=transforms.Compose([
520                                         transforms. ToTensor(),
521                                         transforms.Resize((224,224))
```

```
                                        ]))
    valDataset   = VQA_LSTM(questions_path='Questions/
     ↪ MultipleChoice_abstract_v002_val2015_questions.json',
            annotations_path='Annotations/abstract_v002_val2015_annotations.json',
            images_folder_path='Images/val/',  transform=transforms.Compose([
                                            transforms. ToTensor(),
                                            transforms.Resize((224,224))
                                        ]))
    train_loaderBOW = torch.utils.data.DataLoader(trainDatasetBOW, batch_size=BATCH_SIZE,
     ↪ shuffle=True, num_workers=2)
    val_loaderBOW = torch.utils.data.DataLoader(valDatasetBOW, batch_size=BATCH_SIZE,
     ↪ shuffle=True, num_workers=2)

"""# Generando modelos

## Primer modelo
"""

import torch
import torch.nn as nn
import torchvision.models as models


class ImgEncoderBOW(nn.Module):

    def ___init___(self, embed_size):
        """Red VGG19 preentrenada con los pesos de imagenet
        """
        super(ImgEncoderBOW, self).___init___()
        model = models.vgg19(pretrained=True)
        in_features = model.classifier[-1].in_features  # input size of feature vector
        model.classifier = nn.Sequential(*list(model.classifier.children())[:-1])    # remove last fc layer

        self.model = model                              # loaded model without last fc layer
        self.fc = nn.Linear(in_features, embed_size)    # feature vector of image

    def forward(self, image):
        """Extract feature vector from image vector.
        """
        #image = image.permute(0, 3, 1, 2)
        with torch.no_grad():
            img_feature = self.model(image)                  # [batch_size, vgg16(19)_fc=4096]
        img_feature = self.fc(img_feature)                   # [batch_size, embed_size]

        l2_norm = img_feature.norm(p=2, dim=1, keepdim=True).detach()
        img_feature = img_feature.div(l2_norm)               # l2-normalized feature vector

        return img_feature

class VqaModelBOW(nn.Module):

```

```
570     def __init__(self, embed_size, qst_vocab_size, word_embed_size, num_layers, hidden_size):
571
572         super(VqaModelBOW, self).__init__()
573         self.img_encoder = ImgEncoderBOW(embed_size)
574         self.tanh = nn.Tanh()
575         self.dropout = nn.Dropout(0.5)
576         self.fc1 = nn.Linear(embed_size, 1000)
577         self.fc2 = nn.Linear(1000, 1000)
578
579     def forward(self, img, qst_feature):
580         #se le pasa de forma directa el BoW encoding
581         img_feature = self.img_encoder(img)
582         combined_feature = torch.mul(img_feature, qst_feature)
583         combined_feature = self.tanh(combined_feature)
584         combined_feature = self.dropout(combined_feature)
585         combined_feature = self.fc1(combined_feature)
586         combined_feature = self.tanh(combined_feature)
587         combined_feature = self.dropout(combined_feature)
588         combined_feature = self.fc2(combined_feature)
589
590         return combined_feature
591
592 """## Segundo modelo """
593
594 import torch
595 import torch.nn as nn
596 import torchvision.models as models
597
598
599 class ImgEncoder(nn.Module):
600
601     def __init__(self, embed_size):
602         """Feature vector de la VGG19 preentrenada con los pesos de imagenet
603         """
604         super(ImgEncoder, self).__init__()
605         model = models.vgg19(pretrained=True)
606         in_features = model.classifier[-1].in_features  # input size of feature vector
607         model.classifier = nn.Sequential(*list(model.classifier.children())[:-1])    # remove last fc layer
608
609         self.model = model                              # loaded model without last fc layer
610         self.fc = nn.Linear(in_features, embed_size)    # feature vector of image 4096->1000
611
612     def forward(self, image):
613         """Se extraen las caracteristicas
614         """
615         #image = image.permute(0, 3, 1, 2)
616         with torch.no_grad():
617             img_feature = self.model(image)
618         img_feature = self.fc(img_feature)
619
620         l2_norm = img_feature.norm(p=2, dim=1, keepdim=True).detach()
```

```
621         img_feature = img_feature.div(l2_norm)          # se normaliza
622
623         return img_feature
624
625
626 class QstEncoder(nn.Module):
627     """Red neuronal recurrente LSTM"""
628
629     def ___init___(self, qst_vocab_size, word_embed_size, embed_size, num_layers, hidden_size):
630
631         super(QstEncoder, self).___init___()
632         self.word2vec = nn.Embedding(qst_vocab_size, word_embed_size)
633         self.tanh = nn.Tanh()
634         self.lstm = nn.LSTM(word_embed_size, hidden_size, num_layers)
635         self.fc = nn.Linear(2*num_layers*hidden_size, embed_size)
636         #LSTM con salida de tamaño 1024
637
638     def forward(self, question):
639
640         qst_vec = self.word2vec(question)
641         qst_vec = self.tanh(qst_vec)
642         qst_vec = qst_vec.transpose(0, 1)
643         _, (hidden, cell) = self.lstm(qst_vec)
644         qst_feature = torch.cat((hidden, cell), 2)
645         qst_feature = qst_feature.transpose(0, 1)
646         qst_feature = qst_feature.reshape(qst_feature.size()[0], -1)
647         qst_feature = self.tanh(qst_feature)
648         qst_feature = self.fc(qst_feature)
649
650         return qst_feature
651
652
653 class VqaModel(nn.Module):
654
655     def ___init___(self, embed_size, qst_vocab_size, word_embed_size, num_layers, hidden_size):
656
657         super(VqaModel, self).___init___()
658         self.img_encoder = ImgEncoder(embed_size)
659         self.qst_encoder = QstEncoder(qst_vocab_size, word_embed_size, embed_size, num_layers,
      ↪ hidden_size)
660         self.tanh = nn.Tanh()
661         self.dropout = nn.Dropout(0.5)
662         self.fc1 = nn.Linear(embed_size, 1000)
663         self.fc2 = nn.Linear(1000, 1000)
664
665     def forward(self, img, qst):
666         img_feature = self.img_encoder(img)
667         qst_feature = self.qst_encoder(qst)
668         combined_feature = torch.mul(img_feature, qst_feature)
669         combined_feature = self.tanh(combined_feature)
670         combined_feature = self.dropout(combined_feature)
```

```
671         combined_feature = self.fc1(combined_feature)
672         combined_feature = self.tanh(combined_feature)
673         combined_feature = self.dropout(combined_feature)
674         combined_feature = self.fc2(combined_feature)      #salida de 1000
675 
676         return combined_feature
677 
678 if not bow:
679     qst_vocab_size = train_loader.dataset.qst_vocab.vocab_size
680     ans_vocab_size = 1000
681 embed_size = 1024
682 word_embed_size = 300
683 num_layers = 2
684 hidden_size = 512
685 learning_rate = 0.001
686 
687 print("Tamaño de los vocabularios, question vocab size: ",qst_vocab_size,"answer vocab size",
        ↪ ans_vocab_size)
688 
689 print(len(train_loader.dataset),len(val_loader.dataset))
690 
691 print("Tamaño de los vocabularios, question vocab size: ",qst_vocab_size,"answer vocab size",
        ↪ ans_vocab_size)
692 
693 print(len(train_loader.dataset),len(val_loader.dataset))
694 
695 """# Train function
696 
697 ## Device
698 
699 Se utiliza el entorno GPU.
700 """
701 
702 device = ('cuda' if torch.cuda.is_available() else 'cpu')
703 device
704 
705 """## Función Train para el primer modelo
706 
707 En base a los modelos de la tarea 5 y del curso Deep learning.
708 """
709 
710 import time
711 import copy
712 
713 def trainBOW(net, optimizer, num_epocas):
714     inicio = time.time()
715 
716     #copiamos el modelo utilizando la libreria copy
717     best_model_wts = copy.deepcopy(net.state_dict())
718     train_losses, train_counter, train_accuracy, val_losses,val_accuracy,val_counter  = [],[],[],[],[],[]
719 
```

```
720   best_acc = 0.0
721   best_loss = 2e32
722   for epoch in range(num_epocas):
723       print('Epoch {}/{}'.format(epoch, num_epocas-1))
724       print('-' * 10)
725
726       net.train() #Modo entrenamiento
727
728       running_loss = 0.0
729       running_corrects = 0.0
730       batch_size = 64
731       batch_step_size = len(train_loaderBOW.dataset) / batch_size
732       a=0
733       for batch_idx, batch_sample in enumerate(train_loaderBOW):
734           image = batch_sample[0].to(device).float()
735           question = batch_sample[4].to(device).float()
736           label = batch_sample[5].to(device)
737           optimizer.zero_grad()
738           if a==0:
739             print(image.shape,question.shape,label.shape)
740             print("question",question[0])
741             print("label",label[0])
742             a+=1
743
744           output = net(image,question) #salidas de la red
745           pred_indices = torch.argmax(output,1)  # [batch_size]
746           loss = criterion(output, label)
747
748           loss.backward()
749           optimizer.step()
750
751           running_loss += loss.item()
752           running_corrects += torch.sum(pred_indices == label.data)
753           if batch_idx %100 == 0:
754               print('Batch idx: ',batch_idx,' Batch Step: ' ,int(batch_step_size)) #para ver el avance
755
756       epoch_loss = running_loss /len(train_loaderBOW.dataset) #promedio de error
757       epoch_acc = running_corrects / len(train_loaderBOW.dataset) #promedio de accuracy
758       train_losses.append(epoch_loss)
759       train_counter.append(epoch)
760       train_accuracy.append(epoch_acc)
761
762       print('Train Loss: {:.4f} Acc: {:.4f}'.format(epoch_loss, epoch_acc))
763
764       #Validacion
765       net.eval()
766
767       running_loss = 0.0
768       running_corrects = 0.0
769       for batch_idx, batch_sample in enumerate(val_loaderBOW):
770           image = batch_sample[0].to(device).float()
```

```
            question = batch_sample[4].to(device).float()
            label = batch_sample[5].to(device)
            with torch.set_grad_enabled(False):
                output = net(image,question) #salidas de la red
                _, pred_indices = torch.max(output, 1)  # [batch_size]
                loss = criterion(output, label)

                running_loss += loss.item()
                running_corrects += torch.sum(pred_indices == label)

        epoch_loss = running_loss /len(val_loaderBOW.dataset) #promedio de error
        epoch_acc = running_corrects / len(val_loaderBOW.dataset) #promedio de accuracy
        val_losses.append(epoch_loss)
        val_counter.append(epoch)
        val_accuracy.append(epoch_acc)
        #chekpoint
        if epoch_loss < best_loss:
            best_loss = epoch_loss
            best_model_wts = copy.deepcopy(net.state_dict())

        # early stopping, si el error aumenta más de 5 veces respecto al menor error,
        # terminamos el entrenamiento
        if epoch_loss > best_loss*5:
            print('\n'+'-' * 10+'Early Stopping'+'-' * 10+'\n')
            break

    print('Best val loss: {:.4f}'.format(best_loss))

    final = time.time()
    print('Training complete in {:.0f}m {:.0f}s'.format((final-inicio)//60, (final-inicio) % 60))

    net.load_state_dict(best_model_wts)
    return net, train_counter,train_losses,train_accuracy,val_counter,val_losses,val_accuracy

"""## Función train para el segundo modelo"""

import time
import copy

def train(net, optimizer, num_epocas):
    inicio = time.time()

    ans_unk_idx = train_loader.dataset.ans_vocab.unk2idx

    #copiamos el modelo utilizando la libreria copy
    best_model_wts = copy.deepcopy(net.state_dict())
    train_losses, train_counter, train_accuracy, val_losses,val_accuracy,val_counter  = [],[],[],[],[],[]

    best_acc = 0.0
    best_loss = 2e32
    for epoch in range(num_epocas):
```

```
822         print('Epoch {}/{}'.format(epoch, num_epocas-1))
823         print('-' * 10)
824 
825         net.train() #Modo entrenamiento
826 
827         running_loss = 0.0
828         running_corrects = 0.0
829         batch_size = 64
830         batch_step_size = len(train_loader.dataset) / batch_size
831         for batch_idx, batch_sample in enumerate(train_loader):
832             image = batch_sample[0].to(device).float()
833             question = batch_sample[4].to(device).float()
834             label = batch_sample[5].to(device)
835 
836             optimizer.zero_grad()
837 
838             output = net(image,question) #salidas de la red
839             pred_indices = torch.argmax(output,1)  # [batch_size]
840             loss = criterion(output, label)
841 
842             loss.backward()
843             optimizer.step()
844 
845             #castigamos en caso de que la respuesta sea desconocida
846             pred_indices[pred_indices == ans_unk_idx] = -9999
847             running_loss += loss.item()
848             running_corrects += torch.sum(pred_indices == label.data)
849             if batch_idx %100 == 0:
850                 print('Batch idx: ',batch_idx,' Batch Step: ' ,int(batch_step_size))
851 
852         epoch_loss = running_loss /len(train_loader.dataset) #promedio de error
853         epoch_acc = running_corrects / len(train_loader.dataset) #promedio de accuracy
854         train_losses.append(epoch_loss)
855         train_counter.append(epoch)
856         train_accuracy.append(epoch_acc)
857 
858         print('Train Loss: {:.4f} Acc: {:.4f}'.format(epoch_loss, epoch_acc))
859 
860         #Validacion
861         net.eval()
862 
863         running_loss = 0.0
864         running_corrects = 0.0
865         for batch_idx, batch_sample in enumerate(val_loader):
866             image = batch_sample[0].to(device).float()
867             question = batch_sample[4].to(device).float()
868             label = batch_sample[5].to(device)
869             with torch.set_grad_enabled(False):
870                 output = net(image,question) #salidas de la red
871                 _, pred_indices = torch.max(output, 1)  # [batch_size]
872                 loss = criterion(output, label)
```

```
873
874             #castigamos en caso de que la respuesta sea desconocida
875             pred_indices[pred_indices == ans_unk_idx] = -9999
876             running_loss += loss.item()
877             running_corrects += torch.sum(pred_indices == label)
878
879         epoch_loss = running_loss /len(val_loader.dataset) #promedio de error
880         epoch_acc = running_corrects / len(val_loader.dataset) #promedio de accuracy
881         val_losses.append(epoch_loss)
882         val_counter.append(epoch)
883         val_accuracy.append(epoch_acc)
884         #chekpoint
885         if epoch_loss < best_loss:
886             best_loss = epoch_loss
887             best_model_wts = copy.deepcopy(net.state_dict())
888
889         # early stopping, si el error aumenta más de 5 veces respecto al menor error,
890         # terminamos el entrenamiento
891         if epoch_loss > best_loss*5:
892             print('\n'+'-' * 10+'Early Stopping'+'-' * 10+'\n')
893             break
894
895     print('Best val loss: {:.4f}'.format(best_loss))
896
897     final = time.time()
898     print('Training complete in {:.0f}m {:.0f}s'.format((final-inicio)//60, (final-inicio) % 60))
899
900     net.load_state_dict(best_model_wts)
901     return net, train_counter,train_losses,train_accuracy,val_counter,val_losses,val_accuracy
902
903 """# Inicializando los modelos
904
905 Los demás parámetros se utilizan para generar el modelo del paper.
906
907 * Utilizando Entropia cruzada.
908 * Adam optimizer.
909 * Bath Size de 64.
910 * Learning rate de 0.001 (tambien se realizaron experimentos variando este valor).
911 """
912
913 if BOW == True:
914     net = VqaModelBOW(embed_size, qst_vocab_size, ans_vocab_size, word_embed_size,
        ↪ num_layers, hidden_size)
915     net.cuda()
916     criterion = nn.CrossEntropyLoss()
917     optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)
918
919 else:
920     net = VqaModel(embed_size, qst_vocab_size, ans_vocab_size, word_embed_size, num_layers,
        ↪ hidden_size)
921     net.cuda()
```

```
922     criterion = nn.CrossEntropyLoss()
923     optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)
924     model, train_counter,train_loss,train_accuracy,val_counter,val_loss,val_accuracy = train(net,
          ↪ optimizer, num_epocas=6)
925
926 """# Entrenamos
927
928 Se utilizan 6 epocas solamente debido al alto tiempo que toma entrenar.
929 """
930
931 if BOW == True:
932     model, train_counter,train_loss,train_accuracy,val_counter,val_loss,val_accuracy = trainBOW(
          ↪ net, optimizer, num_epocas=6)
933 else:
934     model, train_counter,train_loss,train_accuracy,val_counter,val_loss,val_accuracy = train(net,
          ↪ optimizer, num_epocas=6)
935
936 plt.figure()
937 #2b. Graficar las curvas de loss de entrenamiento y validación
938 plt.title(f"Loss curve Lr={learning_rate}")
939 plt.plot(train_counter, train_loss, label='Entrenamiento',color='blue')
940 plt.plot(val_counter,val_loss, label='Validacion',color='red')
941 plt.xlabel("Epochs")
942 plt.ylabel("Loss")
943 plt.legend()
944 plt.show()
945
946 train_accuracy_list = []
947 for x in train_accuracy:
948   train_accuracy_list.append(x.cpu())
949
950 val_accuracy_list = []
951 for x in val_accuracy:
952   val_accuracy_list.append(x.cpu())
953
954 plt.figure()
955 plt.title(f"Model Accuracy Lr={learning_rate}")
956 plt.plot(train_counter, train_accuracy_list, label='Entrenamiento',color='blue')
957 plt.plot(val_counter,val_accuracy_list, label='Validacion',color='red')
958 plt.xlabel("Epochs")
959 plt.ylabel("Accuracy")
960 plt.legend()
961 plt.show()
962
963 print("Mayor accuracy en entrenamiento:",max(train_accuracy_list), "\nMayor accuracy en validació
      ↪ n",max(val_accuracy_list))
964
965 print("Menor loss en entrenamiento:",min(train_loss), "\nMenor loss en validación",min(val_loss))
966
967 """# Visualizando los resultados en el conjunto de validación"""
968
```

```
969 model.eval()
970
971 if BOW==True:
972     val_loader = torch.utils.data.DataLoader(valDataset, batch_size=1, num_workers=2,
        ↪ pin_memory=True)
973
974     for batch_idx, batch_sample in enumerate(val_loader):
975         if batch_idx<10:
976             image = batch_sample[0].to(device).float()
977             question = batch_sample[4].to(device)
978             label = batch_sample[5].to(device)
979             with torch.set_grad_enabled(False):
980                 output = net(image,question) #salidas de la red
981                 _, pred_indice = torch.max(output, 1)
982
983             pred_answer = dataset.ans_vocab.idx2word(pred_indice)
984             answer = dataset.ans_vocab.idx2word(label)
985             question_str = ""
986             for x in question[0]:
987                 if x!=0:
988                     if dataset.qst_vocab.idx2word(x)=='<unk>':
989                         question_str+='?'
990                     else:
991                         question_str+=dataset.qst_vocab.idx2word(x)+' '
992
993             question_str
994             print("\n")
995             # Visualizar
996             img = np.squeeze(image.cpu().numpy()).transpose((1, 2, 0))
997             plt.imshow(img)
998             plt.suptitle(f"{question_str}")
999             plt.title(f"Real:{answer} - Pred:{pred_answer}")
1000            plt.axis('off')
1001            plt.show()
1002
1003            time.sleep(1)
1004
1005 """# Se guarda el modelo"""
1006
1007 model = torch.save(model.state_dict(), "modelo.pth")
```