Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Degree in Mathematics

Bachelor's Degree Thesis

Wordnet y Deep Learning: Una posible unin

Author's full name

Supervised by (name of the supervisor/s of the master's thesis)

Month, year

Thanks to...

Abstract

This should be an abstract in english, up to 1000 characters.

Keywords

keyword1, keyword2, keyword3, ...

1. Introduccin

1.1 Conocimientos previos

- Explicar el problema inicial (clasificacin de imgenes) y por que es difcil. - Explicar que es una red convolucional - Explicar que es transfer learning - Explicar el paper An Out-of-the-box Full-network Embedding for Convolutional Neural Networks - Explicar que es wordnet

1.2 Objetivos

La idea principal de tfg es buscar relaciones entre los synsets de wordnet y el full network embedding. Hiptesis iniciales: - cuanto ms concreto sea el synset ms 1 debera tener. - Cuanto ms profundo sea el layer ms 1.

2. Bibliography

A. Cdigo utilizado

```
class Data:
    """
    Esta clase consiste en los datos que voy a necesitar para hacer las estadsticas.
    Que no dependen de los synsets elegidos.

    Attributes:
        version (int): versin del embedding que utilizo puede ser 19, 25 o 31
        embedding_path (str): path
        layers (dict): Un diccionario tal que
            layers[string correspondiente al layer] = [inicio del layer, final del layer]

         labels ()

         :parameter version = Version del embedding que utilizo
    """

    def __init__(self, path, version=25):
        """

        :param version: Es la versin del embedding que queremos cargar (25,31,19)
        """
        self.version = version
        _embedding_path = "../Data/vgg16_ImageNet_ALLlayers_C1avg_imagenet_train.npz"
        self.imagenet_id_path = "../Data/synset.txt"
        if version == 25:
            _embedding = 'vgg16_ImageNet_imagenet_C1avg_E_FN_KSBsp0.15n0.25_Gall_train_.npy'
        elif version == 19:
            _embedding = 'vgg16_ImageNet_imagenet_C1avg_E_FN_KSBsp0.11n0.19_Gall_train_.npy'
        elif version == 31:
            _embedding = 'vgg16_ImageNet_imagenet_C1avg_E_FN_KSBsp0.19n0.31_Gall_train_.npy'
        else:
            _embedding = path
            print('No has puesto un embedding vlido, usando el de defoult (25)')
        self.discretized_embedding_path = '../Data/Embeddings/' + _embedding
        print('Estamos usando ' + _embedding[-20:-16])
        embedding = np.load(_embedding_path)
        self.labels = embedding['labels']
        # self.matrix = self.embedding['data_matrix']
        del embedding
        self.dmatrix = np.array(np.load(self.discretized_embedding_path))
        self.imagenet_all_ids = np.genfromtxt(self.imagenet_id_path, dtype=np.str)
        self.features_category = [-1, 0, 1]
        self.colors = ['#3643D2', 'c', '#722672', '#BF3FBF']
        self.layers = {
```

```
            'conv1_1': [0, 64],  # 1
            'conv1_2': [64, 128],  # 2
            'conv2_1': [128, 256],  # 3
            'conv2_2': [256, 384],  # 4
            'conv3_1': [384, 640],  # 5
            'conv3_2': [640, 896],  # 6
            'conv3_3': [896, 1152],  # 7
            'conv4_1': [1152, 1664],  # 8
            'conv4_2': [1664, 2176],  # 9
            'conv4_3': [2176, 2688],  # 10
            'conv5_1': [2688, 3200],  # 11
            'conv5_2': [3200, 3712],  # 12
            'conv5_3': [3712, 4224],  # 13
            'fc6': [4224, 8320],  # 14
            'fc7': [8320, 12416],  # 15
            'conv1': [0, 128],  # 16
            'conv2': [128, 384],  # 17
            'conv3': [384, 1152],  # 18
            'conv4': [1152, 2688],  # 19
            'conv5': [2688, 4224],  # 20
            'conv': [0, 4224],  # 21
            'fc6tofc7': [4224, 12416],  # 23
            # 'all':[0,12416]           # 24
        }
        self.reduced_layers = {
            'conv1': [0, 128],
            'conv2': [128, 384],
            'conv3': [384, 1152],
            'conv4': [1152, 2688],
            'conv5': [2688, 4224],
            'fc6': [4224, 8320],
            'fc7': [8320, 12416]
        }

    def __del__(self):
        self.embedding = None
        self.dmatrix = None
        self.version = None
        self.embedding_path = None
        self.layers = None
        self.labels = None
        self.features_category = None
        self.colors = None
        gc.collect()

        class Statistics:
```

```
def __init__(self, synsets, data):
    """
        Esta clase genera todas las estadsticas para un conjunto de synsets
    :param synsets: conjunto de synset del que queremos calcualr las estadsticas
    :param synset_in_data[ss_to_text(synset)] =  cantidad de elementos del synset en el to
            synset_in_data['total']
    :param dir_path es el path donde se guardaran todos los datos generados
    :param plot_path es el path donde se guardaran los plots
    :param all_features: es un diccionario tal que all_features[i] = cantidad de features
    """
    self.data = data
    self.synsets = synsets
    self.textsynsets = [str(s)[8:-7] for s in synsets]
    self.dir_path = '../Data/' + str(self.textsynsets) + str(data.version) + '/'
    self.plot_path = self.dir_path + 'plots/'
    if not path.exists(self.dir_path):
        makedirs(self.dir_path)
    if not path.exists(self.plot_path):
        makedirs(self.plot_path)
    self.stats_path = self.dir_path + str(self.textsynsets) + '_stats.txt'
    self.matrix_size = self.data.dmatrix.shape
    self.total_features = self.matrix_size[0] * self.matrix_size[1]
    self.all_features = self.count_features(self.data.dmatrix)
    self.synset_in_data = {}
    self.features_per_synset_path = self.dir_path + 'features_per_synset' + '.pkl'
    self.features_per_synset = {}
    self.features_path = self.dir_path + 'features' + str(self.textsynsets) + '.pkl'
    self.images_per_feature_path = self.dir_path + 'images_per_feature' + '.pkl'
    self.images_per_feature = {}
    self.features_per_layer_path = self.dir_path + 'features_per_layer' + str(self.textsyn
    self.features_per_image_path = self.dir_path + 'features_per_image' + str(self.textsyn
    self.synset_in_data_path = self.dir_path + 'synset_in_data_path' + str()
    self.images_per_feature_per_synset_path = self.dir_path + 'images_per_featre_per_synse
        self.textsynsets) + '.pkl'
    self.images_per_feature_per_synset = {}
    self.features_per_layer = {}
    self.features_per_image = {}
    self.intra_synset = {}
    self.intra_synset_path = self.dir_path + 'intra_synset' + str(self.textsynsets) + '.pk
    self.outlier_path = self.dir_path + 'outliers.txt'
    pathu = self.dir_path + 'latex'
    latex_file = open(pathu, 'w')
    latex_file.write('')
    latex_file.close()
    stats_file = open(self.stats_path, 'w')
    stats_file.write('')
```

```
        stats_file.close()
        plt.rcParams['figure.figsize'] = [8.0, 8.0]

    def get_in_id(self, wordnet_ss):
        """
        Input: Synset
        :param wordnet_ss:
        :return: imagenet id
        """
        # Esta funcion genera la id de imagenet a partir del synset de wordnet
        wn_id = wn.ss2of(wordnet_ss)
        return wn_id[-1] + wn_id[:8]

    def ss_to_text(self, synset):
        """ devuelve el string del nombre del synset en cuestion"""
        return str(synset)[8:-7]

    def get_index_from_ss(self, synset):
        """
        Esta funcin genera un archivo con los ndices(0:999) de la aparicin de un synset y sus
        y otro con los cdigos imagenet de todos los hipnimos
        """
        hypo = lambda s: s.hyponyms()
        path = self.dir_path + self.ss_to_text(synset) + '_index_hyponim' + '.txt'
        hyponim_file = open(path, "w")
        synset_list = []
        for thing in list(synset.closure(hypo)):
            hyponim_file.write(self.get_in_id(thing) + '\n')
            synset_list.append(self.get_in_id(thing))

        index = []
        hyponim_file.close()
        index_path = self.dir_path + self.ss_to_text(synset) + '_' + 'index' + '.txt'
        index_file = open(index_path, 'w')
        i = 0
        for lab in self.data.labels:
            if self.data.imagenet_all_ids[lab] in synset_list:
                index_file.write(str(i) + '\n')
                index.append(i)
            i += 1
        index_file.close()
        return index

    def generate_restricted_labels(self, synset):
        """
        Esta funcin genera un vector de labels con 3 valores, 0, 1 generado particionando el v
```

```
        donde los valores corresponden:
        0 : no pertenece al synset
        1 : pertenece al synset
        :param synset:
        :return:
        """
        restricted_labels = [0] * len(self.data.labels)
        index = self.get_index_from_ss(synset)
        if not path.exists(self.dir_path + '/labels'):
            makedirs(self.dir_path + '/labels')

        rl_npz = self.dir_path + '/labels/imagenet_' + self.ss_to_text(synset) + '_labels' + '
        for i in index:
            restricted_labels[i] = 1

        np.savez(rl_npz, np.array(restricted_labels))
        #return np.array(restricted_labels)

    def printlatex(self, filename):
        path = self.dir_path + 'latex'
        stats_file = open(path, 'a')
        text = r'\b' + 'egin{figure}[h] \n \centering \n \includegraphics[scale=0.5] {Images/'
        ntext = '''
       \\begin{figure}[h]
            \centering
            \\begin{subfigure}[b]{0.3\\textwidth}
            \includegraphics[width=\\textwidth] {''' + str(self.textsynsets) + '19/plots/' + f
            \caption*{Embedding 19}
            \end{subfigure}
            \\begin{subfigure}[b]{0.3\\textwidth}
            \includegraphics[width=\\textwidth] {''' + str(self.textsynsets) + '25/plots/' + f
            \caption*{Embedding 25}
            \end{subfigure}
            \\begin{subfigure}[b]{0.3\\textwidth}
            \includegraphics[width=\\textwidth] {''' + str(self.textsynsets) + '31/plots/' + f
            \caption*{Embedding 31}
            \end{subfigure}
        \end{figure}
        '''
        stats_file.write(ntext)
        stats_file.close()

    def synset_in_data_gen(self):
        """
        This function generates a dictionary with the basic stats
        devuelve synset_in_data donde:
```

```
        synset_in_data[ss_to_text(synset)] = cantidad de elementos del synset en los datos
        synset_in_data['total'] =  cantidad total de elementos

        """

        stats_file = open(self.stats_path, 'a')
        labels_size = self.data.labels.shape[0]
        self.synset_in_data['total'] = labels_size
        for synset in self.synsets:
            synset_path = self.dir_path + self.ss_to_text(synset) + '.txt'
            index_path = self.dir_path + self.ss_to_text(synset) + '_index' + '.txt'
            if path.isfile(index_path):
                index = np.genfromtxt(index_path, dtype=np.int)
            else:
                self.get_index_from_ss(synset)
                index = np.genfromtxt(index_path, dtype=np.int)
            self.synset_in_data[self.ss_to_text(synset)] = index.shape[0]
            text = 'Tenemos ' + str(labels_size) + ' imagenes, de las cuales ' + str(float(ind
                   ', el ' + str(float(index.shape[0]) / labels_size * 100) + ' son ' + self.s
            stats_file.write(text)
        with open(self.synset_in_data_path, 'wb') as handle:
            pickle.dump(self.synset_in_data, handle)
        stats_file.close()

    def plot_synsets_on_data(self):
        """
        Hace un barplot y un pieplot de la ditribucin de los synsets en los datos
        :return:
        """
        if len(self.synset_in_data) == 0:
            self.synset_in_data_gen()
        plt.bar(range(len(self.synset_in_data)), self.synset_in_data.values(), align='center')
        plt.xticks(range(len(self.synset_in_data)), self.synset_in_data.keys())
        plt.title('Distribution of the synsets in the data')
        plt.xlabel('synsets')
        plt.ylabel('Quantity of synsets')
        plt.grid()
        plt.savefig(self.plot_path + 'distribution_of_synsets_bar' + '.png')
        name = 'distribution_of_synsets_bar' + '.png'
        self.printlatex(name)
        plt.cla()
        plt.clf()
        plt.close()
        _aux = {}
        for k in self.synset_in_data.keys():
            if k != 'total':
```

```
            _aux[k] = self.synset_in_data[k]

        plt.pie([float(v) for v in _aux.values()], labels=[k for k in _aux.keys()],
                autopct=None)
        plt.title('Distribution of the synsets in the data')
        plt.grid()
        plt.savefig(self.plot_path + 'distribution_of_synsets_pie' + '.png')
        plt.cla()
        plt.clf()
        name = 'distribution_of_synsets_pie' + '.png'
        self.printlatex(name)

    def count_features(self, matrix):
        """
        Devuelve un diccionario con la cantidad de features de cada tipo de la matriz matrix
        features[category] = cantidad de category de la matriz
        """
        features = {-1: 0, 0: 0, 1: 0}
        features[1] += np.sum(np.equal(matrix, 1))
        features[-1] += np.sum(np.equal(matrix, -1))
        features[0] += np.sum(np.equal(matrix, 0))
        return features

    def plot_all_features(self):
        """
        Genera un bar plot y un pie plot con la distribucin de las features en los datos.
        :return:
        """
        plt.bar(range(len(self.all_features)), self.all_features.values(), align='center')
        plt.xticks(range(len(self.all_features)), self.all_features.keys())
        plt.title('All features')
        plt.xlabel('Categories')
        plt.ylabel('Quantity of features')
        plt.grid()
        plt.savefig(self.plot_path + 'quantity_of_features_bar' + '.png')
        name = 'quantity_of_features_bar' + '.png'
        self.printlatex(name)
        plt.cla()
        plt.clf()
        plt.pie([float(v) for v in self.all_features.values()], labels=[k for k in self.all_fe
                autopct=None)
        plt.title('All features')
        plt.grid()
        plt.savefig(self.plot_path + 'all_features_pie' + '.png')
        name = 'all_features_pie' + '.png'
        self.printlatex(name)
```

```
        plt.cla()
        plt.clf()

    def features_per_synset_gen(self):
        """
        TENGO QUE REESTRUCTURAR ESTA FUNCIN POR QUE ES UN CAOS
        genera un diccionario de la forma
        feaures_per_synset[synset][category] = la cantidad de elementos que tienen el valor ca
        seccion de la matriz correspondiente al synset
        :return:
        """

        stats_file = open(self.stats_path, 'a')
        labels_size = self.data.labels.shape[0]
        # todo: arreglar esto para no tener que hardcodearlo
        text = 'Dentro de las 50k imgenes tenemos: \n un total de ' \
               + str(self.total_features) + 'de la matriz de tamao ' + str(self.matrix_size) \
               + '\n -Features de tipo -1: ' + str(self.all_features[-1]) + ' el ' + str(
            self.all_features[-1] / self.total_features * 100) + ' %' \
               + '\n -Features de tipo 0: ' + str(self.all_features[0]) + ' el ' + str(
            self.all_features[0] / self.total_features * 100) + ' %' \
               + '\n -Features de tipo 1: ' + str(self.all_features[1]) + ' el ' + str(
            self.all_features[1] / self.total_features * 100) + ' %'
        stats_file.write(text)
        for synset in self.synsets:
            synset_path = self.dir_path + self.ss_to_text(synset) + '.txt'
            index_path = self.dir_path + self.ss_to_text(synset) + '_index' + '.txt'

            if path.isfile(index_path):
                index = np.genfromtxt(index_path, dtype=np.int)
            else:
                self.get_index_from_ss(synset)
                index = np.genfromtxt(index_path, dtype=np.int)

            self.features_per_synset[self.ss_to_text(synset)] = self.count_features(self.data.
            synset_total_features = len(index) * self.matrix_size[1]
            """
            Esta parte con el cambio que he hecho iba a petar
            text = '\nEn el ' + self.ss_to_text(synset) + ' tenemos ' + str(synset_total_featu
                   + '\n -Features de tipo -1: ' + str(self.features_per_synset[synset][-1]) +
                   + '\n -Features de tipo 0: ' + str(self.features_per_synset[synset][0]) + '
                   + '\n -Features de tipo 1: ' + str(self.features_per_synset[synset][1]) + '
            stats_file.write(text)
            """
        with open(self.features_per_synset_path, 'wb') as handle:
            pickle.dump(self.features_per_synset, handle)
```

```
    stats_file.close()

def plot_features_per_synset(self):
    """
    Hace un plot para cada synset de la cantidad de features de cada tipo que hay
    :return:
    """
    if path.isfile(self.features_per_synset_path):
        self.features_per_synset = pickle.load(open(self.features_per_synset_path, 'rb'))
    else:
        self.features_per_synset_gen()
        self.features_per_synset = pickle.load(open(self.features_per_synset_path, 'rb'))

    for synset in self.synsets:
        plt.bar(range(len(self.features_per_synset[self.ss_to_text(synset)])),
                self.features_per_synset[self.ss_to_text(synset)].values(), align='center'
        plt.xticks(range(len(self.features_per_synset[self.ss_to_text(synset)])),
                   self.features_per_synset[self.ss_to_text(synset)].keys())
        plt.title('Quantity of features per synset of ' + self.ss_to_text(synset))
        plt.xlabel('Categories')
        plt.ylabel('Quantity of features')
        plt.grid()
        plt.savefig(self.plot_path + 'features_per_synset_bar_' + self.ss_to_text(synset)
        name = 'features_per_synset_bar_' + self.ss_to_text(synset) + '.png'
        self.printlatex(name)
        plt.cla()
        plt.clf()
        plt.close()

def compare_intra_embedding(self, synset):
    index_path = self.dir_path + self.ss_to_text(synset) + '_index' + '.txt'
    syn_index = np.genfromtxt(index_path, dtype=np.int)
    total = 0
    for i, j in combinations(syn_index, 2):
        total += np.sum(np.equal(self.data.dmatrix[i, :], self.data.dmatrix[j, :]))
    return total

def intra_synset_gen(self):
    """
    Genera un diccionario con la relacion interna de los synsets:

    dict[synset][synsethijo] = cantidad de synset hijo en synset
    :return:
    """
    j = 0
    stats_file = open(self.stats_path, 'a')
```

```
        total_embeddings_communes = []
        trol = 0
        self.intra_synset = {}
        for synset in self.synsets:
            index_path = self.dir_path + self.ss_to_text(synset) + '_index' + '.txt'
            syn_index = np.genfromtxt(index_path, dtype=np.int)
            # np.sum(np.in1d(b, a))
            syn_size = syn_index.shape[0]
            self.intra_synset[self.ss_to_text(synset)] = {}
            for i in range(j, len(self.synsets)):
                child_path = self.dir_path + self.ss_to_text(self.synsets[i]) + '_index' + '.t
                child_index = np.genfromtxt(child_path, dtype=np.int)
                child_in_synset = np.sum(np.in1d(child_index, syn_index))
                self.intra_synset[self.ss_to_text(synset)][self.ss_to_text(self.synsets[i])] =
                text = 'Tenemos ' + str(syn_size) + ' ' + self.ss_to_text(synset) + ' de los c
                    child_in_synset) \
                       + ' son ' + str(self.synsets[i]) + ' el ' + str(child_in_synset / syn_s
                # print(text)
                stats_file.write(text)
            j = j + 1
            # print('embedding comn')
        with open(self.intra_synset_path, 'wb') as handle:
            pickle.dump(self.intra_synset, handle)
        stats_file.close()

    def plot_intra_synset(self):
        """
        hace un barplot de la distribucin interna de los synsets para cada synset
        (cuantos mamals hay en living thing por ejemplo)
        :return:
        """
        if path.isfile(self.intra_synset_path):
            self.intra_synset = pickle.load(open(self.intra_synset_path, 'rb'))
        else:
            self.intra_synset_gen()
            self.intra_synset = pickle.load(open(self.intra_synset_path, 'rb'))

        for synset in self.synsets:
            plt.bar(range(len(self.intra_synset[self.ss_to_text(synset)])),
                    self.intra_synset[self.ss_to_text(synset)].values(), align='center')
            plt.xticks(range(len(self.intra_synset[self.ss_to_text(synset)])),
                       self.intra_synset[self.ss_to_text(synset)].keys())
            plt.title('Distribution of the synsets')
            plt.xlabel('Synsets')
            plt.ylabel('Quantity of images')
            plt.grid()
```

```
        plt.savefig(self.plot_path + 'distribution_of_inter_synsets_bar_' + self.ss_to_tex
        name = 'distribution_of_inter_synsets_bar_' + self.ss_to_text(synset) + '.png'
        self.printlatex(name)
        plt.cla()
        plt.clf()
        plt.close()

    def images_per_feature_per_synset_gen(self):
        """
        TARDA INFINITO
        Genera un archivo con el diccionario siguiente:
            Para cada feature(0,...,12k):
                Para cada tipo(-1,0,1)
                    Para cada synset:
                        - cantidad de imgenes del synset que tienen ese tipo en la feature en
            dict[feature][category][synset]

        """
        print('Generando images_per_feature_per_synset, tarda varias horas :( ')
        for feature in range(0, self.data.dmatrix.shape[1]):
            self.images_per_feature_per_synset[feature] = {}
            feature_column = self.data.dmatrix[:, feature]
            for i in self.data.features_category:
                self.images_per_feature_per_synset[feature][i] = {}
                feature_index = np.where(np.equal(feature_column, i))
                for synset in self.synsets:
                    index_path = self.dir_path + self.ss_to_text(synset) + '_index' + '.txt'
                    synset_index = np.genfromtxt(index_path, dtype=np.int)
                    self.images_per_feature_per_synset[feature][i][self.ss_to_text(synset)] =
                        np.in1d(synset_index, feature_index))
        with open(self.images_per_feature_per_synset_path, 'wb') as handle:
            pickle.dump(self.images_per_feature_per_synset, handle)
        print('Ha generado images_per_feature_per_synset')

    def is_in_layer(self, feature, layer):
        return feature in range(layer[0], layer[1])

    def plot_images_per_feature_of_synset_per_layer(self, synset):
        """
        Here I want to plot the images per feature in an histogram per category
        :return:
        """
        if self.images_per_feature_per_synset == {}:
            if path.isfile(self.images_per_feature_per_synset_path):
                self.images_per_feature_per_synset = pickle.load(open(self.images_per_feature_
            else:
```

```
            self.images_per_feature_per_synset_gen()
            self.images_per_feature_per_synset = pickle.load(open(self.images_per_feature_

        for category in self.data.features_category:
            values = {}
            values['conv'] = {}
            values['fc6tofc7'] = {}
            for key in self.images_per_feature_per_synset.keys():
                if self.is_in_layer(key, self.data.layers['conv']):
                    values['conv'][key] = self.images_per_feature_per_synset[key][category][se
                else:
                    values['fc6tofc7'][key] = self.images_per_feature_per_synset[key][category

            plt.hist(list(values['conv'].values()), bins=50, color='#194C33')
            plt.title('Images per feature of ' + str(category) + ' of the synset ' + self.ss_t
                synset) + ' of the convolutional layer')
            plt.xlabel('Quantity of ' + str(category))
            plt.ylabel('Quantity of features')
            plt.grid()
            plt.savefig(self.plot_path + 'Images_per_feature_of_' + str(category) + '_category
                synset) + '_conv.png')
            name = 'Images_per_feature_of_' + str(category) + '_category_' + self.ss_to_text(s
            self.printlatex(name)
            plt.cla()
            plt.clf()

            plt.hist(list(values['fc6tofc7'].values()), bins=50, color='crimson')
            plt.title('Images per feature of ' + str(category) + ' of the synset ' + self.ss_t
                synset) + 'of the full connected layer')
            plt.xlabel('Quantity of ' + str(category))
            plt.ylabel('Quantity of features')
            plt.grid()
            plt.savefig(self.plot_path + 'Images_per_feature_of_' + str(category) + '_category
                synset) + '_fc.png')
            name = 'Images_per_feature_of_' + str(category) + '_category_' + self.ss_to_text(s
            self.printlatex(name)
            plt.cla()
            plt.clf()

            # El histograma acumulativo separado entre conv y fc
            plt.hist([list(values['conv'].values()), list(values['fc6tofc7'].values())], bins=
                     color=['#194C33', 'crimson'], label=['conv', 'fc'])
            plt.title('Images per feature of ' + str(category) + ' of the synset ' + self.ss_t
                synset) + ' of the conv and fc layers')
            plt.xlabel('Quantity of ' + str(category))
            plt.ylabel('Quantity of features')
```

```
            plt.legend()
            plt.grid()
            plt.savefig(self.plot_path + 'Images_per_feature_of_' + str(category) + '_category
                synset) + 'all_layers.png')
            name = 'Images_per_feature_of_' + str(category) + '_category_' + self.ss_to_text(s
            self.printlatex(name)
            plt.cla()
            plt.clf()

    def find_image_without_zero(self):
        """
        Quiero que me devuelva la posicin de las imgenes que no tengan ningun cero
        :return:
        """
        if self.features_per_image == {}:
            if path.isfile(self.features_per_image_path):
                self.features_per_image = pickle.load(open(self.features_per_image_path, 'rb')
            else:
                self.features_per_image_gen()
                self.features_per_image = pickle.load(open(self.features_per_image, 'rb'))
        for i in range(len(self.features_per_image.keys())):
            if self.features_per_image[i][0] == 0:
                print(i)
        print('end')

    def images_per_feature_gen(self):
        """Genera un archivo con el diccionario siguiente:
            Para cada feature(0,...,12k):
                Para cada tipo(-1,0,1)
                    cantidad de imgenes que tienen es categoria en la feature en cuestin
            images_per_feature[feature][category] = cantidad de imagenes que tienen esa catego
        """
        for feature in range(0, self.data.dmatrix.shape[1]):
            self.images_per_feature[feature] = {}
            feature_column = self.data.dmatrix[:, feature]
            for i in self.data.features_category:
                self.images_per_feature[feature][i] = np.sum(np.equal(feature_column, i))
        with open(self.images_per_feature_path, 'wb') as handle:
            pickle.dump(self.images_per_feature, handle)

    def images_per_feature_stats(self):
        """"
        MUERTO
        Aqu debera sacar las estadsticas de las features y guardarlas en features_stats
        """
        if self.images_per_feature == {}:
```

```
            self.images_per_feature = pickle.load(open(self.images_per_feature_path, 'rb'))
        feature_stats_path = self.features_path + '_stats'
        feature_stats_file = open(feature_stats_path, 'a')
        for feature in self.images_per_feature:
            feature_stats_file.write(str(feature) + '\n')
            for i in self.data.features_category:
                feature_stats_file.write(str(i) + ': ' + str(self.images_per_feature[feature][
        feature_stats_file.close()

    def plot_images_per_feature(self):
        """
        Here I want to plot the images per feature in an histogram per category

        En el eje x pone la cantidad de imagenes del dataset que tienen la cantidad de feaures
        :return:
        """
        if self.images_per_feature == {}:
            if path.isfile(self.images_per_feature_path):
                self.images_per_feature = pickle.load(open(self.images_per_feature_path, 'rb')
            else:
                self.images_per_feature_gen()
                self.images_per_feature = pickle.load(open(self.images_per_feature_path, 'rb')

        for category in self.data.features_category:
            values = {}
            for key in self.images_per_feature.keys():
                values[key] = self.images_per_feature[key][category]

            plt.hist(list(values.values()), bins=50)
            plt.title('Images per feature of ' + str(category) + ' category')
            plt.xlabel('Quantity of images')
            plt.ylabel('Quantity of features')
            plt.grid()
            plt.savefig(self.plot_path + 'Images_per_feature_of_' + str(category) + '_category
            name = 'Images_per_feature_of_' + str(category) + '_category' + '.png'
            self.printlatex(name)
            plt.cla()
            plt.clf()
            plt.boxplot(list(values.values()))
            plt.title('Images per feature of ' + str(category) + ' category')
            plt.grid()
            plt.savefig(self.plot_path + 'Images_per_feature_of_' + str(category) + '_category
            name = 'Images_per_feature_of_' + str(category) + '_category_box' + '.png'
            self.printlatex(name)
            plt.cla()
            plt.clf()
```

```
        values = {}
        values['conv'] = {}
        values['fc6tofc7'] = {}

        for key in self.images_per_feature.keys():
            if self.is_in_layer(key, self.data.layers['conv']):
                values['conv'][key] = self.images_per_feature[key][category]
            else:
                values['fc6tofc7'][key] = self.images_per_feature[key][category]

        # El histograma acumulativo separado entre conv y fc
        plt.hist([list(values['conv'].values()), list(values['fc6tofc7'].values())], bins=
                 color=['#194C33', 'crimson'], label=['conv', 'fc'])
        plt.title('Images per feature of ' + str(category) + ' of the conv and fc layers')
        plt.xlabel('Quantity of ' + str(category))
        plt.ylabel('Quantity of features')
        plt.legend()
        plt.grid()
        plt.savefig(self.plot_path + 'Images_per_feature_of_' + str(category) + '_category
        name = 'Images_per_feature_of_' + str(category) + '_category_' + 'all_layers.png'
        self.printlatex(name)
        plt.cla()
        plt.clf()

def find_contradicction_in_synset(self):
    """
    es un copypaste
    Quiero ver si existe algun synset que tenga el valor -1 y 1 en la misma feature
    :return:
    """
    pass

def find_outlier_in_images_per_feature(self):
    """
    Quiero que me defuelva las features outlier

    :return:
    """
    auxlayers = {
        'conv1': [0, 128],
        'conv2': [128, 384],
        'conv3': [384, 1152],
        'conv4': [1152, 2688],
        'conv5': [2688, 4224],
        'fc6': [4224, 8320],
```

```
    'fc7': [8320, 12416]
}

outlier_file = open(self.outlier_path, 'w')
if self.images_per_feature == {}:
    if path.isfile(self.images_per_feature_path):
        self.images_per_feature = pickle.load(open(self.images_per_feature_path, 'rb')
    else:
        self.images_per_feature_gen()
        self.images_per_feature = pickle.load(open(self.images_per_feature_path, 'rb')

outlier_file.write('We are using the embedding ' + str(self.data.version) + '\n')
outlier_file.write('Outliers from the synsets ' + self.ss_to_text(self.synsets) + '\n'
for category in self.data.features_category:
    vals = []
    # print('\n' + str(category) + '\n')
    for feature in range(len(self.images_per_feature.keys())):
        vals.append(self.images_per_feature[feature][category])
    mean = np.mean(vals)
    std = np.std(vals)
    # print('mean:' + str(mean) + '\n')
    # print('std:' + str(std) + '\n')
    downliers = [i for i in range(len(vals)) if vals[i] <= mean - 4 * std]
    upliers = [i for i in range(len(vals)) if vals[i] >= mean + 4 * std]
    outliers = [i for i in range(len(vals)) if vals[i] >= mean + 4 * std or vals[i] <=
    # print('lendown ' + str(len(downliers)) + '\n')
    # print('down:' + str(downliers))
    # print('lenup ' + str(len(upliers)) + '\n')
    # print('up:' + str(upliers))
    layeroutlier = {}
    for k in list(auxlayers.keys()):
        layeroutlier[k] = 0

    for i in downliers:
        for k in list(auxlayers.keys()):
            if i in range(auxlayers[k][0], auxlayers[k][1]):
                layeroutlier[k] += 1

    for i in upliers:
        for k in list(auxlayers.keys()):
            if i in range(auxlayers[k][0], auxlayers[k][1]):
                layeroutlier[k] += 1

    outlier_file.write('category ' + str(category) + '\n')
    outlier_file.write(str(outliers) + '\n Distribution in the layers: \n')
    outlier_file.write(str(layeroutlier) + '\n')
```

```
        # print(layeroutlier)
        plt.bar(range(len(layeroutlier)), layeroutlier.values(), align='center')
        plt.xticks(range(len(layeroutlier)), layeroutlier.keys())
        plt.title('Outliers images per features of ' + str(category))
        plt.xlabel('Features')
        plt.grid()
        plt.savefig(self.plot_path + 'outliers' + str(category) + '.png')
        name = 'outliers' + str(category) + '.png'
        self.printlatex(name)
        plt.cla()
        plt.clf()
    outlier_file.close()

def features_per_layer_gen(self):
    """
    Crea un diccionario de texto con la informacin de features por layer
    :return:features_per_layer[layer][category]  = cantidad de features de la category tal
    """
    for layer in self.data.layers:
        section = self.data.dmatrix[:, range(self.data.layers[layer][0], self.data.layers[
        self.features_per_layer[layer] = self.count_features(section)
    with open(self.features_per_layer_path, 'wb') as handle:
        pickle.dump(self.features_per_layer, handle)

def plot_features_per_layer(self):
    """
    pinta un barplot de las features para cada layer
    :return:
    """
    if path.isfile(self.features_per_layer_path):
        self.features_per_layer = pickle.load(open(self.features_per_layer_path, 'rb'))
    else:
        self.features_per_layer_gen()
        self.features_per_layer = pickle.load(open(self.features_per_layer_path, 'rb'))

    for layer in self.data.layers:
        plt.bar(range(len(self.features_per_layer[layer])), self.features_per_layer[layer]
        plt.xticks(range(len(self.features_per_layer[layer])), self.features_per_layer[lay
        plt.title('Fatures of the layer ' + layer)
        plt.xlabel('Features')
        plt.ylabel('Quantity of features')
        plt.grid()
        plt.savefig(self.plot_path + 'features_per_layer_of_' + layer + '.png')
        name = 'features_per_layer_of_' + layer + '.png'
        self.printlatex(name)
        plt.cla()
```

```
            plt.clf()

    def features_per_image_gen(self):
        """
        Esta funcin debera calcular para cada imagen cuantas features de cada tipo se activan
        Output:
        Un diccionario tal que:
        dic[imagen][tipo]=cantidad de features de este tipo que se activan
        """
        for image in range(0, len(self.data.labels)):
            self.features_per_image[image] = self.count_features(self.data.dmatrix[image, :])
        with open(self.features_per_image_path, 'wb') as handle:
            pickle.dump(self.features_per_image, handle)
        return self.features_per_image

    def plot_features_per_image(self):
        """
        It does a plot of the features per image for each category.
        la cantidad de imagenes que tienen tantas features -1
        :return:
        """
        if path.isfile(self.features_per_image_path):
            self.features_per_image = pickle.load(open(self.features_per_image_path, 'rb'))
        else:
            pass
            self.features_per_image_gen()
            self.features_per_image = pickle.load(open(self.features_per_image_path, 'rb'))

        for category in self.data.features_category:
            values = {}
            for key in self.features_per_image.keys():
                values[key] = self.features_per_image[key][category]
            plt.hist(list(values.values()), bins=50)
            plt.title('Features per image for ' + str(category) + ' category')
            plt.ylabel('Quantity of ' + str(category))
            plt.xlabel('Quantity of images')
            # TODO HACER EL PLOT PARA LAS TRES FEATURES JUNTITAS
            # plt.show()
            plt.grid()
            plt.savefig(self.plot_path + 'features_per_image' + str(category))
            name = 'features_per_image' + str(category)
            self.printlatex(name)
            plt.cla()
            plt.clf()

    def plot_images_per_feature_of_synset(self, synset):
```

```
    """
    Here I want to plot the images per feature in an histogram per category
    :return:
    """
    if self.images_per_feature_per_synset == {}:
        if path.isfile(self.images_per_feature_per_synset_path):
            self.images_per_feature_per_synset = pickle.load(open(self.images_per_feature_
        else:
            print('va a tardar')
            self.images_per_feature_per_synset_gen()
            self.images_per_feature_per_synset = pickle.load(open(self.images_per_feature_

    for category in self.data.features_category:
        values = {}
        for key in self.images_per_feature_per_synset.keys():
            values[key] = self.images_per_feature_per_synset[key][category][self.ss_to_tex
        plt.hist(list(values.values()), bins=50)
        plt.title('Images per feature of ' + str(category) + ' of the synset ' + self.ss_t
        plt.xlabel('Quantity of ' + str(category))
        plt.ylabel('Quantity of features')
        plt.grid()
        plt.savefig(self.plot_path + 'Images_per_feature_of_' + str(category) + '_category
            synset) + '.png')
        name = 'Images_per_feature_of_' + str(category) + '_category_' + self.ss_to_text(s
        self.printlatex(name)
        plt.cla()
        plt.clf()

def plot_changes_between_synset_reps(self):
    """
    Quiero que printe una grfica tal que en el valor de las x tenga los elementos de synse
    un acumulative plot con la  cantidad de 1, 0 y -1 de los representantes del synset en
    changes[synset][-1]
    :return: void
    """
    plt.rcParams['figure.figsize'] = [16.0, 8.0]
    changes_in_synset = {}
    ones = []
    zeros = []
    negones = []
    for synset in self.synsets:
        rep = self.get_representive(synset)
        changes_in_synset[self.ss_to_text(synset)] = self.count_features(rep)
        negones.append(changes_in_synset[self.ss_to_text(synset)][-1])
        zeros.append(changes_in_synset[self.ss_to_text(synset)][0])
        ones.append(changes_in_synset[self.ss_to_text(synset)][1])
```

```
        plot_index = np.arange(len(self.synsets))
        p_negones = plt.bar(plot_index, negones, color='#4C194C')
        p_zeros = plt.bar(plot_index, zeros, color='#7F3FBF', bottom=negones)
        p_ones = plt.bar(plot_index, ones, color='#3F7FBF', bottom=[sum(x) for x in zip(negone

        plt.ylabel('Cantidad')
        plt.title('Comparativa entre las categorias por synset')
        plt.xticks(plot_index, self.textsynsets)
        plt.legend((p_negones[0], p_zeros[0], p_ones[0]), ('-1', '0', '1'))
        plt.grid()
        name = 'Comparative_of_synsets.png'
        plt.savefig(self.plot_path + name)
        self.printlatex(name)
        plt.cla()
        plt.clf()
        plt.rcParams['figure.figsize'] = [8.0, 8.0]

    def plot_changes_between_synset_reps_per_layer(self):
        """
        Quiero que printe una grfica para cada synset tal que en el valor de las x tenga los e
        y en el de las ordenadas un acumulative plot con la  cantidad de 1, 0 y -1 de los repr
        cuestin para cada layer.
        :return: void
        """
        plt.rcParams['figure.figsize'] = [16.0, 8.0]
        for synset in self.synsets:
            changes_in_synset = {}
            ones = []
            zeros = []
            negones = []
            plot_index = np.arange(len(self.data.reduced_layers))
            for layer in self.data.reduced_layers:
                rep = self.get_representive_per_layer(synset, layer)
                changes_in_synset[layer] = self.count_features(rep)
                negones.append(changes_in_synset[layer][-1])
                zeros.append(changes_in_synset[layer][0])
                ones.append(changes_in_synset[layer][1])
            p_negones = plt.bar(plot_index, negones, color='#4C194C')
            p_zeros = plt.bar(plot_index, zeros, color='#7F3FBF', bottom=negones)
            p_ones = plt.bar(plot_index, ones, color='#3F7FBF', bottom=[sum(x) for x in zip(ne
            plt.ylabel('Cantidad')
            plt.title('Comparativa entre las categorias por layer de ' + self.ss_to_text(synse
            plt.xticks(plot_index, list(self.data.reduced_layers))
            plt.legend((p_negones[0], p_zeros[0], p_ones[0]), ('-1', '0', '1'))
            plt.grid()
```

```
            name = 'Comparative_of_synsets_' + self.ss_to_text(synset) + '.png'
            plt.savefig(self.plot_path + name)
            self.printlatex(name)
            plt.cla()
            plt.clf()
        plt.rcParams['figure.figsize'] = [8.0, 8.0]

    def plot_all(self):
        """
        ORDENAR LOS PLOTS
        Esta funcion llama a todos los plots que tengo
        """
        plt.rcParams['figure.figsize'] = [8.0, 8.0]
        self.plot_features_per_image()
        plt.cla()
        plt.clf()
        plt.close("all")
        self.plot_all_features()
        plt.cla()
        plt.clf()
        plt.close("all")
        self.plot_features_per_synset()
        plt.cla()
        plt.clf()
        plt.close("all")
        self.plot_images_per_feature()
        plt.cla()
        plt.clf()
        plt.close("all")
        self.plot_synsets_on_data()
        plt.cla()
        plt.clf()
        plt.close("all")
        self.plot_intra_synset()
        for synset in self.synsets:
            self.plot_images_per_feature_of_synset(synset)
            plt.cla()
            plt.clf()
            plt.close("all")
            self.plot_images_per_feature_of_synset_per_layer(synset)
            plt.cla()
            plt.clf()
            plt.close("all")
        self.plot_features_per_layer()
        plt.cla()
        plt.close("all")
```

```
        plt.clf()
        self.plot_matrix()
        plt.cla()
        plt.clf()
        plt.close("all")
        self.plot_changes_between_synset_reps()
        plt.cla()
        plt.clf()
        plt.close("all")
        self.plot_changes_between_synset_reps_per_layer()
        plt.cla()
        plt.clf()
        plt.close("all")

    def existsfile(self, path):
        """
        TODO: QUIERO QUE NO GENERE LAS IMAGENES SI YA LAS TENGO
        :param path:
        :return:
        """
        return path.isfile(path)

    def get_representive(self, synset):
        """
        Quiero que me devuelva un vector tal que el valor i sea el que tiene mayor proporcin d
        representative[feature] = 1, -1 o 0 segn el valor que se repite ms veces.
        Utilizo como valor auxiliar un diccionario de la siguiente forma, que obtengo de image
        dict[feature] = {1: cantidad de 1, -1: cantidad de -1, 0: cantidad de 0}

        :param synset:
        :return: representative
        """
        if self.images_per_feature_per_synset == {}:
            if path.isfile(self.images_per_feature_per_synset_path):
                self.images_per_feature_per_synset = pickle.load(open(self.images_per_feature_
            else:
                print('Generando images_per_feature_per_synset')
                self.images_per_feature_per_synset_gen()
                self.images_per_feature_per_synset = pickle.load(open(self.images_per_feature_

        representative = []
        # aux[feature][category] = cantidad de valores de la category en cuestion para la feat
        aux = {}
        for feature in self.images_per_feature_per_synset.keys():
            aux[feature] = {}
            for category in self.images_per_feature_per_synset[feature].keys():
```

```
                aux[feature][category] = self.images_per_feature_per_synset[feature][category]
        for feature in aux:
            representative.append(max(aux[feature], key=aux[feature].get))
        return representative

    def get_representive_per_layer(self, synset, layer):
        """
        Quiero que me devuelva un vector tal que el valor i sea el que tiene mayor proporcin d
        representative[feature] = 1, -1 o 0 segn el valor que se repite ms veces.
        Utilizo como valor auxiliar un diccionario de la siguiente forma, que obtengo de image
        dict[feature] = {1: cantidad de 1, -1: cantidad de -1, 0: cantidad de 0}

        :param synset:
        :param layer:
        :return: representative
        """
        if self.images_per_feature_per_synset == {}:
            if path.isfile(self.images_per_feature_per_synset_path):
                self.images_per_feature_per_synset = pickle.load(open(self.images_per_feature_
            else:
                print('Generando images_per_feature_per_synset')
                self.images_per_feature_per_synset_gen()
                self.images_per_feature_per_synset = pickle.load(open(self.images_per_feature_

        representative = []
        # aux[feature][category] = cantidad de valores de la category en cuestion para la feat
        aux = {}
        #section = self.data.dmatrix[:, range(self.data.layers[layer][0], self.data.layers[lay
        for feature in range(self.data.layers[layer][0], self.data.layers[layer][1]):
            aux[feature] = {}
            for category in self.images_per_feature_per_synset[feature].keys():
                aux[feature][category] = self.images_per_feature_per_synset[feature][category]
        for feature in aux:
            representative.append(max(aux[feature], key=aux[feature].get))
        return representative

    def changes_matrix(self, synset1, synset2):
        """
        Genero una matriz de los cambios de los valores para el vector representante del synse

          | -1    0    1
        ------------------
        -1| a     b     c
        0 | d     e     f
        1 | g     h     i
```

```
    donde el valor a sera la cantidad de -1 que se mantienen constantes entre un synset y
    El valor b sera los 0 del synset 1 que pasan a ser -1 en el synset 2
    etd.

    :param synset1:
    :param synset2:
    :return: cambios
    """
    rep1 = self.get_representive(synset1)
    rep2 = self.get_representive(synset2)
    changes = np.zeros([3, 3])
    for feature in range(0, len(rep1)):
        r1 = rep1[feature]
        r2 = rep2[feature]
        if r1 == r2 == -1:
            changes[0][0] += 1
        elif r1 == r2 == 0:
            changes[1][1] += 1
        elif r1 == r2 == 1:
            changes[2][2] += 1

        elif r1 == -1 and r2 == 0:
            changes[1][0] += 1
        elif r1 == -1 and r2 == 1:
            changes[2][0] += 1

        elif r1 == 0 and r2 == -1:
            changes[0][1] += 1
        elif r1 == 0 and r2 == 1:
            changes[2][1] += 1

        elif r1 == 1 and r2 == -1:
            changes[0][2] += 1
        elif r1 == 1 and r2 == 0:
            changes[1][2] += 1

    fig, ax = plt.subplots(figsize=(5, 5))
    diag = np.zeros([3, 3])
    diag[0, 0] += 1
    diag[1, 1] += 1
    diag[2, 2] += 1
    ax.matshow(diag, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(changes.shape[0]):
        for j in range(changes.shape[1]):
            ax.text(x=j, y=i, s=changes[i, j], va='center', ha='center', fontsize=20)
    plt.xticks([0, 1, 2], [-1, 0, 1])
```

```
        plt.yticks([0, 1, 2], [-1, 0, 1])
        plt.xlabel('Original values')
        plt.ylabel('New values')
        plt.title('Changes from ' + self.ss_to_text(synset1) + ' to ' + self.ss_to_text(synset
        plt.tight_layout()
        plt.savefig(
            self.plot_path + 'Changes from ' + self.ss_to_text(synset1) + ' to ' + self.ss_to_
        name = 'Changes from ' + self.ss_to_text(synset1) + ' to ' + self.ss_to_text(synset2)
        self.printlatex(name)
        plt.cla()
        plt.clf()

    def changes_matrix_per_layer(self, synset1, synset2, layer):
        """
        Genero una matriz de los cambios de los valores para el vector representante del synse

          | -1     0    1
        -----------------
        -1| a      b     c
        0 | d      e     f
        1 | g      h     i

        donde el valor a sera la cantidad de -1 que se mantienen constantes entre un synset y
        El valor b sera los 0 del synset 1 que pasan a ser -1 en el synset 2
        etd.

        :param synset1:
        :param synset2:
        :param layer:
        :return: cambios
        """
        rep1 = self.get_representive_per_layer(synset1, layer)
        rep2 = self.get_representive_per_layer(synset2, layer)
        changes = np.zeros([3, 3])
        for feature in range(0, len(rep1)):
            r1 = rep1[feature]
            r2 = rep2[feature]
            if r1 == r2 == -1:
                changes[0][0] += 1
            elif r1 == r2 == 0:
                changes[1][1] += 1
            elif r1 == r2 == 1:
                changes[2][2] += 1

            elif r1 == -1 and r2 == 0:
                changes[1][0] += 1
```

```
            elif r1 == -1 and r2 == 1:
                changes[2][0] += 1

            elif r1 == 0 and r2 == -1:
                changes[0][1] += 1
            elif r1 == 0 and r2 == 1:
                changes[2][1] += 1

            elif r1 == 1 and r2 == -1:
                changes[0][2] += 1
            elif r1 == 1 and r2 == 0:
                changes[1][2] += 1

        fig, ax = plt.subplots(figsize=(5, 5))
        diag = np.zeros([3, 3])
        diag[0, 0] += 1
        diag[1, 1] += 1
        diag[2, 2] += 1
        ax.matshow(diag, cmap=plt.cm.Blues, alpha=0.3)
        for i in range(changes.shape[0]):
            for j in range(changes.shape[1]):
                ax.text(x=j, y=i, s=changes[i, j], va='center', ha='center', fontsize=20)
        plt.xticks([0, 1, 2], [-1, 0, 1])
        plt.yticks([0, 1, 2], [-1, 0, 1])
        plt.xlabel('Original values')
        plt.ylabel('New values')
        plt.title('Changes from ' + self.ss_to_text(synset1) + ' to ' + self.ss_to_text(synset
        plt.tight_layout()
        plt.savefig(
            self.plot_path + 'Changes from ' + self.ss_to_text(synset1) + ' to ' + self.ss_to_
                synset2) + ' of ' + layer + '.png')
        name = 'Changes from ' + self.ss_to_text(synset1) + ' to ' + self.ss_to_text(synset2)
        self.printlatex(name)
        plt.cla()
        plt.clf()

    def plot_matrix(self):
        """
        Quiero pintar la matriz de cambios para cada synset
        :return:
        """
        for synset1 in self.synsets:
            for synset2 in self.synsets:
                self.changes_matrix(synset1, synset2)
                for layer in self.data.reduced_layers:
                    self.changes_matrix(synset1, synset2)
```

```
            self.changes_matrix_per_layer(synset1, synset2, layer)
            plt.cla()
            plt.clf()
            plt.close("all")
```