Clasificación de gráficos usando el modelo AlexNet

Angela Mayhua
Maestría en Ciencia de la Computación
Universidad Católica San Pablo
angela.mayhua@ucsp.edu.pe

***Resumen*—El reconocimiento y análisis de gráficos (*charts*) es un área de investigación relativamente nueva donde se han propuesto técnicas y algoritmos para extraer información a partir de imágenes de gráficos, rediseñarlos para permitir un mejor entendimiento, entre otros; siendo la clasificación de los gráficos una etapa importante. En este trabajo se propone un clasificador de gráficos generado a partir de un proceso de *fine-tuning* de la red neuronal convolucional AlexNet, teniendo como entrada una imagen estática de un gráfico y como salida el tipo de gráfico que se muestra en la imagen.**

I. Introducción

Durante los últimos años los gráficos, diagramas y otros componentes visuales han sido utilizados para transmitir información de una mejor manera y para resumir información de grandes volúmenes de datos. Los gráficos son usados principalmente en reportes, revistas, blogs, libros y en la web; estando disponibles en la mayoría de los casos como imágenes estáticas. Durante los últimos años algunas investigaciones se han enfocado en la clasificación, análisis, rediseño e interpretación de estos gráficos [4, 6]; siendo la clasificación una etapa fundamental antes de proceder con el análisis o rediseño, para que en las etapas posteriores se apliquen las técnicas adecuadas dependiendo del tipo de gráfico; además se ha observado que una de las técnicas más usadas para la clasificación de estos gráficos ha sido SVM (*Support Vector Machine*) [4, 6].

En este trabajo se propone un clasificador de gráficos generado a partir de un proceso de *fine-tuning* de la red neuronal convolucional (CNN) AlexNet pre-entrenada, que permitirá identificar el tipo de gráfico y de esta manera asignarle una de las 10 categorías existentes: gráfico de área, gráfico de barras, gráfico de líneas, mapa, diagrama de Pareto, gráfico circular, gráfica de radar, diagrama de dispersión, tablas y diagramas de Venn.

El presente informe está organizado en cuatro secciones: la Sección II indica algunos trabajos relacionados que plantean algún método de clasificación de gráficos, en la Sección III se detalla la implementación y consideraciones tomadas para clasificar gráficos a partir de imágenes estáticas, en la Sección IV se muestran los resultados de las pruebas realizadas y en la Sección V se encuentran las conclusiones a las que se ha llegado después de realizar este trabajo.

II. Trabajos relacionados

La clasificación de imágenes ha sido un problema de visión computacional bastante analizado; sin embargo, la clasificación y análisis de imágenes extraídas de documentos es un problema nuevo que está siendo analizado por investigadores en estos últimos años, porque con este conocimiento se puede crear herramientas que permitan entender el contenido de un documento y generar mejores resultados para los usuarios [2].

Una de las técnicas más usadas para clasificación de gráficos es SVM, Savva et al. [6] determina el tipo de gráfico usando características de bajo nivel de las imágenes y características extraídas a nivel de texto que generan el patrón característico de una imagen de entrada; el conjunto de patrones entrenan un clasificador SVM multiclase que tiene un porcentaje de aceptación de 80 % para el corpus de 10 categorías. Karthikeyani and Nagarajan [4] proponen un conjunto de características de textura calculados a partir del gráfico localizado dentro de la imagen, estas características conforman el patrón característico del gráfico que son la entrada para los tres clasificadores comparados en este trabajo: kNN, SVM y MLP; el clasificador con mejor resultado fue kNN (*k-Nearest Neighbour*) con un porcentaje de aceptación de 78.06 % para un corpus de 8 categorías. El trabajo de Mishchenko and Vassilieva [5] no requiere aprendizaje supervisado, su método de clasificación se basa en modelos, donde cada gráfico tiene establecido un modelo con la estructura geométrica y topológica de los segmentos que lo conforman (arcos y aristas) y en la etapa de clasificación se extraen los bordes de la imagen de entrada y se compara con los modelos existentes, siendo la salida el modelo al que más se parezca.

III. Clasificación de gráficos

III-A. *Fine-tuning*

El concepto de *fine-tuning* consiste en tomar un modelo entrenado, adaptar la arquitectura y de esta manera reducir el tiempo de entrenamiento porque la red es inicializada con pesos de un modelo pre-entrenado [3]. Este proceso es usado normalmente cuando el corpus no es muy grande y se desea entrenar una red neuronal convolucional. Para crear el clasificador de gráficos se ha utilizado el modelo de la red AlexNet pre-entrenada del framework Caffe [3].

III-B. *Caffe*

Caffe[1] es un framework de *deep learning* desarrollado por Berkeley AI Research en conjunto con una comunidad de colaboradores. Caffe define su propio esquema de modelamiento de las capas de las redes neuronales; almacena, comunica y manipula la información como *blobs* que son arrays estándares e interfaces de memoria unificadas para el framework.

La capa es la unidad fundamental para el funcionamiento de un modelo porque se encargan de aplicar los filtros, realizar las transformaciones, normalizar, entre otras operaciones. En la Figura 1 se puede observar la estructura de una capa en Caffe, cada capa recibe datos como entrada (*bottom blob*) y después del procesamiento genera datos de salida (*top blob*).

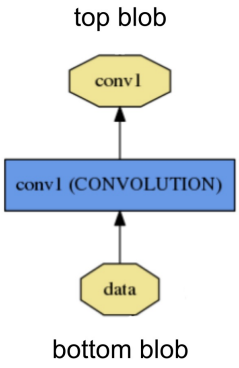
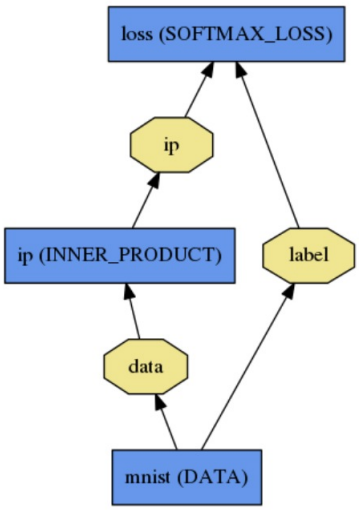


Figura 1. Estructura de una capa en Caffe: los datos de entrada se conocen como *bottom blob* y las salidas como *top blob*

La red se define como un conjunto de capas que interactúan entre sí y Caffe permite su modelamiento en texto plano, lo que lo hace más entendible al usuario. En la Figura 2 se puede observar un ejemplo de un clasificador simple de regresión logística y su modelamiento en texto plano.



(a) Red modelada

```
name: "LogReg"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  data_param {
    source: "input_leveldb"
    batch_size: 64
  }
}
layer {
  name: "ip"
  type: "InnerProduct"
  bottom: "data"
  top: "ip"
  inner_product_param {
    num_output: 2
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```

(b) Modelamiento en texto plano de la red

Figura 2. Estructura de un clasificador simple y su modelamiento en texto plano que puede ser leído directamente por Caffe

Los modelos son almacenados usando el protocolo buffer *prototxt*, mientras que los pesos aprendidos son serializados usando el protocolo binario en archivos .caffemodel. Este tipo de almacenamiento de los modelos en Caffe permite una mayor flexibilidad y extensibilidad, por lo que los modelos pueden usarse en múltiples lenguajes como C++ y Python a través del protocolo de buffer de Google[2].

[1]Página web de Caffe: http://caffe.berkeleyvision.org/

[2]Google Protocol Buffer: https://github.com/google/protobuf

III-C. Implementación

La implementación del clasificador se realizó usando Caffe como framework y Python como lenguaje de programación, en este caso se usó la red pre-entrenada AlexNet que tiene el framework a la cual se le aplicó *fine-tuning* [1] para que pueda clasificar las 10 categorías de gráficos.

La estructura de la red neuronal base del clasificador se muestra en la Figura 4, la estructura es la misma que la red AlexNet, la diferencia con el modelo de Caffe es que la lectura de los datos de entrenamiento y de prueba es a partir de un archivo txt, donde cada línea tiene la ruta de la imagen, seguido de la clase a la que pertenece (ver Figura 3).

```
"""
Create net to chart classification using a txt file (for training or testing).
Each line of this file has the image path and its label
"""
def chart_net(train=True, learn_all=False, subset=None):
    if subset is None:
        subset = 'train' if train else 'test'
    source = path_mydata + num_fold + '_%s.txt' % subset
    transform_param = dict(mirror=train, crop_size=227,
                           mean_file=path_basenet + 'imagenet_mean.binaryproto')
    style_data, style_label = L.ImageData(
        transform_param=transform_param, source=source,
        batch_size=50, new_height=256, new_width=256, ntop=2)
    return caffenet(data=style_data, label=style_label, train=train,
                    num_classes=NUM_CLASSES,
                    classifier_name='fc8_revision',
                    learn_all=learn_all)
```

F5_test.txt | F5_train.txt

```
67  data/AreaGraph/00095.png 0
68  data/AreaGraph/00096.png 0
69  data/AreaGraph/00097.png 0
70  data/AreaGraph/00098.png 0
71  data/AreaGraph/00103.png 0
72  data/AreaGraph/00104.png 0
73  data/BarGraph/00000.png 1
74  data/BarGraph/00004.png 1
75  data/BarGraph/00005.png 1
76  data/BarGraph/00007.png 1
77  data/BarGraph/00008.png 1
```

Figura 3. Estructura del archivo txt usado para cargar los datos a la red neuronal

Como se indicó anteriormente se utilizó *fine-tuning* para crear el clasificador e inicializarlo con pesos de una red pre-entrenada; para realizar la copia de los pesos se utilizó la función *copy_from* teniendo como parámetro de entrada la ruta del archivo que contiene los pesos de la red.

```
weights_pretrained = path_basenet
    + 'bvlc_reference_caffenet.caffemodel'
chart_solver = caffe.get_solver(
    solver(chart_net(train=True)))
chart_solver.net
    .copy_from(weights_pretrained)
```

Caffe también facilita la configuración del CPU o GPU durante el entrenamiento de la red, en nuestro caso se ha configurado para que sea ejecutado en GPU debido a la cantidad de iteraciones que se realizarán y para que se ejecute en menor tiempo.

```
# Execution in GPU
caffe.set_device(0)
caffe.set_mode_gpu()
```

Finalmente, los modelos de la red (para entrenamiento y prueba) son almacenados:

```
mode = 'train' if train else 'test'
filename = path_savemodel + num_fold + '_'
    + typeNet + '_' + mode + '.prototxt'
with open(filename, 'w') as f:
    f.write(str(n.to_proto()))
    return filename
```

También se almacena el archivo de configuración de los parámetros de aprendizaje, conocido en Caffe como *solver*:

```
filename = path_savemodel + num_fold + '_'
    + typeNet + '_solver.prototxt'
with open(filename, 'w') as f:
     f.write(str(s))
     return filename
```

Y los pesos de la red neuronal son guardados para que puedan ser usados en otro lenguaje o para otro tipo de pruebas.

```
weights = {}
for name, s in solvers:
    filename = path_savemodel + num_fold
        + '_weights.%s.caffemodel' % name
    weights[name] = filename
    s.net.save(weights[name])
```

IV. Resultados

IV-A. Dataset

Para la evaluación del clasificador se ha seleccionado el conjunto de datos creado por Savva et al. [6] que consta de 2084 imágenes de gráficos divididas en 10 categorías. En la Tabla I se muestra la distribución por categoría.

En la Figura 5 se muestran algunos ejemplos de imágenes que serán usadas.

Este conjunto de datos fue seleccionado porque el método de clasificación de ReVision [6] es citado en varios trabajos relacionados a análisis de gráficos.

IV-B. Pruebas y resultados

La salida del clasificador es un vector de dimensión 10, donde cada posición indica la probabilidad de que la imagen de entrada pertenezca a esa clase. En la Figura 6 se muestra la distribución de las 10 clases que tienen las más altas probabilidades dada una imagen de entrada; como se puede observar, cuando se inicializa el clasificador

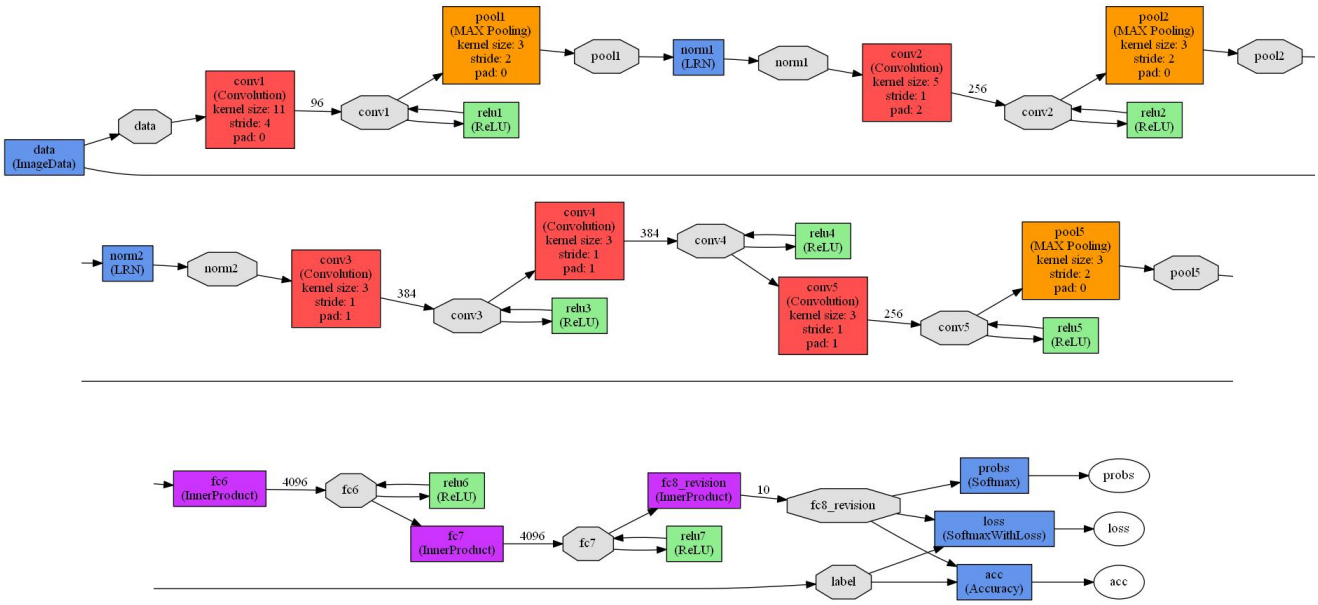
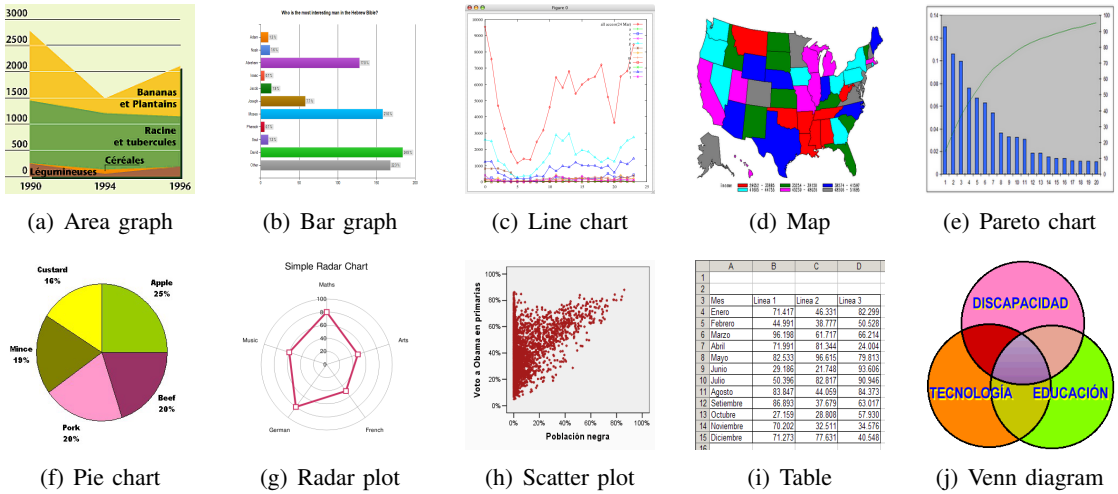


Figura 4. Estructura de la red neuronal durante la etapa de entrenamiento



(a) Area graph (b) Bar graph (c) Line chart (d) Map (e) Pareto chart

(f) Pie chart (g) Radar plot (h) Scatter plot (i) Table (j) Venn diagram

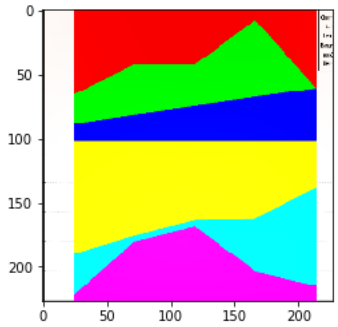
Figura 5. Ejemplos de gráficos contenidos en el dataset

y no se realiza ningún entrenamiento, cada clase tiene la misma probabilidad ($1/NUM\_CLASSES = 10\,\%$) de ser elegido como clase resultante dada una imagen de entrada.

Para realizar las pruebas se han creado dos clasificadores basados en la estructura de AlexNet: uno pre-entrenado que carga los pesos de la red AlexNet (*fine-tuning*) y que sólo entrena la capa clasificadora (*fc8*) para mejorar sus pesos y clasificar de mejor manera; mientras que el otro clasificador inicializa sus pesos aleatoriamente y realiza un entrenamiento total para mejorar los pesos de todas sus capas. En la Figura 7 se observa los valores de pérdida y precisión alcanzados por los clasificadores en las diferentes iteraciones de la etapa de entrenamiento; se puede notar que los valores de precisión para la red pre-entrenada (*pretrained*) convergen más rápido que la red sin pre-entrenamiento (*scratch*); necesitando menos iteraciones para lograr un porcentaje aceptable de precisión.

Adicionalmente, en la Figura 8(a) se puede observar cómo disminuye rápidamente el valor de pérdida para el clasificador pre-entrenado (línea azul) y el clasificador inicializado aleatoriamente (anaranjado) apenas mejora. Con respecto a la exactitud (Figura 8(b)), también se nota un incremento más rápido y estable para el clasificador pre-entrenado en comparación con el otro.

En la Figura 9 se puede observar los filtros de la capa *conv1* del clasificador preentrenado y los filtros apren-



(a) Imagen de entrada

```
Predicted label:  n03291819 envelope
Top 10 predicted ImageNet labels =
        (1) 55.65% n03291819 envelope
        (2)  8.51% n04507155 umbrella
        (3)  5.86% n02840245 binder, ring-binder
        (4)  3.04% n04136333 sarong
        (5)  2.44% n03958227 plastic bag
        (6)  1.76% n04525038 velvet
        (7)  1.71% n03871628 packet
        (8)  1.59% n03485794 handkerchief, hankie, hanky, hankey
        (9)  1.41% n03355925 flagpole, flagstaff
        (10)  1.30% n04548362 wallet, billfold, notecase, pocketbook
```

(b) Distribución de probabilidades en AlexNet

```
Predicted label:  AreaGraph
Top 10 predicted ChartNet Untrained labels =
        (1) 10.00% AreaGraph
        (2) 10.00% BarGraph
        (3) 10.00% LineGraph
        (4) 10.00% Map
        (5) 10.00% ParetoChart
        (6) 10.00% PieChart
        (7) 10.00% RadarPlot
        (8) 10.00% ScatterGraph
        (9) 10.00% Table
        (10) 10.00% VennDiagram
```

(c) Distribución de probabilidades en clasificador no entrenado

Figura 6. Distribución de las 10 clases que tienen las más altas probabilidades de ser elegidas para la imagen de entrada, la clase resultante es la clase con mayor probabilidad

Tabla I
DISTRIBUCIÓN DE IMÁGENES POR CATEGORÍAS DE LA BASE DE DATOS CREADA EN [6]

Categoría	**Nro Imágenes**
Gráficos de área (*Area graph*)	90
Gráficos de barras (*Bar graph*)	169
Gráfico de líneas (*Line chart*)	318
Mapa (*Map*)	249
Diagrama de Pareto (*Pareto chart*)	168
Gráfico circular (*Pie chart*)	210
Gráfica de radar (*Radar plot*)	137
Gráfico de dispersión (*Scatter plot*)	372
Tablas (*Tables*)	263
Diagramas de Venn (*Venn diagrams*)	108
Total	2084

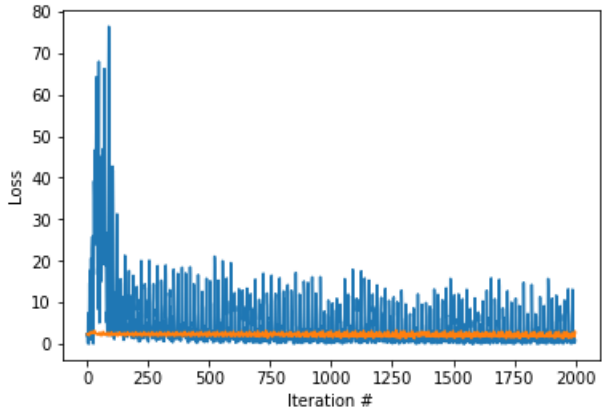
```
Running solvers for 2000 iterations...
  0) pretrained: loss=2.303, acc= 0%; scratch: loss=2.303, acc= 0%
 50) pretrained: loss=5.054, acc=52%; scratch: loss=2.443, acc= 0%
100) pretrained: loss=12.772, acc=32%; scratch: loss=2.826, acc= 0%
150) pretrained: loss=0.896, acc=82%; scratch: loss=2.361, acc= 0%
200) pretrained: loss=3.507, acc=70%; scratch: loss=2.850, acc= 0%
250) pretrained: loss=0.698, acc=94%; scratch: loss=2.354, acc= 0%
300) pretrained: loss=1.996, acc=68%; scratch: loss=2.893, acc= 0%
350) pretrained: loss=0.830, acc=90%; scratch: loss=2.350, acc= 0%
400) pretrained: loss=1.729, acc=78%; scratch: loss=2.929, acc= 0%
450) pretrained: loss=0.180, acc=98%; scratch: loss=2.347, acc= 0%
500) pretrained: loss=1.827, acc=74%; scratch: loss=2.952, acc= 0%
550) pretrained: loss=0.567, acc=88%; scratch: loss=2.344, acc= 0%
```

Figura 7. Pérdida y precisión alcanzada en las iteraciones del proceso de entrenamiento de los clasificadores: *pretrained* es el clasificador con *fine-tuning* y *scratch* es el clasificador sin *fine-tuning*

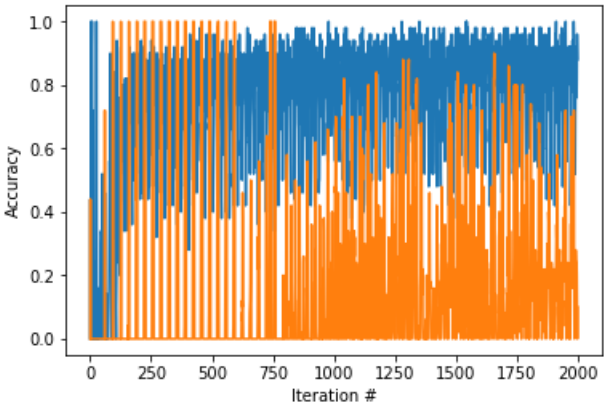
didos por el clasificador sin preentrenamiento; también se puede observar las salidas de esta capa después de aplicar los filtros correspondientes.

La evaluación de la red neuronal se realizó usando el método *5-fold cross-validation* multiclase, sobre las 10 categorías del conjunto de datos. Por cada clase se realiza 5 particiones aleatorias que éstas se unen a las particiones de las otras clases para crear un *fold*; en cada iteración del *cross-validation* se escogen 4 *folds* para ser usados en la etapa de entrenamiento y 1 *fold* para la etapa de pruebas. Los resultados mostrados en la Tabla II corresponden al promedio de la evaluación de los 5 *folds*; la primera columna corresponde a los resultados obtenidos en el paper ReVision [6], la segunda columna muestra los resultados obtenidos por un clasificador basado en AlexNet sin *fine-tuning* y la última columna corresponde a los resultados obtenidos por un clasificador basado en AlexNet con *fine-tuning*; el número de iteraciones para ambas CNN fue de 2000 iteraciones.

Después de realizar las pruebas se puede visualizar que el clasificador basado en AlexNet con *fine-tuning* presenta el mayor valor de precisión para la clasificación de gráficos, logrando un valor mayor al clasificador propuesto por Savva et al. [6], se necesitaría incrementar



(a) Gráfico de pérdida



(b) Gráfico de exactitud

Figura 8. Gráficos comparativos de los valores de pérdida y exactitud de los dos clasificadores: clasificador preentrenado (línea azul) y clasificador sin pre-entrenamiento (línea anaranjada)

Tabla II
CUADRO DE COMPARACIÓN DEL PORCENTAJE DE PRECISIÓN OBTENIDOS POR REVISION Y LOS CLASIFICADORES BASADOS EN CNN PROPUESTOS EN ESTE TRABAJO

Categoría	**ReVision**	**AlexNet sin *fine-tuning***	**AlexNet con *fine-tuning***
Area graph	88 %	31.8 %	77.2 %
Bar graph	78 %	1 %	80.6 %
Line chart	73 %	43.2 %	58.2 %
Map	84 %	27.2 %	83 %
Pareto chart	85 %	39.6 %	84.6 %
Pie chart	79 %	79 %	93.2 %
Radar plot	88 %	0.6 %	91 %
Scatter plot	79 %	47.4 %	92.8 %
Table	86 %	25.4 %	89.2 %
Venn diagram	75 %	0 %	87.4 %
Promedio	80 %	29.52 %	83.72 %

el número de iteraciones para mejorar los pesos en la capa clasificadora de la red y lograr una mayor precisión, otra consideración es balancear el número de imágenes por clase para un mejor entrenamiento. El clasificador sin pre-entrenamiento tiene una precisión muy baja debido a que se ha inicializado con pesos aleatorios, por lo que sería necesario incrementar el número de iteraciones de entrenamiento para mejorar su precisión.

V. CONCLUSIONES

- El proceso de *fine-tuning* es de utilidad cuando la base de datos no cuenta con un gran número de elementos para la etapa de entrenamiento.
- El uso de una red pre-entrenada permite reducir el número de iteraciones en la etapa de entrenamiento porque converge más rápido, logrando en muchos casos un porcentaje de precisión aceptable en menor número de iteraciones. Cuando una red es inicializada con pesos aleatorios, se necesita un gran número de iteraciones para alcanzar un valor de precisión aceptable.
- Los pesos iniciales asignados a una red neuronal convolucional son un factor importante que influencia en el número de iteraciones necesarias para converger a la solución del problema; dependiendo qué tan buenos sean esos pesos se puede reducir o incrementar la cantidad de iteraciones.
- Utilizar una CNN evita la tarea de pensar en las características adecuadas que representen una imagen u objeto en particular para generar un patrón característico adecuado.
- Caffe es un framework de mucha utilidad cuando se desea modelar una red neuronal convolucional porque permite serializar los modelos y usarlos con otros lenguajes de programación a través del protocolo buffer de Google.
- Cuando se tiene una gran cantidad de datos o el número de iteraciones es grande, es conveniente usar la GPU para reducir el tiempo de ejecución de la etapa de entrenamiento.

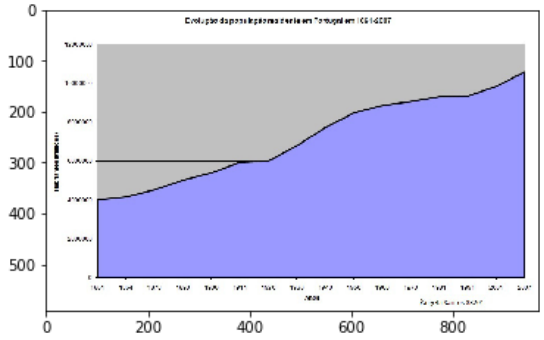
REFERENCIAS

[1] Caffe. Fine-tuning a Pretrained Network for Style Recognition. http://nbviewer.jupyter.org/github/BVLC/caffe/blob/master/examples/02-fine-tuning.ipynb.

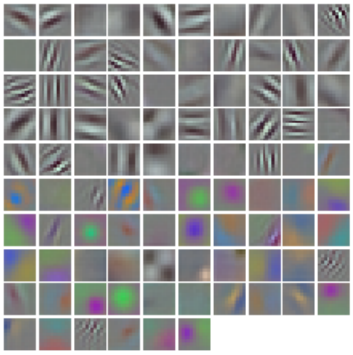
[2] H. Hamraz. Classification of chart images. resreport, Department of Computer Science, University of Kentucky, 2014. URL http://cs.uky.edu/~hhamraz/reports/chart_image_classification.pdf.

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

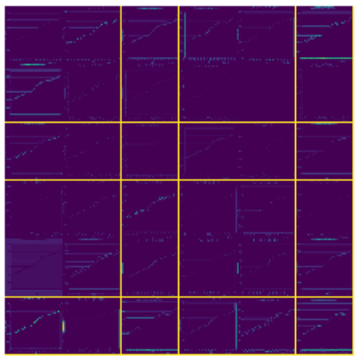
[4] V. Karthikeyani and S. Nagarajan. Machine learning classification algorithms to recognize chart types in



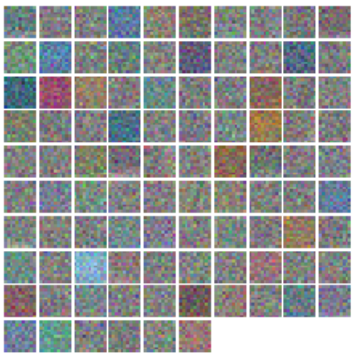
(a) Imagen de entrada



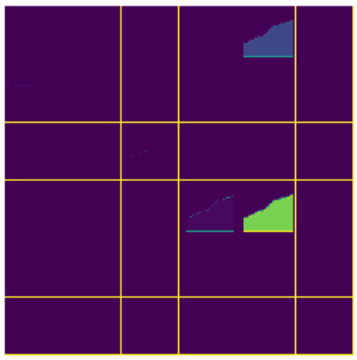
(b) Filtros de *conv1* del clasificador preentrenado



(c) Salida de *conv1* del clasificador preentrenado



(d) Filtros de *conv1* del clasificador no preentrenado



(e) Salida de *conv1* del clasificador no preentrenado

Figura 9. Visualización de los filtros de la capa *conv1* de los dos clasificadores creados al finalizar las 2000 iteraciones y la salida de los 36 primeros filtros teniendo como entrada la imagen de (a)

portable document format (pdf) files. *International Journal of Computer Applications*, 39(2):1–5, 2012.

[5] A. Mishchenko and N. Vassilieva. *Model-Based Chart Image Classification*, pages 476–485. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[6] M. Savva, N. Kong, A. Chhajta, L. Fei-Fei, M. Agrawala, and J. Heer. ReVision: Automated Classification, Analysis and Redesign of Chart Images. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 393–402, Santa Barbara, California, USA, 2011. ACM.