Universitat Autònoma
de Barcelona

INFORME INICIAL

Reconocimiento de caracteres usando Capsule Nets

López Hernández, Ancor

Tabla de contenido

Estado del arte

La clasificación de imágenes es un problema central en machine learning, es por esto que surgen las Redes Neuronales Convolucionales (CNN), la idea básica de estas redes es entrenar a la red con imágenes etiquetadas para que el algoritmo pueda ir abstrayendo las características(features) que le permitan clasificar correctamente esa imagen y una vez entrenada, poder utilizar ese modelo de red para clasificar nuevas imágenes sin etiquetar. A lo largo de los años estas redes han ido siendo mejoradas hasta llegar al estado del arte actual, a modo de pequeño repaso y puesta en contexto se muestran algunas de las redes más importantes:

LeNeT-5

Fue una CNN de 7 capas pionera introducida por LeCun en 1988 [1] y fue utilizada por la gran mayoría de bancos para clasificar los dígitos escritos a mano en los cheques. Una de sus claves para el éxito fue utilizar convoluciones para extraer características espaciales y fue la base sobre las que se desarrollaron las siguientes arquitecturas.

AlexNet

En 2012, Alex Krizhevsky [2] presentó esta red que introducía nuevas mejoras como el uso de ReLU que ayudó a prevenir The vanishing gradient problem [3] e introdujo el concepto de dropout que consiste en activar y desactivar aleatoriamente las neuronas de cada capa para prevenir el overfitting. También introdujo el concepto de data augmentation que consiste en entrenar a la red neuronal con imágenes con diferente rotación y ángulo.

VGGNet

Fue presentada en 2014 por Karen Simonyan and Andrew Zisserman [4], su principal mejora fue añadir más capas a la red neuronal.

GoogleNet

Esta red [5] creada en 2015 permite que en cada capa se puedan realizar operaciones convoluciones y pooling a la vez concatenado sus salidas y propagándolas a la siguiente capa lo que se traduce en un mejor aprendizaje de las features en cada capa.

Microsoft ResNet

Creada en 2015 [6], la idea detrás de esta red neuronal es que, en cierto punto, añadir más capas no mejora el rendimiento de la red, de hecho, ocurre todo lo contrario debido al gradiente. Para evitar esto cada dos capas un identity mapping addition lo cual ayuda a propagar el error.

Problemas con las CNN

Uno de los problemas de las CNN es que no guardan la relación espacial entre las features de bajo nivel y las de alto nivel, es aquí donde entra la red neuronal Capsule Nets que añade esta relación a las features [7] (pose, en esencia translación más rotación.) lo que le permite detectar objetos rotados y trasladados en la escena, también llamada equivariancia.

Otra ventaja respecto a las CNN es que Capsule Nets solo necesita una fracción de los datos que usa una CNN para lograr el mismo rendimiento, además utiliza el Route by Agreement para cambiar los pesos entre neuronas.

Capsule Nets

Esta red neuronal [8] se divide en 2 partes: Encoder y Decoder. Las primeras 3 capas forman parte del Encoder y las 3 siguientes forman el Decoder.

Capa 1. Convolutional

Capa 2. PrimaryCaps

Capa 3. DigitCaps

Capa 4. Fully connected #1

Capa 5. Fully connected #2

Capa 6. Fully connected #3

Encoder

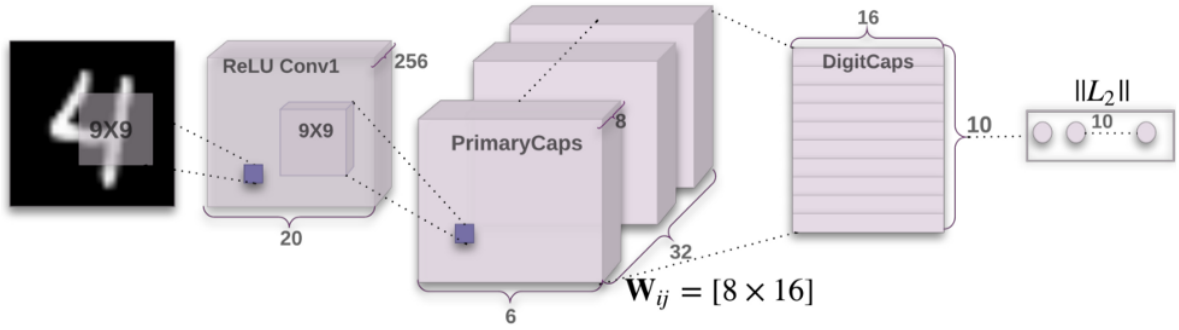


Figure 1 Arquitectura del Encoder. Fuente [8]

Trabajaremos con imágenes de 28x28 (MNIST dataset), por lo tanto, éste será el input del Encoder y nos dará una salida de vectores de 16 Dimensiones

Capa1. Convolutional Layer

Detecta las basic features de nuestras imágenes de entrada. Consta de 256 feature maps con kernels de 9x9x1 y stride 1 con función de activación ReLU.

Capa 2 Primary Caps Layer

Cada primaryCap está formada por un grid de 6x6 capsulas de vectores de 8 Dimensiones, las cuáles tienes 32 maps o PrimaryCaps, en total tendremos 1152 capsulas (6x6x32) con kernels de convolución de 9x9x256

Capa 3 DigitCaps Layer

Está formada por 10 DigitCapsules, una por cada número que queremos reconocer, con un vector de output de 16 Dimensiones.

Como reciben el vector de 8Dimensiones de las PrimaryCaps, cada uno de estos vectores tiene su matriz de pesos de 8x16 para transformar los vectores 8D en 16D.

Loss Function

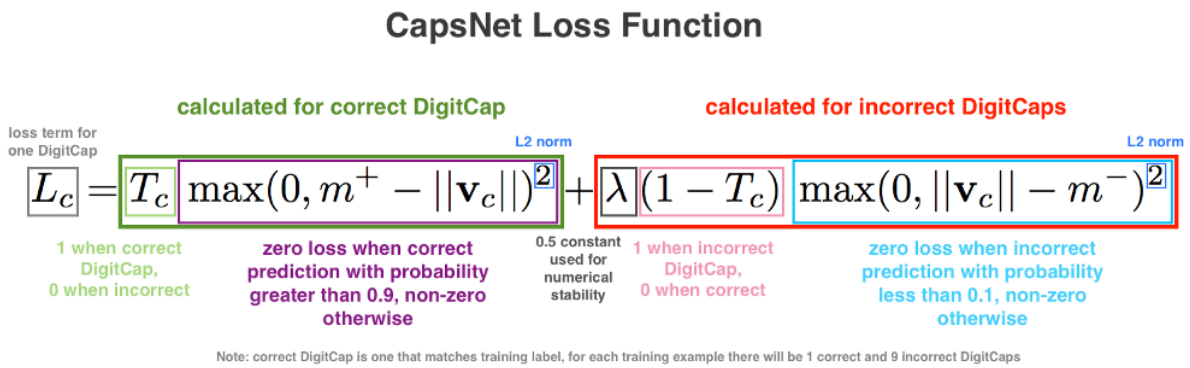


Figure 2 Loss Function. Source: [9]

Esto quiere decir que, si un objeto de la clase k está presente en la imagen, la capsula correspondiente de nivel más alto tendrá un output vector cuyo ||v||² (Norma) será como mínimo de 0.9, de lo contrario el output vector tendrá una Norma inferior a 0.1.

Decoder

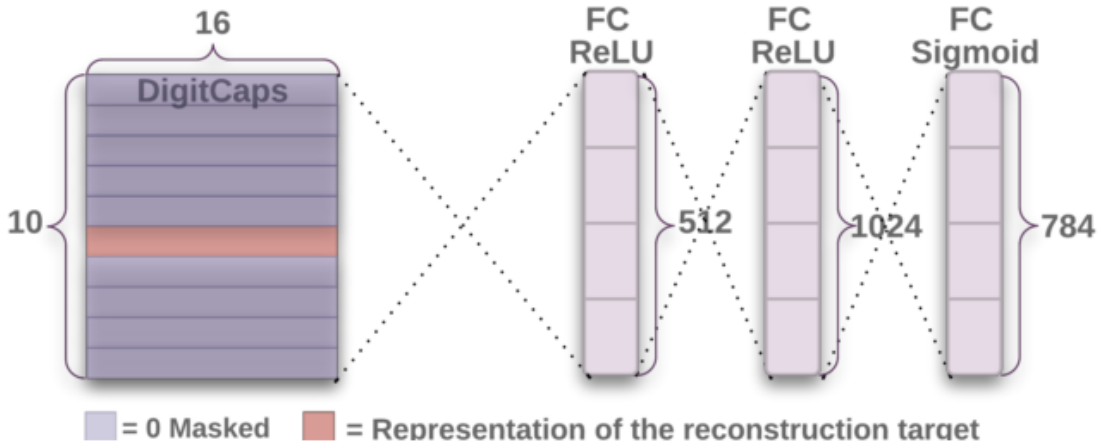


Figure 3 Arquitecture del Decoder. Source: [8].

Su función es coger el vector de 16D que tiene como output el Encoder y aprender a decodificarlo como la imagen de un número y recrea una imagen de 28x28 como la que teníamos en la entrada de Encoder.

Consta de tres fully connected layers de las cuales:

- La primera consta de 512 neuronas con activación ReLU.
- La segunda consta de 1024 neuronas con activación ReLU.
- La tercera tiene 784 neuronas con activación Sigmoid que después de hacer un reshape nos dará la imagen de 28x28.

¿Como funcionan las capsulas?

En esta tabla se pueden ver las principales diferencias entre una capsula y una neurona.

Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector($\mathbf{u}_i$)	scalar(x_i)
Operation	Affine Transform	$\hat{\mathbf{u}}_{j\|i} = \mathbf{W}_{ij}\mathbf{u}_i$	–
	Weighting	$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j\|i}$	$a_j = \sum_i w_i x_i + b$
	Sum		
	Nonlinear Activation	$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1+\|\mathbf{s}_j\|^2}\frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}$	$h_j = f(a_j)$
Output		vector($\mathbf{v}_j$)	scalar(h_j)

Figure 4 Tabla con diferencias entre una neurona y una capsula. Source: [10]

Podemos describir una neurona con 4 pasos:

1. Multiplicación matricial de los input vectors
2. scalar weighting of input vectors
3. Suma de los vectores pesados (weighted)
4. vector-to-vector activación no lineal.

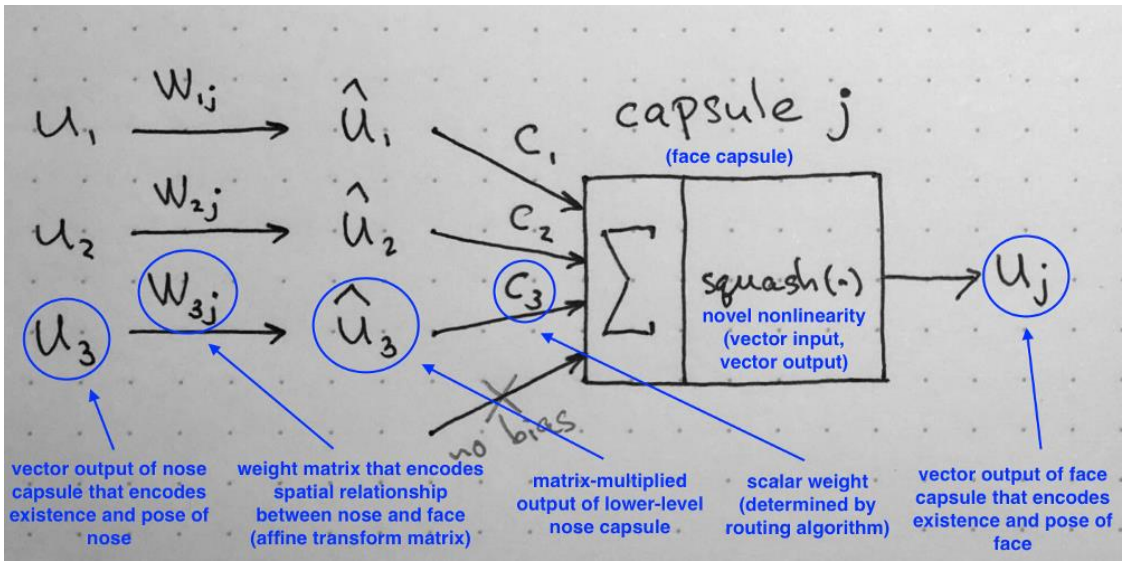


Figure 5 Esquema de como funciona una capsula. Source: [10] [11]

Route By Agreement

Algoritmo que utilizan las capsulas para modificar sus pesos y determinar hacia que capsula del nivel superior envían sus outputs.

Procedure 1 Routing algorithm.

1: **procedure** ROUTING($\hat{\boldsymbol{u}}_{j|i}, r, l$)
2: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3: **for** r iterations **do**
4: for all capsule i in layer l: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$ ▷ `softmax` computes Eq. 3
5: for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6: for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$ ▷ `squash` computes Eq. 1
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

Figure 6 Algoritmo de Routing. Source: [8]

Para este algoritmo necesitaremos los outputs de las capsulas en la capa l y las iteraciones que queremos hacer.

- Iniciamos los pesos entre capsulas de bij a 0.
- Hacemos un for de tantas iteraciones como r.
- Asignamos un peso de ruta (ci) a cada capsula de la capa l utilizando la función softmax, la cual nos asegura que cada peso será un número no negativo y la suma de todos dará 1.
- Hacemos un sumatorio de todos los vectores de predicción multiplicados por su peso en la siguiente capa de capsulas (sj).
- Como probablemente el vector sj sea mayor que 1, le aplicamos una función de squash para dejarlo entre 0 y 1.
- Finalmente actualizamos los pesos multiplicando los vectores de output de predicciones de las capsulas de la capa l por su vector de output actual vj y lo sumamos al valor que tenía la variable bij. Al multiplicar los dos vectores lo que conseguimos es dar mucho peso a las capsulas que tienen inputs y outputs parecidos y viceversa.

Objetivos

- Aprender Pytorch, ya que es el framework que se utilizará para implementar esta red neuronal.

- Entender y ejecutar el código de una de las implementaciones de la red neuronal CapsuleNets. https://github.com/higgsfield/Capsule-Network-Tutorial

- Una vez ejecutado el código y obtenido los resultados, utilizar otro dataset adaptando el código ya existente, por ejemplo, EMNIST o CIFAR10 y analizar los resultados.

- Sintetizar datos, generando imágenes de manera automática y cambiar la arquitectura de Capsule Nets modificando su arquitectura.

Metodología y Planificación

Metodología

Primeramente, aprenderé Pytorch utilizando los tutoriales proporcionados en su página web. Una vez hecho esto, se procederá a coger el código de https://github.com/higgsfield/Capsule-Network-Tutorial donde está implementada la red Capsulenets y ejecutarlo para obtener los primeros resultados y ver el accuracy que se consigue con el dataset MNIST y compraralo con el accuracy obtenido con otros modelos de redes neuronales, para poder ejecutar el código se necesitarán las siguientes tecnologías:

- Python 3 ya que la red neuronal CapsuleNets está implementada en este lenguaje.
- Pytorch que es un framework pensado para desarrollar redes neuronales sobre Python.
- TorchVision para poder descargar y acceder a los datasets sobre los cuales entrenaremos y evaluaremos nuestra red neuronal.
- Numpy para la gestión de matrices.
- Matplotlib para poder printar por pantalla los resultados obtenidos de nuestra red neuronal y poder compararlos con el dataset original.

Seguidamente ejecutaremos nuestro código sobre otro dataset (MNIST o CIFAR10) y analizaremos sus resultados.

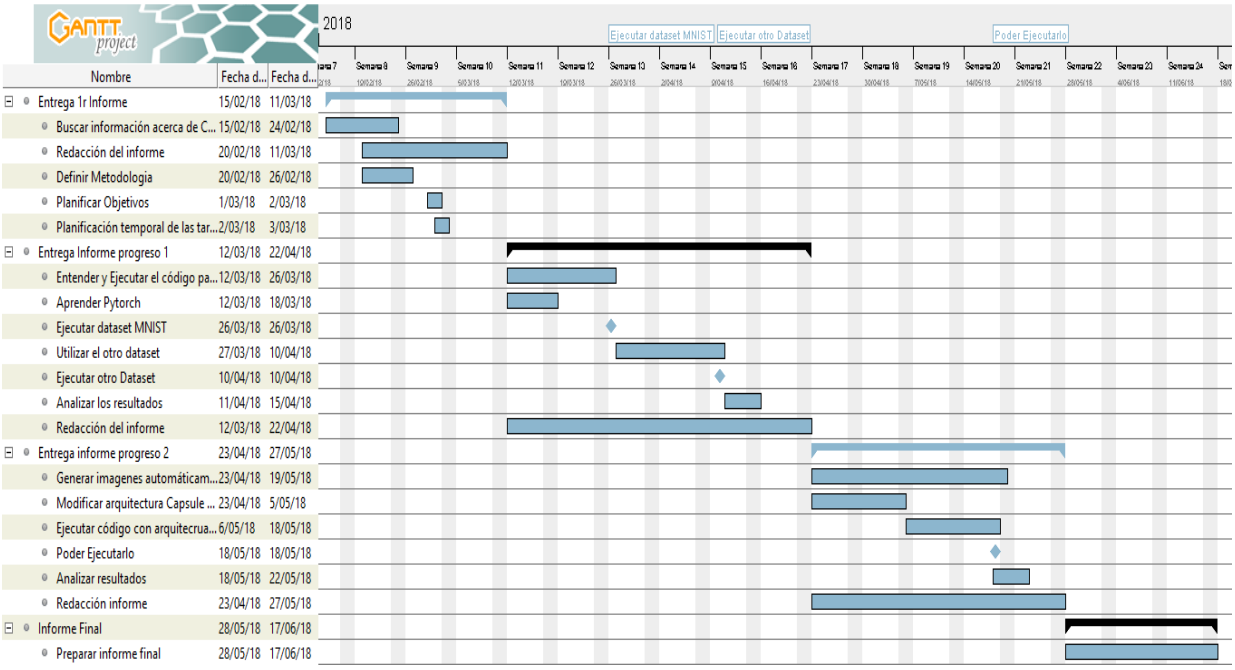
Finalmente definiremos nuevas arquitecturas de Capsule Nets añadiendo más capas a nuestra red y observando cómo cambian los resultados sobre los mismo datasets probados anteriormente respecto de la arquitectura inicial y trataremos de generar nuevas imágenes automáticamente usando diferentes fuentes del ordenador, modificando la rotación, el ruido, el tamaño….

Para llevar el control de toda la documentación que se irá generando durante todas las fases del proyecto, ya sean informes, resultados, ficheros de código… se utilizará la herramienta Github.

Planificación

- Para planificar las fases utilizaré un diagrama de Gantt ya que me permitirá ver con claridad las tareas a seguir y el tiempo que necesitaré para realizar cada una de ellas.
  Marcaré cada objetivo, excepto el primero, como un milestone que estará al final de cada fase, las cuales estarán definidas en base a las fechas de entregas de los informes que se deben realizar.

Diagrama de Gantt



Bibliografía

[1] Y. LeCun, L. Bottou, Y. Bengio y P. Haffner, «Gradient-Based Learning Applied to Document Recognition,» 1988. [En línea]. Available: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.

[2] G. H. I. S. Alex Krizhevsky, «ImageNet Classification with Deep Convolutional,» 2012. [En línea]. Available: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[3] Wikipedia, «Vanishing gradient problem,» [En línea]. Available: https://en.wikipedia.org/wiki/Vanishing_gradient_problem.

[4] A. Z. Karen Simonyan, «Very Deep Convolutional Networks for Large-Scale Image Recognition,» 2014. [En línea]. Available: https://arxiv.org/pdf/1409.1556.pdf.

[5] W. L. Y. J. P. S. S. R. D. A. ,. D. E. V. V. A. R. G. I. U. o. N. C. C. H. U. o. M. A. A. Christian Szegedy, «Going Deeper with Convolutions,» 2015. [En línea]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf.

[6] X. Z. R. J. S. Kaiming He, «Deep Residual Learning for Image Recognition,» 2015. [En línea]. Available: https://arxiv.org/pdf/1512.03385v1.pdf.

[7] M. Pechyonkin, «Understanding Hinton's Capsule Networks. Part I: Intuition,» Medium, 2017. [En línea]. Available: https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuition-b4b559d1159b.

[8] N. F. G. E. H. Sara Sabour, «Dynamic Routing Between Capsules,» [En línea]. Available: https://arxiv.org/abs/1710.09829.

[9] M. Pechyonkin, «Understanding Hinton's Capsule Networks. Part IV: CapsNet Architecture,» Medium, 2017. [En línea]. Available: https://medium.com/@pechyonkin/part-iv-capsnet-architecture-6a64422f7dce.

[10] M. Pechyonkin, «Understanding Hinton’s Capsule Networks. Part II: How Capsules Work.,» Medium, 2017. [En línea]. Available: https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-ii-how-capsules-work-153b6ade9f66.

[11] M. Pechyonkin, «Understanding Hinton’s Capsule Networks. Part III: Dynamic Routing Between Capsules.,» Medium, 2017. [En línea]. Available: https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-iii-dynamic-routing-between-capsules-349f6d30418.