**INSTITUTO TECNÓLOGICO DE LA PAZ**

**DIVISIÓN DE ESTUDIOS PROFESIONALES**

**Reporte semanal de residencia semana 02**

**“Reconocimiento y seguimiento de objetos en entornos controlados”**

**CARRERA**

**Ingeniería en Sistemas Computacionales**

**PRESENTA**

**17310793 – Eloy Antonio Clemente Rosas**

La Paz, Baja California Sur, México, 05 de Febrero del 2021

# Índice

Contenido

[Índice 2](#_Toc94984442)

[Introducción 3](#_Toc94984443)

[Desarrollo de actividades 4](#_Toc94984444)

[Redes neuronales convolucionales 4](#_Toc94984445)

[Guardar, exportar y cargar un modelo. 7](#_Toc94984446)

[Conclusión 13](#_Toc94984447)

# Introducción

El presente documento contiene las actividades, y aprendizajes realizados durante la segunda semana en mi residencia profesional. Esta semana fue un poco más teórica que la anterior, enfocando principalmente la atención en lo que son las redes neuronales convolucionales, de las cuales se explicara más adelante dentro del documento.

Durante la semana también, me toco investigar sobre las diversas maneras en las que se puede exportar un modelo de red neuronal, ver las diferentes características que tiene cada una de las formas de exportar dicho modelo y seleccionar los mejores formas como candidatas para implantarse y usarse en otro ambiente.

Esta semana en el tecnológico, preparamos una computadora, instalando Cudnn y Cuda que sean compatibles, esto para poder entrenar nuestra red neuronal en un equipo desde la GPU, optimizando la velocidad de entrenamiento.

*“CUDA son las siglas de Compute Unified Device Architecture que hace referencia a una plataforma de computación en paralelo incluyendo un compilador y un conjunto de herramientas de desarrollo creadas por Nvidia que permiten a los programadores usar una variación del lenguaje de programación C para codificar algoritmos en GPU de Nvidia”.*

*“Deep Neural Network (cuDNN) es una biblioteca de primitivas con aceleración de GPU para usar con redes neuronales profundas”.*

*Instalar NVIDIA cuDNN. (26 de Agosto ,2021). VMWare Docs. Recuperado de:* [*https://docs.vmware.com/es/VMware-vSphere-Bitfusion/4.0/Example-Guide/GUID-B69A5522-3E3E-408A-9901-ED4C14168CA5.html*](https://docs.vmware.com/es/VMware-vSphere-Bitfusion/4.0/Example-Guide/GUID-B69A5522-3E3E-408A-9901-ED4C14168CA5.html)

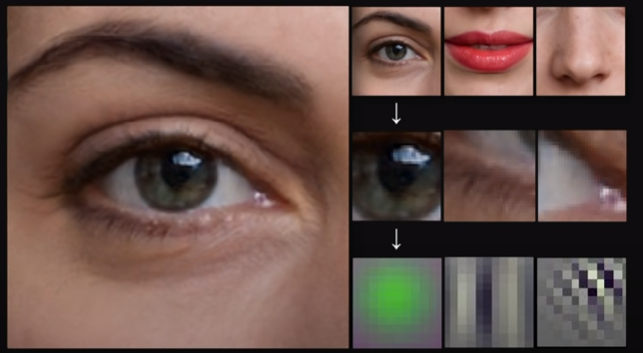
# Desarrollo de actividades

## Redes neuronales convolucionales

Es un tipo de red neuronal artificial donde las neuronas artificiales, corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico.

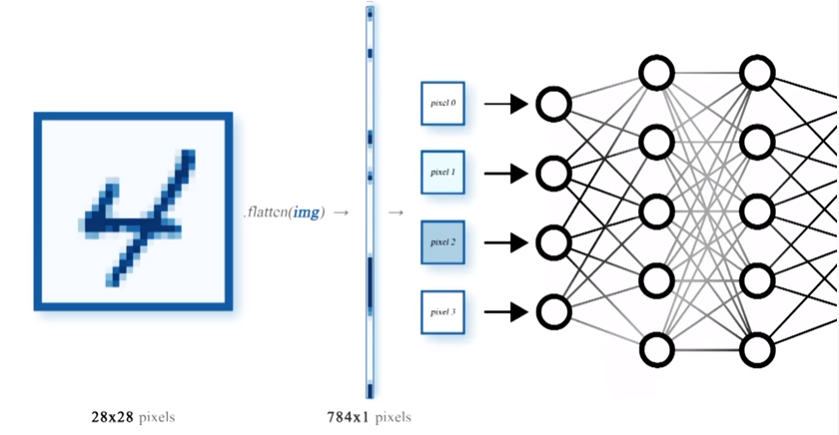
Son redes neurales capaces de construir funciones completas a partir de otras menos complejas. Un ejemplo es la detección facial, en la que primeras capas eligen líneas horizontales y verticales mientras que las capas posteriores encuentran narices y bocas.

Las redes neuronales convolucionales están especializadas para trabajar con imágenes, la importancia de estas redes viene de la capacidad que tienen de poder descifrar los patrones más complejos en enormes data sets de imágenes, proporcionando visión a una computadora, que tanto puede observar rostros de personas, radiografías de personas o peatones que se cruzan en un coche autónomo.



Si identificamos un rostro primero identificamos los patrones básicos necesarios para saber que es un rostro.

En una red neuronal convencional introducimos todos los pixeles de una foto como se muestra en la imagen como si de una variable convencional se tratara, como si fuera un vector plano, y algo tan importante como la posición misma de cada pixel se pierde. La red neuronal multicapa solo vería un vector plano de pixeles.

Cada posición de cada pixel es importante, de cada uno nacen nuevos pixeles, tanto en el ancho como en el alto, esto lo que logra es que surjan estructuras, formas y patrones que analizadas correctamente sirven para analizar que estamos viendo.

A partir de esta idea surgen las redes neuronales convolucionales.

Una red neuronal convolucional es un tipo de red neuronal multicapa que se caracteriza por aplicar un tipo de capa donde se realiza una operación matemática conocida como convolución.

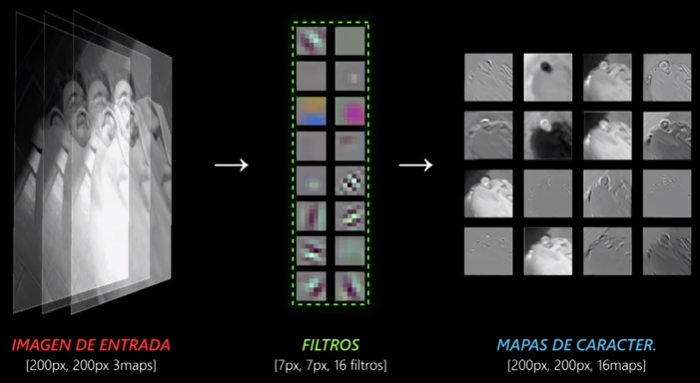
Cada pixel nuevo se calculara multiplicando la matriz de pixeles, como se puede apreciar en la imagen anterior facilitando su posterior análisis. En el análisis puede ayudar a detectar cosas diferentes según cuales sean los valores el filtro que se definan. Esto lo que estará aprendiendo la red neuronal poco a poco para hacer mejor su tarea.



Aprender estos filtros para detectar patrones es el principal trabajo de la red neuronal convolucional.

A cada una de las imágenes generadas se le conoce como mapa de caracteristicas. Actua como un mapa donde se nos indica en que parte de la imagen se ha detectado la caracteristica buscada por dicho filtro, cada pixel blanco sera una activacion que nos indique que el elemento buscado esta.

Esta operación se realiza secuencialmente donde la salida de una capa se convierte en la entrada de la siguiente capa.



La operación de convolucion cada vez se va a ir haciendo mas potente por ej.

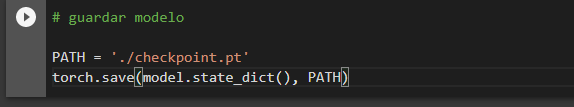
Donde antes existia una region de 9 pixeles ahora es, un unico pixel de informacion, y si se aplica una convolucion sobre estos mapas de caracteristicas estaremos accediendo a mas informacion de la imagen original.

## Guardar, exportar y cargar un modelo.

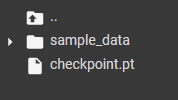
Existen diferentes formas que nos ofrece pytorch para guardar y exportar los modelos, con el fin de luego usarlos en la aplicación que querramos.

La primera opción consiste en guardar sólo los parámetros de la red. Pytorch nos permite guardar el *state\_dict* del modelo, un dict de Python que contiene una relación directa entre todas las capas con parámetros de la red y sus valores.

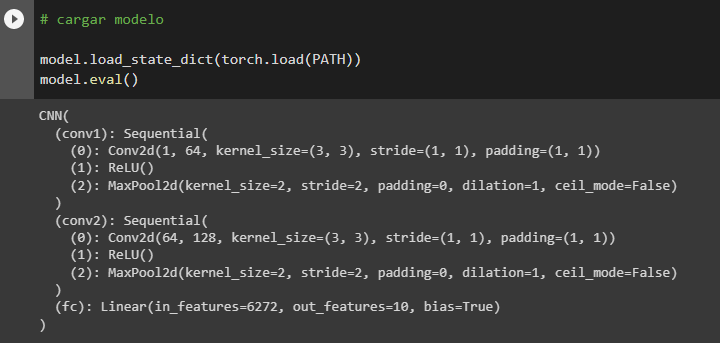
Donde indicamos un PATH, es el nombre que le dara pytorch a nuestro modelo.



Se guarda en los archivos con el nombre que asignamos en el path, es posible descargarlo para poder usarlo donde se necesite.



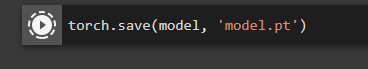
Para cargar el modelo se usa el siguiente comando que lee el archivo que le indiquemos en la variable PATH, y los mete directamente capa por capa en el modelo. Esta listo para realizar predicciones.

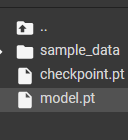


Si bien de esta manera podemos guardar y cargar el modelo de manera eficiente, necesitamos tener un modelo instanciado para poder llamar a la función *model.load\_state\_dict().* Esto significa que necesitaremos la definición de nuestro modelo allá dónde queramos importarlo (lo cual es poco flexible).

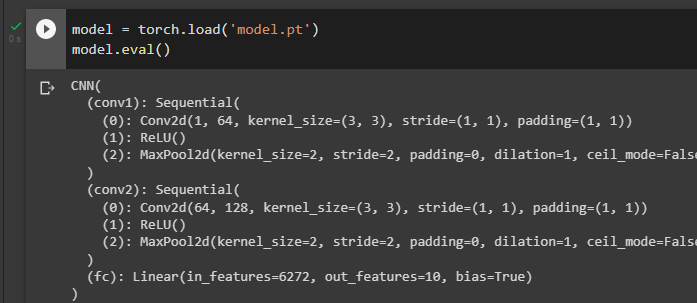
Esto es mas flexible para cuando ya tenemos un modelo ya entrenado y no queremos entrenar el modelo desde 0, y solo deseamos entrenar desde un punto de partida que hemos guardado. Sirve para aprovechar los entrenamientos ya preparados.

Alternativamente, Pytorch nos permite guardar el modelo entero, y no solo el *state\_dict*, de la siguiente manera.



De esta forma no guarda solo el state, si no que guarda todo el modelo.

Es el archivo model.pt que se ha guardado como resultado.

Y podemos cargar y evaluar nuestro modelo así:

Esto nos devuelve el modelo entero sin la necesidad de instanciarlo.

Si bien de esta forma no necesitamos que nuestro modelo esté instanciado, seguimos necesitando su definición. Es por este motivo que la opción anterior es la recomendada, ya que es más eficiente (sólo guardamos los pesos) y también más flexible (podemos guardar otra información además del *state\_dict* de nuestro modelo). Esta opción es ideal para guardar y cargar modelos durante el entrenamiento del mismo, quizás incluso junto al estado del optimizador, de manera que podemos entrenar modelos a partir de estos checkpoints en lugar de empezar de cero cada vez. Otro ejemplo consistiría en guardar el *state\_dict* del modelo durante el entrenamiento solo cuando mejore una métrica determinada y cargar el mejor modelo al final del entrenamiento (que no tiene porqué coincidir con el último).

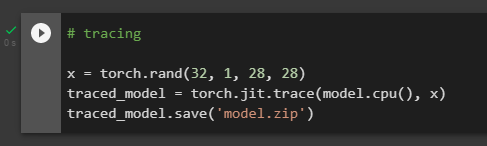
Ahora si queremos guardar el modelo para usarlo en otra aplicación, ya sea una aplicación web, una aplicación movil, etc. Porque para los dos anteriores es necesario tener Pytorch, y por ende, abrir el modelo desde phyton. Entonces no podremos usar el modelo. Para esto el siguiente comando nos puede ayudar.

**Exportar un modelo.**

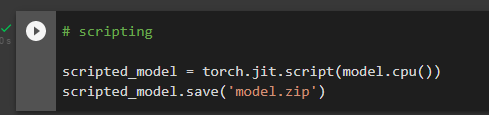
En pytorch podemos extraer un modelo de dos formas diferentes, la primera es utilizando torchscript y la segunda utilizando la libreria Onix.

Torchscript es una representación intermedia de un modelo de Pytorch que puede ejecutarse en diferentes entornos sin la necesidad de Python, por ejemplo en C++. Un modelo de Pytorch exportado en torchscript contiene los pesos de la red así como su definición (todas las operaciones que aplicaremos a un tensor desde la entrada hasta la salida). Tenemos dos maneras de exportar un modelo con torchscript:

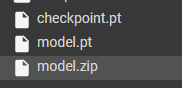
* Tracing: Dada un entrada, se genera una representación del modelo de manera dinámica registrando todas las operaciones aplicadas al tensor hasta la salida. En este caso no seremos capaces de capturar diferentes caminos en nuestra red (control flow). Es la alternativa más eficiente, pero menos flexible.



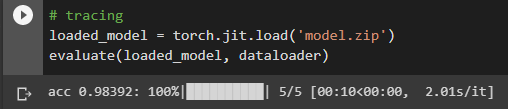
* Scripting: Genera la representación intermedia de nuestra red neuronal directamente a partir del análisis de la misma, siendo capaz de capturar de manera fiel cualquier ramificación en la misma. No es tan eficiente, pero si más flexible.



En los dos casos nos genera este archivo de tipo modelo:



Para cargar el modelo en tracing y en torchscript, usamos el siguiente comando y tendremos el modelo ya listo para realizar predicciones.

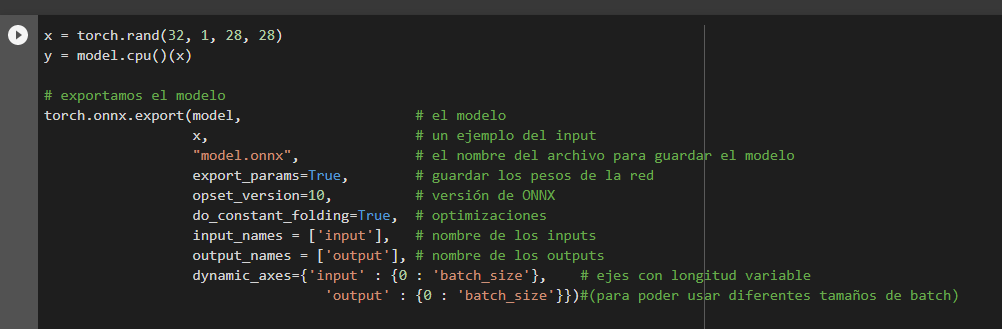


Exportar nuestro modelo nos aporta varias ventajas:

* Ahora nuestro modelo puede ejecutarse en cualquier entorno capaz de interpretar la representación intermedia generada por torchscript, independientemente del hardware o software utilizado.
* Nuestro modelo contiene los pesos y la definición de las operaciones, evitando tener que guardar código extra.
* Esta representación intermedia puede ser optimizada de manera independiente, haciendo que nuestros modelos sean más rápidos.
* La principal desventaja es que al estar "traduciendo" Python a otro lenguaje, es posible que no todas las operaciones que queramos hacer estén soportadas.

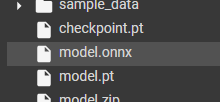
ONNX

Si bien ahora nuestro modelo es capaz de ser importado y ejecutado de forma más flexible y eficiente, seguimos limitados por el hecho de necesitar Pytorch (o la librería Torchscript en C++) para ello. Por este motivo, Pytorch nos ofrece una última manera de exportar nuestros modelos a otra forma de representación intermedia conocida como ONNX. Éste es un formato abierto con el espíritu de convertirse en un estándar de representación de redes neuronales.

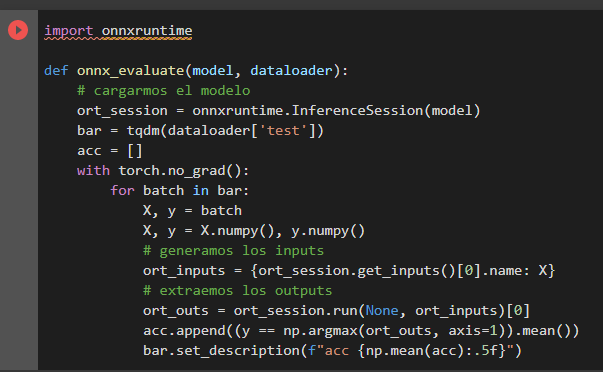
La gran mayoría de librerías y frameworks de deep learning soportan este formato, lo que implica que, por ejemplo, podemos entrenar un modelo en Pytorch, exportarlo en formato ONNX e importarlo en Tensorflow para ponerlo en producción (aunque ONNX también ofrece soluciones optimizadas para ello: ONNX Runtime). Así pues, exportar nuestros modelos a formato ONNX nos proporcionará la máxima flexibilidad, permitiéndonos, entre muchas otras cosas, ejecutar nuestras redes neuronales en entornos tales como navegadores web o IoT. Por contra, esta librería es la menos flexible en cuanto a cantidad y tipo de operaciones que podemos exportar, lo cual puede imponer unas restricciones muy grandes sobre nuestros modelos (aunque cada vez se soportan más).

Para poder ejecutar nuestro modelo necesitamos la librería de ONNX Runtime para Python.

Se genera este archivo .onnx en donde se almacena de manera general nuestro modelo.



Ahora el modelo puede realizar predicciones en cualquier lenguaje de programación, la forma de cargarlo en phyton es la siguiente.



# Conclusión

Las redes neuronales convolucionales son la opción ideal para la clasificación de imágenes puesto que funcionan mejor en datos (en lugar de usar redes neuronales densas normales) en los que existe una fuerte correlación entre, por ejemplo, píxeles porque el contexto espacial no se pierde.

Si en nuestro entorno de producción podemos usar Python e instalar Pytorch, entonces podemos simplemente guardar el state\_dict de nuestro modelo, o incluso el modelo completo, y luego cargarlo en la aplicación. Teniendo en cuenta que se necesitará la definición de la red neuronal que define las capas y operaciones que se aplican dada una entrada. Sin embargo, una opción más eficiente, consiste en exportar nuestro modelo con torch.jit.trace o torch.jit.script ya que luego es posible cargar nuestro modelo sin necesidad de arrastrar código y, además, podremos ejecutarlo en otros entornos como C++.

Para la próxima semana está planeado implementar una red neuronal convolucional y darle una aplicación que nos ayude a lograr clasificar objetos con una mayor precisión.