 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

# RECONOCIMIENTO DE SOMBRAS EN PANELES SOLARES CON IA

Jaiber Pino Acevedo

Sebastián Serna Vera

Tecnología en electrónica

Director del trabajo de grado Julián Peláez

**INSTITUTO TECNOLÓGICO METROPOLITANO**

**15 de Noviembre de 2019**

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

## RESUMEN

---

Las sombras en un panel solar representan pérdida de energía y a mayor tamaño de la sombra más pérdidas obtenemos, así que por medio de una plataforma de inteligencia artificial llamada Jetson Nano, la cual es un pequeño ordenador de gran alcance que nos permite ejecutar múltiples redes neuronales en paralelo para aplicarla en reconocimiento de imágenes, la utilizaremos para determinar qué porcentaje de sombra cubre un panel solar. La tarjeta es programada con Python y por medio de la cámara hará la identificación del sombreado. Es un valioso aporte, ya que la sombra no permite el buen desempeño del panel, y a gran escala sería más fácil identificar estas sombras con esta clase de dispositivos.

Esta plataforma debe ser entrenada con las imágenes de diferentes tipos de sombras y por medio de IA podrá hacer el reconocimiento de la imagen, en caso de que haya una sombra que no conozca, la tarjeta deberá hacer una inferencia, esto quiere decir que por medio de los conocimientos que adquirió arrojará el porcentaje de sombra más cercano al real. Para realizar esta tarea se debe conocer el funcionamiento de la plataforma y el lenguaje para programarla, además crear una base de datos y entrenar la plataforma para que haga la inferencia. Al examinar los resultados, después de que la tarjeta se entrenó con dos porcentajes de sombreado diferentes, se espera que al enseñarle alguna sombra similar nos arroje el porcentaje deseado, lo cual nos indicara que el dispositivo obtuvo un entrenamiento aceptable.

Con los resultados obtenidos se puede concluir que esta plataforma puede ser muy útil al momento de identificar sombras en un panel solar, lo cual sería útil para no tener pérdidas energéticas en una red alimentada por paneles solares.

*Palabras clave:* Inteligencia artificial, Jetson nano, Python, sombras, paneles solares, inferencia.

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

## RECONOCIMIENTOS

---

Agradecimientos al profesor Julián Peláez por compartir sus conocimientos y por introducirnos al mundo de la inteligencia artificial.

Agradecimiento al laboratorio de energías alternativas por permitirnos usar sus instalaciones y recursos para realizar este proyecto.

Agradecemos a los compañeros del laboratorio por asesorarnos en algunos casos específicos.

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

# ACRÓNIMOS

---

IA Inteligencia Artificial

GPU Unidad de procesamiento grafico

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

## TABLA DE CONTENIDO

1. INTRODUCCIÓN .....	7
2. MARCO TEÓRICO .....	8
3. METODOLOGÍA.....	15
4. RESULTADOS Y DISCUSIÓN.....	27
5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO .....	30
REFERENCIAS .....	33
APÉNDICE.....	34

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

# 1. INTRODUCCIÓN

En las redes eléctricas alimentadas por paneles solares uno de los aspectos que interfieren con el funcionamiento adecuado de los mismos es la sombra, y a mayor área que cubra la sombra, mayor es la pérdida de energía y baja la eficacia del este.

Con la ayuda de la tarjeta Jetson nano, la cual fue diseñada para IA, y por medio de Python, más el conocimiento adquirido en machine learning, entendiendo como una computadora puede entrenarse y podrá hacer inferencia reconociendo imágenes; vamos a utilizar redes neuronales para entrenar la tarjeta, y así poder identificar qué porcentaje de sombra está cubriendo al panel solar afectando su eficacia.

Crearemos una base de datos con diferentes clases de sombras, con la cual se entrenará la Jetson Nano, esto le dará la capacidad de inferir que porcentaje de sombra está afectando el panel.

Se harán pruebas que constatarán que la tarjeta está reconociendo las áreas sombradas obteniendo resultados deseados.

Obtuvimos nuevos conocimientos en el área de la inteligencia artificial, abordamos una plataforma relativamente nueva, tratamos de entender como aprendía a identificar diferentes clases de imágenes para poder enseñarle como distinguir el porcentaje de sombra en un panel, para obtener mejores resultados se investigó más sobre la manera de entrenarla, esto con el objetivo de tener más precisión.

En conclusión, la Jetson Nano es una poderosa herramienta, que al obtener el conocimiento de su manejo, es muy útil para resolver problemas donde se necesita IA, en nuestro caso cumplió a la hora de identificar el área sombreada.

## 2. MARCO TEÓRICO

---

### Tarjeta de desarrollo Jetson Nano

La plataforma para desarrolladores NVIDIA Jetson Nano es una pequeña computadora dotada con inteligencia artificial, la cual permite al usuario construir aplicaciones prácticas de inteligencia artificial. (Nvidia Corporation, 2019)

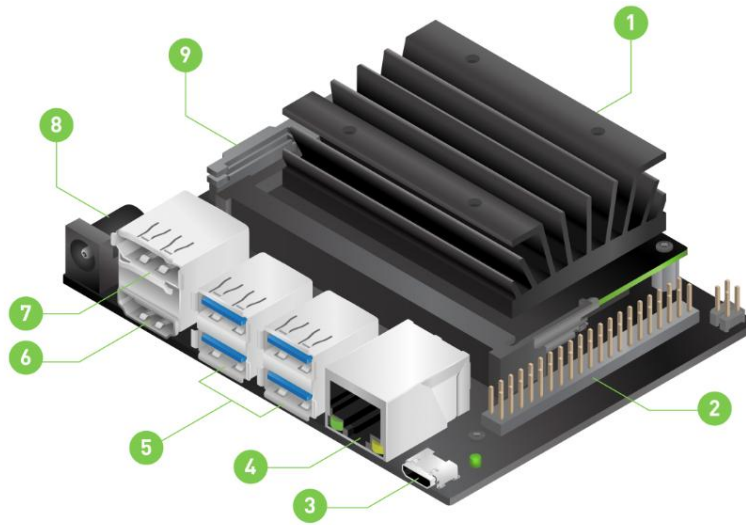


Figura 1. Jetson Nano y sus puertos. Fuente: Tomado <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit> consultada 7/11/2019

1. Ranura para tarjeta SD
2. Expansión de 40 pines
3. Puerto micro USB
4. Puerto Gigabit Ethernet
5. Puertos USB
6. Salida HDMI
7. DisplayPort
8. Conector DC
9. Conector de la cámara



	<b>INFORME FINAL DE TRABAJO DE GRADO</b>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

El dispositivo, mucho más potente que el Mini PC más popular hoy en día, la Raspberry Pi, no presenta sin embargo un consumo energético mayor que ésta, lo que facilitará su integración en robots y dispositivos para hogares inteligentes.

Pequeña en cuanto a tamaño (69,6 x 45 mm), lo que no le impide albergar una CPU ARM Cortex-A57 MPCore de 4 núcleos (capaz de proporcionarnos 472 gigaflops de potencia), una GPU Nvidia Maxwell con 128 núcleos CUDA (capaz de ejecutar la librería de procesamiento de datos CUDA-X AI), 4 Gb de RAM, 16 GB de almacenamiento y 4 puertos USB 3.0.

En cuanto al software, cabe destacar que Jetson Nano es compatible con los frameworks de IA más populares del mercado: TensorFlow, PyTorch, Caffe, Keras, MXNet, etc. Además, la suite de simulación Isaac Sim (incluida en el SDK Isaac proporcionado por Nvidia) tiene el objetivo de proporcionar un entorno de entrenamiento para máquinas autónomas. (Nvidia Corporation, 2019)

### **Machine learning usando Python**

Machine Learning es un tipo de inteligencia artificial (IA), que proporciona a las computadoras la capacidad de aprender sin ser programadas explícitamente. El aprendizaje de forma automática, se centra en el desarrollo de programas informáticos que pueden cambiar cuando se les presenta un nuevo dato.

Machine Learning implica que la computadora aprenda utilizando un conjunto de datos ya determinado, y use este aprendizaje para predecir las propiedades de un nuevo dato dado. Un ejemplo sería entrenar la computadora con 1000 imágenes de gatos y mil que no sean gatos, y enseñarle si la imagen es de un gato o no. Después de este proceso, al enseñarle

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

una imagen nueva la computadora podrá saber si es gato o no haciendo este proceso por inferencia.

Haremos un ejemplo para tener más claridad respecto al tema: Usaremos un conjunto de datos de flores de iris para entrenar la computadora, y luego le damos un nuevo valor a la computadora para hacer la inferencia. El conjunto de datos consta de 50 muestras de cada una de las tres especies de Iris (Iris, cetosa, Iris virginica e Iris versicolor). Se miden cuatro características de cada muestra: la longitud y el ancho de sépalos y pétalos, en centímetros. Entrenamos nuestro programa usando este conjunto de datos, y luego usamos este entrenamiento para predecir las especies de una flor de iris con las mediciones dadas.

Código de Python para clasificación KNN (K-neares-neighbor):

```

1 # Python program to demonstrate
2 # KNN classification algorithm
3 # on IRIS dataset
4
5 from sklearn.datasets import load_iris
6 from sklearn.neighbors import KNeighborsClassifier
7 import numpy as np
8 from sklearn.model_selection import train_test_split
9
10 iris_dataset=load_iris()
11
12 X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset["target"], random_state=0)
13
14 kn = KNeighborsClassifier(n_neighbors=1)
15 kn.fit(X_train, y_train)
16
17 x_new = np.array([[5, 2.9, 1, 0.2]])
18 prediction = kn.predict(x_new)
19
20 print("Predicted target value: {}\n".format(prediction))
21 print("Predicted feature name: {}\n".format
22       (iris_dataset["target_names"][prediction]))
23 print("Test score: {:.2f}".format(kn.score(X_test, y_test)))

```

Figura 2. Código de Python para clasificación. Fuente: Tomado de Python Data Science Handbook (pág. 541). consultada 14/11/2019

Toma datos de prueba y encuentra K valores de datos más cercanos a estos datos del conjunto de datos de prueba. Luego selecciona el vecino más cercano y da como resultado el que tenga propiedades similares al dato ingresado. (VanderPlas, 2016)

	<b>INFORME FINAL DE TRABAJO DE GRADO</b>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

### Entrenando el conjunto de datos

- La primera línea importa el conjunto de datos de iris que ya está predefinido en el módulo sklearn. El conjunto de datos de iris es básicamente una tabla que contiene información sobre diversas variedades de flores de iris.
- Importamos el algoritmo `KNeighborsClassifier` y la clase `train_test_split` de sklearn y el módulo `numpy` para usar en este programa.
- Luego encapsulamos el método `load_iris ()` en la variable `iris_dataset`. Además, dividimos el conjunto de datos en datos de entrenamiento y datos de prueba usando el método `train_test_split`. El prefijo `X` en la variable indica los valores de las características (por ejemplo, longitud del pétalo, etc.) y el prefijo `y` indica los valores objetivo (por ejemplo, 0 para *setosa*, 1 para *virginica* y 2 para *versicolor*).
- Este método divide el conjunto de datos en datos de entrenamiento y prueba al azar en una proporción de 75:25. Luego encapsulamos el método `KNeighborsClassifier` en la variable `kn` mientras mantenemos el valor de `k = 1`. Este método contiene el algoritmo `K Nearest Neighbor` en él.
- En la siguiente línea, ajustamos nuestros datos de entrenamiento en este algoritmo para que la computadora pueda entrenarse usando estos datos. Ahora la parte de entrenamiento está completa. (VanderPlas, 2016)

### Probar el conjunto de datos

- Ahora tenemos las dimensiones de una nueva flor en una matriz `numpy` llamada `x_new` y queremos predecir la especie de esta flor. Hacemos esto usando el método de predicción que toma esta matriz como entrada y devuelve el valor objetivo previsto como salida.
- Entonces, el valor objetivo previsto resulta ser 0, que significa *setosa*. Entonces esta flor tiene buenas posibilidades de ser de especies de *setosa*.
- Finalmente encontramos el puntaje de la prueba, que es la relación del no. de predicciones encontradas predicciones correctas y totales hechas. Hacemos esto

	<b>INFORME FINAL DE TRABAJO DE GRADO</b>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

usando el método de puntaje que básicamente compara los valores reales del conjunto de pruebas con los valores predichos. (VanderPlas, 2016)

### **TensorRT**

TensorRT es una plataforma para inferencia de aprendizaje profundo de alto rendimiento. Incluye un optimizador de inferencia de aprendizaje profundo y tiempo de ejecución que ofrece baja latencia y alto rendimiento para aplicaciones de inferencia de aprendizaje profundo. Las aplicaciones basadas en TensorRT funcionan hasta 40 veces más rápido que las plataformas solo de CPU durante la inferencia. Con TensorRT, puede optimizar modelos de redes neuronales capacitados en todos los marcos principales, calibrar para una precisión más baja con alta precisión y finalmente implementar en centros de datos de hiperescala, plataformas integradas o de productos automotrices. (Nvidia Corporation, 2019)

### **ONNX**

ONNX es un formato abierto de intercambio de redes neuronales, su objetivo será que los desarrolladores de IA puedan mover más fácilmente sus modelos entre herramientas de Deep Learning y elegir cual combinación sería la mejor (figura 18). La idea es ofrecer a los usuarios la libertad de innovar y poder transferir los modelos fácilmente.

### **Inteligencia artificial (IA)**

La idea de construir una máquina que pueda realizar tareas que requieran de inteligencia humana es uno de los objetivos de la IA, ya que esta pretende que estas máquinas no solo realicen la tarea con inteligencia, sino que también aprendan o adquieran conocimientos, que sean capaces de realizar tareas que normalmente requieren de inteligencia humana.

La computadora más simple puede superar ampliamente a la persona más inteligente cuando se trata de algo como una ecuación matemática complicada. Sin embargo, las computadoras más poderosas del mundo, fallaban realizando tareas que los humanos consideran triviales, como reconocer rostros, identificar objetos en imágenes, entre otros,

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

tomando una definición más sencilla, la IA es la simulación de procesos de inteligencia humana por parte de máquinas, especialmente sistemas informáticos como la Jetson Nano. Estos procesos incluyen el aprendizaje (la adquisición de información y reglas para el uso de la información), el razonamiento (usando las reglas para llegar a conclusiones aproximadas o definitivas) y la autocorrección. (alvarez, 2018)

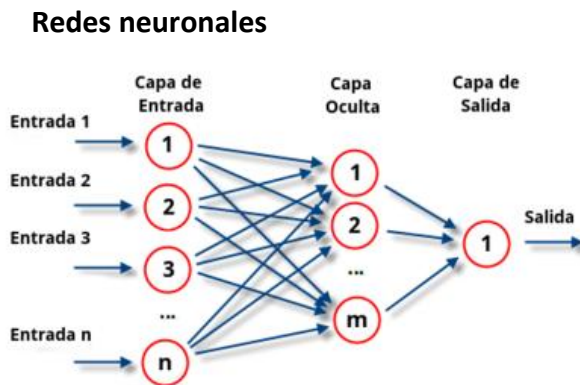


Figura 3. Capas de las redes neuronales Fuente: Tomado de <http://agentemello007.blogspot.com/2011/09/informe-sobre-redes-neuronales.html> consultada 9/11/2019

Las Redes Neuronales son un campo muy importante dentro de la Inteligencia Artificial (IA). Inspirándose en el comportamiento conocido del cerebro humano (principalmente el referido a las neuronas y sus conexiones), trata de crear modelos artificiales que solucionen problemas difíciles de resolver mediante técnicas algorítmicas convencionales.

Estas redes intercambian información entre las neuronas lo cual después de ingresar un dato y pasar por varias capas arroja un resultado. Por ejemplo, al ingresar una imagen desconocida para maquina a la red neuronal, debe pasar por estas capas, y a la salida podrá inferir que imagen se le ingreso.

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

### **Efectos de la sombra en un panel solar**

Siempre se debe evitar el sombreado en el panel solar ya que la generación de energía eléctrica se ve afectada de forma drástica, la existencia de sombras sobre parte de un panel fotovoltaico produce la entrada en funcionamiento de los diodos “Bypass” de la caja de conexiones y esto provoca escalones en la curva I-V del mismo, es decir, su producción se reduce.

Si la sombra es de gran tamaño cierta parte del panel se convertirá en una carga, lo que generará calentamiento o simplemente el panel dejará de funcionar, por lo tanto, la sombra siempre será poco deseada en redes de paneles solares.

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

### 3. METODOLOGÍA

A partir de la necesidad de identificar si alguna indeseada sombra cubría el panel solar en algún lapso del día, y poder definir qué porcentaje de área que esta cubría, y con la nueva tecnología en tarjetas de inteligencia artificial y conocimientos adquiridos nos dimos a la tarea de identificar las sombras con la ayuda de IA y la Jetson Nano.

En el laboratorio de energías renovables y con la asesoría del profesor Julián Peláez, ayuda de los demás compañeros del laboratorio, con la plataforma de desarrollo Jetson Nano suministrada por el profesor y algunos recursos extras, empezamos este proyecto.

Al comienzo lo primordial era conocer las características, funcionamiento y manipulación de la Jetson Nano ya que esta plataforma apenas fue lanzada al mercado en marzo del 2019 y comenzó a enviarse en junio del 2019 (Perez, 2019), lo cual hace, que haya muy poca información, así que para empezar a interactuar con ella nos basamos en el página web de Nvidia, los cuales son sus desarrolladores.

La tarjeta viene en blanco y necesitamos una tarjeta SD mínimo de 16 Gb, para nuestro proyecto usamos una de 64 Gb, así que lo primero es instalar el sistema operativo en la tarjeta SD. Para esta tarea necesitamos un PC, descargamos y ejecutamos la aplicación Etcher, este nos instalará Linux en nuestra tarjeta. Retiramos la tarjeta SD del PC y la ingresamos a Jetson nano, creamos usuario y contraseña y ya con esto estará lista para usarse en él proyecto. (Nvidia, corporation, 2019)

Ahora configuramos la Jetson Nano con el Jetpack, lo cual instala una serie de útiles herramientas y simplificara la instalación de Linux y sus controladores, este paquete contiene:

- Kernel L4T
- Kit de herramientas CUDA
- cuDNN
- TensorRT
- OpenCV

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

- VisionWorks

Ya con el sistema operativo y las herramientas instaladas, clonamos el Repo el cual nos creara carpetas con paquetes y librerías de Python, aunque en la tarjeta ya descargamos con anterioridad Python 2.7 el cual estaba incluido en el sistema operativo. Estas herramientas serán de gran utilidad para hacer reconocimiento de imágenes, así que las usaremos luego para entrenar la tarjeta.

Ahora debemos descargar unos modelos predeterminados, los cuales usaremos para recrear algunos ejemplos y entender más cómo funciona la Jetson Nano. Según la necesidad se selecciona el modelo, en nuestro caso el modelo que nos interesa es el de recognition, ya que este es usado para reconocer imágenes, igualmente exploramos los otros modelos con el fin de poder hallar más posibles soluciones.

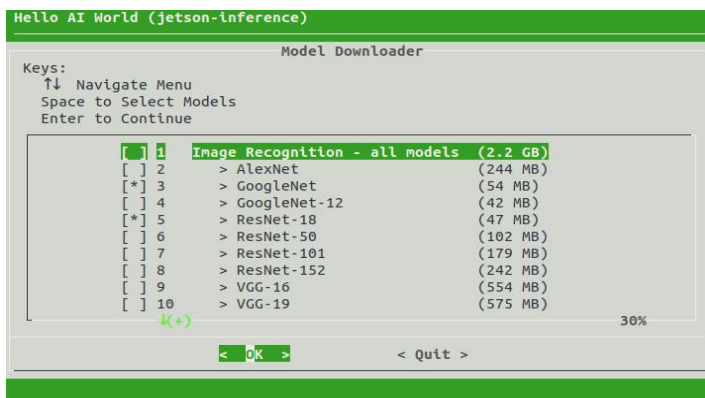


Figura 4. Tabla de modelos predeterminados. Fuente: Tomado <https://github.com/dusty-nv/jetson-inference/blob/master/docs/building-repo-2.md> consultada 14/11/2019

El modelo le dice a la tarjeta que entradas y salidas posibles tendrá nuestra red neuronal, para nuestro caso debemos crear nuestro propio modelo ya que las posibles salidas serán pocas a comparación de estos grandes modelos predeterminados.

Ahora debemos instalar PyTorch, el cual es una importante herramienta a la hora de entrenar redes con transfer learning, debemos prestar atención y seleccionar la versión de Python que estamos usando para evitar futuros errores, en nuestro caso usamos Python 2.7, luego creamos enlaces de extensión de Python y otras bibliotecas.

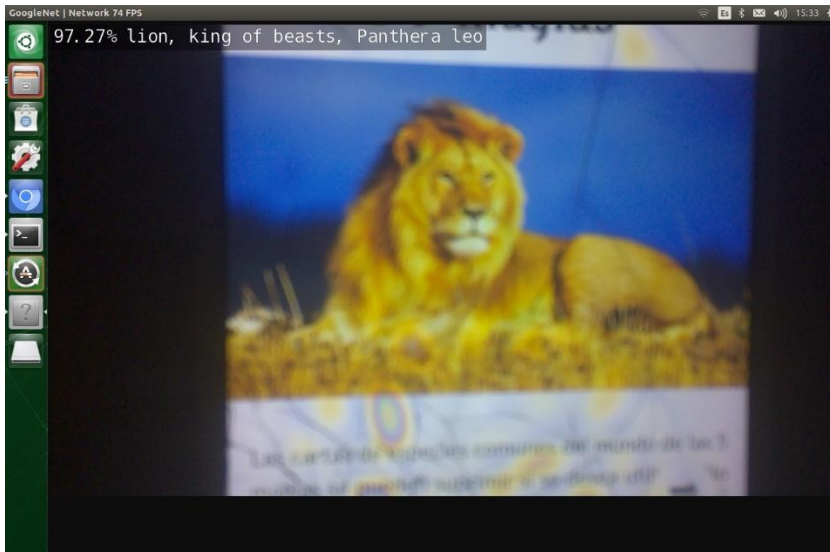


	<b>INFORME FINAL DE TRABAJO DE GRADO</b>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

En la carpeta de Python podremos ver el código que se encargara de entrenar la tarjeta.  
(Franklin, 2019)

### **Ejemplos de reconocimiento de imágenes**

Ahora creamos nuestro programa de reconocimiento de imágenes con un ejemplo paso a paso para poder identificar cómo funciona el programa y saber dónde tendremos que modificar el código de Python para poder usarlo en la identificación de las sombras en el panel, en este ejemplo podremos reconocer 1000 diferentes objetos, ya que los modelos de clasificación están entrenados en el conjunto de datos ILSVRC ImageNet que contiene 1000 clases de objetos. La asignación de nombres para los 1000 tipos de objetos, se pueden encontrarla en el repositorio. (Franklin, github, 2019)



*Figura 5 Probando programa de reconocimiento de imágenes.*

El ejemplo nos muestra como reconoce un león con alta eficiencia, por lo tanto, nos queda claro que el modelo predeterminado de imagenet identifica imágenes fácilmente.

También podemos identificar la imagen por medio de la cámara en tiempo real, le mostramos la imagen a la tarjeta y esta deberá identificarla.

	<b>INFORME FINAL DE TRABAJO DE GRADO</b>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

### **Transferir aprendizaje con Pytorch**

Ahora utilizaremos aprendizaje por transferencia la cual es una técnica para volver a entrenar un modelo DNN con un nuevo conjunto de datos, y esto llevara menos tiempo que entrenar la red desde cero.

Con el aprendizaje por transferencia, los pesos de un modelo previamente entrenado se ajustan para clasificar un conjunto de datos personalizado. Para nuestra red, utilizaremos la red ResNet-18.

### **Creando nuestro propio conjunto de datos para reconocer 2 clases de sombra**

Para poder hacer inferencia con la Jetson, necesitamos crear nuestro propio conjunto de datos, y entrenar la red con estas imágenes. Lo primero es tomar medidas del panel solar y definir cuanto será una sombra del 10% y cuanto será una sombra del 20%, luego crearemos figuras con esa misma área y tomaremos fotografías del panel con las sombras en diferentes posiciones, para obtener un entrenamiento adecuado tomamos 430 fotos con diferentes figuras del 10% del área del panel y 489 fotos con el 20%, las cuales harán sombra sobre el panel solar en diferentes posiciones. A continuación, se muestran algunas figuras usadas para hacer la sombra sobre el panel.

Imágenes con el 10% del área sombreada



*Figura 6. Sombra 10 %*



*Figura 7. Sombra 10 %*



*Figura 8. Sombra 10 %*

	<p>INFORME FINAL DE TRABAJO DE GRADO</p>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27



Figura 9. Sombra 10 %

Imágenes con el 20% del área sombreada

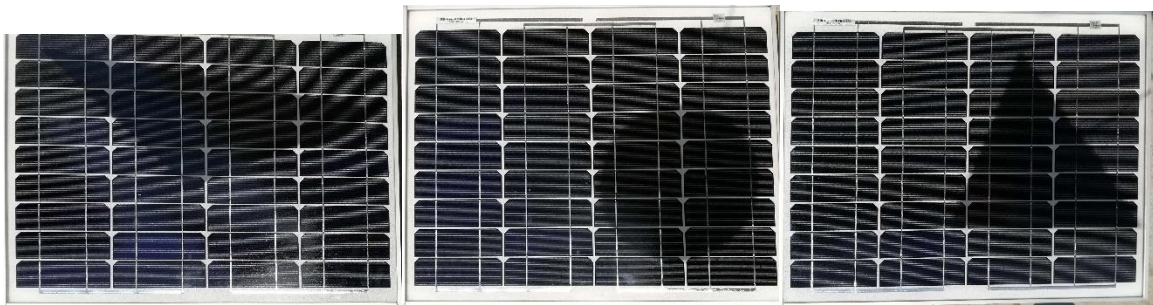


Figura 10. Sombra 20 %

Figura 11. Sombra 20 %

Figura 12. Sombra 20 %



Figura 13. Sombra 20 %

Figura 14. Sombra 20 %

Ya con las imágenes crearemos una carpeta la cual será nuestro Dataset esta carpeta la llamamos panel, en su interior crearemos otras tres carpetas, las cuales serán:

**Train:** en esta carpeta ingresaremos todas las imágenes de entrenamiento con las cuales la tarjeta aprenderá a inferir. Estarán separadas en 2 carpetas las cuales llamamos: sombra 10%, sombra 20%.

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

**Val:** En esta carpeta la tarjeta hará validación de lo entrenado, allí ingresaremos algunas imágenes de sombras al azar para así la Jetson poder comparar con las imágenes de entreno.

**Test:** En esta carpeta ingresamos las imágenes para luego del entrenamiento, corroborar si la tarjeta quedo bien entrenada y verificar si hace bien la inferencia según la imagen de testeo.

Ahora creamos otro archivo tipo txt, en el cual ingresaremos las clases de las imágenes que obtendremos a la salida, en nuestro caso serán sombra del 10% y sombra del 20%, en otras palabras, vinculamos los nombres según la imagen de salida.

### Comienza el entrenamiento

Ya con nuestro Dataset listo, empezamos el entrenamiento. Antes de comenzar debemos cambiar en el código de Python el learning rate, una de las tareas más arduas de entrenar una red neuronal profunda es elegir la tasa de aprendizaje (LR). A menudo esto se hace al tanteo. Elegimos un valor y vemos cómo funciona. Repetimos hasta que obtengamos buenos resultados, esta es una tarea tediosa, ya que cada vez que cambiamos el valor, debemos volver a entrenarla y esto tarda 5 horas aproximadamente. Es difícil saber si los resultados subóptimos podrían mejorarse aumentando el número de épocas o cambiando el learning rate, ya que, si asignamos un valor muy alto o muy bajo, la tarjeta no hará bien la inferencia.

Como podemos observar en la figura 15, el valor que se utiliza debe estar en la pendiente, o sea cerca de cero. En nuestro caso después de intentar con diferentes valores, el que arrojó los mejores resultados fue cuando utilizamos un learning rate de 0.01, así que modificamos este valor en el código de python.

En esta línea de código cambiamos el learning rate:

```
parser.add_argument('--lr', '--learning-rate', default=0.01, type=float,
```

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

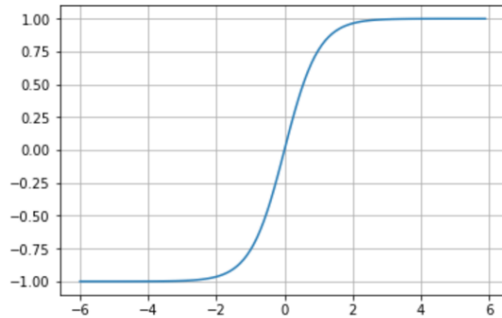


Figura 15. Gráfica de aprendizaje según learning rate seleccionado.

También reasignamos el valor a las épocas que por defecto tiene asignado 35, (las épocas son las veces que la red interactúa con los datos), ya que una buena cantidad de estas garantiza una mayor precisión en los resultados. Como podemos observar en la figura 16, en la época 60 o más, la precisión llega a una estabilidad, así que, en nuestro proyecto la entrenamos con 70 épocas para obtener mejores resultados. Para modificar este valor en código de Python con el cual hace la inferencia en la línea del número de épocas, le reasignamos el nuevo valor.

En esta línea configuramos las épocas de entrenamiento:

```
parser.add_argument('--epochs', default=70, type=int, metavar='N',
                    help='number of total epochs to run')
```

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

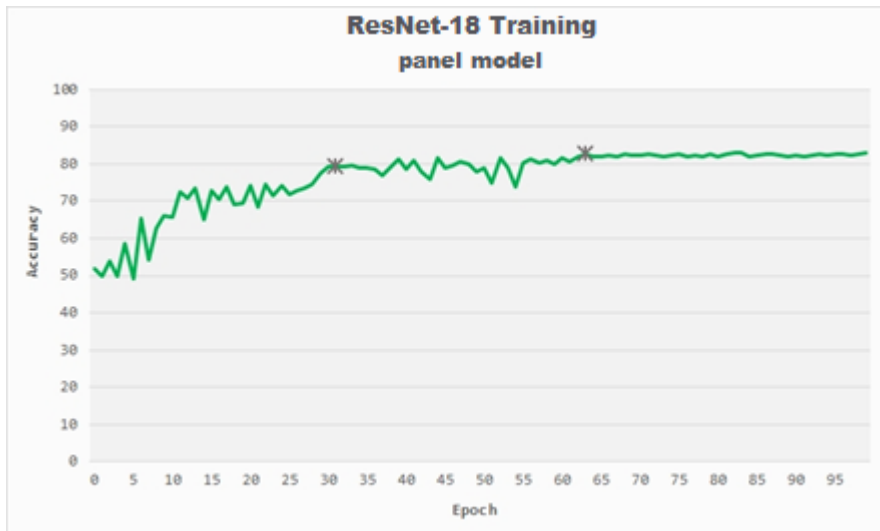


Figura 16. Grafica de precisión vs épocas.

Cuando se va a realizar el entrenamiento, con el código de Python, este tiene un valor predeterminado para tamaño del lote de 8, pero para mejor inferencia cogeremos por lotes más grande de 10 imágenes, así que también modificamos esta línea en el código:

```
parser.add_argument('-b', '--batch-size', default=10, type=int,
                    metavar='N',
                    help='mini-batch size (default: 10), this is the total '
```

Una vez configurado lo anterior en el código de Python, usamos los siguientes comandos para que la tarjeta comience con el entrenamiento:

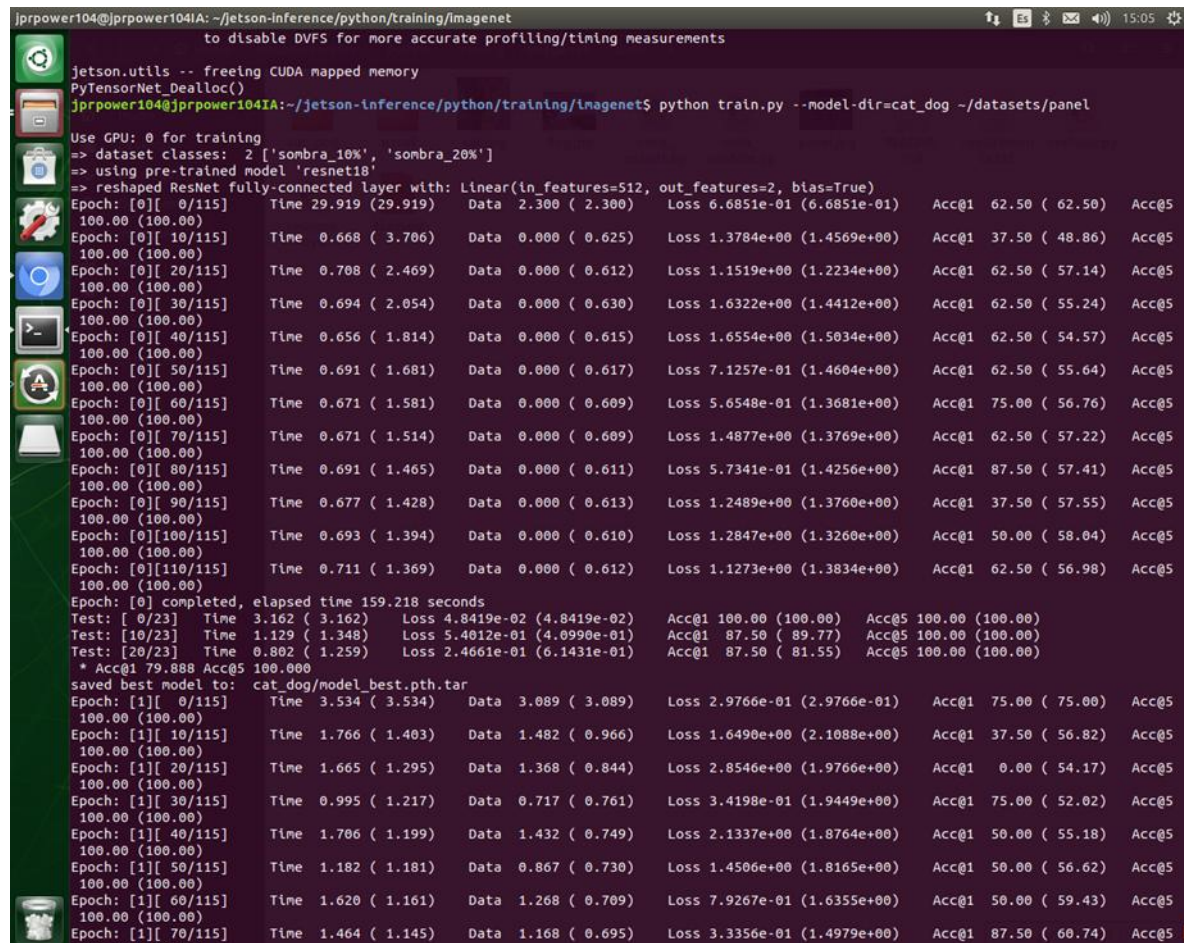
En la primera línea ingresamos a la carpeta donde está el código de Python el cual es el encargado del aprendizaje de la tarjeta, luego en la segunda línea creamos nuestro modelo que se llamará panel y será entrenado con nuestra base de datos también llamada panel.

```
$ cd jetson-inference / python / training / imagenet
$ python train.py --model-dir = panel ~/ datasets / panel
```



 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

A medida que comienza el aprendizaje en la consola nos muestra la siguiente imagen:



```

jprpower104@jprpower104IA: ~/jetson-inference/python/training/imagenet
to disable DVFS for more accurate profiling/timing measurements

jetson.utils -- freeing CUDA mapped memory
PyTensorNet Dealloc()
jprpower104@jprpower104IA:~/jetson-inference/python/training/imagenet$ python train.py --model-dir=cat_dog ~/datasets/panel

Use GPU: 0 for training
=> dataset classes: 2 ['sombra_10%', 'sombra_20%']
=> using pre-trained model 'resnet18'
=> reshaped ResNet fully-connected layer with: Linear(in_features=512, out_features=2, bias=True)

Epoch: [0][ 0/115]   Time 29.919 (29.919)   Data 2.300 ( 2.300)   Loss 6.6851e-01 (6.6851e-01)   Acc@1 62.50 ( 62.50)   Acc@5
100.00 (100.00)
Epoch: [0][ 10/115]  Time 0.668 ( 3.706)   Data 0.000 ( 0.625)   Loss 1.3784e+00 (1.4569e+00)   Acc@1 37.50 ( 48.86)   Acc@5
100.00 (100.00)
Epoch: [0][ 20/115]  Time 0.708 ( 2.469)   Data 0.000 ( 0.612)   Loss 1.1519e+00 (1.2234e+00)   Acc@1 62.50 ( 57.14)   Acc@5
100.00 (100.00)
Epoch: [0][ 30/115]  Time 0.694 ( 2.054)   Data 0.000 ( 0.630)   Loss 1.6322e+00 (1.4412e+00)   Acc@1 62.50 ( 55.24)   Acc@5
100.00 (100.00)
Epoch: [0][ 40/115]  Time 0.656 ( 1.814)   Data 0.000 ( 0.615)   Loss 1.6554e+00 (1.5034e+00)   Acc@1 62.50 ( 54.57)   Acc@5
100.00 (100.00)
Epoch: [0][ 50/115]  Time 0.691 ( 1.681)   Data 0.000 ( 0.617)   Loss 7.1257e-01 (1.4604e+00)   Acc@1 62.50 ( 55.64)   Acc@5
100.00 (100.00)
Epoch: [0][ 60/115]  Time 0.671 ( 1.581)   Data 0.000 ( 0.609)   Loss 5.6548e-01 (1.3681e+00)   Acc@1 75.00 ( 56.76)   Acc@5
100.00 (100.00)
Epoch: [0][ 70/115]  Time 0.671 ( 1.514)   Data 0.000 ( 0.609)   Loss 1.4877e+00 (1.3769e+00)   Acc@1 62.50 ( 57.22)   Acc@5
100.00 (100.00)
Epoch: [0][ 80/115]  Time 0.691 ( 1.465)   Data 0.000 ( 0.611)   Loss 5.7341e-01 (1.4256e+00)   Acc@1 87.50 ( 57.41)   Acc@5
100.00 (100.00)
Epoch: [0][ 90/115]  Time 0.677 ( 1.428)   Data 0.000 ( 0.613)   Loss 1.2489e+00 (1.3760e+00)   Acc@1 37.50 ( 57.55)   Acc@5
100.00 (100.00)
Epoch: [0][100/115]  Time 0.693 ( 1.394)   Data 0.000 ( 0.610)   Loss 1.2847e+00 (1.3260e+00)   Acc@1 50.00 ( 58.04)   Acc@5
100.00 (100.00)
Epoch: [0][110/115]  Time 0.711 ( 1.369)   Data 0.000 ( 0.612)   Loss 1.1273e+00 (1.3834e+00)   Acc@1 62.50 ( 56.98)   Acc@5
100.00 (100.00)
Epoch: [0] completed, elapsed time 159.218 seconds
Test: [ 0/23]   Time 3.162 ( 3.162)   Loss 4.8419e-02 (4.8419e-02)   Acc@1 100.00 (100.00)   Acc@5 100.00 (100.00)
Test: [10/23]  Time 1.129 ( 1.348)   Loss 5.4012e-01 (4.0990e-01)   Acc@1 87.50 ( 89.77)   Acc@5 100.00 (100.00)
Test: [20/23]  Time 0.802 ( 1.259)   Loss 2.4661e-01 (6.1431e-01)   Acc@1 87.50 ( 81.55)   Acc@5 100.00 (100.00)
* Acc@1 79.888 Acc@5 100.000
saved best model to: cat_dog/model_best.pth.tar
Epoch: [1][ 0/115]   Time 3.534 ( 3.534)   Data 3.089 ( 3.089)   Loss 2.9766e-01 (2.9766e-01)   Acc@1 75.00 ( 75.00)   Acc@5
100.00 (100.00)
Epoch: [1][ 10/115]  Time 1.766 ( 1.403)   Data 1.482 ( 0.966)   Loss 1.6490e+00 (2.1088e+00)   Acc@1 37.50 ( 56.82)   Acc@5
100.00 (100.00)
Epoch: [1][ 20/115]  Time 1.665 ( 1.295)   Data 1.368 ( 0.844)   Loss 2.8546e+00 (1.9766e+00)   Acc@1 0.00 ( 54.17)   Acc@5
100.00 (100.00)
Epoch: [1][ 30/115]  Time 0.995 ( 1.217)   Data 0.717 ( 0.761)   Loss 3.4198e-01 (1.9449e+00)   Acc@1 75.00 ( 52.02)   Acc@5
100.00 (100.00)
Epoch: [1][ 40/115]  Time 1.706 ( 1.199)   Data 1.432 ( 0.749)   Loss 2.1337e+00 (1.8764e+00)   Acc@1 50.00 ( 55.18)   Acc@5
100.00 (100.00)
Epoch: [1][ 50/115]  Time 1.182 ( 1.181)   Data 0.867 ( 0.730)   Loss 1.4506e+00 (1.8165e+00)   Acc@1 50.00 ( 56.62)   Acc@5
100.00 (100.00)
Epoch: [1][ 60/115]  Time 1.620 ( 1.161)   Data 1.268 ( 0.709)   Loss 7.9267e-01 (1.6355e+00)   Acc@1 50.00 ( 59.43)   Acc@5
100.00 (100.00)
Epoch: [1][ 70/115]  Time 1.464 ( 1.145)   Data 1.168 ( 0.695)   Loss 3.3356e-01 (1.4979e+00)   Acc@1 87.50 ( 60.74)   Acc@5
100.00 (100.00)

```

Figura 17. Entrenamiento de nuestra red neuronal.

La tabla nos informa en cual época del entrenamiento se encuentra (epochs), las imágenes se procesan por lotes para mejorar el rendimiento lo podemos ver en [10/115], dependiendo la cantidad de datos seleccionamos el lote, en nuestro caso tomamos lotes de 10 imágenes así que vemos como toma lotes de 10 en 10, en la tabla también muestra la diferencia entre el valor predicho por su modelo y el valor verdadero (loss) y la precisión (Acc), la idea es que el loss sea cada vez menor y la precisión vaya subiendo cada época, este proceso de entrenamiento dura aproximadamente 5 horas.

	<b>INFORME FINAL DE TRABAJO DE GRADO</b>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

En esta tabla constatamos si el learning rate si fue el más adecuado para el entrenamiento, porque mientras entrena nos muestra la precisión y el loss.

Si termina de entrenar y la precisión fue en aumento y el loss en decrecimiento significa que ya está listo nuestro modelo en PyTorch, ahora debemos convertirlo a formato ONNX para que TensorRT pueda cargarlo. ONNX es un formato modelo abierto que admite PyTorch lo cual permite la transferencia de modelos entre herramientas. (Gill, 2019)

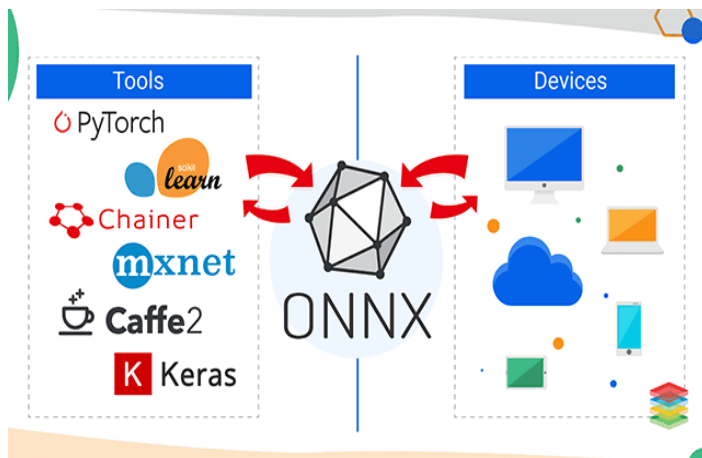


Figura 18. Exportación a ONNX Fuente: Tomado <https://www.xenonstack.com/blog/artificial-neural-network-applications/> consultada 8/11/2019

Para exportarlo a ONNX usamos el siguiente comando:

```
python onnx_export.py --model-dir = panel
```

Esto crea un modelo llamado resnet18.onnx en la carpeta de panel y debido a esto podemos hacer la inferencia usando tensorRT.

### Realizamos las pruebas

Ahora vamos a realizar las pruebas para comprobar si el entrenamiento fue exitoso.

Con el siguiente comando llamamos una imagen de la carpeta de test y con el modelo panel que fue el que creamos, podremos constatar si hace bien la inferencia.



 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

```
imagenet-console.py --model = panel / resnet18.onnx --input_blob = input_0 --
output_blob = output_0 --labels = $ DATASET /labels.txt $ DATASET
/test/sombra_10%/01.jpg
```

```
jprpower104@jprpower104IA:~/jetson-inference/python/training/imagenet$ imagenet-console.py --model=panel/resnet18.onnx --input_blob=i
input_0 --output_blob=output_0 --labels=$DATASET/labels.txt $DATASET/test/sombra_10%/01.jpg
jetson.inference.__init__.py
jetson.inference -- initializing Python 2.7 bindings...
jetson.inference -- registering module types...

[TRT] binding to input 0 input_0 binding index: 0
[TRT] binding to input 0 input_0 dims (b=1 c=3 h=224 w=224) size=602112
[TRT] binding to output 0 output_0 binding index: 1
[TRT] binding to output 0 output_0 dims (b=1 c=2 h=1 w=1) size=8
device GPU, panel/resnet18.onnx initialized.
[TRT] panel/resnet18.onnx loaded
imageNet -- loaded 2 class info entries
panel/resnet18.onnx initialized.
)class 0000 - 0.881579 (sombra_10%
class 0001 - 0.118421 (sombra_20%)
' (class #0) with 88.157922% confidence

[TRT] -----
[TRT] Timing Report panel/resnet18.onnx
[TRT] -----
[TRT] Pre-Process CPU 0.07370ms CUDA 1.76370ms
[TRT] Network CPU 59.72840ms CUDA 57.37339ms
[TRT] Post-Process CPU 0.06756ms CUDA 0.06750ms
[TRT] Total CPU 59.86966ms CUDA 59.20459ms
[TRT] -----

[TRT] note -- when processing a single image, run 'sudo jetson_clocks' before
to disable DVFS for more accurate profiling/timing measurements

jetson.utils -- freeing CUDA mapped memory
PyTensorNet_Dealloc()
```

Figura 20. Resultado obtenido con imagen 01 del 10 % de la carpeta de test, el resultado fue 88.1% de acierto.

```
r104@jprpower104IA: ~/jetson-inference/python/training/imagenet
jprpower104@jprpower104IA:~/jetson-inference/python/training/imagenet$ imagenet-console.py --model=panel/resnet18.onnx --input_blob=i
input_0 --output_blob=output_0 --labels=$DATASET/labels.txt $DATASET/test/sombra_10%/11.jpg
jetson.inference.__init__.py

[TRT] binding to input 0 input_0 dims (b=1 c=3 h=224 w=224) size=602112
[TRT] binding to output 0 output_0 binding index: 1
[TRT] binding to output 0 output_0 dims (b=1 c=2 h=1 w=1) size=8
device GPU, panel/resnet18.onnx initialized.
[TRT] panel/resnet18.onnx loaded
imageNet -- loaded 2 class info entries
panel/resnet18.onnx initialized.
)class 0000 - 0.961743 (sombra_10%
class 0001 - 0.038257 (sombra_20%)
' (class #0) with 96.174300% confidence

[TRT] -----
[TRT] Timing Report panel/resnet18.onnx
[TRT] -----
[TRT] Pre-Process CPU 0.10558ms CUDA 1.76281ms
[TRT] Network CPU 95.59968ms CUDA 93.28713ms
[TRT] Post-Process CPU 0.27721ms CUDA 0.27375ms
[TRT] Total CPU 95.98248ms CUDA 95.32369ms
[TRT] -----

[TRT] note -- when processing a single image, run 'sudo jetson_clocks' before
to disable DVFS for more accurate profiling/timing measurements

jetson.utils -- freeing CUDA mapped memory
PyTensorNet_Dealloc()
jprpower104@jprpower104IA:~/jetson-inference/python/training/imagenet$
```

Figura 21. resultado obtenido con imagen 11 del 10 % de la carpeta de testeo, el resultado fue 96.1% de acierto.

```
jprpower104@jprpower104IA:~/jetson-inference/python/training/imagenet$ imagenet-console.py --model=panel/resnet18.onnx --input_blob=i
nput_0 --output_blob=output_0 --labels=$DATASET/labels.txt $DATASET/test/sombra_20%/18.jpg
jetson.inference.__init__.py
```

 Institución Universitaria	<b>INFORME FINAL DE TRABAJO DE GRADO</b>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

```
[TRT] binding to input 0 input_0 dims (b=1 c=3 h=224 w=224) size=602112
[TRT] binding to output 0 output_0 binding index: 1
[TRT] binding to output 0 output_0 dims (b=1 c=2 h=1 w=1) size=8
device GPU, panel/resnet18.onnx initialized.
[TRT] panel/resnet18.onnx loaded
ImageNet -- loaded 2 class info entries
panel/resnet18.onnx initialized.
)lass 0000 - 0.264995 (sombra_10%)
class 0001 - 0.735005 (sombra_20%)
image is recognized as 'sombra 20%' (class #1) with 73.500466% confidence

[TRT] -----
[TRT] Timing Report panel/resnet18.onnx
[TRT] -----
[TRT] Pre-Process CPU 0.08359ms CUDA 1.73338ms
[TRT] Network CPU 65.84721ms CUDA 63.52385ms
[TRT] Post-Process CPU 0.07745ms CUDA 0.07719ms
[TRT] Total CPU 66.00825ms CUDA 65.33443ms
[TRT] -----

[TRT] note -- when processing a single image, run 'sudo jetson_clocks' before
to disable DVFS for more accurate profiling/timing measurements

jetson.utils -- freeing CUDA mapped memory
PyTensorNet_Dealloc()
```

Figura 22. resultado obtenido con imagen 18 del 20 % de la carpeta de test, el resultado fue 73.5% de acierto.

```
er104@jprpower104IA: ~/jetson-inference/python/training/imagenet
jprpower104@jprpower104IA:~/jetson-inference/python/training/imagenet$ imagenet-console.py --model=panel/resnet18.onnx --input_blob=i
nput_0 --output_blob=output_0 --labels=$DATASET/Labels.txt $DATASET/test/sombra_20%/07.jpg
jetson.inference.__init__.py

[TRT] binding to input 0 input_0 binding index: 0
[TRT] binding to input 0 input_0 dims (b=1 c=3 h=224 w=224) size=602112
[TRT] binding to output 0 output_0 binding index: 1
[TRT] binding to output 0 output_0 dims (b=1 c=2 h=1 w=1) size=8
device GPU, panel/resnet18.onnx initialized.
[TRT] panel/resnet18.onnx loaded
ImageNet -- loaded 2 class info entries
panel/resnet18.onnx initialized.
)lass 0000 - 0.090640 (sombra_10%)
class 0001 - 0.909360 (sombra_20%)
image is recognized as 'sombra 20%' (class #1) with 90.936011% confidence

[TRT] -----
[TRT] Timing Report panel/resnet18.onnx
[TRT] -----
[TRT] Pre-Process CPU 0.09942ms CUDA 1.75964ms
[TRT] Network CPU 66.65221ms CUDA 64.33547ms
[TRT] Post-Process CPU 0.08723ms CUDA 0.08646ms
[TRT] Total CPU 66.83887ms CUDA 66.18156ms
[TRT] -----

[TRT] note -- when processing a single image, run 'sudo jetson_clocks' before
to disable DVFS for more accurate profiling/timing measurements

jetson.utils -- freeing CUDA mapped memory
PyTensorNet_Dealloc()
```

Figura 23. resultado obtenido con imagen 07 del 20 % de la carpeta de test, el resultado fue del 90.9% de acierto.

Con el anterior comando podemos ingresar cualquier imagen con sombra del 10% o del 20% y la tarjeta podrá hacer inferencia y seleccionar el resultado más similar a la imagen ingresada.

En caso de que el resultado no sea el esperado se deberá revisar la calidad de las imágenes, el learning rate, aumentar las épocas y revisar el loss no sea muy alto.

	<p>INFORME FINAL DE TRABAJO DE GRADO</p>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

## 4. RESULTADOS Y DISCUSIÓN

Cuando empezamos el proyecto (agosto 2019) teníamos claro que la Jetson nano no llevaba más de 2 meses en el mercado, por lo tanto, teníamos un dispositivo donde se habían realizado muy pocos proyectos, así que lo primordial fue conocer sus capacidades y realizar pruebas donde nos quedara la certeza de que con esta tarjeta de IA se podría realizar el reconocimiento de sombras en paneles solares. Realizamos pruebas reconociendo entre perros y gatos, también reconociendo diferentes clases de osos, luego de realizadas se pudo constatar que la plataforma podía clasificar imágenes así fuesen similares, así que es la plataforma más adecuada para nuestro proyecto.

Después de terminado el proyecto se logró que la plataforma de desarrollo Jetson Nano hiciera la inferencia de manera correcta cuando se le ingresaban imágenes con fotos de sombreado del 10% y del 20% de sombra, las cuales eran desconocidas para ella, sin embargo, la precisión cuando la imagen era del 20% no fue tan elevada como las del 10%. Claro que estas sombras son de similar tamaño, y es difícil de ver la diferencia, no obstante, la Jetson logro ver la diferencia y dar el porcentaje de mayor valor al tipo de sombra que pertenecía.

Para obtener estos resultados se debió entrenar la tarjeta varias ocasiones, cambiando las épocas y el learning rate, ya que este valor se calcula de manera experimental porque no hay una formula general para calcularlo.

Después de tener el conocimiento del código de Python con el cual hace la inferencia, se pudo modificar algunos valores para mejorar los resultados.

Queda claro que podemos identificar sombras por medio de la IA, pero se deben seguir haciendo más pruebas para poder obtener mayor precisión y mejorar la identificación de las sombras.

Unas de las debilidades del proyecto son cuando hay poco sol y la sombra no está muy definida, la tarjeta no la percibirá, por lo tanto, hay otro método llamado segmentación, el

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

cual realiza las imágenes y en ese caso la sombra se verá nítida y podremos identificarla más fácilmente, claro que esta herramienta se habilito para Python apenas hace 1 mes.

Igualmente se obtuvieron los resultados esperados, y la idea es ampliar la base de datos para aumentar la cantidad de sombras y que pueda identificar cualquier porcentaje.

*Tabla1.*

*Resultados de las pruebas con 10% de sombra*

Imágenes de texteo 10%	porcentaje de similitud con sombras del 20%
Imagen 1	88,10%
Imagen 2	99,60%
Imagen 3	99,80%
Imagen 4	99,90%
Imagen 5	99,70%
Imagen 6	99,70%
Imagen 7	84,20%
Imagen 8	77,10%
Imagen 9	80,80%
Imagen 10	81,70%
Imagen 11	96,10%
Imagen 12	98,10%
Imagen 13	98,10%
Imagen 14	98,80%
Imagen 15	99,90%
Imagen 16	98,20%
Imagen 17	99,60%
Imagen 18	98%
Imagen 19	98,80%
Imagen 20	99,60%
Imagen 21	98,40%
Imagen 22	96,60%

*Tabla2.*

*Resultados de las pruebas con 20% de sombra*

Imágenes de texteo 20%	porcentaje de similitud con sombras del 20%
Imagen 1	78,90%
Imagen 2	54,20%
Imagen 3	64,50%
Imagen 4	59,30%
Imagen 5	60,10%
Imagen 6	55,70%
Imagen 7	90,90%
Imagen 8	70,40%
Imagen 9	69,20%
Imagen 10	64,90%
Imagen 11	57%
Imagen 12	66,10%
Imagen 13	57,50%
Imagen 14	55%
Imagen 15	53,70%
Imagen 16	66,70%
Imagen 17	52,60%
Imagen 18	73,50%
Imagen 19	64,10%
Imagen 20	61,40%
Imagen 21	57,70%
Imagen 22	54,90%

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

En las tablas se logra observar que al ingresar todas las imágenes de testeo (22 del 10% y 22 del 20%), la tarjeta diferencio las sombras del 10% y el 20%, tuvo un poco de dificultad con las del 20%, sin embargo, las identificó de buena manera.

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

## 5. CONCLUSIONES, RECOMENDACIONES Y TRABAJO FUTURO

---

Algo que se debe tener en cuenta es que la plataforma de desarrollo Jetson Nano es un dispositivo que salió al mercado aproximadamente hace 7 meses, por lo tanto, la información de su funcionamiento y como usarla es muy reciente, sin embargo, se logro programar y se pudo conocer más a fondo el código de Python con el cual la plataforma aprende a inferir, se modificaron algunas líneas para mejorar el aprendizaje y se dio más precisión a los resultados. La tarjeta tiene gran capacidad de identificar imágenes así la diferencia sea pequeña, aunque, se deben tomar imágenes de buena calidad ya que si esta no está muy clara la tarjeta no podrá hacer la inferencia o simplemente arrojará un resultado no deseado.

En conclusión, el proyecto es viable y se puede seguir mejorando para poder identificar cualquier porcentaje de sombra, sin embargo, es necesario ampliar la base de datos con fotografías de las diferentes sombras para que la plataforma aprenda a reconocerlas y se pueda obtener resultados más satisfactorios.

La idea es continuar con el proyecto, mejorando la calidad de la base de datos con muchas más fotos con más porcentaje de sombra, y seguir estudiando como variar el código de Python para mejorar más la precisión y explorar otras herramientas que faciliten el proceso de identificación.

### **Recomendaciones**

Para poder eliminar el problema cuando la sombra no está muy definida se lanzó hace poco una nueva herramienta, la cual funciona también con el código de Python y permite hacer segmentación y logra realzar la sombra y así será más fácil de identificarla, esta herramienta se tendrá en cuenta para futuros proyectos, otra recomendación es tratar de obtener las

	<p>INFORME FINAL DE TRABAJO DE GRADO</p>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

imágenes con una alta calidad ya que la tarjeta las reconocerá más fácil y se podrán obtener mejores resultados.

### Trabajo futuro

Se deberá incluir las otras herramientas con las que cuenta la Jetson Nano a la hora de reconocer imágenes, para perfeccionar los resultados una de estas herramientas es el aprendizaje profundo con segmentación semántica. La segmentación semántica se basa en el reconocimiento de imágenes, excepto que las clasificaciones ocurren a nivel de píxeles en lugar de la imagen completa (figura 25). Esto se logra mediante la convolucionización de una red troncal de reconocimiento de imágenes previamente capacitada, que transforma el modelo en una Red Completamente Convolutiva (FCN) capaz de etiquetar por píxel. Especialmente útil para la percepción ambiental, la segmentación produce densas clasificaciones por píxel de muchos objetos potenciales diferentes por escena, incluidos los primeros planos y fondos de la escena. (Justin, 2019) Luego de realzar la sombra se podrá identificar más fácil el área de esta.

Otra herramienta muy útil cuando tengamos una planta con muchos paneles solares es detectNet (figura 24), la cual analizaría cada panel por separado y si en alguno hay sombra aparecerá en la parte superior el porcentaje de sombra que lo cubre.

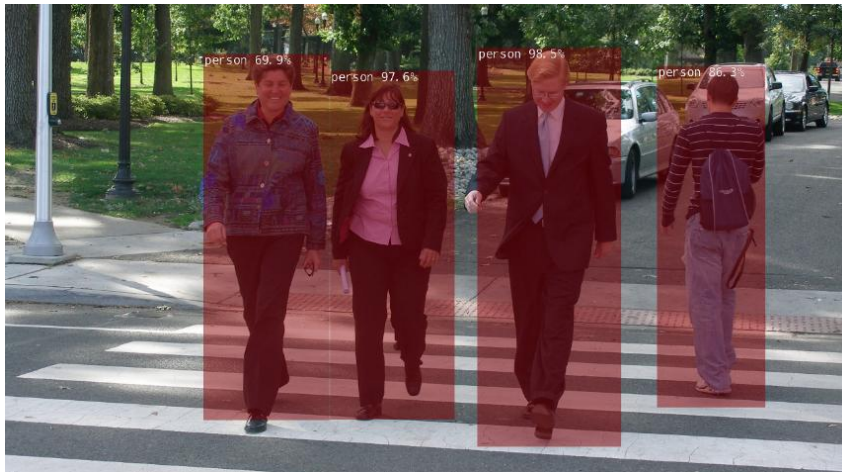


Figura 24. Imagen usando detectnet Tomado de <https://github.com/dusty-nv/jetson-inference/blob/master/docs/detectnet-console-2.md#detecting-objects-from-the-command-line> consultada 14/11/2019.

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27



Figura 25. Imagen usando segmentación. Tomado de <https://github.com/dusty-nv/jetson-inference/blob/master/docs/segnet-console-2.md#segmenting-images-from-the-command-line> consultada 14/11/2019



 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

## REFERENCIAS

- 
- alvarez, M. (9 de enero de 2018). *ONNX Open Neural Network*. Obtenido de <http://blog.josemarioalvarez.com/2018/01/09/onnx-open-neural-network-exchange/>
- Franklin, J. (5 de julio de 2019). *gibhub*. Obtenido de <https://github.com/dusty-nv/pytorch-classification/blob/6b8fcd38fee76cae26e43b9bd547491813bf423d/train.py>
- Franklin, J. (19 de julio de 2019). *gibhub*. Obtenido de <https://github.com/dusty-nv/jetson-inference/blob/master/docs/imagenet-camera-2.md>
- Gill, N. S. (01 de marzo de 2019). *xenonstack*. Obtenido de <https://www.xenonstack.com/blog/artificial-neural-network-applications/>
- Justin, F. (25 de septiembre de 2019). *gibhub*. Obtenido de <https://github.com/dusty-nv/jetson-inference/blob/master/docs/segnet-console-2.md#segmenting-images-from-the-command-line>
- Nvidia Corporation. (marzo de 2019). *Nvidia*. Obtenido de <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#intro>
- Nvidia, corporation. (abril de 2019). *Nvidia*. Obtenido de <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write>
- Perez, A. (19 de marzo de 2019). *top10 game*. Obtenido de <https://top10games.es/hardware/smartthings/nvidia-anuncia-jetson-nano-su-nuevo-mini-pc-por-100e/>
- VanderPlas, J. (2016). Python Data Science Handbook. En J. VanderPlas, *Python Data Science Handbook* (pág. 541). California: O'Reilly Media.

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

## APÉNDICE

Para poder escribir la imagen de la tarjeta SD necesitamos utilizar un programa gráfico como Etcher.

### Instrucciones de grabado

1. Descargamos, instalamos y ejecutamos la aplicación Etcher .

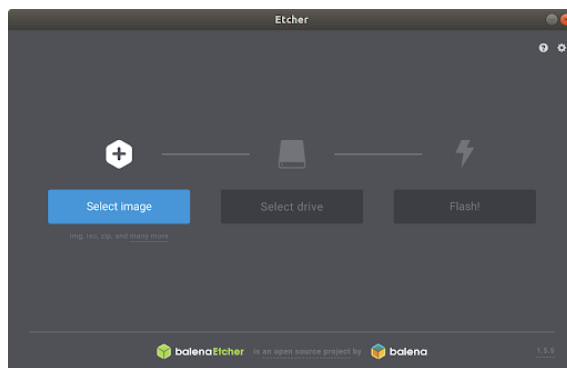


Figura 25. Proceso descarga. Tomado de <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write> consultada 14/11/2019

2. Hacemos clic en "Seleccionar imagen" y elija el archivo de imagen comprimido descargado anteriormente.
3. Inserta tu tarjeta microSD. Si no tiene otras unidades externas conectadas, Etcher seleccionará automáticamente la tarjeta microSD como dispositivo de destino. De lo contrario, haga clic en "Cambiar" y elija el dispositivo correcto.

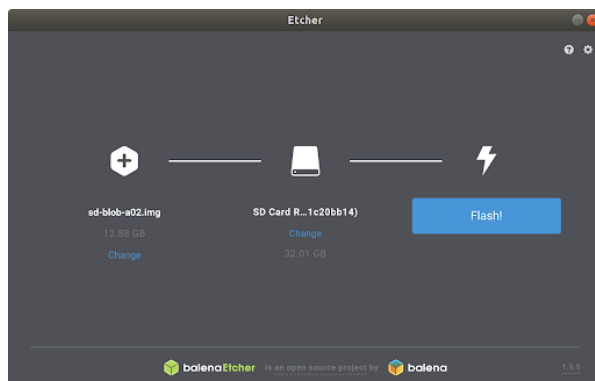


Figura 26. Proceso de instalación. Tomado de <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write> consultada 14/11/2019

	<b>INFORME FINAL DE TRABAJO DE GRADO</b>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

- Haga clic en "Flash". Su sistema operativo puede solicitar su nombre de usuario y contraseña antes de permitir que Etcher continúe.

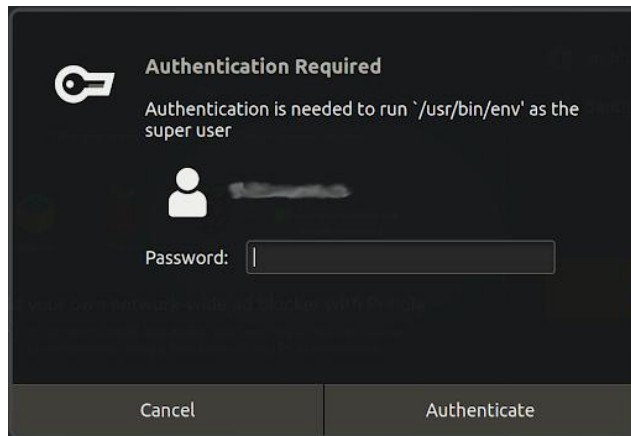


Figura 27. Usuario y contraseña. Tomado de <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write> consultada 14/11/2019

Etcher tardará entre 10 y 15 minutos en escribir y validar la imagen si su tarjeta microSD está conectada a través de USB3.

- Luego expulsamos la tarjeta SD del PC y la retiramos.
- Ingresamos la tarjeta SD a la Jetson Nano y la iniciamos, debe aparecer una pantalla así:

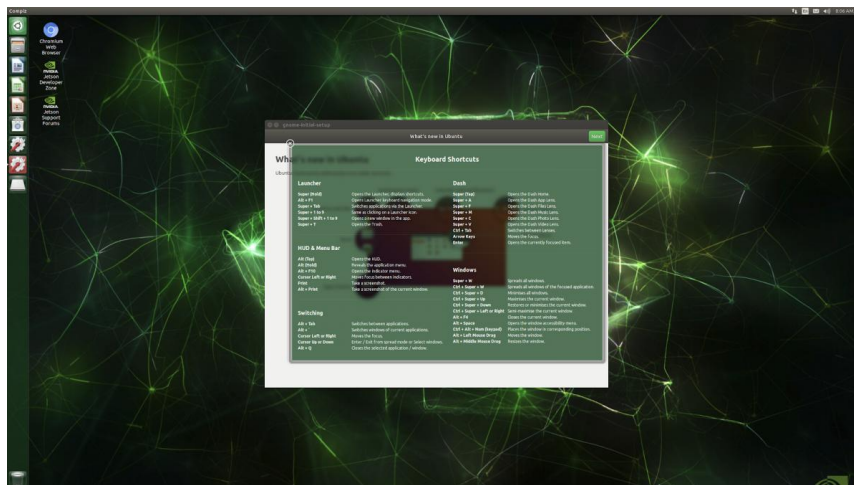


Figura 28. Pantalla que mostrara si es exitoso el proceso de instalación. Tomado de <https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit#write> consultada 14/11/2019

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

Ya quedo configurada para trabajar.

Se recomienda realizar este corto curso en el cual usaremos Python en nuestra Jetson Nano para construir un proyecto de clasificación de aprendizaje profundo con modelos de visión por computadora.

Link del curso: <https://courses.nvidia.com/courses/course-v1:DLI+C-RX-02+V1/about>

el Aquí hay una forma condensada de los comandos para descargar, compilar e instalar proyecto:

```
$ sudo apt-get update
$ sudo apt-get install git cmake libpython3-dev python3-numpy
$ git clone --recursive https://github.com/dusty-nv/jetson-inference
$ cd jetson-inference
$ mkdir build
$ cd build
$ cmake ../
$ make
$ sudo make install
$ sudo ldconfig
```

Para descargar el código, navegue a la carpeta que elija en el Jetson. Primero, asegúrese de que git y cmake estén instalados:

```
$ sudo apt-get update
$ sudo apt-get install git cmake
```

Luego clonamos el jetson-inference proyecto:

```
$ git clone https://github.com/dusty-nv/jetson-inference
$ cd jetson-inference
$ git submodule update --init
```

A continuación, cree un directorio de compilación dentro del proyecto y ejecútelo

```
$ cd jetson-inference # omitir si el directorio de trabajo ya es jetson-
inference / desde arriba
```

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

```
$ mkdir build
$ cd build
$ cmake ../
```

Ahora descargamos los modelos:

```
$ cd jetson-inference / tools
$ ./download-models.sh
```

Ahora instalamos pytorch

```
$ cd jetson-inference/build
$ ./install-pytorch.sh
```

Luego compilamos el proyecto:

```
$ cd jetson-inference/build          # omit if working directory is already
build/ from above
$ make
$ sudo make install
$ sudo ldconfig
```

A continuación se muestra la salida abreviada del objeto pydoc de imagenet y el código de Python del paquete de jetson.inference :

```
jetson.inference.imageNet = class imageNet(tensorNet)
|   Image Recognition DNN - classifies an image
|
|   __init__(...)
|       Loads an image recognition model.
|
|       Parameters:
|           network (string) -- name of a built-in network to use
|                               values can be: 'alexnet', 'googlenet',
'googlenet-12', 'resnet-18`, ect.
|                               the default is 'googlenet'
|
|           argv (strings) -- command line arguments passed to imageNet,
|                               for loading a custom model or custom settings
|
|   Classify(...)
|       Classify an RGBA image and return the object's class and confidence.
|
|       Parameters:
|           image (capsule) -- CUDA memory capsule
|           width (int) -- width of the image (in pixels)
|           height (int) -- height of the image (in pixels)
```

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

```

|
| Returns:
|     (int, float) -- tuple containing the object's class index and
confidence
|
| GetClassDesc(...)
|     Return the class description for the given object class.
|
| Parameters:
|     (int) -- index of the class, between [0, GetNumClasses()]
|
| Returns:
|     (string) -- the text description of the object class
|
| GetNumClasses(...)
|     Return the number of object classes that this network model is able to
classify.
|
| Parameters: (none)
|
| Returns:
|     (int) -- number of object classes that the model supports
|
-----

```

```

importar
jetson.inference

```

```

importar jetson.utils

```

```

importar argparse
importación sys

```

```

# analizar la línea de comando
parser = argparse.ArgumentParser ( description = " Clasificar una
imagen usando un DNN de reconocimiento de imagen " ,

```

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

```

                                formatter_class =
                                argparse.RawTextHelpFormatter, epilog =
                                jetson.inference.imageNet.Usage ())

parser.add_argument ( " file_in " , type = str , help = " nombre de
archivo de la imagen de entrada para procesar " )
parser.add_argument ( " file_out " , type = str , default = None ,
nargs = ' ? ' , help = " nombre de archivo de la imagen de salida
para guardar " )
parser.add_argument ( " --network " , type = str , default = "
googlenet " , help = " modelo pre-entrenado para cargar (vea las
opciones a continuación) " )

prueba :
    opt = parser.parse_known_args () [ 0 ]
excepto :
    imprimir ( " " )
    parser.print_help ()
    sys.exit ( 0 )

# cargar una imagen (en la memoria compartida de la CPU / GPU)
img, ancho, alto = jetson.utils.loadImageRGBA (opt.file_in)

# cargar la red de reconocimiento
net = jetson.inference.imageNet (opt.network, sys.argv)

# clasifica la imagen
class_idx, confianza = net.Classify (img, ancho, alto)

# encuentra la descripción del objeto
class_desc = net.GetClassDesc (class_idx)

# imprimir el resultado

```

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

```

print ( "la imagen se reconoce como ' { : s } ' (clase # { : d } )
con { : f } % c onfidence \ n " .format (class_desc, class_idx,
confianza * 100 ))

# imprimir información de tiempo
net.PrintProfilerTimes ()

# superpone el resultado en la imagen
si opt.file_out no es Ninguno :
    font = jetson.utils.cudaFont ( tamaño =
jetson.utils.adaptFontSize (ancho))
    font.OverlayText (img, ancho, alto, " { : f } % { : s } "
.format (confianza * 100 , class_desc), 10 , 10 , font.White,
font.Gray40)
    jetson.utils.cudaDeviceSynchronize ()
    jetson.utils.saveImageRGBA (opt.file_out, img, width,
height)

```

Cuando vamos a realizar la inferencia, escribimos `imagenet console.py` esto es porque estamos llamando el código anterior para ejecutarlo y hacer la inferencia.

Despues de construir el proyecto, asegurese de estar en la carpeta `aarch64/bin` :

```
$ cd jetson-inference / build / aarch64 / bin
```

Ahora ejecutamos el programa de ejemplo para verificar que la tarjeta haya quedado bien configurada:

```
$ ./imagenet-console.py --network=googlenet images/orange_0.jpg output_0.jpg
```

Ya verificado el buen funcionamiento crearemos nuestro propio programa de reconocimiento de imágenes.

Verificamos que el `pytorch` este funcionando correctamente e importamos `torchvision`, una importante herramienta de `pythorch`:



 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

```
>>> antorcha de importación
>>> print (antorcha.__version__)
>>> print ( ' CUDA disponible: ' + str (torch.cuda.is_available ()))
>>> a = torch.cuda.FloatTensor (2) .zero_ ()
>>> print ( ' Tensor a = ' + str (a))
>>> b = torch.randn (2) .cuda ()
>>> print ( ' Tensor b = ' + str (b))
>>> c = a + b
>>> print ( ' Tensor c = ' + str (c))

>>> importación torchvision
>>> print (torchvision.__version__)
```

Ahora creamos un espacio de memoria adicional de 4Gb las cuales se usaran en el entrenamiento, ya que este consume mucha memoria:

```
sudo fallocate -l 4G /mnt/4GB.swap
sudo mkswap /mnt/4GB.swap
sudo swapon /mnt/4GB.swap
```

Ahora creamos la base de datos, la cual constara de todas fotos con las cuales entrenara, validara y se harán los testeos. En nuestro caso se usaron las siguientes carpetas.

Tabla 3.

*Nombres y cantidad de fotos por carpeta.*

Porcentaje de sombra	Carpeta Train N° de imágenes	Carpeta test N°de imágenes	Carpeta Validación N° de imágenes
10% DE SOMBRA	430	22	80
20% DE SOMBRA	489	22	90

Esta carpeta la ingresamos a una sola carpeta llamada Panel.

Ahora para una mayor precisión cambiamos en el código de Python de train, el numero de épocas de 35 a 70 epocas:

```
parser.add_argument('--epochs', default=70, type=int, metavar='N',
```

Y también cambiamos el learning rate para un optimo entrenamiento, bien de 0.1 y le asignamos 0.01:

```
parser.add_argument('--lr', '--learning-rate', default=0.01, type=float,
```

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

La tarjeta entrenara por lotes, para una respuesta mas confiable tomaremos lotes grandes de fotos, cambiando de 8 a 10.

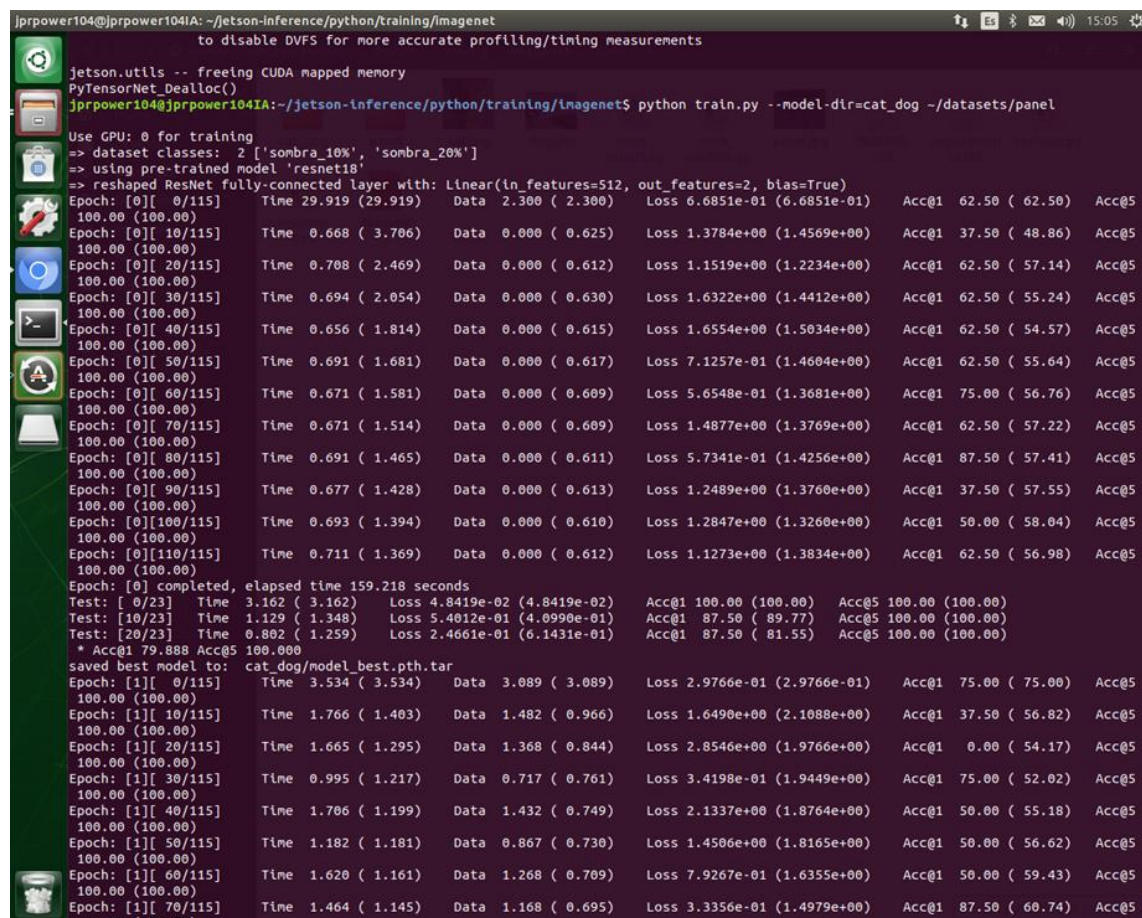
```
parser.add_argument('-b', '--batch-size', default=10, type=int,
```

Con estos dos parámetros modificado podemos empezar el entrenamiento.

Cuando haya recopilado una gran cantidad de datos, puede intentar entrenar un modelo en él, tal como lo hemos hecho antes. El proceso de capacitación es el mismo que en los ejemplos anteriores, y se utilizan los mismos scripts PyTorch:

```
$ cd jetson-inference/python/training/classification
$ python train.py --model-dir=panel ~/datasets/panel
```

Despues se vera como empieza a entrenar, se debe tener cuidado e ir monitoreando que los parámetros que arroja en el entrenamiento sean los esperados.



```

jetsonpower104@jetsonpower104IA: ~/jetson-inference/python/training/imagenet
to disable DVFS for more accurate profiling/timing measurements

jetson.utils -- freeing CUDA mapped memory
PyTensorNet_Dealloc()
jetsonpower104@jetsonpower104IA:~/jetson-inference/python/training/imagenet$ python train.py --model-dir=cat_dog ~/datasets/panel

Use GPU: 0 for training
=> dataset classes: 2 ['sombra_10%', 'sombra_20%']
=> using pre-trained model 'resnet18'
=> reshaped ResNet fully-connected layer with: Linear(in_features=512, out_features=2, bias=True)
Epoch: [0][ 0/115]   Time 29.919 (29.919)   Data 2.300 ( 2.300)   Loss 6.6851e-01 (6.6851e-01)   Acc@1 62.50 ( 62.50)   Acc@5
100.00 (100.00)
Epoch: [0][ 10/115]  Time 0.668 ( 3.706)   Data 0.000 ( 0.625)   Loss 1.3784e+00 (1.4569e+00)   Acc@1 37.50 ( 48.86)   Acc@5
100.00 (100.00)
Epoch: [0][ 20/115]  Time 0.708 ( 2.469)   Data 0.000 ( 0.612)   Loss 1.1519e+00 (1.2234e+00)   Acc@1 62.50 ( 57.14)   Acc@5
100.00 (100.00)
Epoch: [0][ 30/115]  Time 0.694 ( 2.054)   Data 0.000 ( 0.630)   Loss 1.6322e+00 (1.4412e+00)   Acc@1 62.50 ( 55.24)   Acc@5
100.00 (100.00)
Epoch: [0][ 40/115]  Time 0.656 ( 1.814)   Data 0.000 ( 0.615)   Loss 1.6554e+00 (1.5034e+00)   Acc@1 62.50 ( 54.57)   Acc@5
100.00 (100.00)
Epoch: [0][ 50/115]  Time 0.691 ( 1.681)   Data 0.000 ( 0.617)   Loss 7.1257e-01 (1.4604e+00)   Acc@1 62.50 ( 55.64)   Acc@5
100.00 (100.00)
Epoch: [0][ 60/115]  Time 0.671 ( 1.581)   Data 0.000 ( 0.609)   Loss 5.6548e-01 (1.3681e+00)   Acc@1 75.00 ( 56.76)   Acc@5
100.00 (100.00)
Epoch: [0][ 70/115]  Time 0.671 ( 1.514)   Data 0.000 ( 0.609)   Loss 1.4877e+00 (1.3769e+00)   Acc@1 62.50 ( 57.22)   Acc@5
100.00 (100.00)
Epoch: [0][ 80/115]  Time 0.691 ( 1.465)   Data 0.000 ( 0.611)   Loss 5.7341e-01 (1.4256e+00)   Acc@1 87.50 ( 57.41)   Acc@5
100.00 (100.00)
Epoch: [0][ 90/115]  Time 0.677 ( 1.428)   Data 0.000 ( 0.613)   Loss 1.2489e+00 (1.3760e+00)   Acc@1 37.50 ( 57.55)   Acc@5
100.00 (100.00)
Epoch: [0][100/115]  Time 0.693 ( 1.394)   Data 0.000 ( 0.610)   Loss 1.2847e+00 (1.3260e+00)   Acc@1 50.00 ( 58.04)   Acc@5
100.00 (100.00)
Epoch: [0][110/115]  Time 0.711 ( 1.369)   Data 0.000 ( 0.612)   Loss 1.1273e+00 (1.3834e+00)   Acc@1 62.50 ( 56.98)   Acc@5
100.00 (100.00)
Epoch: [0] completed, elapsed time 159.218 seconds
Test: [ 0/23]   Time 3.162 ( 3.162)   Loss 4.8419e-02 (4.8419e-02)   Acc@1 100.00 (100.00)   Acc@5 100.00 (100.00)
Test: [10/23]  Time 1.129 ( 1.348)   Loss 5.4012e-01 (4.0990e-01)   Acc@1 87.50 ( 89.77)   Acc@5 100.00 (100.00)
Test: [20/23]  Time 0.802 ( 1.259)   Loss 2.4661e-01 (6.1431e-01)   Acc@1 87.50 ( 81.55)   Acc@5 100.00 (100.00)
* Acc@1 79.888 Acc@5 100.000
saved best model to: cat_dog/model_best.pth.tar
Epoch: [1][ 0/115]   Time 3.534 ( 3.534)   Data 3.089 ( 3.089)   Loss 2.9766e-01 (2.9766e-01)   Acc@1 75.00 ( 75.00)   Acc@5
100.00 (100.00)
Epoch: [1][ 10/115]  Time 1.766 ( 1.403)   Data 1.482 ( 0.966)   Loss 1.6490e+00 (2.1088e+00)   Acc@1 37.50 ( 56.82)   Acc@5
100.00 (100.00)
Epoch: [1][ 20/115]  Time 1.665 ( 1.295)   Data 1.368 ( 0.844)   Loss 2.8546e+00 (1.9766e+00)   Acc@1 0.00 ( 54.17)   Acc@5
100.00 (100.00)
Epoch: [1][ 30/115]  Time 0.995 ( 1.217)   Data 0.717 ( 0.761)   Loss 3.4198e-01 (1.9449e+00)   Acc@1 75.00 ( 52.02)   Acc@5
100.00 (100.00)
Epoch: [1][ 40/115]  Time 1.706 ( 1.199)   Data 1.432 ( 0.749)   Loss 2.1337e+00 (1.8764e+00)   Acc@1 50.00 ( 55.18)   Acc@5
100.00 (100.00)
Epoch: [1][ 50/115]  Time 1.182 ( 1.181)   Data 0.867 ( 0.730)   Loss 1.4506e+00 (1.8165e+00)   Acc@1 50.00 ( 56.62)   Acc@5
100.00 (100.00)
Epoch: [1][ 60/115]  Time 1.620 ( 1.161)   Data 1.268 ( 0.709)   Loss 7.9267e-01 (1.6355e+00)   Acc@1 50.00 ( 59.43)   Acc@5
100.00 (100.00)
Epoch: [1][ 70/115]  Time 1.464 ( 1.145)   Data 1.168 ( 0.695)   Loss 3.3356e-01 (1.4979e+00)   Acc@1 87.50 ( 60.74)   Acc@5
100.00 (100.00)

```

Figura 29. Proceso de entrenamiento.

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

Ahora convertimos el modelo a ONNX, para poder hacer la inferencia con tensorRT:

```
python onnx_export.py --model-dir=cat_dog
```

Esto creara un modelo llamado: resnet18.onnx bajo:

```
jetson-inference/python/training/classification/panel/
```

Ahora probamos el entrenamiento haciendo una prueba con las imágenes de testeo:

```
imagenet-console.py --model = panel / resnet18.onnx --input_blob = input_0 --  
output_blob = output_0 --labels = $ DATASET /labels.txt $ DATASET  
/test/sombra_10%/03.jpg panel.jpg
```



Este comando arrojará el resultado de la inferencia dando mayor porcentaje al valor de sombra deseado.

Es decir, si ingresamos una imagen de testeo de la carpeta de 10 % arroja el resultado de la inferencia. Para que sea correcto la tarjeta debe asignar el mayor porcentaje de acierto a la imagen del 10 % de sombra en este caso.

 Institución Universitaria	<b>INFORME FINAL DE TRABAJO DE GRADO</b>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

Institución Universitaria		MODALIDAD TRABAJO DE GRADO PRODUCTO OBTENIDO EN TALLERES O LABORATORIOS DEL ITM		Código	FDE 146
Registro de actividades y cumplimiento de horas / Talleres o Laboratorios de DOCENCIA		Versión	02	Fecha	2015-02-30
Documento de Identidad:		7024573758			
Nombre completo del estudiante:		SROASTIAN SEMA VERA			
Programa académico ITM:		Tecnología en Electrónica			
Nombre completo del Docente Asesor:		Julian Perez Restrepo			
Fecha de iniciación del producto (aaaa/mm/dd):		2014/09/07		Fecha de terminación del producto (aaaa/mm/dd):	2014/11/11
Nombre Taller o Laboratorio:		Electrónica y Energías Renovables			
Ubicación:		Parque I			
Campus:		Fraternidad			

A	M	D	Actividad desempeñada por el estudiante	Hora Ingreso	Hora salida	Total horas	Firma Laboratorista	Firma Estudiante
14	8	7	Objetivos del producto de laboratorio	7:00	7:12	2	[Firma]	[Firma]
14	8	8	Familiarización con la Tetsen Nano.	7:14	7:16	2	[Firma]	[Firma]
14	8	18	Familiarización con la Tetsen Nano.	8	7:12	4	[Firma]	[Firma]
14	8	16	Introducción al mundo de la Tetsen Nano.	7:14	7:16	2	[Firma]	[Firma]
14	8	20	Introducción de linux como sistema operativo.	8	7:12	4	[Firma]	[Firma]
14	8	23	Metodos de programación para la Tetsen Nano (python).	7:14	7:16	2	[Firma]	[Firma]
14	8	27	Tensor Flow y bases en Inteligencia artificial.	8	7:12	4	[Firma]	[Firma]
14	8	30	Análisis de diferentes ejemplos con AI y Tetsen Nano.	7:14	7:16	2	[Firma]	[Firma]
14	9	3	Emulación de python y otras Herramientas.	8	7:12	4	[Firma]	[Firma]
14	9	6	Instalación de modelos (GoogleNet, AlexNet, VGGNet).	7:14	7:16	2	[Firma]	[Firma]
14	9	10	Corrección de errores al intentar compilar el programa.	8	7:12	4	[Firma]	[Firma]
14	9	13	Corrección de errores al intentar compilar el programa.	7:14	7:16	2	[Firma]	[Firma]
14	9	17	Familiarización y prueba con los ejemplos.	8	7:12	4	[Firma]	[Firma]
14	9	20	Análisis de resultados de los ejemplos.	7:14	7:16	2	[Firma]	[Firma]
14	9	24	Abordamos nuestro proyecto.	8	7:12	4	[Firma]	[Firma]
14	9	27	Toma de fotos para base de datos.	7:14	7:16	2	[Firma]	[Firma]
14	9	30	Toma de fotos para base de datos.	8	7:12	4	[Firma]	[Firma]
14	10	4	Ajuste de la base de datos.	7:14	7:16	2	[Firma]	[Firma]
14	10	8	Entrenamiento de nuestra red neuronal.	8	7:12	4	[Firma]	[Firma]
14	10	11	Pruebas y análisis de nuestro primer resultado.	7:14	7:16	2	[Firma]	[Firma]
14	10	15	Ajustes en el código de python para mejor resultados.	8	7:12	4	[Firma]	[Firma]
14	10	19	Pruebas y ajustes finales del proyecto.	7:14	7:16	2	[Firma]	[Firma]
14	10	22	Toma de datos de los resultados de la inferencia.	8	7:12	4	[Firma]	[Firma]
14	10	25	Análisis de los resultados del proyecto.	7:14	7:16	2	[Firma]	[Firma]
TOTAL HORAS								

  
Firma Estudiante  
  
Nombre y Firma Laboratorista

Nombre y Firma Profesional Universitario - Centro de Laboratorios

 Institución Universitaria	<b>INFORME FINAL DE TRABAJO DE GRADO</b>	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

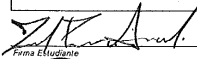
 Institución Universitaria		MODALIDAD TRABAJO DE GRADO PRODUCTO OBTENIDO EN TALLERES O LABORATORIOS DEL ITM				Código FDE 146
Registro de actividades y cumplimiento de horas / Talleres o Laboratorios de DOCENCIA						Versión 02
Fecha 2015-09-30						Fecha 2015-09-30


  

Documento de Identidad:	1017127676					
Nombre completo del estudiante:	Jaiber Pineda Acevedo					
Programa académico ITM:	Tecnología en electrónica					
Nombre completo del Docente Asesor:	Juliano Velazquez Restrepo					
Fecha de iniciación del producto (aaaa/mm/dd):	2014/08/07		Fecha de terminación del producto (aaaa/mm/dd):	2014/11/11		
Nombre Taller o Laboratorio:	Electrónica y energías renovables					
Ubicación:	Parque					
Campus:	Fraternidad					


Fecha A M D	Actividad desempeñada por el estudiante	Hora Ingreso	Hora salida	Total horas	Firma Laboratorista	Firma Estudiante
19/8/7	Objetivos del producto de laboratorio	10	12	2	[Firma]	[Firma]
19/8/8	Familiarización con la Jetson Nano	14	16	2	[Firma]	[Firma]
19/8/13	Familiarización con Python.	8	12	4	[Firma]	[Firma]
19/8/16	Introducción al manejo de la Jetson Nano.	14	16	2	[Firma]	[Firma]
19/8/20	Instalación de linux como sistema operativo	8	12	4	[Firma]	[Firma]
19/8/23	metodos de programación para la Jetson Nano (Python)	14	16	2	[Firma]	[Firma]
19/8/27	Tensor Flow y bases en inteligencia artificial.	8	12	4	[Firma]	[Firma]
19/8/30	Analizar diferentes ejemplos con IA y Jetson Nano	14	16	2	[Firma]	[Firma]
19/9/3	Instalación de Pytorch y otras herramientas.	8	12	4	[Firma]	[Firma]
19/9/6	Instalación de modelos (cocoapi, tensorflow, alexnet, etc.)	14	16	2	[Firma]	[Firma]
19/9/10	Corrección de los errores al intentar compilar el programa.	8	12	4	[Firma]	[Firma]
19/9/13	Corrección de los errores al intentar compilar el programa.	14	16	2	[Firma]	[Firma]
19/9/17	Familiarización y pruebas con los ejemplos.	8	12	4	[Firma]	[Firma]
19/9/20	Análisis de resultados de los ejemplos.	14	16	2	[Firma]	[Firma]
19/9/24	Abordamos nuestro proyecto.	8	12	4	[Firma]	[Firma]
19/9/27	Toma de fotos para base de datos.	14	16	2	[Firma]	[Firma]
19/10/1	Toma de fotos para base de datos.	8	12	4	[Firma]	[Firma]
19/10/4	Ajuste de la base de datos.	14	16	2	[Firma]	[Firma]
19/10/8	Entrenamiento de nuestra red neuronal.	8	12	4	[Firma]	[Firma]
19/10/11	Pruebas y análisis de nuestro primer resultado.	14	16	2	[Firma]	[Firma]
19/10/15	Ajustes en el código de python para mejorar resultados.	8	12	4	[Firma]	[Firma]
19/10/18	Pruebas y ajustes finales del proyecto.	14	16	2	[Firma]	[Firma]
19/10/22	Toma de datos de los resultados de la inferencia.	8	12	4	[Firma]	[Firma]
19/10/25	Análisis de los resultados del proyecto.	14	16	2	[Firma]	[Firma]
TOTAL HORAS						70

  
Firma Estudiante

  
Nombre y Firma Laboratorista

Nombre y Firma Profesional Universitario - Centro de Laboratorios

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

 Institución Universitaria	INFORME FINAL DE TRABAJO DE GRADO	Código	FDE 089
		Versión	03
		Fecha	2015-01-27

FIRMA ESTUDIANTES	 _____ _____ 
FIRMA ASESOR	 _____ 
FECHA ENTREGA: 15/11/2014	

FIRMA COMITÉ TRABAJO DE GRADO DE LA FACULTAD _____		
RECHAZADO____	ACEPTADO____	ACEPTADO CON MODIFICACIONES____
ACTA NO. _____		
FECHA ENTREGA: _____		

FIRMA CONSEJO DE FACULTAD _____
ACTA NO. _____
FECHA ENTREGA: _____