Proyecto - Investigación

Integrantes:

> Juan Carlos Herrera

> Steven Jimenez Bustamante

Clasificación de imágenes (Celulas sin Parásitos vs Celulas con Parásitos)

Background:

Los parásitos intracelulares son microparásitos que son capaces de crecer y reproducirse dentro de las células de un huésped. Algunos de estos parácitos son patogénicos, lo que quiere decir que puede provocar enfermedades en las células y en el organismo portador. Por mencionar algunos de estos parásitos patogénicos, se encuentran Salmonella Typhi, Staphylococcus aureus, Clamidia, entre otros.

Es por esto que confirmar la presencia o ausencia de un parásito en una célula es de suma importancia para prevenir enfermedades.

Para este ejercicio, utilizaremos un dataset de imagenes con presencia de parásitos intracelular y también de imágenes con ausencia de parásitos intracelulares. Este dataset se obtuvo de la siguiente página:

https://www.kaggle.com/brsdincer/cell-images-parasitized-or-not

A continuación se describen los dos escenarios (etiquetas):

> Célula con Parásito (**Infectada**)

> Célula sin Parásito (**No Infectada**)

Nuestro problema consiste en clasificar una imagen en dos escenarios, celula **Infectada** y célula **No Infectada**.

La propuesta de estrategia para abordar este problema de clasificación es el siguiente:

1. Preprocesar la imagenes
2. Decidir cuales features de las imágenes son relevantes para el modelo de clasificación
3. Definir el modelo de clasificación

> > 3.1 Entrenar una red neuronal pre-entrenada mediante Fast AI y ResNet34

> > 3.2 Crear y entrenar una red neuronal desde cero

> > > Se harán pruebas con varias capas ocultas de redes neuronales, y en base a los datos de accuracy,
> > > se seleccionará el mejor modelo de red neuronal desde cero.

4. Se validará cual es el método más eficiente para la clasificación de imagenes. Model entrenado desde cero VS Modelo Pre-entrenado

Creacion de una CNN para clasificar imagenes desde 0

```
#Importación de libreías
```

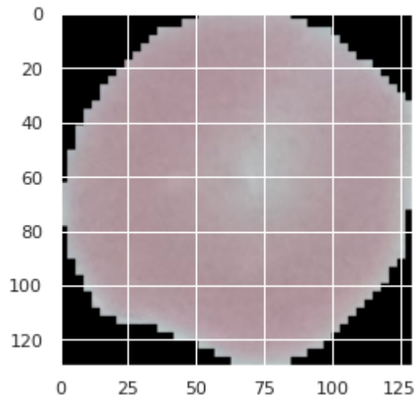
```
#importacion de librerias
from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.keras import optimizers
from tensorflow.keras.models import Sequential
from tensorflow.python.keras.layers import Dropout, Flatten, Dense, Activation
from tensorflow.python.keras.layers import  Convolution2D, MaxPooling2D
from tensorflow.python.keras import backend as K
import numpy as np
from keras.preprocessing.image import load_img, img_to_array
from keras.models import load_model
import tensorflow as tf
from tensorflow import keras

# Librerias de ayuda
import cv2
import os
import seaborn as sns
%matplotlib inline
sns.set(color_codes=True)
import matplotlib.pyplot as plt
```

```

# Creacion del acceso a google drive
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
from IPython import display

```

```
Mounted at /content/drive
```

```
# Se muestran ejemplos de celulas infectadas y no Infectadas
print("Célula sin Parásito (no Infectada)")
image = plt.imread('/content/drive/My Drive/Colab Notebooks/cell_images/test/uninfected/C241NThinF_IMG_20151207_124608_cell_171.png')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))


```

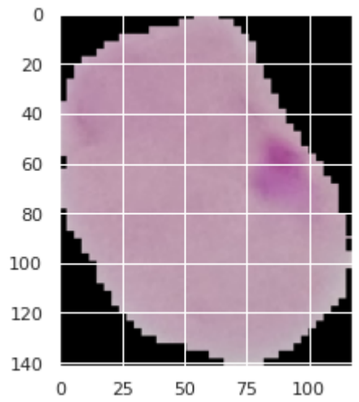
```
Célula sin Parásito (no Infectada)
<matplotlib.image.AxesImage at 0x7f1a5c0a15d0>
```




```
print("Célula con Parásito (Infectada)")
image = plt.imread('/content/drive/My Drive/Colab Notebooks/cell_images/test/parasitized/C189P150ThinF_IMG_20151203_141901_cell_82.png')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

```
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

```
Célula con Parásito (Infectada)
<matplotlib.image.AxesImage at 0x7f1a5f207090>
```



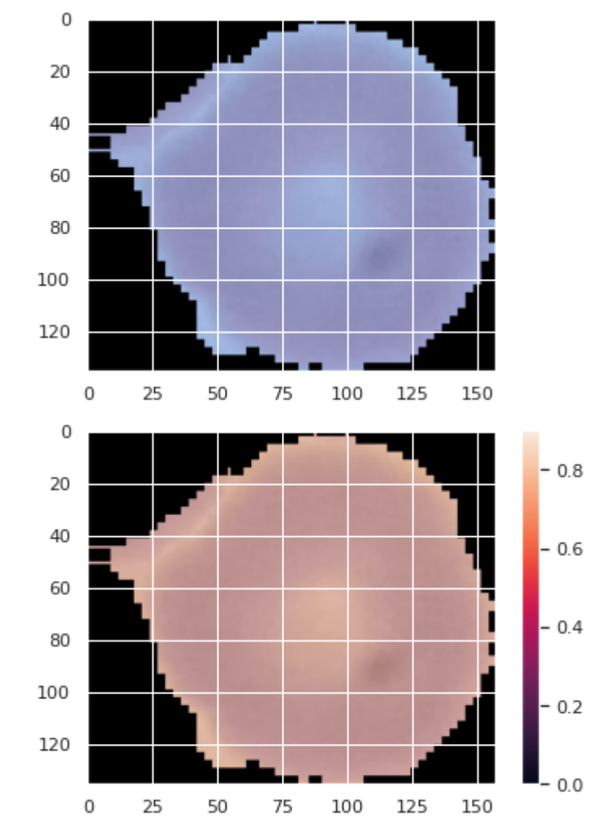

```
K.clear_session() #Borrar sesiones remanentes
```

```
# Se definen los directorios que contienen las imagenes para train y test
Entrenamiento = '/content/drive/My Drive/Colab Notebooks/cell_images/train'
Validacion = '/content/drive/My Drive/Colab Notebooks/cell_images/test'


```

```
# a continuancion definiremos ciertos parametros que van ser de utilidad durante la construccion de las diferentes capa de la CNN
epochs=15 ## define la cantidad de epochs de cada red que decidamos incorporar
length, width= 100, 100 # se define el tamano de 100X100 para las imagenes
Inputshape = 100, 100,3 # Se define el tamano para la red y ademas que es de 3 canales RGB o a color en resumidas palabras
batch_size = 32 # tamano de los bactches a procesar en cada CNN

# numero de veces que se va a procesar la informacion en este caso vamos a definir
# num_samples de training 24987 / batch_size 32
steps = 780
# numero de veces que se va a procesar la informacion en este caso vamos a definir
# num_samples de validacio 2600/ batch_size 32
validation_steps = 81

## se definen los filtros a usar en cada capa de nuestra CNN
filtroConv1 = 32
filtroConv2 = 64
filtroConv3 = 128
filtroConv4 = 256
###
filtro1 = (3, 3)
filtro2 = (3, 3)
filtro3 = (2, 2)
filtro4 = (2, 2)
tamano_pool = (2, 2)
clases = 2 # se definen la cantidad de clases del modelo en este caso 2 con parasito y sin parasito
lr = 0.002 # se define el learning rate del modelo
```

```
## a continuacion se pplotea una imagen de ejemplo para ver su dimension y colores
image = plt.imread('/content/drive/My Drive/Colab Notebooks/cell_images/test/uninfected/prueba.png')
```

```
image = plt.imread('/content/drive/My Drive/Colab Notebooks/cell_images/test/uninfected/pruebu.png')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.show()
plt.imshow(image)
plt.colorbar()
plt.grid(True)
plt.show()
image.shape
```




```
(136, 157, 3)
```

Preparacion y procesado de imagenes

```
# Definimos en una variable los parametro a configurar y se realiza el ajuste de las imagenes RGB para obtener valores entre 0 y 1...
# Zoom range del 10% , pero dependiendo del accuracy se puede variar para obtener un resultado diferente

set_entrenamiento = ImageDataGenerator(rescale=1. / 255,shear_range=0.1,zoom_range=0.1,horizontal_flip=True)
set_validacion = ImageDataGenerator(rescale=1. / 255, zoom_range=0.1, shear_range=0.1 ,horizontal_flip=True)

```

```
##Se utiliza la funcion flow from directory para procesar las imagenes del directorio deseado
#con todos los parametros con el que sera preprocesadas las imagenes
entrenamiento_procesado = set_entrenamiento.flow_from_directory(Entrenamiento,target_size=(length, width),
    batch_size=batch_size,color_mode='rgb',class_mode='categorical')
```

```
Found 24987 images belonging to 2 classes.
```

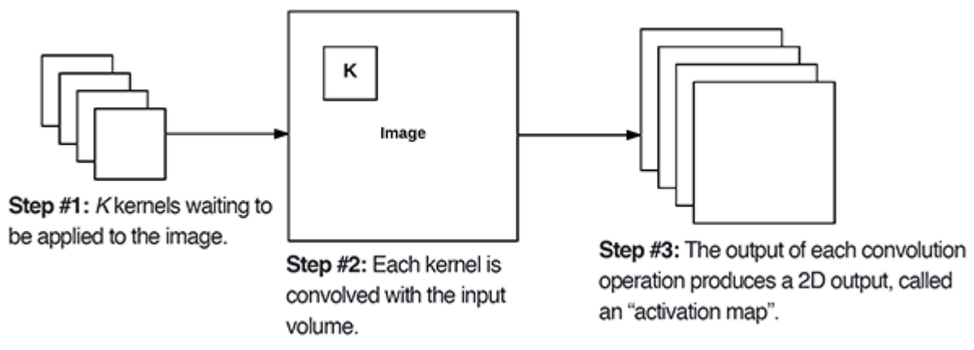
```
#Se realiza un proceso similir de preprocesado para las imagenes de validacion
```

```
#Se realiza un proceso similar de preprocesado para las imagenes de validacion
validacion_procesado = set_validacion.flow_from_directory(Validacion,target_size=(length, width),
    batch_size=batch_size, color_mode='rgb',class_mode='categorical')

```

```
Found 2600 images belonging to 2 classes.
```

Creacion de la CNN

La siguiente figura muestra como el prametro Keras Conv2D determina el numero de kernels a convulocionar con respecto a la entrada del sistemas para generar un mapa de activacion en 2 dimensiones. En nuestro caso los filtros estan previamente definidos como variables filtroConv1 = 32 filtroConv2 = 64 filtroConv3 = 128 filtroConv4 = 256



La figura de abajo describe como funciona el tamano del kernel y las dimensiones mas comunes son 1×1, 3×3, 5×5, and 7×7, para el caso de nuestra CNN tambien se definieron previamente como las siguientes variables, al ser imagenes menores de 128x128 se recomienda usar filtros no mayores a 3X3 filtro1 = (3, 3) filtro2 = (3, 3) filtro3 = (2, 2) filtro4 = (2, 2)

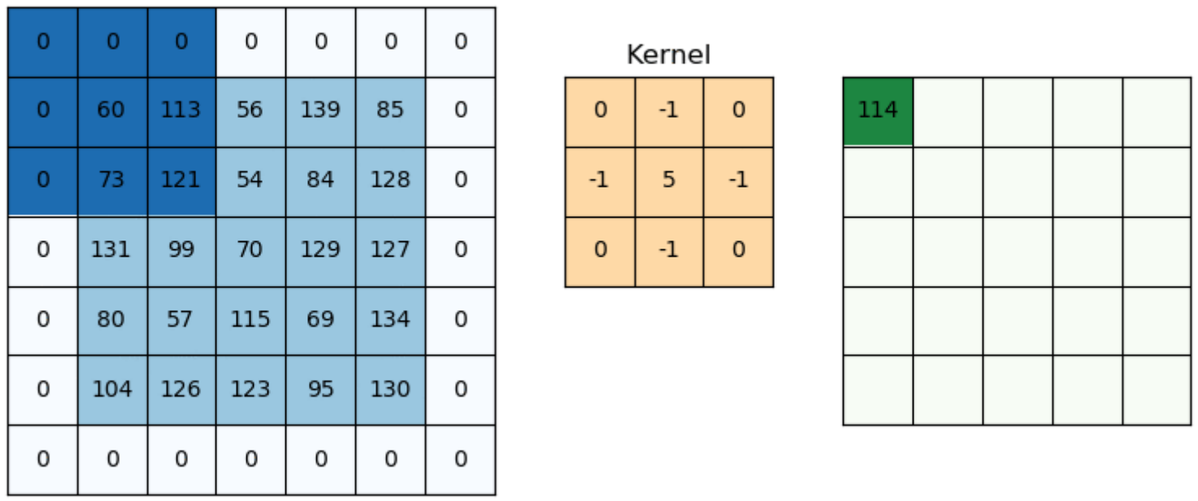
3x3

0.91	0.32	0.07
0.73	0.26	0.81
0.53	0.68	0.14

5x5

0.27	0.64	0.44	0.84	0.29
0.28	0.06	0.89	0.99	0.33
0.64	0.67	0.08	0.38	0.03
0.04	0.31	0.16	0.57	0.08
0.87	0.85	0.97	0.71	0.96

La funcion de padding acontinuacion mostrada en la siguiente figura muestra un kernel de 3×3 aplicado a una imagen con padding.



El fomato de los datos tambien debe ser seleccionado para nuestras imagenes y en este caso el numero 3 significa que son imagenes RGB, caso contrario si fuera blanco y negro seria un 2

Channels Last Ordering

Height	Width	Depth
96	96	3

Channels First Ordering

Depth	Height	Width
3	96	96

```
## Se define CNN como secuencial para que las capas se ejecuten en secuencia
## se utiliza la capa convolution2d de la libreria Keras por su performance en procesado de imagenes
## primer capa CNN
# se aplica una capa de pooling luego de
Celulascnn = Sequential()
Celulascnn.add(Convolution2D(filtroConv1,filtro1, padding ="same",input_shape=Inputshape , activation='relu'))
Celulascnn.add(MaxPooling2D(pool_size=tamano_pool))
```

```
#segunda capa de CNN
Celulascnn.add(Convolution2D(filtroConv3,filtro2, padding ="same",activation='relu'))
Celulascnn.add(MaxPooling2D(pool_size=tamano_pool))
```

```
#tercera capa de CNN
Celulascnn.add(Convolution2D(filtroConv3, filtro3, padding ="same",activation='relu'))
Celulascnn.add(MaxPooling2D(pool_size=tamano_pool))
```

```
#cuarta capa de la CNN
Celulascnn.add(Convolution2D(filtroConv4,filtro4, padding ="same",activation='relu'))
Celulascnn.add(MaxPooling2D(pool_size=tamano_pool))
```

```
### se usa flatten par aplanar el modelo,una nueva capa de procesado y dropout del 50% para prevenir el overfitting  y
# luego una nueva capa de procesado y para finalizar softmax como capa de salida del modelo
```

```
# luego una nueva capa de procesado y para finalizar softmax como capa de salida del modelo
Celulascnn.add(Flatten())
Celulascnn.add(Dense(128, activation='relu'))
Celulascnn.add(Dropout(0.5))
Celulascnn.add(Dense(clases, activation='softmax'))
```

Entrenamiento del modelo

```
# se utliza el optimizador Adam para optimizacion y compilacion del modelo
opt = keras.optimizers.Adam(learning_rate=lr)
Celulascnn.compile(loss='categorical_crossentropy',optimizer = opt, metrics=['accuracy'])
```

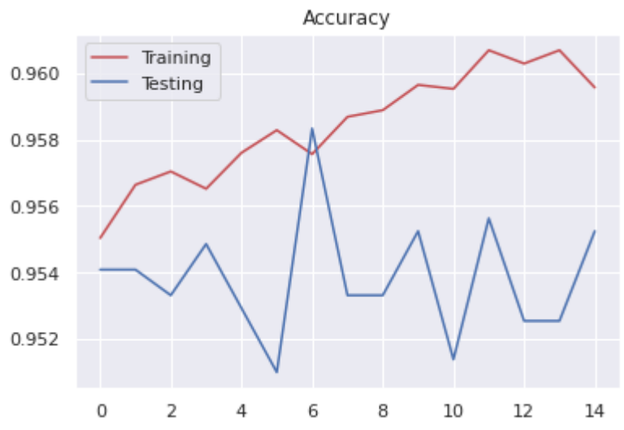
```
1
grafico = Celulascnn.fit(entrenamiento_procesado,steps_per_epoch=steps,epochs=epochs, validation_data=validacion_procesado,validation_steps=validation_steps)
```

```
Epoch 1/15
780/780 [==============================] - 1075s 1s/step - loss: 0.1508 - accuracy: 0.9550 - val_loss: 0.1460 - val_accuracy: 0.9541
Epoch 2/15
780/780 [==============================] - 915s 1s/step - loss: 0.1434 - accuracy: 0.9566 - val_loss: 0.1437 - val_accuracy: 0.9541
Epoch 3/15
780/780 [==============================] - 909s 1s/step - loss: 0.1409 - accuracy: 0.9570 - val_loss: 0.1388 - val_accuracy: 0.9533
Epoch 4/15
780/780 [==============================] - 913s 1s/step - loss: 0.1405 - accuracy: 0.9565 - val_loss: 0.1441 - val_accuracy: 0.9549
Epoch 5/15
780/780 [==============================] - 892s 1s/step - loss: 0.1391 - accuracy: 0.9576 - val_loss: 0.1445 - val_accuracy: 0.9529
Epoch 6/15
780/780 [==============================] - 743s 953ms/step - loss: 0.1337 - accuracy: 0.9583 - val_loss: 0.1466 - val_accuracy: 0.9510
Epoch 7/15
780/780 [==============================] - 740s 948ms/step - loss: 0.1337 - accuracy: 0.9576 - val_loss: 0.1431 - val_accuracy: 0.9583
Epoch 8/15
780/780 [==============================] - 752s 965ms/step - loss: 0.1301 - accuracy: 0.9587 - val_loss: 0.1337 - val_accuracy: 0.9533
Epoch 9/15
780/780 [==============================] - 745s 955ms/step - loss: 0.1272 - accuracy: 0.9589 - val_loss: 0.1344 - val_accuracy: 0.9533
Epoch 10/15
780/780 [==============================] - 744s 953ms/step - loss: 0.1262 - accuracy: 0.9596 - val_loss: 0.1301 - val_accuracy: 0.9552
Epoch 11/15
780/780 [==============================] - 748s 958ms/step - loss: 0.1247 - accuracy: 0.9595 - val_loss: 0.1410 - val_accuracy: 0.9514
Epoch 12/15
780/780 [==============================] - 743s 952ms/step - loss: 0.1240 - accuracy: 0.9607 - val_loss: 0.1261 - val_accuracy: 0.9556
Epoch 13/15
780/780 [==============================] - 744s 954ms/step - loss: 0.1226 - accuracy: 0.9603 - val_loss: 0.1252 - val_accuracy: 0.9525
Epoch 14/15
780/780 [==============================] - 748s 959ms/step - loss: 0.1188 - accuracy: 0.9607 - val_loss: 0.1369 - val_accuracy: 0.9525
Epoch 15/15
780/780 [==============================] - 747s 957ms/step - loss: 0.1226 - accuracy: 0.9596 - val_loss: 0.1343 - val_accuracy: 0.9552
```

```
# imprimimos un remunen del modelo para ver su comportamiento
#print(Celulascnn.summary())
```

```
#Se realiza un grafico para poder visualmente el loss del modelo entrenado
plt.figure(0)
plt.title("Loss")
plt.plot(grafico.history['loss'], 'r', label='Training')
plt.plot(grafico.history['val_loss'], 'b', label='Testing')
plt.legend()
plt.show()
```


```
#se realiza un grafico para ver el comportamiento del accuracy durante los epochs
plt.figure(1)
plt.title("Accuracy")
plt.plot(grafico.history['accuracy'], 'r', label='Training')
plt.plot(grafico.history['val_accuracy'], 'b', label='Testing')
plt.legend()
plt.show()
```




```
# Se genera una funcion para almacenar o crear un directorio para guardar el modelo
import os
# Creacion del acceso a google drive
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
from IPython import display
target_dir = '/content/drive/My Drive/Colab Notebooks/cell_images/modelo'
if not os.path.exists(target_dir):
    os.mkdir(target_dir)
Celulascnn.save('/content/drive/My Drive/Colab Notebooks/cell_images/modelo/modelo.h5')
Celulascnn.save_weights('/content/drive/My Drive/Colab Notebooks/cell_images/modelo/pesos.h5')
```

```
Mounted at /content/drive
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-c569230f0a49> in <module>()
      8 if not os.path.exists(target_dir):
      9     os.mkdir(target_dir)
---> 10 Celulascnn.save('/content/drive/My Drive/Colab Notebooks/cell_images/modelo/modelo.h5')
     11 Celulascnn.save_weights('/content/drive/My Drive/Colab Notebooks/cell_images/modelo/pesos.h5')
```

Seccion de prediccion y prueba

SEARCH STACK OVERFLOW

```
longitud, altura = 100, 100
modelo = '/content/drive/My Drive/Colab Notebooks/cell_images/modelo/modelo.h5'
pesos_modelo = '/content/drive/My Drive/Colab Notebooks/cell_images/modelo/pesos.h5'
cnn = load_model(modelo)
cnn.load_weights(pesos_modelo)

def predict(file):
  x = load_img(file, target_size=(longitud, altura))
  x = img_to_array(x)
  x = np.expand_dims(x, axis=0)
  array = cnn.predict(x)
  result = array[0]
  answer = np.argmax(result)
  if answer == 0:
    print("pred: celula con parasitos")
  elif answer == 1:
    print("pred:celula no infectada")
  return answer
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-7af5e9919ed4> in <module>()
      2 modelo = '/content/drive/My Drive/Colab Notebooks/cell_images/modelo/modelo.h5'
      3 pesos_modelo = '/content/drive/My Drive/Colab Notebooks/cell_images/modelo/pesos.h5'
----> 4 cnn = load_model(modelo)
      5 cnn.load_weights(pesos_modelo)
      6

NameError: name 'load_model' is not defined
```

SEARCH STACK OVERFLOW

```
#Prueba de una imagen para ver el resultado
path= '/content/drive/My Drive/Colab Notebooks/cell_images/test/uninfected/C3thin_original_IMG_20150608_162835_cell_101.png'
plt.imshow(image)
predict(path)
```

Construccion de modelo con CNN preentrenada RESNET34

```
# importar las librerias de Fastai y se monta el google drive para poder accesar contenido desde ahi
from fastai.vision import *
from fastai.metrics import error_rate, accuracy
import warnings
```

```
import warnings
warnings.filterwarnings('ignore')
```

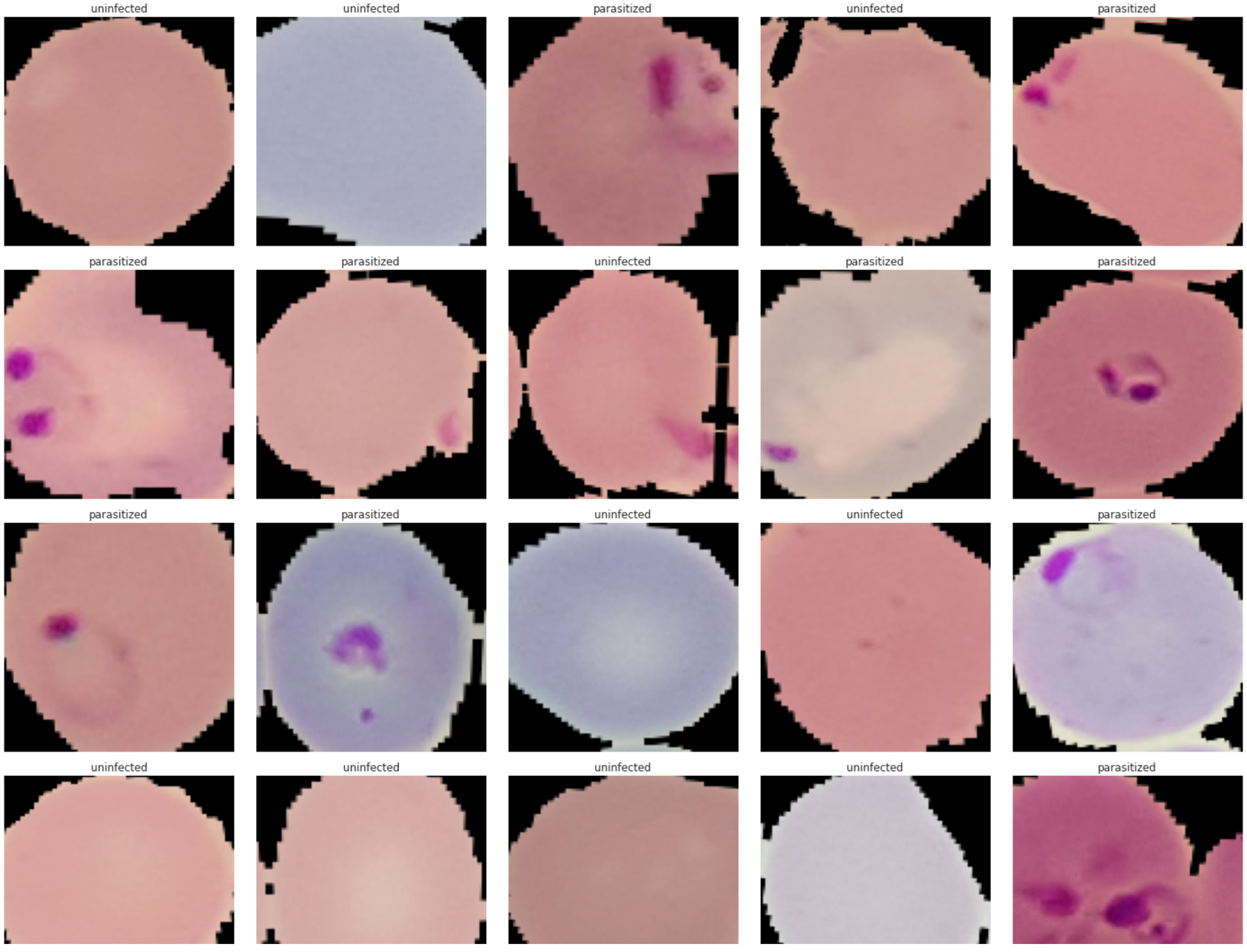
```
# Se define el camino al directorio de google drive donde se encuentra el data ser con las imagenes
# ademas se imprimen todos los directorios en esa direccion
path = Path('/content/drive/My Drive/Colab Notebooks/cell_images')
path.ls()
```

```
[PosixPath('/content/drive/My Drive/Colab Notebooks/cell_images/test'),
 PosixPath('/content/drive/My Drive/Colab Notebooks/cell_images/train'),
 PosixPath('/content/drive/My Drive/Colab Notebooks/cell_images/modelo')]
```

```
# se debe generar un paquete de datos para su posterior procesado
# se corre el modelo inicialmente con una resolucion de 352*352
# pero estos parametros pueden manipularse para obtener mejores resultados
Celulas = ImageDataBunch.from_folder(path, train='train', valid='test', ds_tfms=get_transforms(do_flip=False), size=100, bs=32, num_workers=8)
```

```
# Se muestras las imagenes y las clases respectivas ademas de la cantidad de cada una de las clases
Celulas.show_batch()
print(Celulas.classes,Celulas.c)
print(len(Celulas.train_ds), len(Celulas.valid_ds))
```

```
['parasitized', 'uninfected'] 2
24987 2600
```




```
1 # Construir un modelo de CNN con una red preeentrenada para reducir el tiempo de ejucion  se uso la resnet34 de la libreria Pytorch
2 # se define el Error rate = 1 - accuracy
3 Celulas_modelo= cnn_learner(Celulas, models.resnet34, metrics = [accuracy])
4 # Se entrena el modelo basado en 4 epochs de datos con un learning rate por default
5 Celulas_modelo.fit_one_cycle(10)
```

```
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to /root/.cache/torch/hub/checkpoints/resnet34-b627a593.pth
100%                                                 83.3M/83.3M [00:01<00:00, 55.9MB/s]
```

epoch	train_loss	valid_loss	accuracy	time
0	0.321380	0.242810	0.908462	38:33
1	0.225369	0.178343	0.929615	38:06
2	0.190363	0.139787	0.951538	38:29
3	0.176698	0.133757	0.951154	38:37
4	0.158591	0.116340	0.959231	38:19
5	0.152809	0.119924	0.960385	38:01

```
# se almacena el modelo y luego se carga
Celulas_modelo.save('stage-1')
Celulas_modelo.load('stage-1')
```

```
      (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (5): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (7): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (8): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (9): ReLU(inplace=True)
      (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (11): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (13): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (14): ReLU(inplace=True)
      (15): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (16): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (17): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (18): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (19): ReLU(inplace=True)
      (20): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (21): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (22): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (23): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (24): ReLU(inplace=True)
      (25): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (26): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (27): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (28): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (29): ReLU(inplace=True)
      (30): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (31): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (32): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (33): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (34): ReLU(inplace=True)
      (35): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (36): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (37): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
      (38): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (39): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (40): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (41): ReLU(inplace=True)
      (42): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (43): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (44): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (45): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
  (46): ReLU(inplace=True)
  (47): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (48): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
), Sequential(
  (0): AdaptiveAvgPool2d(output_size=1)
  (1): AdaptiveMaxPool2d(output_size=1)
  (2): Flatten()
  (3): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (4): Dropout(p=0.25, inplace=False)
  (5): Linear(in_features=1024, out_features=512, bias=True)
  (6): ReLU(inplace=True)
  (7): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (8): Dropout(p=0.5, inplace=False)
  (9): Linear(in_features=512, out_features=2, bias=True)
)], add_time=True, silent=False)
```

```
# Unfreeze all layers of the CNN
Celulas_modelo.unfreeze()
# Find the optimal learning rate and plot a visual
Celulas_modelo.lr_find()
Celulas_modelo.recorder.plot(suggestion=True)
```

0.00% [0/1 00:00<00:00]

epoch	train_loss	valid_loss	accuracy	time

9.62% [75/780 05:15<49:23 0.3668]

```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
Min numerical gradient: 1.10E-06
Min loss divided by 10: 6.92E-07
```


```
# Fit the model over 2 epochs
Celulas_modelo.fit_one_cycle(2, max_lr=slice(3e-5, 3e-9))
```

0.00% [0/2 00:00<00:00]

epoch	train_loss	valid_loss	accuracy	time

27.82% [217/780 16:07<41:50 0.1276]

```
# Rebuild interpreter and replot confusion matrix
interp = ClassificationInterpretation.from_learner(Celulas_modelo)
interp.plot_confusion_matrix(figsize=(12,12), dpi=60)
```

```
path = Path('/content/drive/My Drive/Colab Notebooks/cell_images/test/uninfected/C3thin_original_IMG_20150608_162835_cell_101.png')
img = open_image(path)
pred_class,pred_idx,outputs = Celulas_modelo.predict(img)
print(pred_class)
img.show()
pred_class,pred_idx,outputs = Celulas_modelo.predict(img)

```

Analisis entre modelos

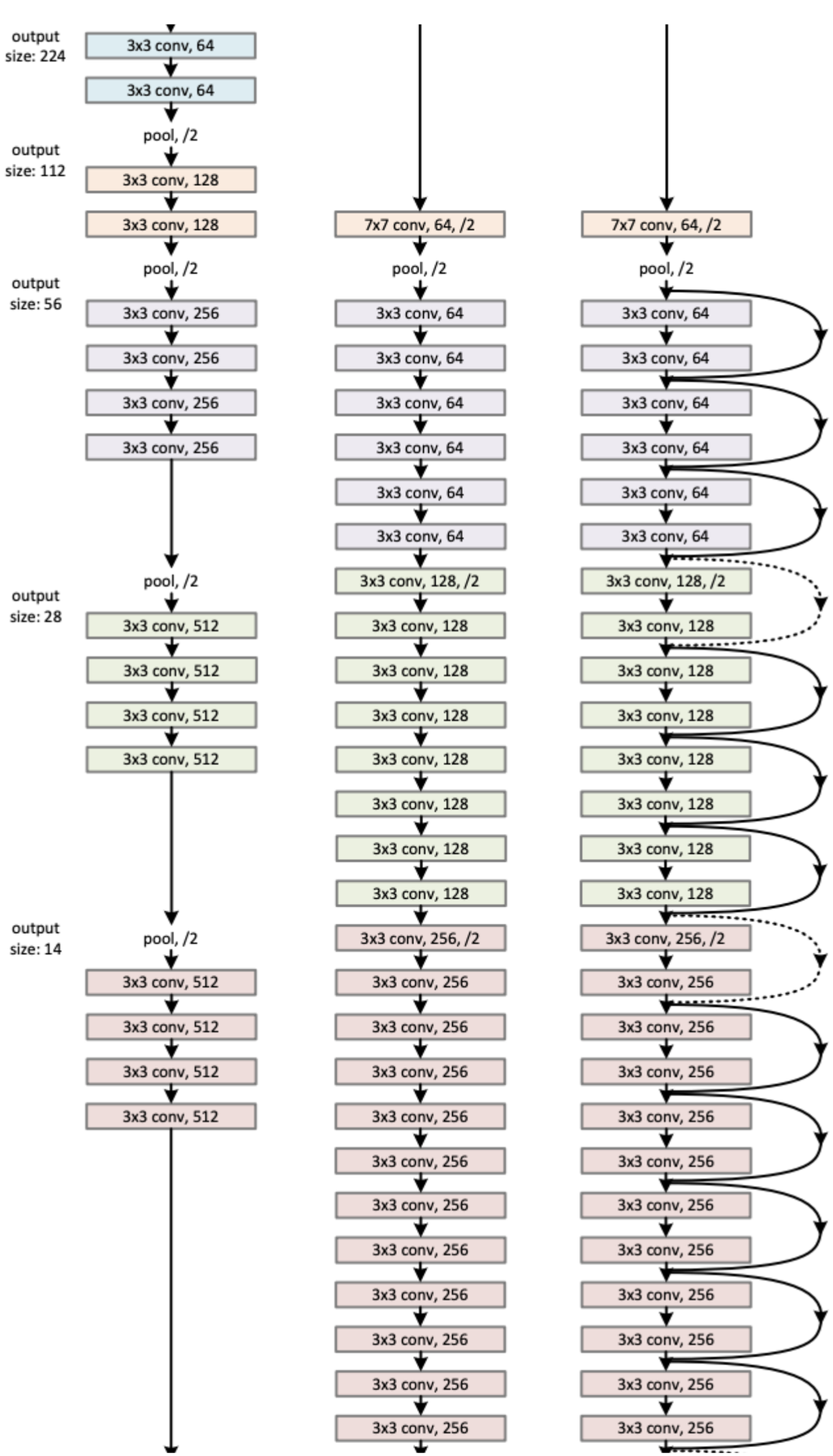
A realizar una comparativa entre ambas redes CNN generada por capas y la red preentrenada RESNET34 se puede observar en los resultados del accuracy de ambas se encuentra entre 95-96% tanto para training como para testing por lo que ambas redes funcionas apropiadamente con el set de imagenes utilizados. Basado en esto procedemos a realizar un analisis mas detallado para explicar ambas redes:

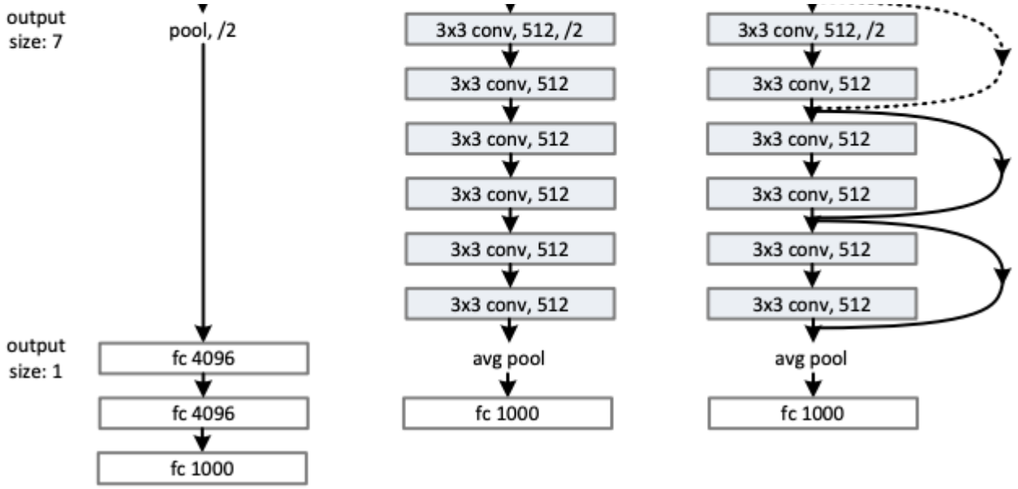
First column name	CNN custom	Resnet 34
Tiempo entrenamiento Approx	4.5	6.3 H
Mejor acuracy	95.83%	96.03%
Facilidad de implementacion	Media-Alta	Baja
Facilidad para modificar	Alta	No modificable

Conclusion:

Ambas redes presentan excelentes resultados y tiempos muy similares de ejecucion, por un tema de profundidad y capacidad para seguir mejorando el modelo y adaptabilidad a diferentes set de datos y tamanos de imagenes la CNN custom es la que mejor funciona para el proposito de este proyecto. Sin embargo las redes preentrenadas como la resnet 34 pueden ser una opcion valida para un modelado rapido y efectivo cuando el tipo de proyecto no amerite un detalle en profundidad. Como parte de la investigacion aca mencionamos otros tipos de redes que pueden ser utilizadas para el procesamiento de imagenes Utilizando fast AI:

- resnet18
- resnet50,
- resnet101,
- resnet152,
- squeezenet1_0,
- squeezenet1_1
- densenet121,
- densenet169,
- densenet201,
- densenet161,
- vgg16_bn,
- vgg19_bn
- alexnet


output size: 224
3x3 conv, 64
3x3 conv, 64
pool, /2
output size: 112
3x3 conv, 128
3x3 conv, 128
pool, /2
output size: 56
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
pool, /2
output size: 28
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
pool, /2
output size: 14
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256


output size: 7
pool, /2
output size: 1
fc 4096
fc 4096
fc 1000
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000