SPRING 2020 MACHINE LEARNING 4610-MINI-PROJECT 2

UPDATES

Logs of updates (I want the code in Tesorflow/keras V2)

- Extra-explanations: Textbook chapter 13, starting in pag 378 with ResNet. But reading the entire chapter 13 will help a lot.

  Download the textbook code: Chapter 13
  https://github.com/ageron/handson-ml

- Extra tutorial: Extra tutorial for ResNet with CIFAR-10 (our dataset), Here:
  https://towardsdatascience.com/resnets-for-cifar-10-e63e900524e0

- CODE: I asked permission to the author of this code and co-author of some of my papers to re-use its code. So, you can modify and re-use this implementation (unless you write your own code):
  https://github.com/yuengdelahoz/Resnet/blob/master/detailed_implementations/resnet_50.py

Key elements to modify (for now, I haven't check more in details) :

Line 128: `model = get_model((224,224,3)) //this model was created for ImageNet where images have some size (read the project)`

Line 69 `def get_model(input_shape,num_class=1000):` Obviously this is the number of classes in ImagesNet in your case you have to train with CIFAR-10. (How many classes in CIFAR-10?)

This code doesn't have a loss function to measure how good or bad your predictions are doing.

You need to read the data, and train.

You need to predict, and measure accuracy.

Other resources (Not for using directly, but for getting ideas)

https://keras.io/examples/cifar10_resnet/

https://github.com/seansoleyman/cifar10-resnet

Rubric

Report 30%
Presentation 20 %
Code and documentation 50%

Note: The rubric above takes place just after the presentation. No presentations on time will result in an automatic grade of zero.

Participants name	Code Section	Report Section	Documentation Sec	Presentation Sec

Submission deadline: February 18 at 11:59 pm.
Presentation deadline: February 18 at class time.

Deliveries:

- Code: A Jupyter Notebook that includes: project (code), documentation.
- Report: The same Jupyter notebook works as report as long as you use Markdown cells to provide format. This will allow
- Presentation: Presentations at class time (More details coming soon)

Resources

- K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016. https://arxiv.org/abs/1512.03385

- Tutorial 1: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

- Tutorial 2: https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8

- Video: https://www.youtube.com/watch?v=qhXZsFVxGKo

General description

In this project, you will be implementing a residual network with 50 layers (The one described in the paper: https://arxiv.org/abs/1512.03385) and testing its accuracy using the *CIFAR-10 dataset* (https://www.cs.toronto.edu/~kriz/cifar.html).

Here, some details about the network. ResNet is part of group of rock start deep CNNs that include networks such as VGG Net, ResNet, Dense Net, Inception Net, Xception Net. As a fanny note, Inception Net which was proposed by Christian Szegedy and his team at Google has been always associated with the 2010 movie Inception. A brief description of these networks can be found here (https://towardsdatascience.com/cnn-architectures-a-deep-dive-a99441d18049).

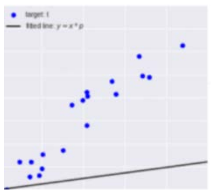
ResNet address the problem of plain deep CNNs, namely vanishing/exploding gradients. As you remember from the class, the way in which Neural Networks works is not very different from the basic idea of linear regression.

BRIEF REVIEW OF LINEAR REGRESSION

1. Use hypothesis function to compute some preliminary prediction.

$$h_\theta = \theta_0 + \theta_1 x$$

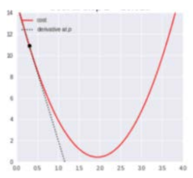
2. Plot the result (does not look good)



3. Quantify how bad or good your hypothesis function is fitting the data (in this case bad bad). Use the cost function J shown in next equation.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

4. You can plot your parameters (theta 1) for instance vs the cost function J (doesn't look good) you want the se black dot line horizontal in the button.



5. Compute the partial derivative of the cost function (J)

$$\frac{\partial}{\partial\theta_0} J(\theta_0,\theta_1) = \frac{\partial}{\partial\theta_0}\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Then you will obtain the partial derivatives of theta 0, and theta 1 like shown in the following two equations

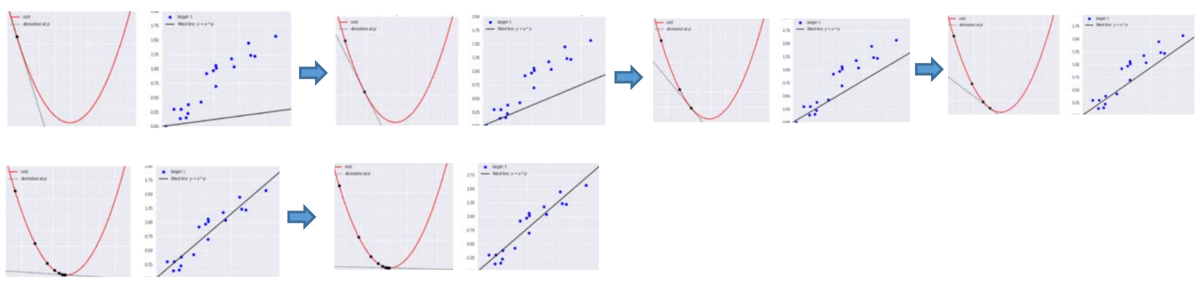
$$\frac{\partial}{\partial\theta_0} J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\frac{\partial}{\partial\theta_1} J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x^{(i)}$$

8 Update your parameters iteratively until reaching convergency using these two derivatives and gradient decent, also use the learning rate alpha.

$$\begin{aligned}
&\text{repeat until convergence } \{\\
&\quad \theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\\
&\quad \theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x^{(i)}\\
&\}
\end{aligned}$$

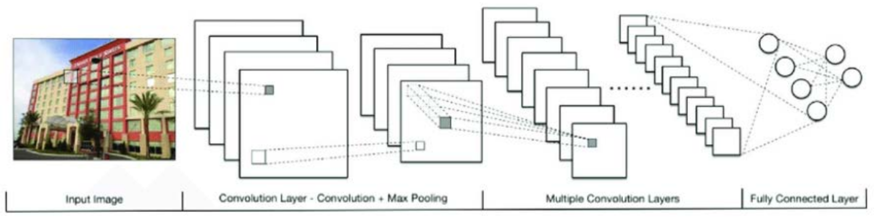
As result you will observe something like that every time that you update theta 0, and theta 1, namely the linear function is fitting very well the training data (right blue), while error or loss (left red, theta 1 vs cost) is getting its minimum.



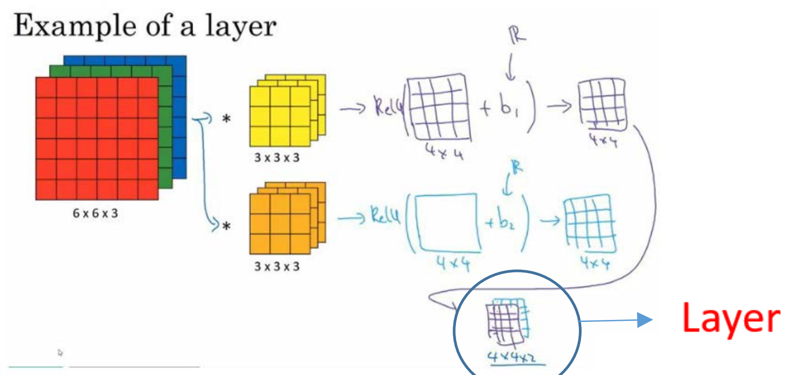
SIMILAR PROCESS WITH CNNs.

Basically, if you understand well linear regression, you already understand CNNs.

1. What is the hypothesis function here?
   The hypothesis function is your network architecture. That's means, the number of hidden layers, the type of activation functions (RELU, logistic regression, tanh, linear regression, SoftMax, etc). Let's take a look of the following CNN architecture


Input Image
Convolution Layer - Convolution + Max Pooling
Multiple Convolution Layers
Fully Connected Layer

2. Which are the parameters that we want to update? Let's take a look of what a CNN layer means. The following graph (from the class slides illustrates the concept)



Given a RGB picture (left), we may apply to filters of 3x3x3 just because the original image has three channels (R,G,B), thus the output is a 4x4 matrix per filter. Then, we add a bias (theta 0), and apply and activation function like RELU. Finally, we stack the partial 4x4 results (two 4x4 outputs) which result in a 4x4x2 array (a layer). If we analyze each element of this 4x4x2 output, we will find that each elements correspond to a element-wise product between the filter, and the input the pixel of the picture (here, the set of pixels corresponds to X or the features). Thus, each element 4x4x2 layer corresponds to RELU(X*theta 1 + Theta 0 (bias)).

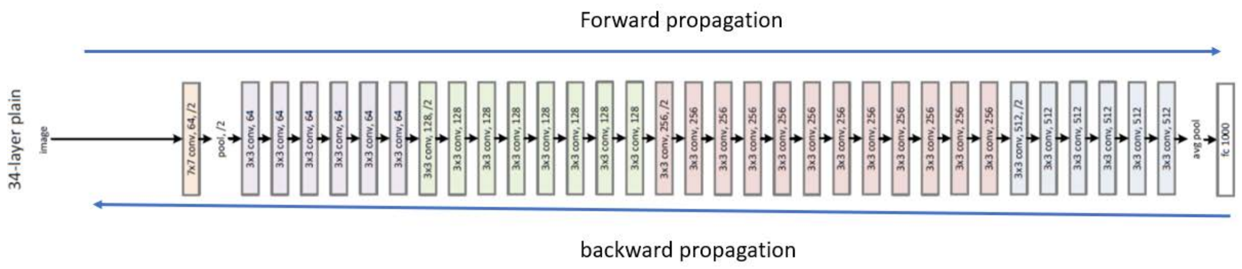
So, what is theta 1?

(Sorry, I can't give you points for the answer ☹)

Did you get it? **YES,** theta 1 corresponds to the elements of the filters. YES, so you don't have to be a computer vision or signal processing expert to find the filter that better find the vertical, horizontal or diagonal features. You will LEARN these parameters or filters.

What is the challenge here (No for you, but in general)

One of the problems with deep CNNs is to compute the partial derivatives of the cost function with respect to the parameters (See step 5 above in the linear regression example). Next figure illustrates the process



Here, the process of updating the parameters for CNNs

1. Move forward, this process is call forward propagation. Here, we read the input picture's pixels (X), and apply the different filters, bias and activation functions (this is one layer), the output of this layer is then the input of the next layer. Thus, this process continues layer after layer until reaching the last layer where the prediction is output.
2. **Evaluate your prediction**, same as in linear regression, use a loss function to compute how good or bad is your prediction. Here, the loss function is independent from the model. You already know that, if you want to do:

- For Regression use Mean Square Error (MSE) (Check canvas linear regression)

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

- For binary classification (cat or not cat) you can use Cross Entropy Loss (Check slides logistic regression in canvas) as shown in the next formula

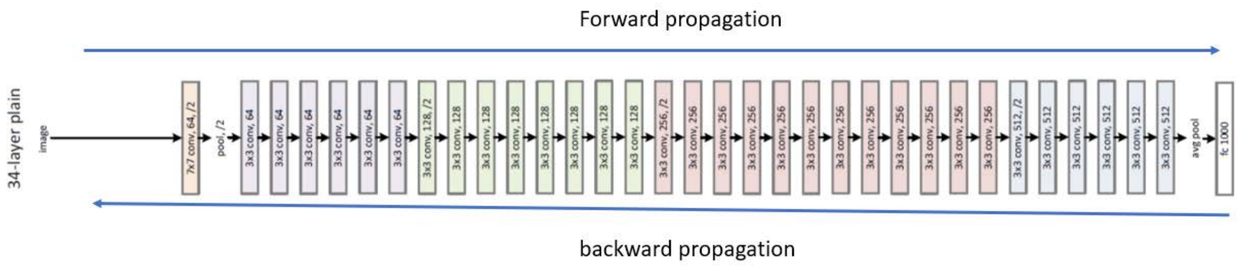
$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

For this project, your training set has the following classes:

- airplane
- automobile
- bird
- cat
- deer
- dog
- frog
- horse
- ship
- truck

Meaning, given a picture your CNN will output one of thesse classes. In other words, what you have here is a multi-class classification problem, In this case, you can use a Categorical Cross Entropy loss function (DO YOUR RESEARCH HERE)

3. Computing the GRADIENTS, meaning computing the partial derivative with respect to your parameters. In CNNs we do that by using a process call BACK PROPAGATION



In plain deep CNNs computing this partial derivative has well known problem: vanishing/exploding gradients basically the problem consists in that the partial derivatives become very small or very large making the job o gradient decent (GD) very difficult or impossible. In other words, by using the partial derivatives gradient decent is not able to improve the parameters and reaching convergency. In order to illustrate the idea, let's take a look of the following example for regression. Here, are the partial derivatives of the loss function for regression with respect of theta 0, and theta 1.

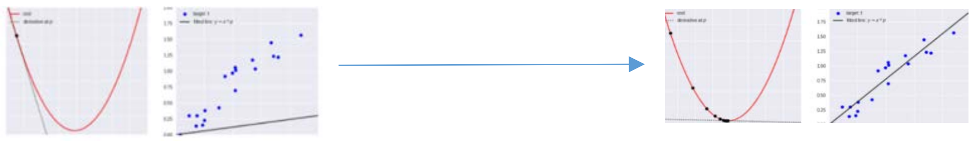
$$\frac{\partial}{\partial\theta_0}J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)$$

$$\frac{\partial}{\partial\theta_1}J(\theta_0,\theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x^{(i)}$$

Then gradient decent uses these gradients to update the parameters.

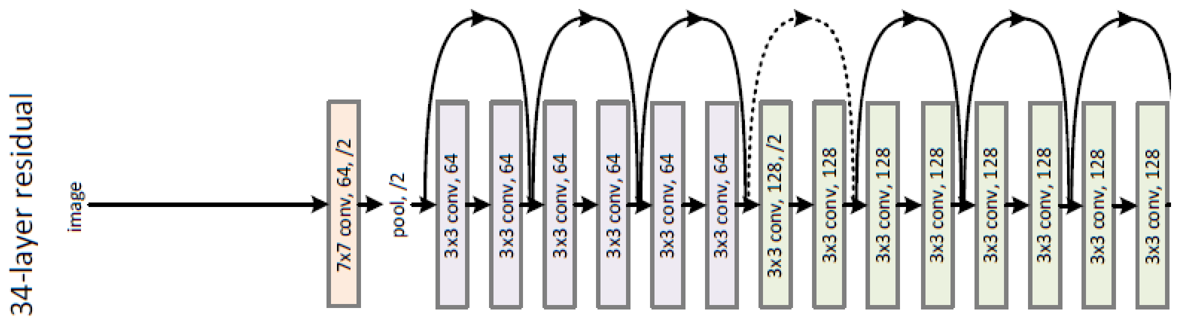
```
repeat until convergence {
    $\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)$
    $\theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\cdot x^{(i)}$
}
```

Then, by using the new theta 0, and theta 1 resulting from gradient decent, we will go from the figure from left to the one to right.



The problem is that when the gradients vanishes (close to zero) or explode (become huge) GD is not able to improve theta 0, and theta 1, so we are not getting the convergence show in the previous Graph.

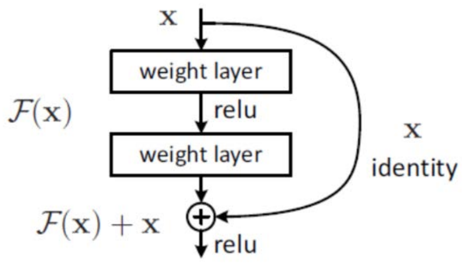
This video of Andre Ng provides nice intuition about the problem of vanishing/exploding gradients, in the context of fully connected networks.
https://www.youtube.com/watch?v=qhXZsFVxGKo



HOW ResNet address the problem of vanishing/exploding gradients?

1. Normalizing the input
2. Using Bash normalization (BN) right after each convolution and before activation, following
3. Using Short shortcut connections

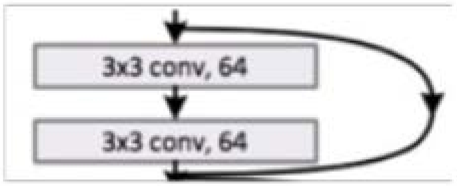
The idea behind shortcut connection is that is easier to learn the residual than the original function. But before let's take a look here to this structure. Here, we have three layers, one per color (orange, purple, and green) . The interesting ones are the light purple, and the **green one**. Each layer is compose for blocks (each loop). Inside, each block we have some operation like convolutions.

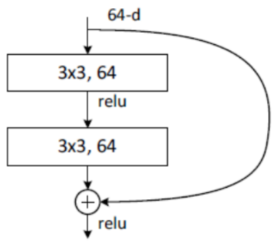


The main idea here is that is easier to learn *H(x)= F(x) + x than Just F(x) (this is actually residual learning). Part of your project is to do some research and provide an intuition about why is easier to learn H(x) than F(x). But, a good intuition.*

Note, that we have two types of shortcuts (the solid ones, and the dot ones).

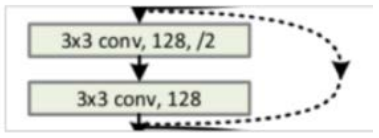
<u>Let's take a look of solid one (A identity shortcut)</u>

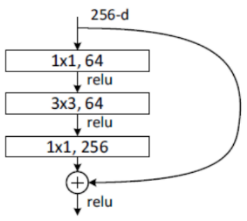




Here, we add the output of the previous layer to the output of the block. In this basic block, the size of input of the previous layer match the size of the output layer.

Let’s take a look of dot one or projection shortcut (A identity shortcut)





The projection shortcut performs a convolution operation to ensure the volumes at this addition operation are the same size. In other words, fix the problem of miss-matching sizes between the input X and the F(X) the block output. There are two option 1) padding the input volume or perform 1x1 convolutions as shown previous picture right or 2) down sampling performed by increasing the stride to 2 previous pictures left.

A detail explanation here.

https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8

and here:

https://towardsdatascience.com/resnets-for-cifar-10-e63e900524e0

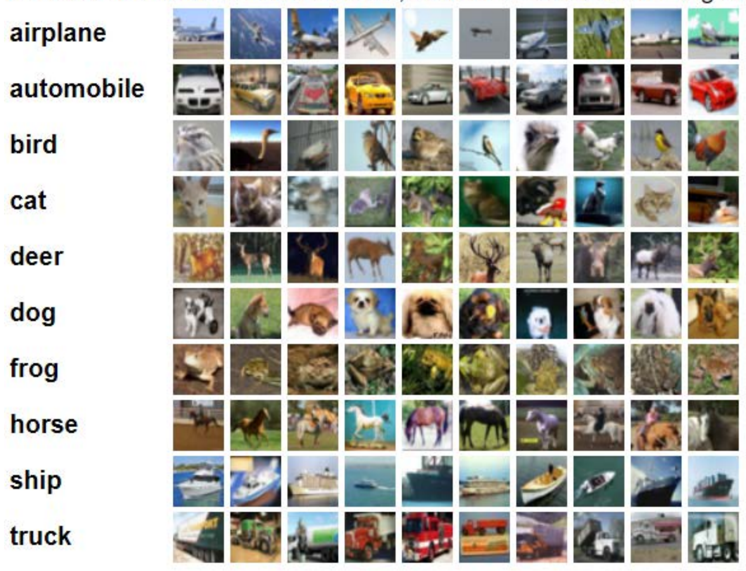
Let’s start by following the next table, Assuming a input RGB picture of 224 x 224 pixel by 3 channels.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Data set.

In this dataset the images are 32 by 32 so instead of 224 x 224 x 3 of the paper example using ImageNet your input will be 32 x 32 x 3 using CIFAR 10.

Here are the classes in the dataset, as well as 10 random images from each:



You need to recompute the numbers in the table for the new dataset.