

Estudiante: Ricardo Andrey Sánchez Delgado
Carné: 2014010852

Tutoriales

1. [Image Recognition](#)

Conclusiones

- El reconocimiento de imágenes y su clasificación son tareas que dan sus resultados con base en probabilidades, puede que no exista un 100% de que algo es algo, no obstante se pueden obtener porcentajes muy buenos para problemas que no sean tan estrictos.
- Es preferible que las imágenes que se desean clasificar tengan un único objeto clasificable, ya que si se tiene 2 o más puede que la respuesta sea errónea.

Cómo podría aplicarlo a un problema real

Esto definitivamente es un problema que existe en el mundo real, sería de gran ayuda en lo que se refiere a clasificación de imágenes por ejemplo en el área médica, se podría lograr un ahorro de trabajo y brindar resultados más rápido a los pacientes y con esto incluso salvarles la vida.

De igual manera hay otra clase de problemas los cuales se pueden resolver, como es el análisis estadístico de imágenes que fueron tomadas automáticamente y de ahí inferir información que sea útil.

Screenshots de resultados

2. [Image Retraining](#)

Conclusiones

- En caso de que se quiera entrenar una red neuronal, los datos con lo que se hará esto no son fácilmente conseguibles dado que se debe de tener una clasificación anticipada de todos estos datos, además de que la información debe de encontrarse normalizada para que el programa pueda realizar su lectura correctamente.
- A la hora de normalizar los datos, estos se deben de normalizar inteligentemente, ya que el tamaño de los archivos, las variables a tomar en cuenta, puede hacer que el tiempo de entrenamiento sea muy extenso.

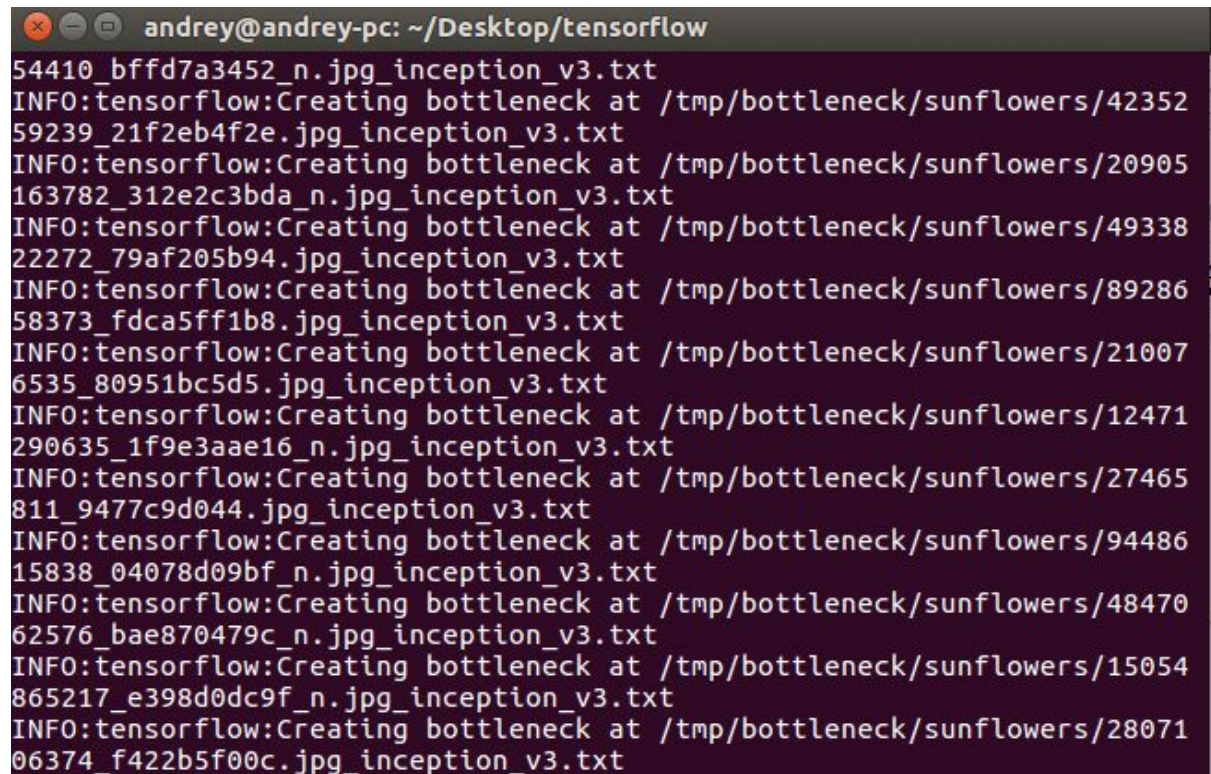
Cómo podría aplicarlo a un problema real

Se debe de buscar un problema el cual pueda generar información la cual pueda ser clasificada. Hoy en día existe el ejemplo de Snapchat, la cual es una red social la cual puede aplicar filtros, básicamente si el programa detecta una cara y el usuario toma una foto entonces se está confirmando a snapchat qué es una cara, a como se puede observar es

un problema que alimenta una red y que lo hace de forma automática con ayuda de los usuarios, esto lo hace muy rentable.

También igual que en el caso anterior existe el caso en el que se puede entrenar con imágenes que existan en los hospitales, imágenes las cuales hayan sido clasificadas, entonces una vez se le dio un pronóstico acertado a un usuario entonces se puede afirmar que la imagen es de este tipo y puede utilizarse para entrenar la red neuronal.

Screenshots de resultados

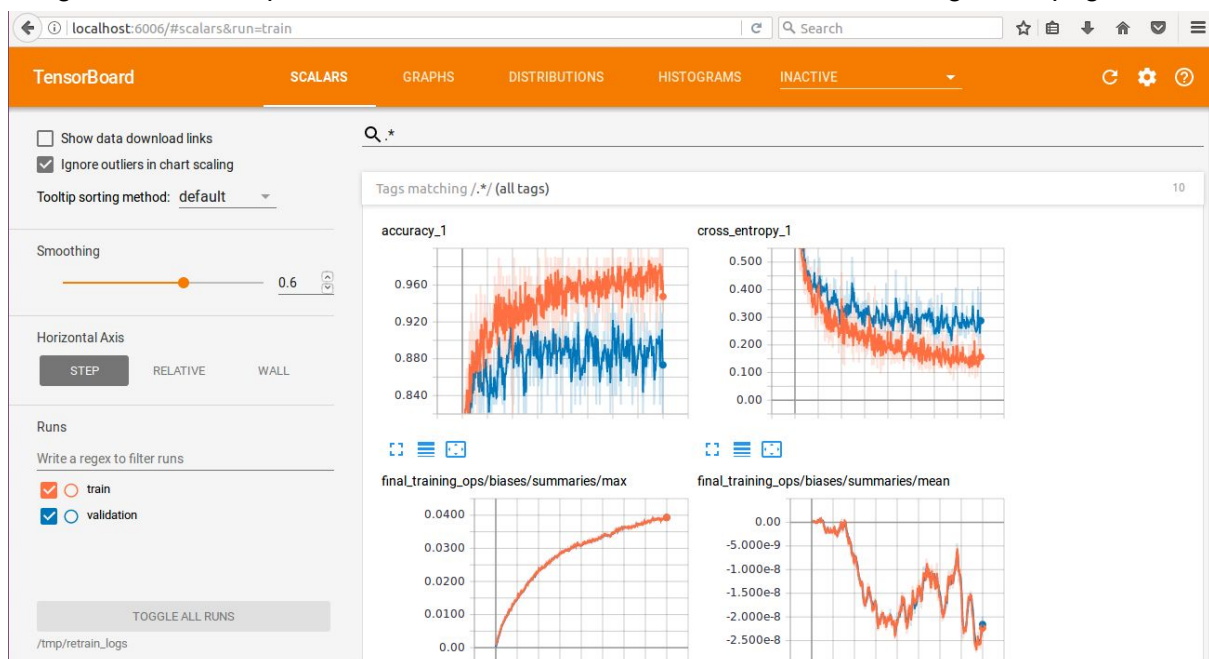


```
andrey@andrey-pc: ~/Desktop/tensorflow
54410_bffd7a3452_n.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/sunflowers/42352
59239_21f2eb4f2e.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/sunflowers/20905
163782_312e2c3bda_n.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/sunflowers/49338
22272_79af205b94.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/sunflowers/89286
58373_fdca5ff1b8.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/sunflowers/21007
6535_80951bc5d5.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/sunflowers/12471
290635_1f9e3aae16_n.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/sunflowers/27465
811_9477c9d044.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/sunflowers/94486
15838_04078d09bf_n.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/sunflowers/48470
62576_bae870479c_n.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/sunflowers/15054
865217_e398d0dc9f_n.jpg_inception_v3.txt
INFO:tensorflow:Creating bottleneck at /tmp/bottleneck/sunflowers/28071
06374_f422b5f00c.jpg_inception_v3.txt
```

```
andrey@andrey-pc: ~/Desktop/tensorflow
98.0%
INFO:tensorflow:2017-11-10 11:53:12.842111: Step 2700: Cross entropy =
0.258848
INFO:tensorflow:2017-11-10 11:53:13.042297: Step 2700: Validation accur
acy = 90.0% (N=100)
INFO:tensorflow:2017-11-10 11:53:15.215103: Step 2710: Train accuracy =
97.0%
INFO:tensorflow:2017-11-10 11:53:15.215530: Step 2710: Cross entropy =
0.149880
INFO:tensorflow:2017-11-10 11:53:15.442196: Step 2710: Validation accur
acy = 87.0% (N=100)
INFO:tensorflow:2017-11-10 11:53:17.568778: Step 2720: Train accuracy =
94.0%
INFO:tensorflow:2017-11-10 11:53:17.569302: Step 2720: Cross entropy =
0.205299
INFO:tensorflow:2017-11-10 11:53:17.818842: Step 2720: Validation accur
acy = 86.0% (N=100)
INFO:tensorflow:2017-11-10 11:53:20.533238: Step 2730: Train accuracy =
98.0%
INFO:tensorflow:2017-11-10 11:53:20.533589: Step 2730: Cross entropy =
0.110307
INFO:tensorflow:2017-11-10 11:53:20.762021: Step 2730: Validation accur
acy = 86.0% (N=100)
INFO:tensorflow:2017-11-10 11:53:23.127534: Step 2740: Train accuracy =
96.0%

INFO:tensorflow:2017-11-10 11:58:29.883259: Step 3999: Validation accur
acy = 87.0% (N=100)
INFO:tensorflow:Final test accuracy = 91.9% (N=372)
INFO:tensorflow:Froze 2 variables.
Converted 2 variables to const ops.
andrey@andrey-pc:~/Desktop/tensorflow$
```

De igual manera se pueden ver los resultados del entrenamiento en la siguiente página:



3. [A Guide to TF Layers: Building a Convolutional Neural Network](#)

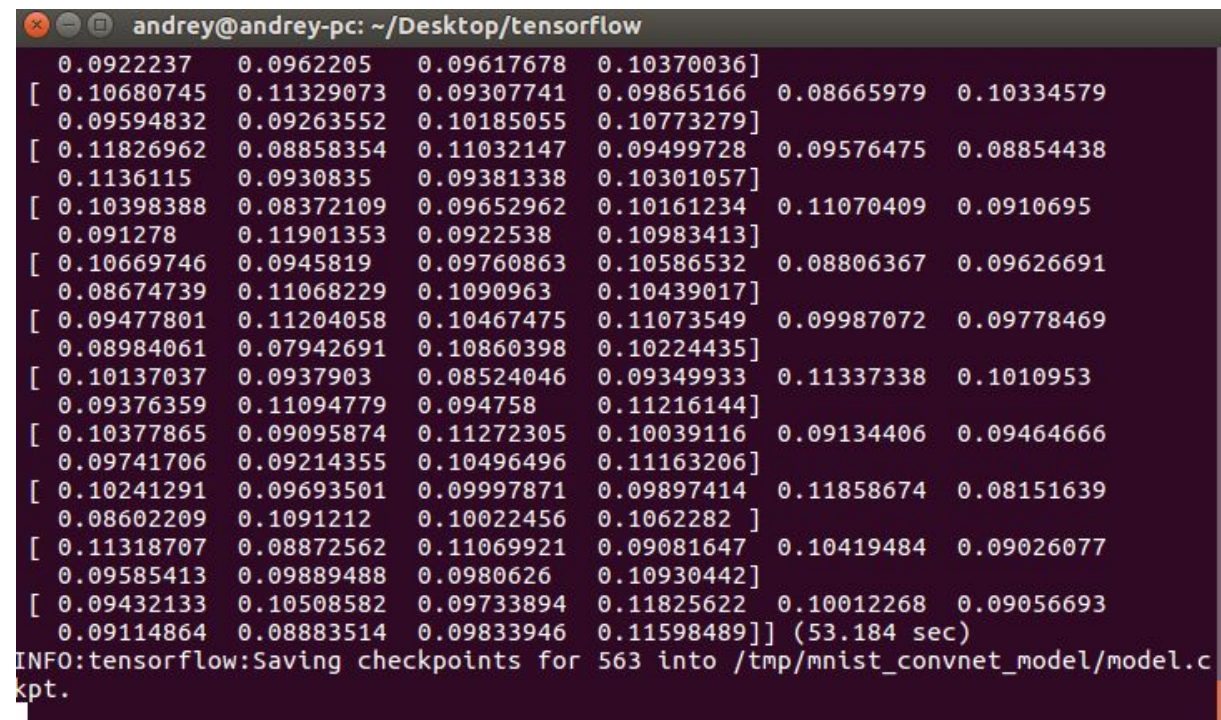
Conclusiones

- Para crear una red neuronal se debe de tener un conocimiento abstracto del sistema del sistema que se quiere crear muy bueno, saber mucho hablando de aspectos técnicos también
-

Cómo podría aplicarlo a un problema real

Análogamente a los casos anteriores existe una gran cantidad de problemas a los que se podría aplicar este tutorial, ejemplo de esto podría ser con los famosos captchas los cuales los seres humanos clasifican, pero análogamente al ejemplo expuesto en el tutorial (que es identificar una serie de números escritos a mano) esto se puede generalizar a letras y diferentes tipos de fuentes. y así facilitar la transcripción de imágenes a texto.

Screenshots de resultados



```
andrey@andrey-pc: ~/Desktop/tensorflow
0.0922237 0.0962205 0.09617678 0.10370036]
[ 0.10680745 0.11329073 0.09307741 0.09865166 0.08665979 0.10334579
0.09594832 0.09263552 0.10185055 0.10773279]
[ 0.11826962 0.08858354 0.11032147 0.09499728 0.09576475 0.08854438
0.1136115 0.0930835 0.09381338 0.10301057]
[ 0.10398388 0.08372109 0.09652962 0.10161234 0.11070409 0.0910695
0.091278 0.11901353 0.0922538 0.10983413]
[ 0.10669746 0.0945819 0.09760863 0.10586532 0.08806367 0.09626691
0.08674739 0.11068229 0.1090963 0.10439017]
[ 0.09477801 0.11204058 0.10467475 0.11073549 0.09987072 0.09778469
0.08984061 0.07942691 0.10860398 0.10224435]
[ 0.10137037 0.0937903 0.08524046 0.09349933 0.11337338 0.1010953
0.09376359 0.11094779 0.094758 0.11216144]
[ 0.10377865 0.09095874 0.11272305 0.10039116 0.09134406 0.09464666
0.09741706 0.09214355 0.10496496 0.11163206]
[ 0.10241291 0.09693501 0.09997871 0.09897414 0.11858674 0.08151639
0.08602209 0.1091212 0.10022456 0.1062282 ]
[ 0.11318707 0.08872562 0.11069921 0.09081647 0.10419484 0.09026077
0.09585413 0.09889488 0.0980626 0.10930442]
[ 0.09432133 0.10508582 0.09733894 0.11825622 0.10012268 0.09056693
0.09114864 0.08883514 0.09833946 0.11598489]] (53.184 sec)
INFO:tensorflow:Saving checkpoints for 563 into /tmp/mnist_convnet_model/model.ckpt.
```

```

andrey@andrey-pc: ~/Desktop/tensorflow
0.00000085 0.00135554 0.00036054 0.99696869]
[ 0.00006547 0.00000002 0.00000117 0.00001539 0.0000531 0.00006779
0.00000002 0.9704324 0.00125305 0.02811165]
[ 0.00011233 0.00015004 0.00001695 0.01950749 0.0004152 0.95190167
0.01313469 0.00000014 0.0140341 0.00072743]
[ 0.01714906 0.0181533 0.10817594 0.03321341 0.00034095 0.02783495
0.13708226 0.00000865 0.65797538 0.00006611]
[ 0.00002068 0.0000216 0.00001721 0.00005718 0.98233485 0.00340687
0.00009754 0.00078575 0.0003621 0.0128961 ]
[ 0.00000369 0.00000169 0.00000012 0.00054534 0.00000259 0.99863786
0.00000231 0.00000926 0.00004596 0.00075113]
[ 0.00019939 0.00000252 0.00000158 0.0004209 0.00000053 0.98756999
0.00114751 0.00000003 0.01065162 0.00000598]] (54.032 sec)
INFO:tensorflow:Saving checkpoints for 20000 into /tmp/mnist_convnet_model/model.ckpt.
INFO:tensorflow:Loss for final step: 0.0944601.
INFO:tensorflow:Starting evaluation at 2017-11-11-06:32:19
INFO:tensorflow:Restoring parameters from /tmp/mnist_convnet_model/model.ckpt-20000
INFO:tensorflow:Finished evaluation at 2017-11-11-06:32:55
INFO:tensorflow:Saving dict for global step 20000: accuracy = 0.9695, global_step = 20000, loss = 0.0992864
{'loss': 0.099286444, 'global_step': 20000, 'accuracy': 0.96950001}
andrey@andrey-pc:~/Desktop/tensorflow$

```

4. [Convolutional Neural Networks](#)

Conclusiones

- el tiempo de entrenamiento de una red neuronal con imágenes depende del problema puede durar mucho tiempo.

Ejemplo aplicado a la vida real

Igual que en el caso anterior este modelo sirve para crear redes neuronales pero en este caso un poco más sofisticadas, por lo que se puede decir que es posible utilizarlas para machine learning para cualquier tipo de imágenes, en casos anteriores de dio ejemplos como el de snapchat, en este caso se puede utilizar para clasificar imágenes de matrículas por ejemplo y así ver quien rompe las leyes de tránsito.

Screenshots


```
andrey@andrey-pc: ~/Desktop/models/tutorials/image/cifar10
c/batch)
2017-11-11 14:25:12.318252: step 20740, loss = 0.88 (56.4 examples/sec; 2.270 se
c/batch)
2017-11-11 14:25:34.907884: step 20750, loss = 0.79 (56.7 examples/sec; 2.259 se
c/batch)
2017-11-11 14:25:57.321435: step 20760, loss = 0.80 (57.1 examples/sec; 2.241 se
c/batch)
2017-11-11 14:26:20.017984: step 20770, loss = 0.67 (56.4 examples/sec; 2.270 se
c/batch)
2017-11-11 14:26:42.629573: step 20780, loss = 0.74 (56.6 examples/sec; 2.261 se
c/batch)
2017-11-11 14:27:05.332118: step 20790, loss = 0.90 (56.4 examples/sec; 2.270 se
c/batch)
2017-11-11 14:27:28.263883: step 20800, loss = 0.82 (55.8 examples/sec; 2.293 se
c/batch)
2017-11-11 14:27:50.842919: step 20810, loss = 0.74 (56.7 examples/sec; 2.258 se
c/batch)
2017-11-11 14:28:13.508263: step 20820, loss = 0.75 (56.5 examples/sec; 2.267 se
c/batch)
2017-11-11 14:28:36.349808: step 20830, loss = 0.87 (56.0 examples/sec; 2.284 se
c/batch)
2017-11-11 14:29:00.298389: step 20840, loss = 0.93 (53.4 examples/sec; 2.395 se
c/batch)
```

```
andrey@andrey-pc: ~/Desktop/models/tutorials/image/cifar10
Filling queue with 20000 CIFAR images before starting to train. This will take a
few minutes.
2017-11-11 00:49:45.047311: I tensorflow/core/platform/cpu_feature_guard.cc:137]
Your CPU supports instructions that this TensorFlow binary was not compiled to
use: SSE4.1 SSE4.2 AVX AVX2 FMA
2017-11-11 00:49:57.835936: step 0, loss = 4.67 (96.7 examples/sec; 1.324 sec/ba
tch)
2017-11-11 00:50:21.022842: step 10, loss = 4.62 (55.2 examples/sec; 2.319 sec/b
atch)
2017-11-11 00:50:44.134070: step 20, loss = 4.49 (55.4 examples/sec; 2.311 sec/b
atch)
2017-11-11 00:51:07.155113: step 30, loss = 4.63 (55.6 examples/sec; 2.302 sec/b
atch)
2017-11-11 00:51:30.038958: step 40, loss = 4.32 (55.9 examples/sec; 2.288 sec/b
atch)
2017-11-11 00:51:53.052915: step 50, loss = 4.27 (55.6 examples/sec; 2.301 sec/b
atch)
2017-11-11 00:52:16.002323: step 60, loss = 4.13 (55.8 examples/sec; 2.295 sec/b
atch)
2017-11-11 00:52:39.144835: step 70, loss = 4.16 (55.3 examples/sec; 2.314 sec/b
atch)
2017-11-11 00:53:02.123623: step 80, loss = 4.07 (55.7 examples/sec; 2.298 sec/b
atch)
2017-11-11 00:53:25.133295: step 90, loss = 4.17 (55.6 examples/sec; 2.301 sec/b
```

```
andrey@andrey-pc: ~/Desktop/models/tutorials/image/cifar10
c/batch)
2017-11-11 13:06:58.052856: step 18680, loss = 0.64 (56.5 examples/sec; 2.264 se
c/batch)
2017-11-11 13:07:20.794817: step 18690, loss = 0.86 (56.3 examples/sec; 2.274 se
c/batch)
2017-11-11 13:07:43.577731: step 18700, loss = 0.74 (56.2 examples/sec; 2.278 se
c/batch)
2017-11-11 13:08:06.198516: step 18710, loss = 0.81 (56.6 examples/sec; 2.262 se
c/batch)
2017-11-11 13:08:28.945498: step 18720, loss = 0.81 (56.3 examples/sec; 2.275 se
c/batch)
2017-11-11 13:08:51.500341: step 18730, loss = 0.76 (56.8 examples/sec; 2.255 se
c/batch)
2017-11-11 13:09:14.145715: step 18740, loss = 0.84 (56.5 examples/sec; 2.265 se
c/batch)
2017-11-11 13:09:36.747197: step 18750, loss = 0.77 (56.6 examples/sec; 2.260 se
c/batch)
2017-11-11 13:09:59.284987: step 18760, loss = 0.82 (56.8 examples/sec; 2.254 se
c/batch)
2017-11-11 13:10:21.881854: step 18770, loss = 0.95 (56.6 examples/sec; 2.260 se
c/batch)
2017-11-11 13:10:46.431143: step 18780, loss = 0.91 (52.1 examples/sec; 2.455 se
c/batch)
```

Dado que era un millón de pasos entonces a modo de ejemplo se probó el mismo ejemplo pero con solamente 200 pasos para que se pueda observar que si corre exitosamente el programa.

5. [Vector Representations of Words](#)

Conclusiones

- Siempre que sea posible se deben de elegir las entradas que consuman menos recursos para que así el entrenamiento sea rápido

Ejemplo aplicado a la vida real

En este caso se pueden utilizar versiones digitales de libros para averiguar patrones de escritura entre ellos y a partir de esto inferir información, esto también se puede hacer con números por ejemplo de información de una ciudad para así facilitar lo que es la toma de decisiones. Con eso se puede observar que la inteligencia artificial se puede utilizar para una gran cantidad de problemas que puede ser difícil resolver simplemente con la lógica.

Screenshots


```
Average loss at step 92000 : 4.68552202749
Average loss at step 94000 : 4.72314978397
Average loss at step 96000 : 4.69173931253
Average loss at step 98000 : 4.59526500225
Average loss at step 100000 : 4.70809893656
Nearest to this: it, which, the, that, ursus, thaler, microcebus, callithrix,
Nearest to war: aztlán, gates, memoirs, criminology, imparted, accretion, roque, peacocks,
Nearest to not: still, they, generally, now, it, also, to, arbitrary,
Nearest to for: symbolics, wct, immaterial, operatorname, shatila, peacocks, pontificia, of,
Nearest to six: seven, eight, five, four, three, nine, two, dasyprocta,
Nearest to be: been, have, by, was, were, become, is, being,
Nearest to many: some, several, these, all, thaler, those, most, other,
Nearest to can: may, could, would, will, should, cannot, might, must,
Nearest to all: both, some, many, michelob, these, several, dasyprocta, ursus,
Nearest to he: it, she, they, who, there, but, never, ursus,
Nearest to five: four, seven, three, six, eight, two, zero, nine,
Nearest to such: these, well, known, many, certain, unless, ursus, abbreviated,
Nearest to is: was, has, are, although, ursus, wct, became, be,
Nearest to will: may, can, would, might, could, should, must, to,
Nearest to american: german, retroviral, english, british, microcebus, altenberg, and, char,
Nearest to had: has, have, was, were, since, moc, cebus, having,
```

```
andrey@andrey-pc: ~/Desktop/models/tutorials/image/cifar10
atch)
2017-11-11 14:34:40.195327: step 90, loss = 4.13 (45.0 examples/sec; 2.848 sec/b
atch)
2017-11-11 14:35:06.754066: step 100, loss = 4.43 (48.2 examples/sec; 2.656 sec/
batch)
2017-11-11 14:35:35.201913: step 110, loss = 4.27 (45.0 examples/sec; 2.845 sec/
batch)
2017-11-11 14:36:01.212803: step 120, loss = 4.13 (49.2 examples/sec; 2.601 sec/
batch)
2017-11-11 14:36:30.328619: step 130, loss = 3.94 (44.0 examples/sec; 2.912 sec/
batch)
2017-11-11 14:36:59.147156: step 140, loss = 3.87 (44.4 examples/sec; 2.882 sec/
batch)
2017-11-11 14:37:25.610120: step 150, loss = 4.03 (48.4 examples/sec; 2.646 sec/
batch)
2017-11-11 14:37:53.000080: step 160, loss = 4.02 (46.7 examples/sec; 2.739 sec/
batch)
2017-11-11 14:38:18.661407: step 170, loss = 4.06 (49.9 examples/sec; 2.566 sec/
batch)
2017-11-11 14:38:45.759276: step 180, loss = 4.05 (47.2 examples/sec; 2.710 sec/
batch)
2017-11-11 14:39:15.797728: step 190, loss = 4.03 (42.6 examples/sec; 3.004 sec/
batch)
andrey@andrey-pc: ~/Desktop/models/tutorials/image/cifar10$
```