Cat or Dog?

Aaron Rubenacker, Rita Tong, Tai Yu Pan

Project overview

- Binary image classification :
  - cat
  - dog
- Data available on Kaggle website
- Two models are implemented in this project

http://spcasuncoast.org/wp-content/uploads/2017/02/?SD

Softmax

- Logistic regression function

- The function contains K-dimension vector, and all entries sum up to 1

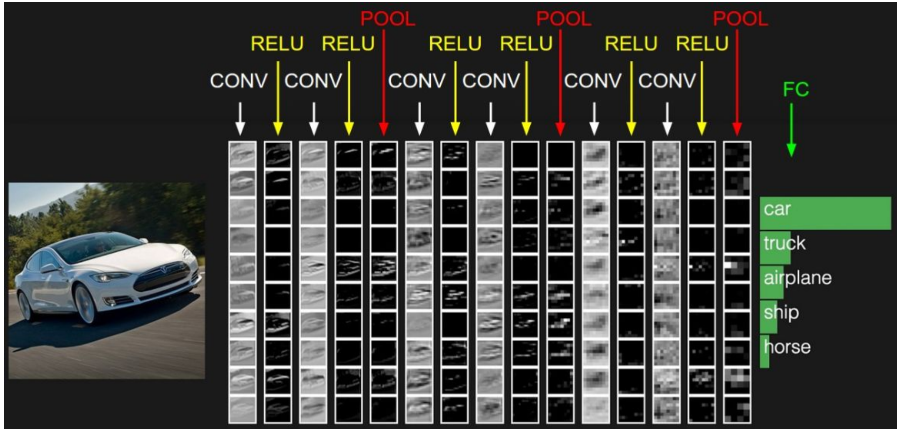
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- Multi-class classification

Convolutional Neural Networks

There are four basic layers in CNN:

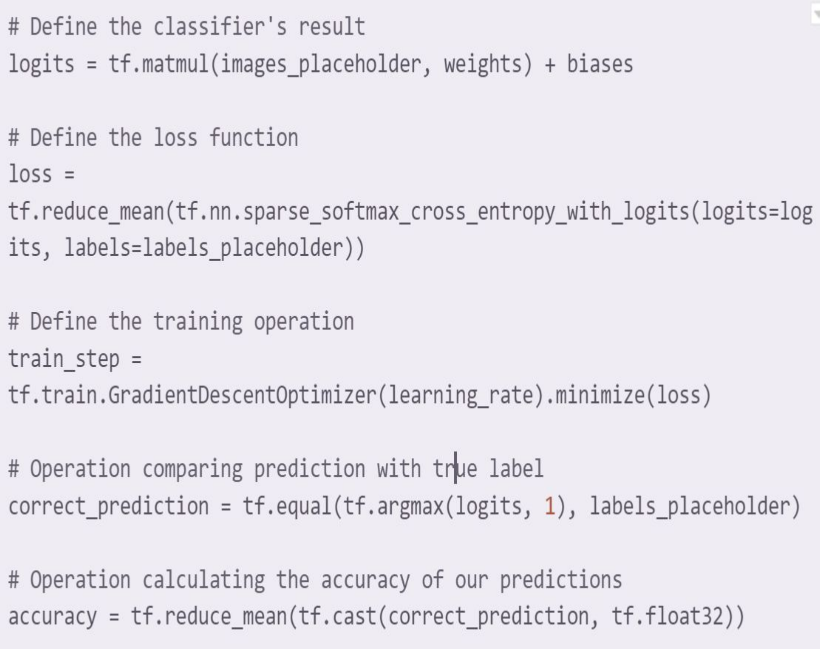
1. Convolution layer
2. ReLU function
3. Pooling layer
4. Fully connected layer



http://cs231n.github.io/convolutional-networks/

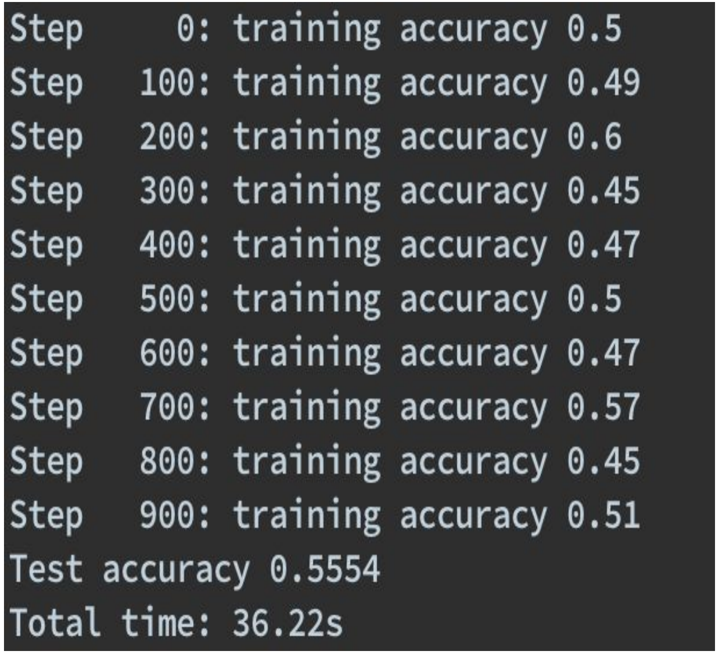
First Model - Softmax

Code



```
# Define the classifier's result
logits = tf.matmul(images_placeholder, weights) + biases

# Define the loss function
loss =
tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=log
its, labels=labels_placeholder))

# Define the training operation
train_step =
tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

# Operation comparing prediction with true label
correct_prediction = tf.equal(tf.argmax(logits, 1), labels_placeholder)

# Operation calculating the accuracy of our predictions
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Reslut



```
Step     0: training accuracy 0.5
Step   100: training accuracy 0.49
Step   200: training accuracy 0.6
Step   300: training accuracy 0.45
Step   400: training accuracy 0.47
Step   500: training accuracy 0.5
Step   600: training accuracy 0.47
Step   700: training accuracy 0.57
Step   800: training accuracy 0.45
Step   900: training accuracy 0.51
Test accuracy 0.5554
Total time: 36.22s
```

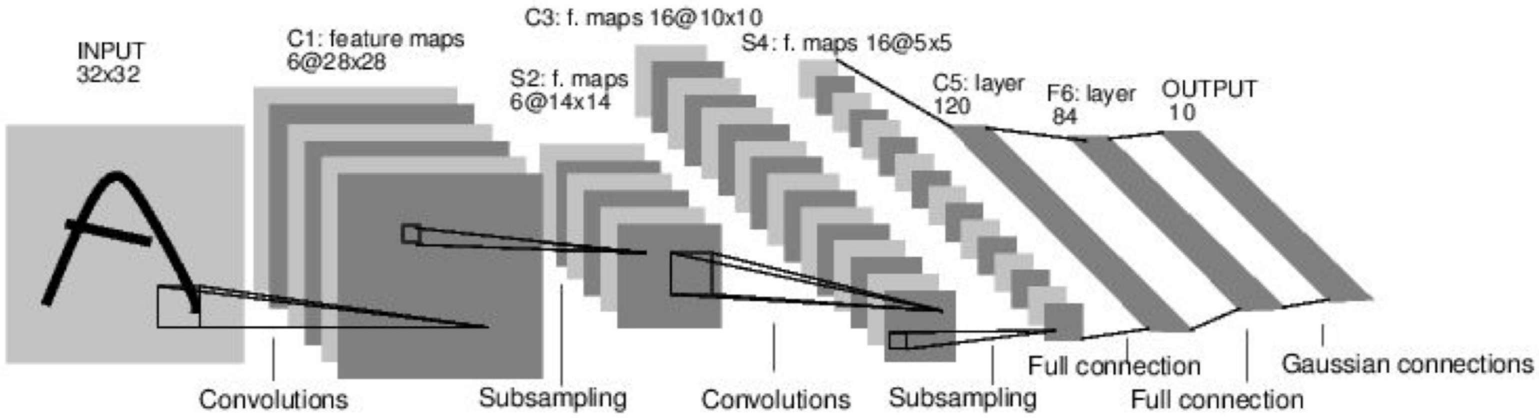
Reasons for a low accuracy

1. Colors
2. Features
3. The place of pixels

Second Model - CNN

Case Study: LeNet-5

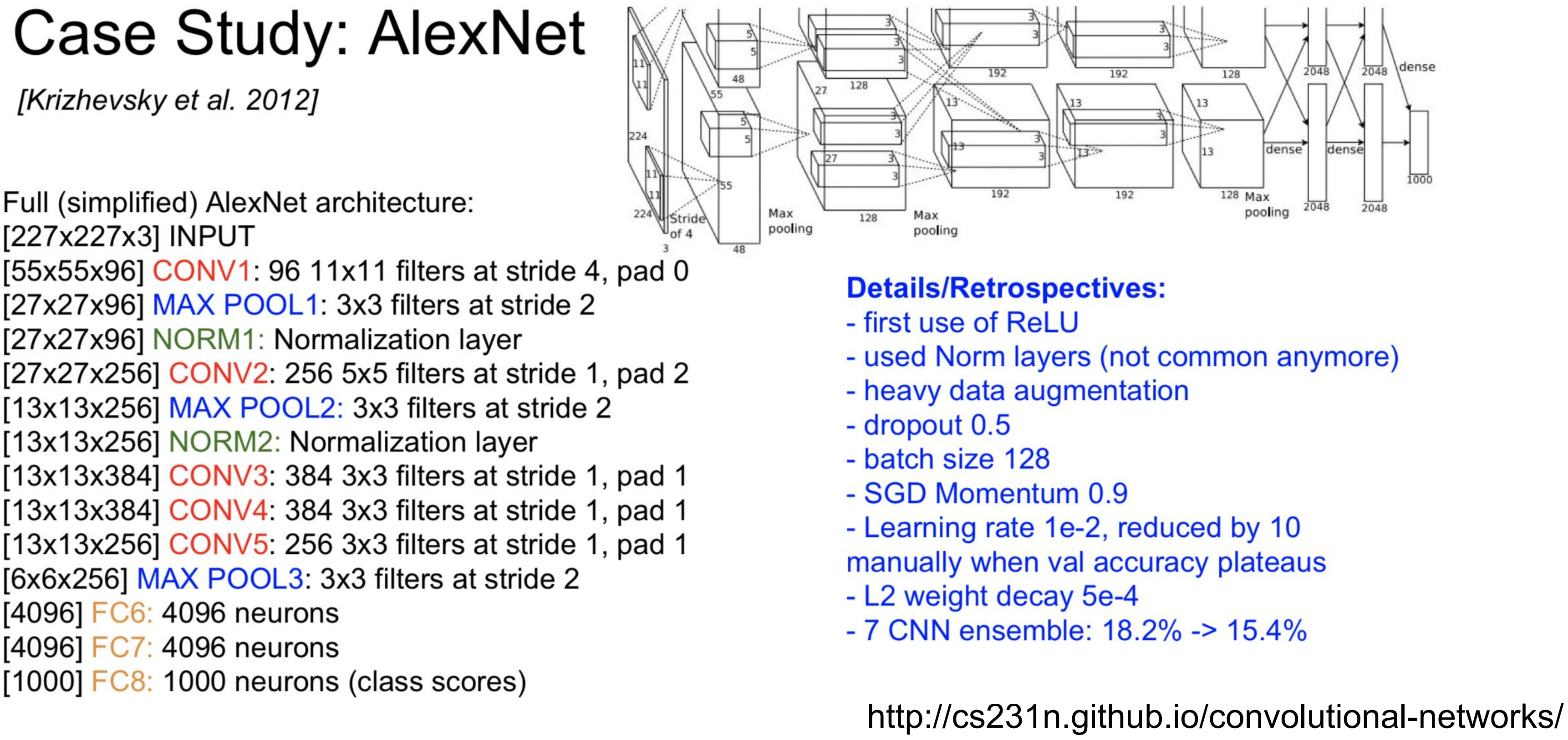
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

http://cs231n.github.io/convolutional-networks/

Second Model - CNN



Case Study: AlexNet

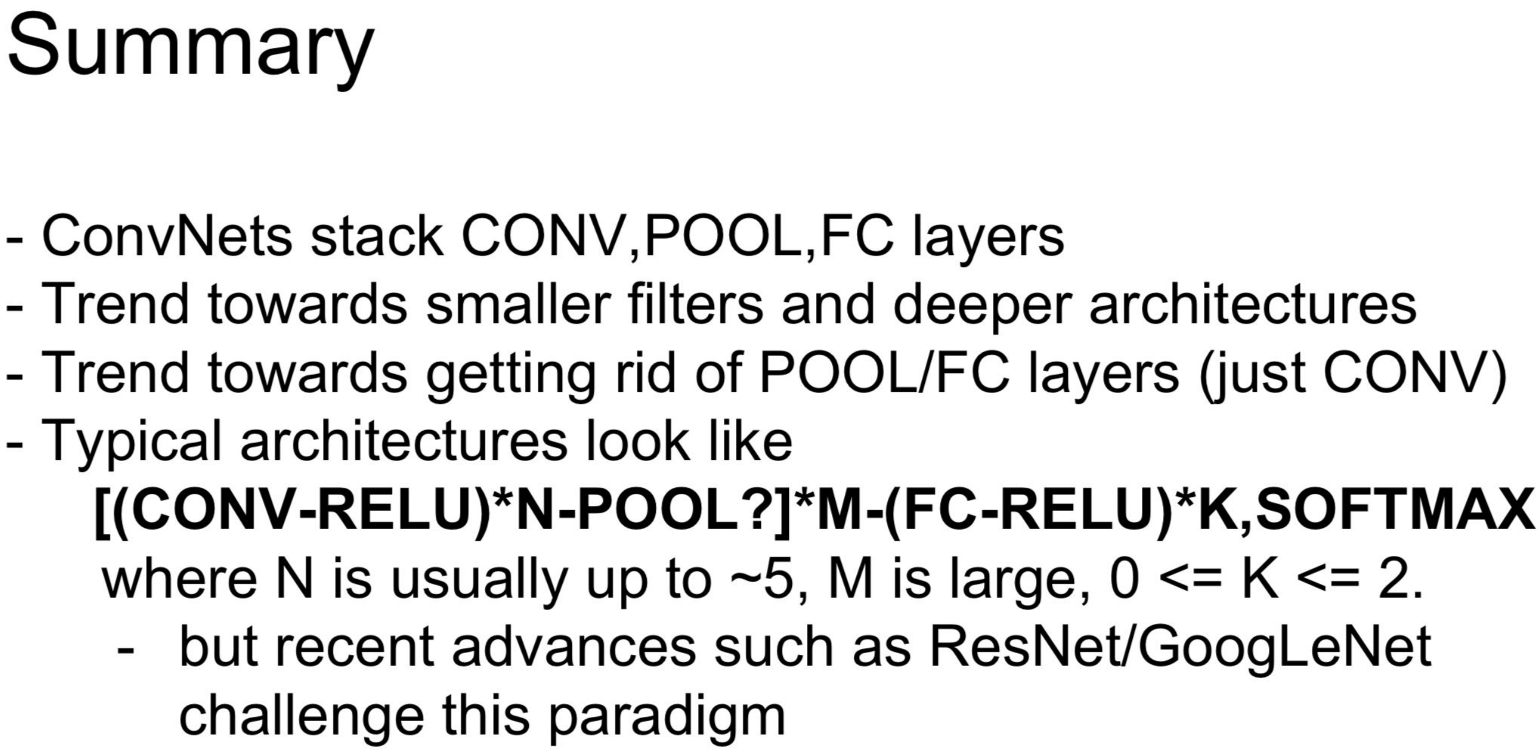
[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

http://cs231n.github.io/convolutional-networks/

Second Model - CNN



Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like

  [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX

  where N is usually up to ~5, M is large, 0 <= K <= 2.

  - but recent advances such as ResNet/GoogLeNet challenge this paradigm

http://cs231n.github.io/convolutional-networks/

Second Model - CNN

- Environments:

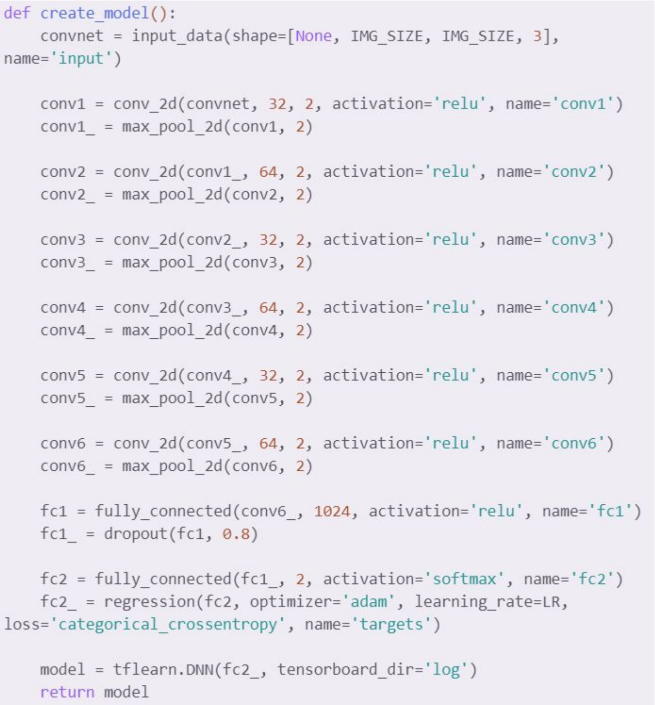
Python == 3.5.6

tensorflow == 1.10.0

tflearn == 0.3.2

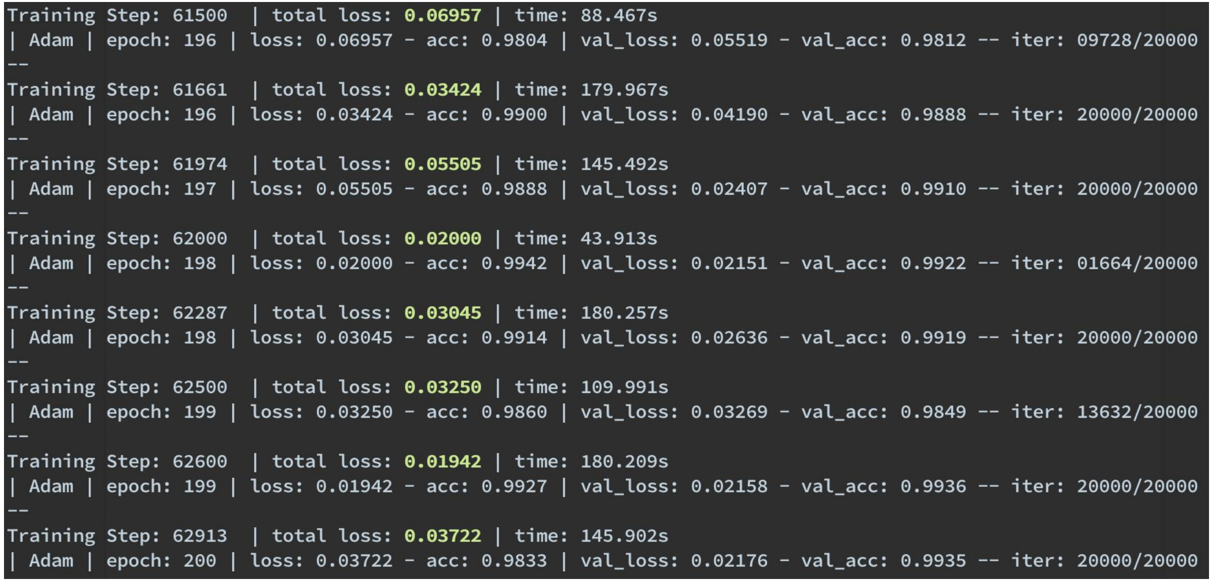
- Input image: 32*32*3

Second Model - CNN



```
def create_model():
    convnet = input_data(shape=[None, IMG_SIZE, IMG_SIZE, 3],
name='input')

    conv1 = conv_2d(convnet, 32, 2, activation='relu', name='conv1')
    conv1_ = max_pool_2d(conv1, 2)

    conv2 = conv_2d(conv1_, 64, 2, activation='relu', name='conv2')
    conv2_ = max_pool_2d(conv2, 2)

    conv3 = conv_2d(conv2_, 32, 2, activation='relu', name='conv3')
    conv3_ = max_pool_2d(conv3, 2)

    conv4 = conv_2d(conv3_, 64, 2, activation='relu', name='conv4')
    conv4_ = max_pool_2d(conv4, 2)

    conv5 = conv_2d(conv4_, 32, 2, activation='relu', name='conv5')
    conv5_ = max_pool_2d(conv5, 2)

    conv6 = conv_2d(conv5_, 64, 2, activation='relu', name='conv6')
    conv6_ = max_pool_2d(conv6, 2)

    fc1 = fully_connected(conv6_, 1024, activation='relu', name='fc1')
    fc1_ = dropout(fc1, 0.8)

    fc2 = fully_connected(fc1_, 2, activation='softmax', name='fc2')
    fc2_ = regression(fc2, optimizer='adam', learning_rate=LR,
loss='categorical_crossentropy', name='targets')

    model = tflearn.DNN(fc2_, tensorboard_dir='log')
    return model
```

Second Model - CNN

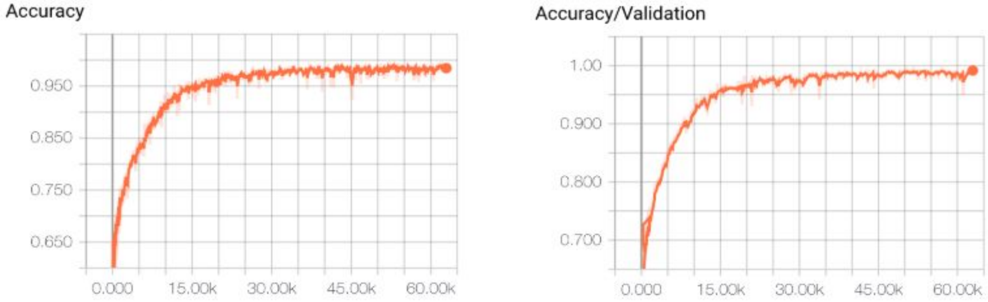


```
Training Step: 61500  | total loss: 0.06957 | time: 88.467s
| Adam | epoch: 196 | loss: 0.06957 - acc: 0.9804 | val_loss: 0.05519 - val_acc: 0.9812 -- iter: 09728/20000
--
Training Step: 61661  | total loss: 0.03424 | time: 179.967s
| Adam | epoch: 196 | loss: 0.03424 - acc: 0.9900 | val_loss: 0.04190 - val_acc: 0.9888 -- iter: 20000/20000
--
Training Step: 61974  | total loss: 0.05505 | time: 145.492s
| Adam | epoch: 197 | loss: 0.05505 - acc: 0.9888 | val_loss: 0.02407 - val_acc: 0.9910 -- iter: 20000/20000
--
Training Step: 62000  | total loss: 0.02000 | time: 43.913s
| Adam | epoch: 198 | loss: 0.02000 - acc: 0.9942 | val_loss: 0.02151 - val_acc: 0.9922 -- iter: 01664/20000
--
Training Step: 62287  | total loss: 0.03045 | time: 180.257s
| Adam | epoch: 198 | loss: 0.03045 - acc: 0.9914 | val_loss: 0.02636 - val_acc: 0.9919 -- iter: 20000/20000
--
Training Step: 62500  | total loss: 0.03250 | time: 109.991s
| Adam | epoch: 199 | loss: 0.03250 - acc: 0.9860 | val_loss: 0.03269 - val_acc: 0.9849 -- iter: 13632/20000
--
Training Step: 62600  | total loss: 0.01942 | time: 180.209s
| Adam | epoch: 199 | loss: 0.01942 - acc: 0.9927 | val_loss: 0.02158 - val_acc: 0.9936 -- iter: 20000/20000
--
Training Step: 62913  | total loss: 0.03722 | time: 145.902s
| Adam | epoch: 200 | loss: 0.03722 - acc: 0.9833 | val_loss: 0.02176 - val_acc: 0.9935 -- iter: 20000/20000
```

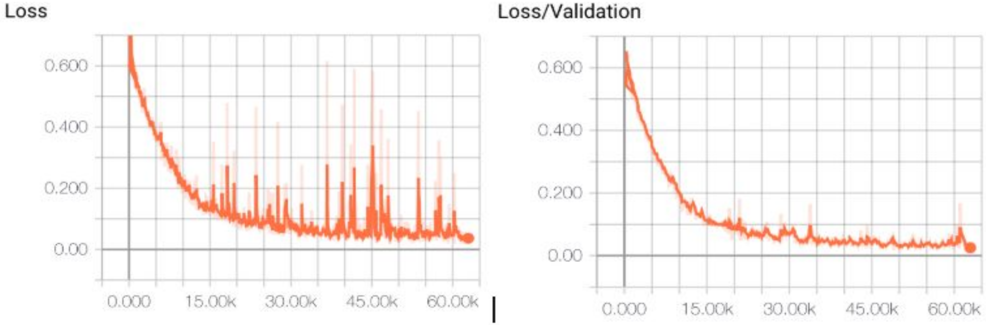
Computer Environment:
model == MacBook Pro (15-inch, 2016)
processor == 2.9 GHz Intel Core i7
Graphics == Radeon Pro 460 4096 MB
It takes in total 16 hrs 33 mins to train the model for 200 epoch containing 62913 steps.

Second Model

Accuracy vs steps on the training set and test set respectively:

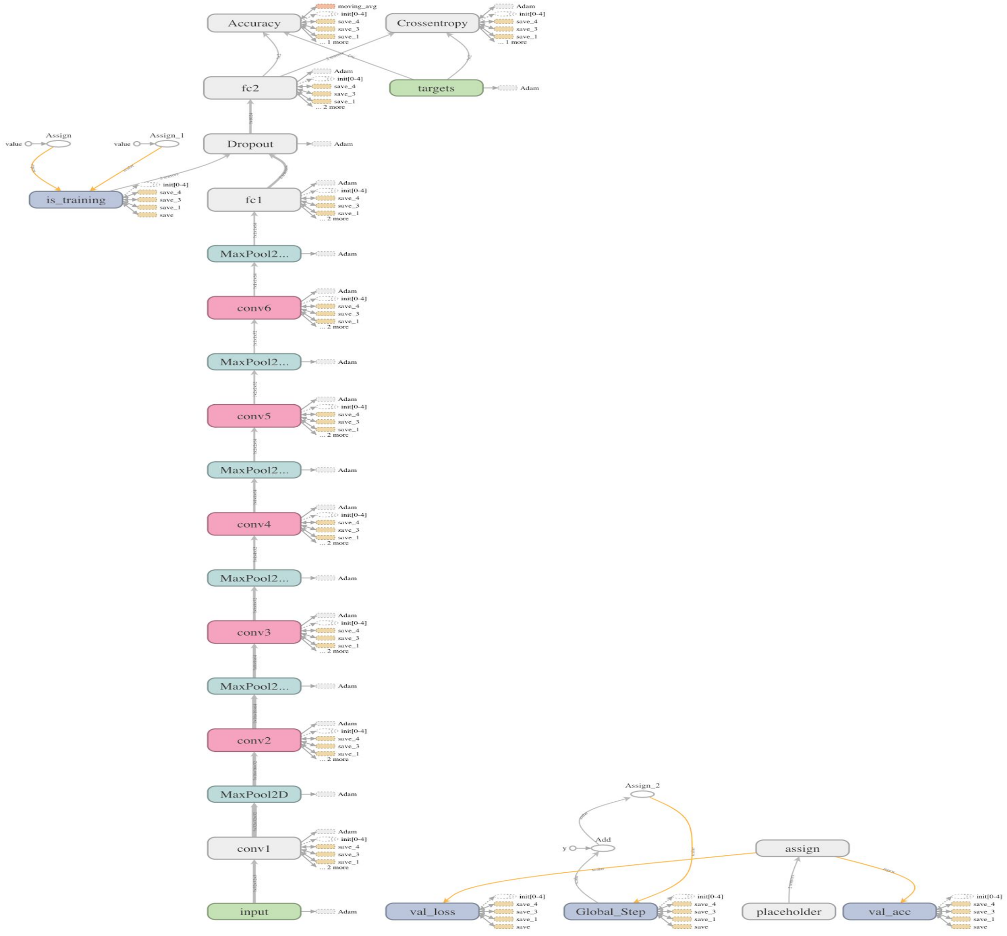


Loss values vs steps on training set and test set respectively:



Evalua

CNN Model

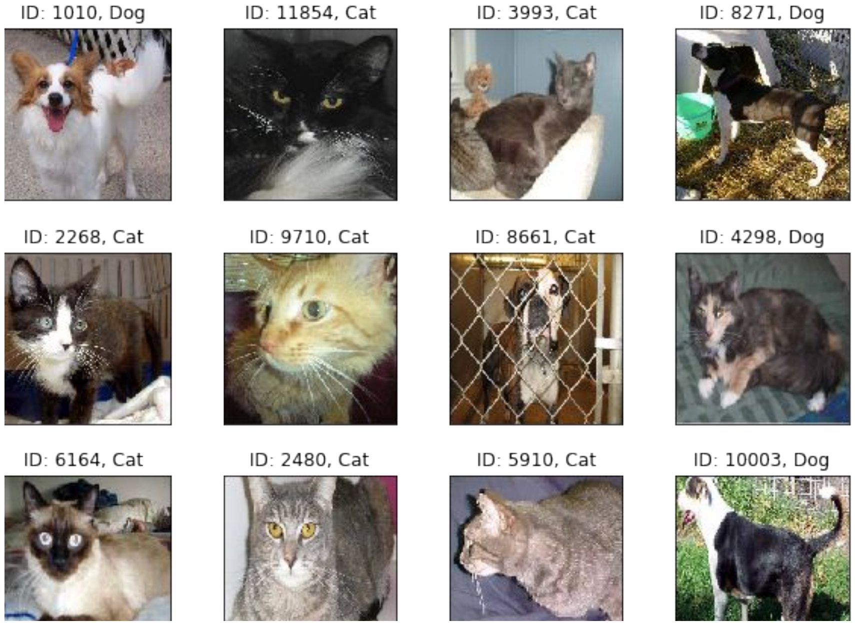


Result from CNN

1. Solve problems from Softmax
   - colors
   - features
   - place of pixels
2. 99% accuracy
3. Small images

Evaluation

Sample output:



Demo

Thank you for listening