Identifying a Whale by it’s Tail

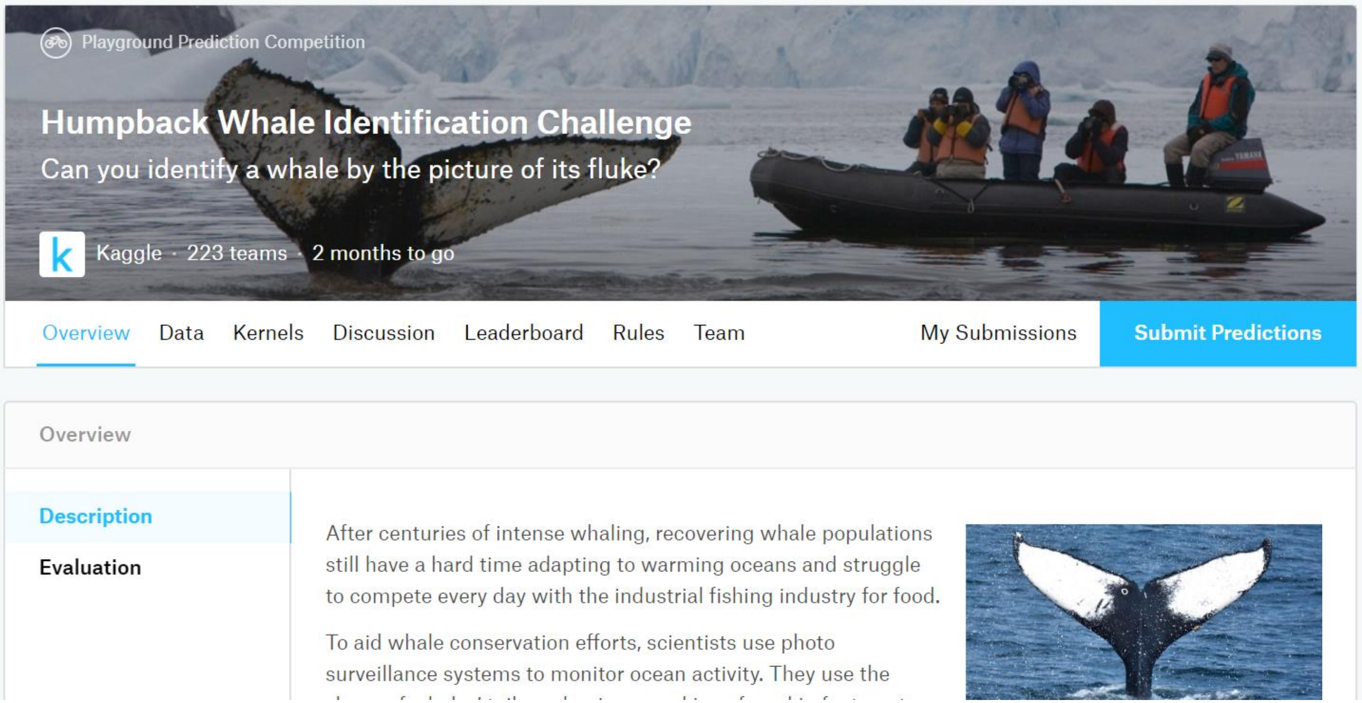
COMPARISON OF VARIOUS IMAGE CLASSIFICATION METHODS

About the Dataset

FROM KAGGLE COMPETITION

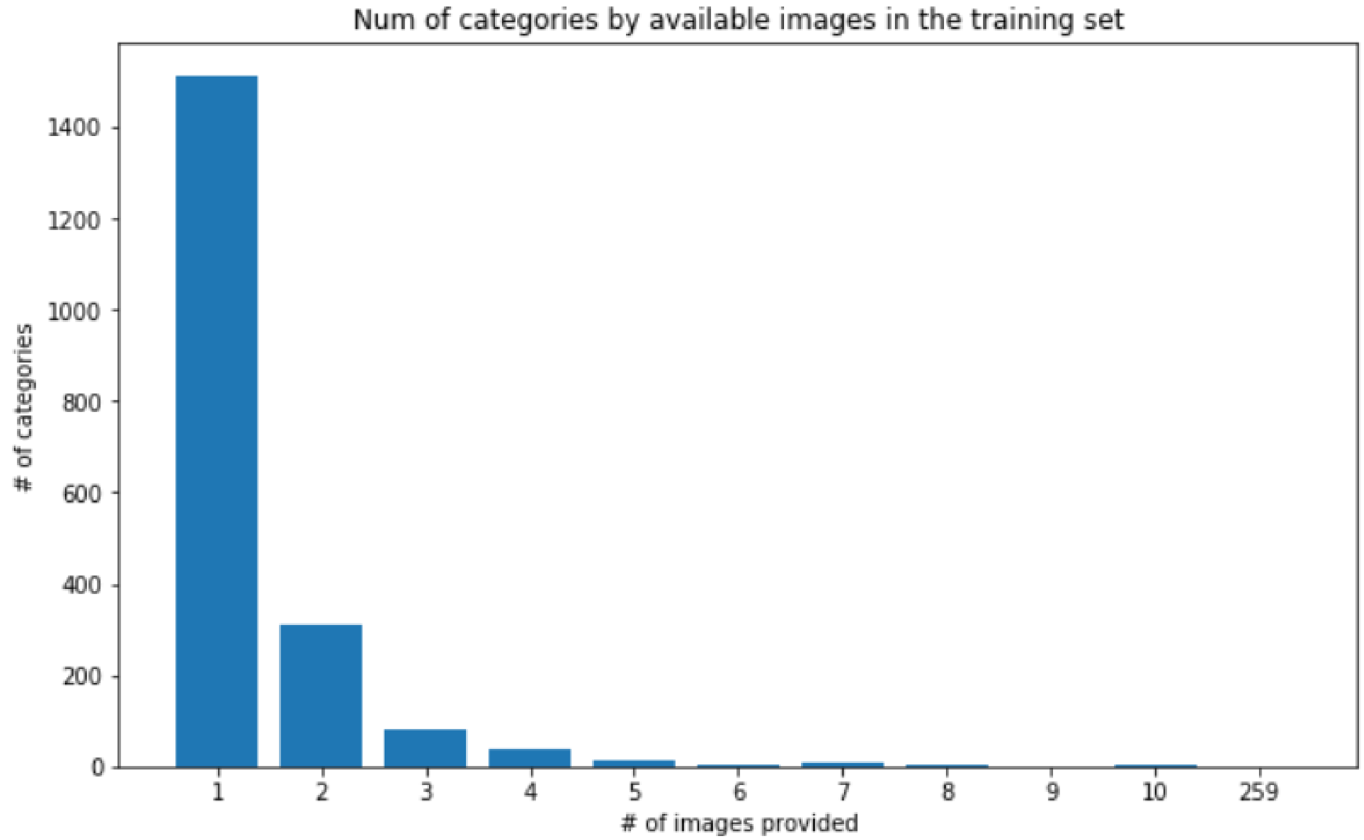
9000+ IMAGES

1900+ GROUPS



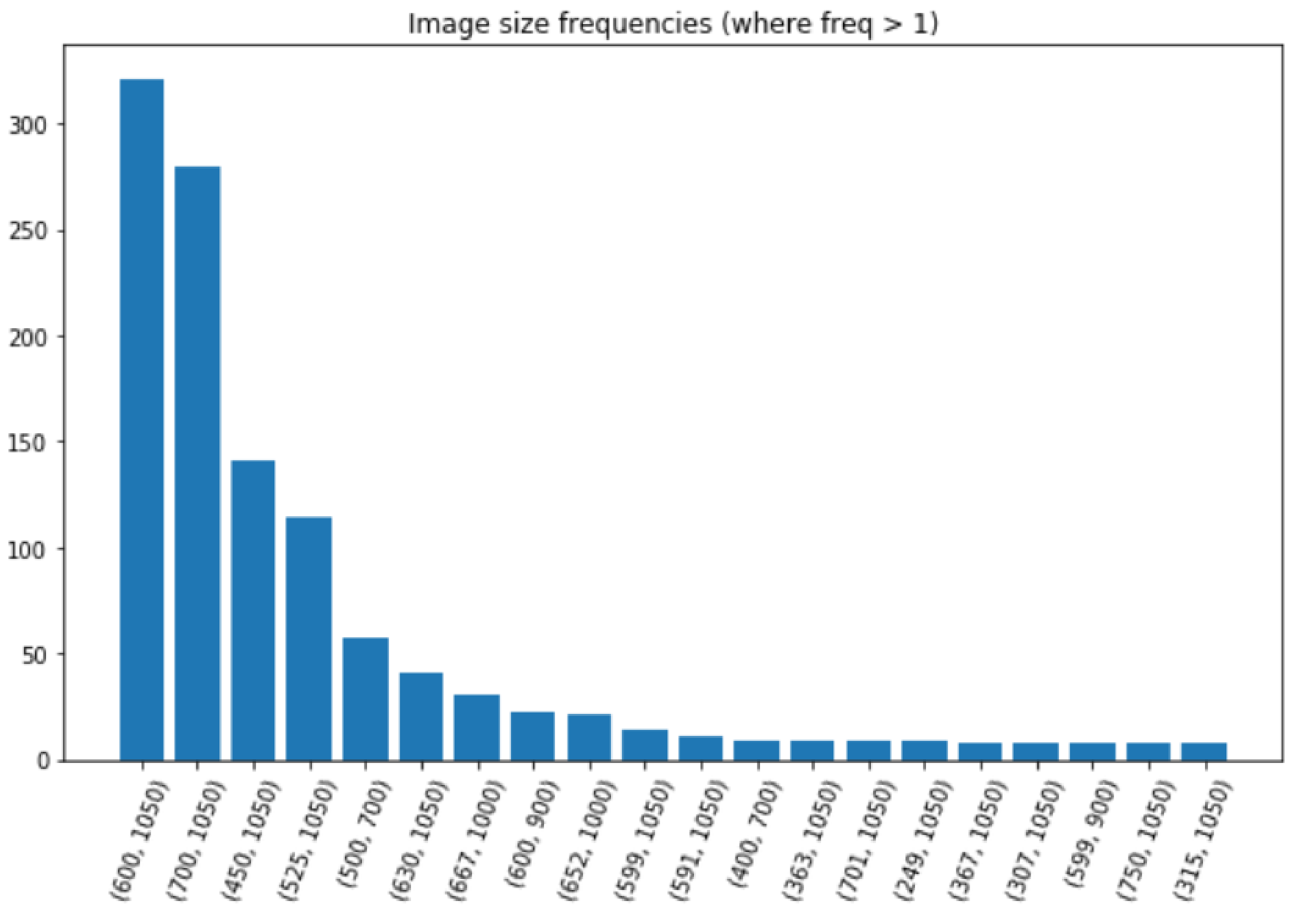
Problems

- The dataset includes a catch-all category called "new whale" which includes the vast majority of the images
- Technically speaking, you could just classify all images as new whale and be correct roughly 8% of the time
- A lot of classes contained only 1 image



More Problems

- Images varied greatly in size from 1050x1574 to 64x30
- Needed to somehow standardize size



Even More Problems

- Quite a few of the images were duplicated and placed either:
  - under the same label
  - Placed under several labels
  - placed under "new whale" in addition to other labels
- Found by translating images into hashes and grouping

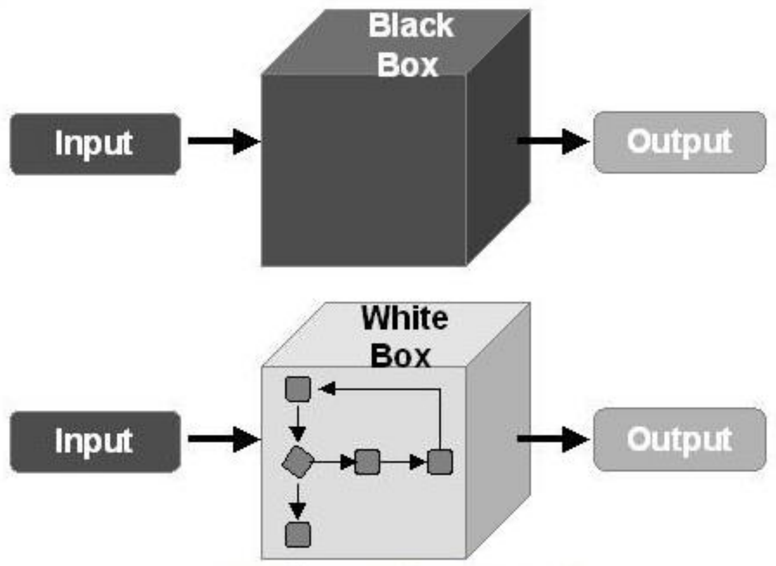
	Hash	Ids	Ids_count	Ids_contain_new_whale
bb8ec43039cb663c	3	{w_cae7677, new_whale}	2	True
9e1bc5d0bc4e0bc3	2	{new_whale, w_7185713}	2	True
af8fd0fcd3702940	2	{new_whale, w_1f09cdd}	2	True
e9889673ed9d5a02	2	{new_whale, w_a365757}	2	True
ee9ac1f47a4b8470	2	{new_whale, w_17a2610}	2	True
ed088dab92f0e3f0	2	{new_whale, w_ab4cae2}	2	True
932c5ac3a4b9ac5b	2	{w_2f54c3c}	1	False
8b90a4633b5cd29f	2	{w_dcb1f2a}	1	False
84717a9ec1a4717d	2	{w_ee948c6}	1	False
96bd2dc2c8c272f4	2	{w_5ba417d, new_whale}	2	True

Solutions for Data Preparation

- Removed the new whale category entirely
- Removed category's with < 10 images
- Removed duplicates from remainder
- Result was 48 classes which we padded with augmented data (over 2,000 images)

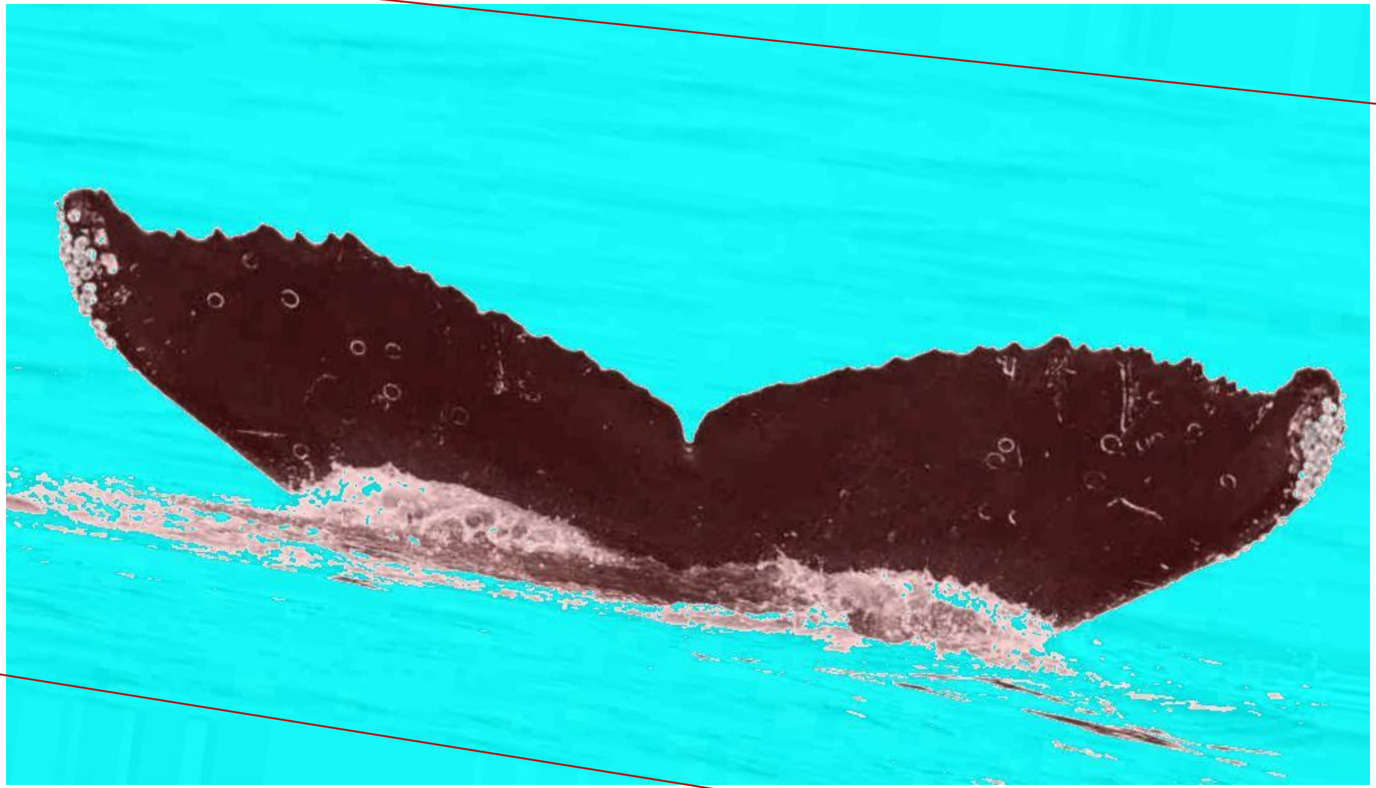
Classification Methods

- White Box Classification
  - SVM, KNN, Decision Tree, etc.
  - Needs feature extraction
  - Needs more data processing
  - More easy to be implemented
- Black Box
  - Deep Neural Network
  - No manual data extraction
  - Multiple feature methods applied automatically
  - Extremely resource intensive



Data Augmentation

- Set to high contrast and then applied greyscale
- Applied random rotations and shifts to create additional data
- Found a mean size then resized to 100 x 100 while maintaining aspect ratio
- Filled remaining area with random noise based on nearest cells



L2-Histogram of Gradients (HOG)

- HOG convolves the images with two filters that are sensitive to horizontal and vertical brightness gradients--allowing us to capture
  - Edge
  - Contour
  - Texture information
- HOG also subdivides the images into cells of a predetermined size, and computes histograms of the gradient orientations within each cell.
- Data is normalized in each cell and the result is a one dimensional feature vector made from the information in each cell of an image.
- Numpy array of (2256, 8100)

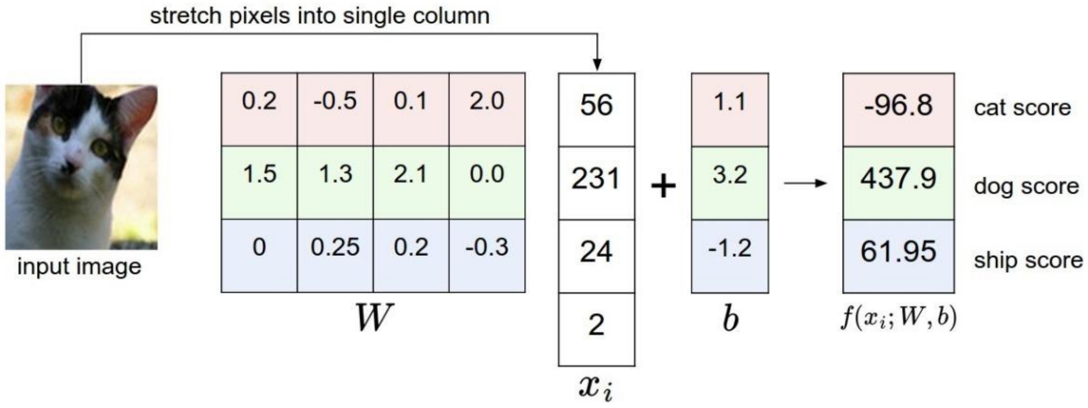
White Box Classifiers

- We started by implementing 5 basic classifiers without parameter tuning
  - K Nearest Neighbors
  - SVC (C-Support SVM). C is penalty value
  - NuSVC: similar to SVC but uses a parameter to control the number of support vectors
  - Decision Tree
  - Random Forest (messed up… supposed to use with decision tree)

```
==============================
KNeighborsClassifier
****Results****
Accuracy: 19.9409%
Log Loss: 17.849415387393172
==============================
SVC
****Results****
Accuracy: 41.8021%
Log Loss: 3.1330715145841355
==============================
NuSVC
****Results****
Accuracy: 6.6470%
Log Loss: 3.7058826960689952
==============================
DecisionTreeClassifier
****Results****
Accuracy: 10.6352%
Log Loss: 30.865523957047262
==============================
RandomForestClassifier
****Results****
Accuracy: 14.3279%
Log Loss: 19.576194929467743
==============================
```

Parameter Tuning of SVM

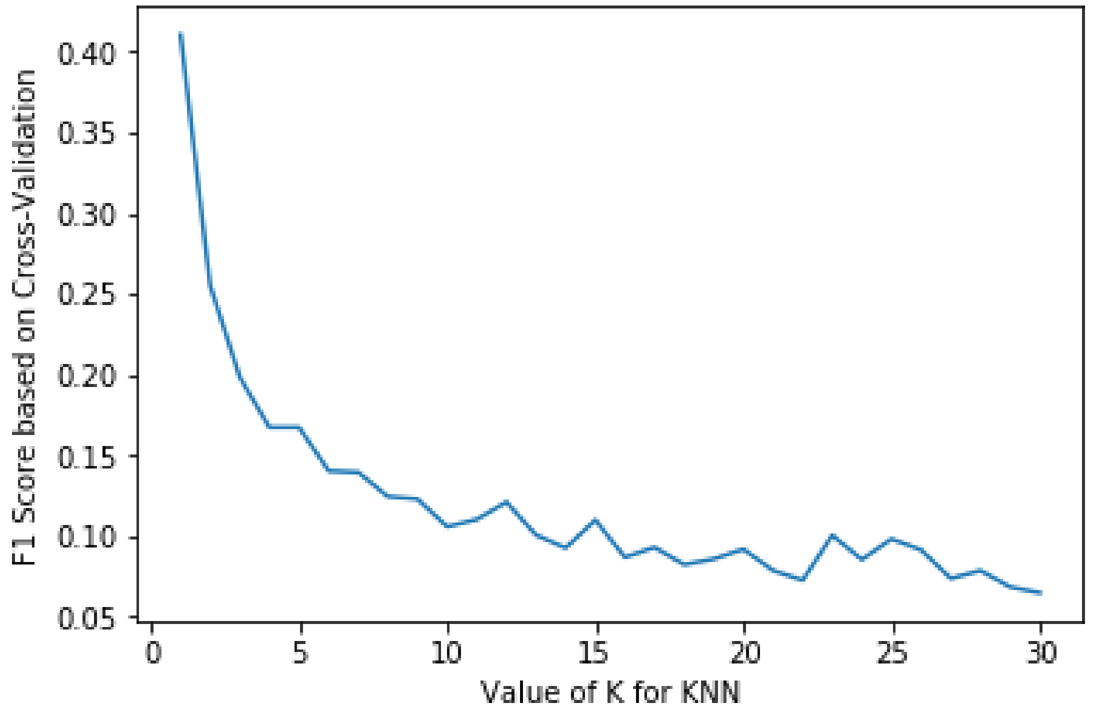
- SVM:
  - Tried both rbf and linear through Grid Search
  - Best Para: C = 100, linear kernel
  - Linear kernel: the images are stretched into high-dimensional column (8100), each image is a single point in this space. Linear Classifiers computes the score of a class as a weighted sum of all of its pixels values across the column.



Result of SVM

- F1-Score avg 0.40
- Precision avg 0.44
- Recall avg 0.41
- For 48 classes using a linear kernel it's not horrible but not good

	precision	recall	f1-score
0	0.44	0.69	0.54
1	0.33	0.39	0.36
2	0.22	0.20	0.21
3	0.40	0.33	0.36
4	0.46	0.69	0.55
5	0.25	0.08	0.12
6	0.29	0.38	0.33
7	0.39	0.54	0.45
8	0.50	0.39	0.44
9	0.20	0.08	0.12
10	0.45	0.71	0.56
11	0.80	0.80	0.80
12	0.63	0.80	0.71
13	0.27	0.31	0.29
14	0.33	0.30	0.32
15	0.30	0.73	0.42
16	0.28	0.24	0.26
17	0.33	0.25	0.29
18	0.45	0.38	0.41
19	0.75	0.27	0.40
20	0.46	0.40	0.43
21	1.00	0.17	0.29
22	0.70	0.58	0.64
23	0.25	0.29	0.27
24	0.43	0.25	0.32
25	0.54	0.54	0.54
26	0.43	0.30	0.35
27	0.50	0.14	0.22
28	0.43	0.53	0.48
29	0.62	0.67	0.64
30	0.47	0.53	0.50
31	0.45	0.53	0.49
32	0.25	0.27	0.26
33	0.40	0.20	0.27
34	0.40	0.17	0.24
35	0.67	0.27	0.38
36	0.50	0.15	0.24
37	0.67	0.55	0.60
38	0.31	0.31	0.31
39	0.42	0.29	0.34
40	0.14	0.29	0.19
41	0.08	0.10	0.09
42	0.45	0.45	0.45
43	0.33	0.46	0.39
44	0.53	0.62	0.57
45	1.00	0.27	0.43
46	0.56	0.50	0.53
47	0.40	0.36	0.38
avg / total	0.44	0.41	0.40



Parameter Tuning of KNN

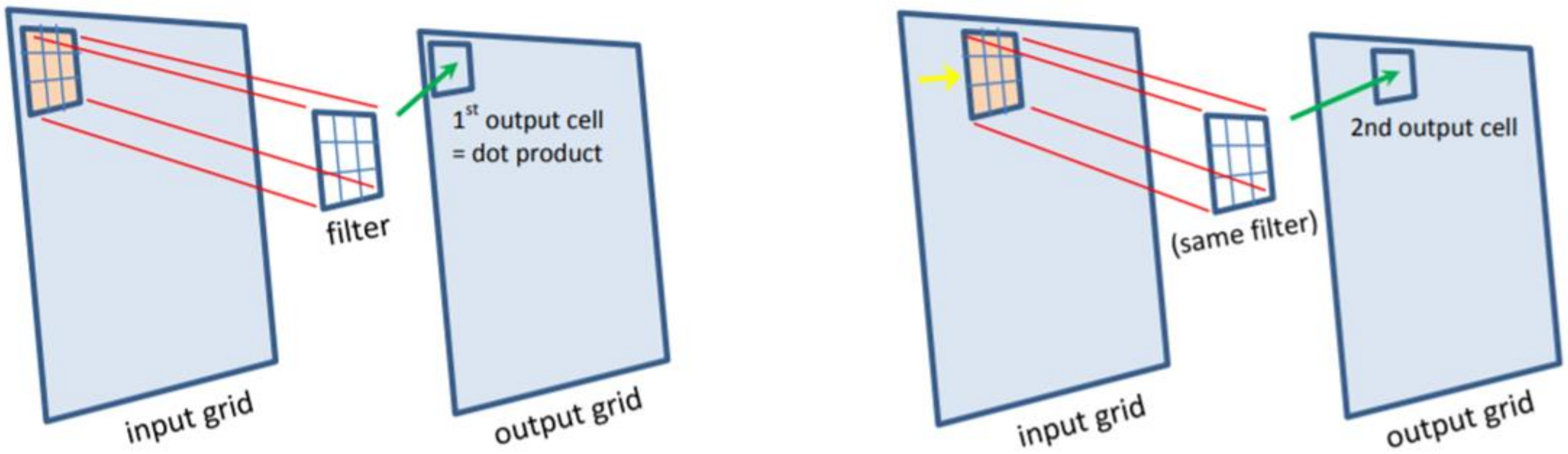
KNN Tuning Results

- F1-Score avg 0.43
- Precision avg 0.48
- Recall avg 0.43
- Better…

	precision	recall	f1-score
0	0.67	0.54	0.60
1	0.71	0.28	0.40
2	0.57	0.40	0.47
3	0.70	0.58	0.64
4	0.23	0.50	0.31
5	0.47	0.58	0.52
6	0.62	0.38	0.48
7	0.18	0.54	0.27
8	0.46	0.33	0.39
9	0.31	0.33	0.32
10	0.53	0.57	0.55
11	0.67	0.40	0.50
12	0.40	0.40	0.40
13	0.22	0.31	0.26
14	0.56	0.50	0.53
15	0.22	0.45	0.29
16	0.32	0.29	0.30
17	0.29	0.17	0.21
18	0.35	0.29	0.32
19	0.40	0.18	0.25
20	0.26	0.40	0.32
21	0.40	0.17	0.24
22	1.00	0.42	0.59
23	0.91	0.71	0.80
24	0.33	0.42	0.37
25	0.44	0.54	0.48
26	0.40	0.40	0.40
27	0.46	0.43	0.44
28	0.44	0.58	0.50
29	0.71	0.83	0.77
30	0.73	0.53	0.62
31	0.50	0.47	0.49
32	0.30	0.20	0.24
33	0.44	0.40	0.42
34	0.67	0.15	0.25
35	0.50	0.47	0.48
36	0.56	0.42	0.48
37	0.45	0.45	0.45
38	0.67	0.46	0.55
39	0.64	0.53	0.58
40	0.24	0.36	0.29
41	0.60	0.60	0.60
42	0.64	0.45	0.53
43	0.25	0.38	0.30
44	0.78	0.54	0.64
45	0.43	0.27	0.33
46	0.50	0.30	0.37
47	0.28	0.45	0.34
avg / total	0.48	0.43	0.43

Convolutional Layer

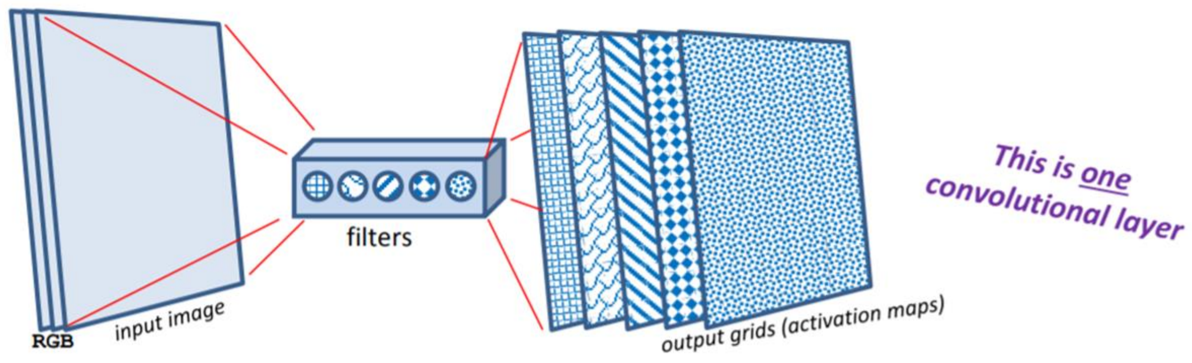
Typically used for input data that is organized into square 2D grids (such as for image recognition).
A convolutional layer has a set of some number of “filters”. Each filter performs a series of dot products:

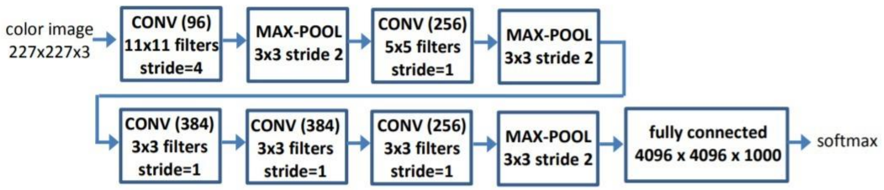


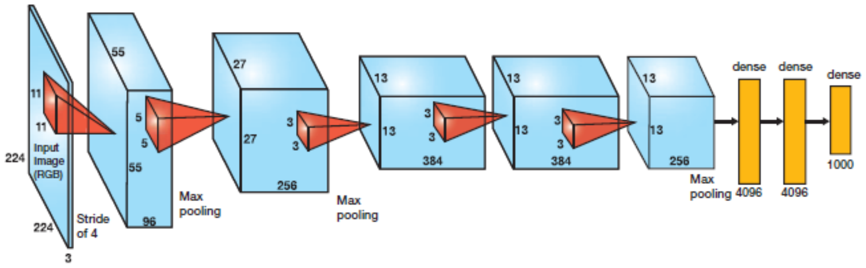
Convolutional Neural Network

CNN

- Each filter extract one feature
- Early stage detect low level features
- Train on themselves

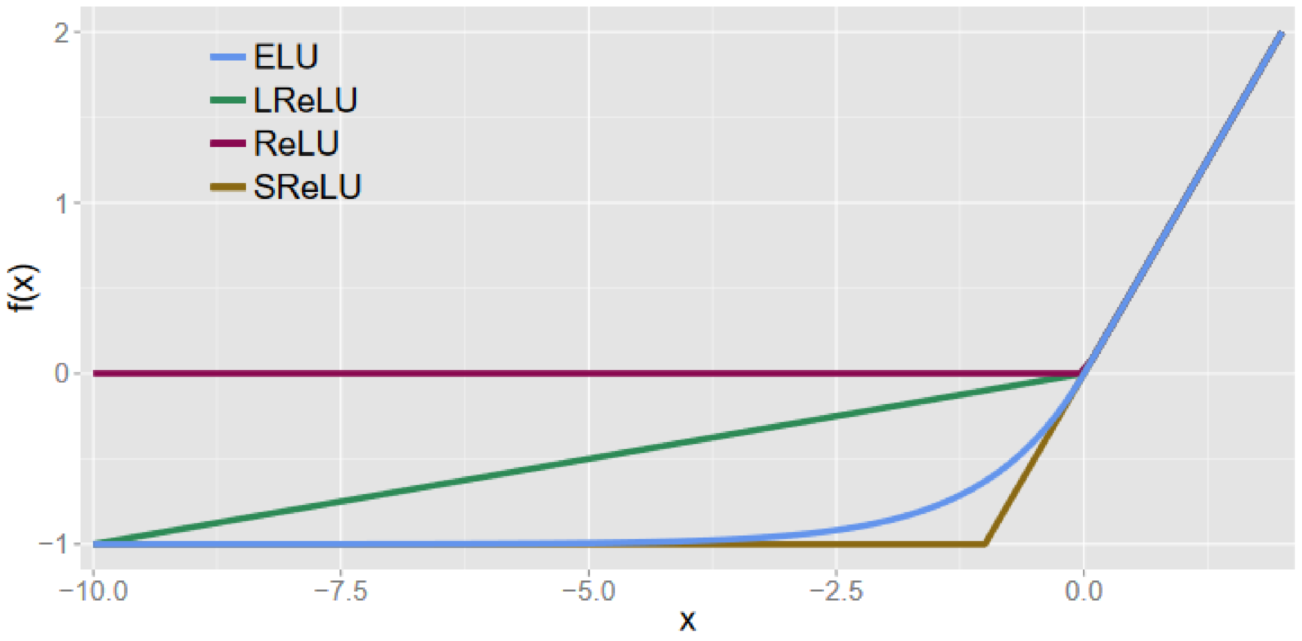



color image 227x227x3
CONV (96) 11x11 filters stride=4
MAX-POOL 3x3 stride 2
CONV (256) 5x5 filters stride=1
MAX-POOL 3x3 stride 2
CONV (384) 3x3 filters stride=1
CONV (384) 3x3 filters stride=1
CONV (256) 3x3 filters stride=1
MAX-POOL 3x3 stride 2
fully connected 4096 x 4096 x 1000
softmax


Input Image (RGB)
224
224
3
11
11
Stride of 4
55
55
96
5
5
Max pooling
27
27
256
3
3
Max pooling
13
13
384
13
13
384
13
13
256
Max pooling
dense
4096
dense
4096
dense
1000

Modifications

- Based on paper: https://arxiv.org/pdf/1511.07289.pdf
- Use Exponential Linear Units instead of Rectified ELU (RELU)
- Add extra Layer
- Supposedly good for smaller datasets



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

import tflearn
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.estimator import regression
from tflearn.metrics import Accuracy

acc = Accuracy()
network = input_data(shape=[None, 100, 100, 3])

# Conv layers ------------------------------------
network = conv_2d(network, 64, 3, strides=1, activation='elu')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 64, 3, strides=1, activation='elu')
network = max_pool_2d(network, 2, strides=2)
network = conv_2d(network, 64, 3, strides=1, activation='elu')
network = conv_2d(network, 64, 3, strides=1, activation='elu')
network = conv_2d(network, 64, 3, strides=1, activation='elu')
network = max_pool_2d(network, 2, strides=2)

# Fully Connected Layers -------------------------
network = fully_connected(network, 1024, activation='tanh')
network = dropout(network, 0.5)
network = fully_connected(network, 1024, activation='tanh')
network = dropout(network, 0.5)

# CHANGE BELOW FOR NUMBER OF CLASSES
network = fully_connected(network, 57, activation='softmax')
network = regression(network, optimizer='momentum', loss='categorical_crossentropy',
learning_rate=0.001, metric=acc)

model = tflearn.DNN(network, tensorboard_verbose=3, tensorboard_dir="logs")
```

Results

- 2 hours to run on a Nvidia 1050ti
- F1-Score avg 0.89
- Precision avg 0.89
- Recall avg 0.89

	precision	recall	f1-score
0	0.87	0.79	0.83
1	0.83	0.91	0.87
2	0.88	0.95	0.92
3	1.00	0.82	0.90
4	0.91	0.83	0.87
5	0.72	1.00	0.84
6	1.00	0.95	0.97
7	0.90	0.64	0.75
8	0.89	1.00	0.94
9	0.91	0.71	0.80
10	1.00	0.71	0.83
11	0.93	1.00	0.96
12	0.85	0.89	0.87
13	1.00	0.67	0.80
14	0.94	0.84	0.89
15	0.87	0.87	0.87
16	1.00	0.73	0.84
17	0.79	0.88	0.84
18	0.88	0.91	0.89
19	0.92	0.92	0.92
20	0.86	0.89	0.87
21	1.00	0.85	0.92
22	0.84	0.94	0.89
23	1.00	0.69	0.82
24	1.00	0.54	0.70
25	1.00	0.94	0.97
26	1.00	0.92	0.96
27	0.81	0.81	0.81
28	0.92	0.92	0.92
29	0.88	0.88	0.88
30	0.91	0.91	0.91
31	1.00	0.92	0.96
32	0.85	0.92	0.88
33	0.92	0.73	0.81
34	0.93	0.78	0.85
35	0.85	0.81	0.83
36	0.93	0.76	0.84
37	1.00	0.79	0.88
38	1.00	0.91	0.95
39	0.79	1.00	0.88
40	0.83	0.88	0.86
41	0.75	0.80	0.77
42	0.87	0.93	0.90
43	0.90	0.64	0.75
44	0.88	0.71	0.79
45	1.00	0.91	0.95
46	0.93	0.72	0.81
47	0.91	0.91	0.91
48	0.92	1.00	0.96
49	0.86	0.55	0.67
50	0.88	0.88	0.88
51	0.88	0.94	0.91
52	0.85	0.85	0.85
53	1.00	0.67	0.80
54	0.91	0.91	0.91
55	1.00	0.69	0.82
56	0.90	0.75	0.82
avg / total	0.89	0.89	0.89

Conclusions

- CNN Provide best results for image classification on this scale
- SVM and KNN apply a SoftMax and are essentially just one layer of CNN
- CNN takes a very long time even with GPU
- CNN doesn’t reveal features extracted, however more are extracted
- BE CONSISTENT WITH LIBRARIES
- If doing images, don’t do what we did. Just do CNN and maybe apply SVM at the end to improve performance