Team Raspberry - Image Classification

Kristian Wahlroos Ilkka Vähämaa Sean Lang

December 15th, 2017

Overview

Methods

Data representation

- Treat every image as a 3D-tensor (RGB)
  - Repeat the value of grayscale images three times
  - Colorized are handled as the original tensors
- Original data has 14 labels, we used 15
  - Extra one for the unclassified images
  - One-hot encoded labels

Methods

Data processing

- Read images in batches of size 2000
  - Helps to avoid filling the RAM
- Normalize the pixel values between $[0.0, 1.0]$
- For every batch augmenting the data
  - Provided by Keras
  - Centerify, shear, zoom, rotate and flip
  - To get more variation and samples from classes with few labels

Methods

Class weights 1/2

- Classes are very unbalanced

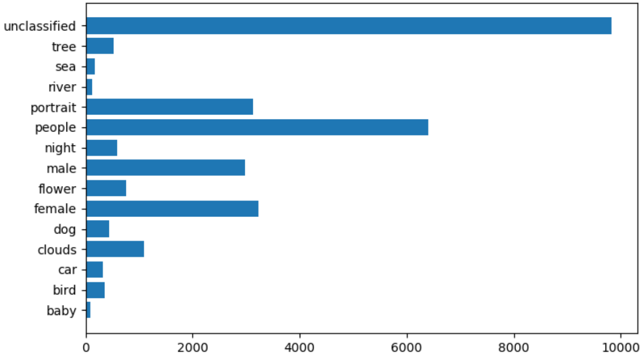


Figure: Class distribution

Methods

Class weights 2/2

- We tackled this problem by custom weights per class
  - Giving them at training phase

Class weight function

$$S(c_i; \lambda) = \ln\left(\lambda \frac{\sum_c |c|}{|c_i|}\right)$$

$$W(c_i; \lambda) = \max(S(c_i; \lambda), 1)$$

Methods

Network topology

- One network that outputs 15 classes
- Four convolution layers all followed by max pooling
  - Filters 16, 32, 32, 64
  - Kernel size 3x3
  - Max pool size 2x2
  - ReLU as activation function
- After pooling flattening via dropout to dense layer with sigmoid activation
  - Dropout value: 0.4
- Very simple network

Methods

Loss function 1/2

- Categorical crossentropy wouldn't work as one image can be in many classes
- Binary crossentropy was suggested in many forum posts
  - Still not viable solution when there are many overlapping categories
  - Loss is too forgiving for giving 0 labels

Methods

Loss function 2/2

- Solution: "custom" loss function **BP-MLL***
  - Actually taken directly from the paper [1]†
  - Designed for multi-label problems
  - Implementation for Keras can be found from internet
  - Punishes more from just giving 0 labels

$$E = \sum_{i=1}^{m} \frac{1}{|Y_i||\bar{Y}_i|} \sum_{(k,l)\in Y_i \times \bar{Y}_i} \exp(-(c_k^i - c_l^i))$$

*Backpropagation for Multilabel Learning

†[1] *Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization*, 2006

Methods

Validation

- Per batch, 10% of the data is randomly selected
- This subset is left out from the training phase
- Validated against in the final step
- With F1-score, we also inspected
  - Binary accuracy
  - Categorical accuracy
  - Hamming loss
  - Micro averaged precision score

Parametrization

Parameterization Tweaks

1. "Default"
2. Increased deeply-connected layers
3. Adagrad optimizer
4. Nadam optimizer
5. More convolutions
6. Even more convolutions
7. Reverse convolution triangle
8. Learning Rate Adjustments
   - 8.1 `lr=0.0005`
   - 8.2 `lr=0.000333`
   - 8.3 `lr=0.002`
   - 8.4 `lr=0.005`
9. Activation Functions
   - 9.1 Leaky ReLU ($\alpha = 0.3$)
   - 9.2 `tanh`

Parametrization results

Table: Parameter Tryouts

Model n.	**F1**	**HL**[‡]
1	0.462	0.125
2	0.452	0.128
3	0.459	0.125
4	0.469	0.123
5	0.462	0.124
6	0.462	0.126
7	0.463	0.125
8.1	0.464	0.124
8.2	0.457	0.126
8.3	0.465	0.124
8.4	0.457	0.125
9.1	0.083	0.796
9.2	0.378	0.184

[‡] Hamming Loss

Final System

Hyper-parameters of the final system

- Same as described in Methods section
- 1 epoch
- BP_MLL_Loss
- RMSprop optimizer (learning rate of 0.002 [twice default])

Results

- Needs some fancy plots here

Results

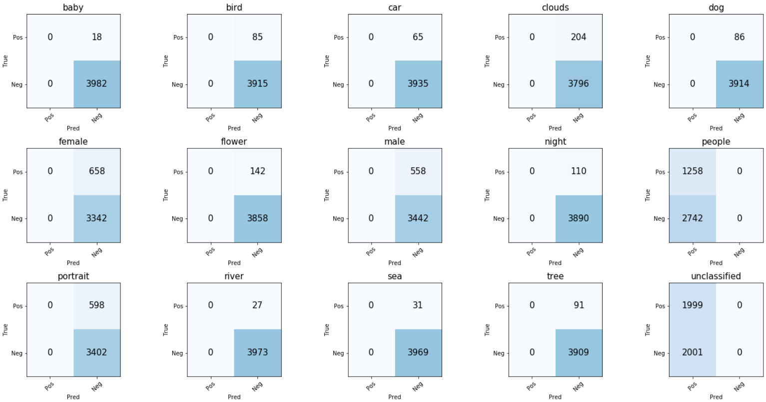
Sample Output

```
0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0
```

. . .

Results

Confusion Matrix



- Always predicts same result for each label on every picture

Results

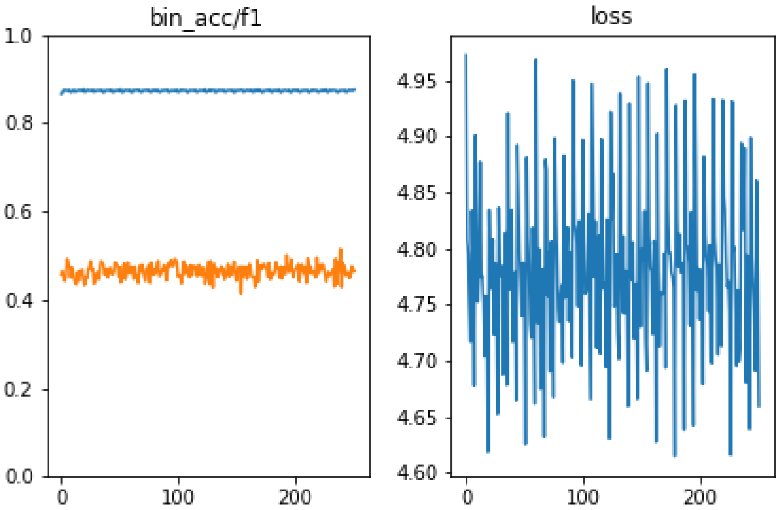


Figure: Soemthing