DEEP LEARNING

CSE 599 N1: Modern Mobile Systems

modernmobile.cs.washington.edu

Content borrowed from Tianqi Chen, Fei-Fei Li
Image Classification: A core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

This image by Nkita is licensed under CC-BY 2.0
The Problem: Semantic Gap

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)
**Distance Metric to compare images**

**L1 distance:**

\[ d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p| \]

<table>
<thead>
<tr>
<th>test image</th>
<th>training image</th>
<th>pixel-wise absolute value differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>56 32 10 18</td>
<td>10 20 24 17</td>
<td>46 12 14 1</td>
</tr>
<tr>
<td>90 23 128 133</td>
<td>8 10 89 100</td>
<td>82 13 39 33</td>
</tr>
<tr>
<td>24 26 178 200</td>
<td>12 16 178 170</td>
<td>12 10 0 30</td>
</tr>
<tr>
<td>2 0 255 220</td>
<td>4 32 233 112</td>
<td>2 32 22 108</td>
</tr>
</tbody>
</table>

Add up to get 456
Parametric Approach: Linear Classifier

\[ f(x, W) = Wx + b \]

- Image: Array of 32x32x3 numbers (3072 numbers total)
- Parameters or weights: \( W \)
- 3072x1 matrix
- 10x1 bias vector
- 10 numbers giving class scores
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Input image

- 56
- 231
- 24
- 2

Stretch pixels into column

\[
\begin{align*}
W &= \begin{bmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0 & 0.25 & 0.2 & -0.3
\end{bmatrix} \\
b &= \begin{bmatrix}
56 \\
231 \\
24 \\
2
\end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{Cat score} &= 0.2 \times 56 - 0.5 \times 231 + 0.1 \times 24 + 2.0 \times 2 + 1.1 \\
\text{Dog score} &= 1.5 \times 56 + 1.3 \times 231 + 2.1 \times 24 + 0.0 \times 2 + 3.2 \\
\text{Ship score} &= 0 \times 56 + 0.25 \times 231 + 0.2 \times 24 + (-0.3) \times 2 + (-1.2)
\end{align*}
\]

\[
\begin{align*}
\text{Cat score} &= -96.8 \\
\text{Dog score} &= 437.9 \\
\text{Ship score} &= 61.95
\end{align*}
\]
Suppose: 3 training examples, 3 classes. With some $W$ the scores $f(x, W) = WX$ are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>3.2</td>
<td>1.3</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>5.1</td>
<td>4.9</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>-1.7</td>
<td>2.0</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

A **loss function** tells how good our current classifier is. Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

where $x_i$ is image and $y_i$ is (integer) label.

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_{i} L_i(f(x_i, W), y_i)$$
Suppose: 3 training examples, 3 classes. With some \( W \) the scores \( f(x, W) = Wx \) are:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3.2</td>
<td>1.3</td>
</tr>
<tr>
<td>car</td>
<td>5.1</td>
<td>4.9</td>
</tr>
<tr>
<td>frog</td>
<td>-1.7</td>
<td>2.0</td>
</tr>
</tbody>
</table>

**Losses:** 2.9 0 12.9

**Multiclass SVM loss:**

Given an example \((x_i, y_i)\) where \(x_i\) is the image and where \(y_i\) is the (integer) label, and using the shorthand for the scores vector: \(s = f(x_i, W)\)

the SVM loss has the form:

\[
L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)
\]

Loss over full dataset is average:

\[
L = \frac{1}{N} \sum_{i=1}^{N} L_i
\]

\[
L = \frac{(2.9 + 0 + 12.9)}{3} = 5.27
\]
Regularization

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \]

**Data loss:** Model predictions should match training data

**Regularization:** Prevent the model from doing too well on training data

**Simple examples**

- **L2 regularization:** \( R(W) = \sum_k \sum_l W_{k,l}^2 \)
- **L1 regularization:** \( R(W) = \sum_k \sum_l |W_{k,l}| \)
- **Elastic net (L1 + L2):** \( R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}| \)

**More complex:**

- Dropout
- Batch normalization
- Stochastic depth, fractional pooling, etc
Regularization

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W) \]

\( \lambda \) = regularization strength (hyperparameter)

**Data loss:** Model predictions should match training data

**Regularization:** Prevent the model from doing too well on training data

Why regularize?
- Express preferences over weights
- Make the model *simple* so it works on test data
- Improve optimization by adding curvature
Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as probabilities

\[ s = f(x_i; W) \]

\[ P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \]

Softmax Function

Probabilities must be \( \geq 0 \)
Probabilities must sum to 1

\[ L_i = -\log P(Y = y_i | X = x_i) \]

\begin{align*}
\text{cat} & : 3.2 & 24.5 & 0.13 & 1.00 \\
\text{car} & : 5.1 & 164.0 & 0.87 & 0.00 \\
\text{frog} & : -1.7 & 0.18 & 0.00 & 0.00
\end{align*}

Unnormalized log-probabilities / logits
unnormalized probabilities
probabilities
Correct probs

\( \exp \) and normalize for probabilities
Neural networks: without the brain stuff

(Before) Linear score function: \( f = Wx \)

(Now) 2-layer Neural Network or 3-layer Neural Network

\[ f = W_2 \max(0, W_1 x) \]

\[ f = W_3 \max(0, W_2 \max(0, W_1 x)) \]
Activation functions

**Sigmoid**
\[ \sigma(x) = \frac{1}{1+e^{-x}} \]

**tanh**
\[ \tanh(x) \]

**ReLU**
\[ \max(0, x) \]

**Leaky ReLU**
\[ \max(0.1x, x) \]

**Maxout**
\[ \max(w_1^T x + b_1, w_2^T x + b_2) \]

**ELU**
\[ \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases} \]
Elements of Machine Learning

Model

$$x_i = \begin{bmatrix} \text{feature}_0 \\ \text{feature}_1 \\ \vdots \\ \text{feature}_m \end{bmatrix}$$

$$\hat{y}_i = \frac{1}{1 + \exp(-w^T x_i)}$$

Objective

$$L(w) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \lambda \|w\|^2$$

Training

$$w \leftarrow w - \eta \nabla_w L(w)$$
What’s Special About Deep Learning

Compositional Model

End to End Training

\[ \hat{y}_i = \frac{1}{1 + \exp(-w^T x_i)} \]
Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

Example training set

```python
def train(images, labels):
    # Machine learning!
    return model

def predict(model, test_images):
    # Use model to predict labels
    return test_labels
```
People started to use deep convolutional neural networks
Fully Connected Layer

\[ h_i = \sum_{j=1}^{5} W_{ij} x_i \]

\[ h_1 = W_{11} x_1 + W_{21} x_2 + W_{31} x_3 + W_{41} x_4 + W_{51} x_5 \]
Convolution = Spatial Locality + Sharing

Spatial Locality

Without Sharing

\[ h_i = W_{1,i}x_{i-1} + W_{2,i}x_i + W_{3,i}x_{i+1} \]

With Sharing

\[ h_i = W_1x_{i-1} + W_2x_i + W_3x_{i+1} \]
Convolution with Multiple Channels

Source: http://cs231n.github.io/convolutional-networks/
Pooling Layer

Can be replaced by strided convolution

224x224x64

pool

112x112x64

downsampling

112

224

Single depth slice

\[
\begin{bmatrix}
1 & 1 & 2 & 4 \\
5 & 6 & 7 & 8 \\
3 & 2 & 1 & 0 \\
1 & 2 & 3 & 4 \\
\end{bmatrix}
\]

max pool with 2x2 filters and stride 2

\[
\begin{bmatrix}
6 & 8 \\
3 & 4 \\
\end{bmatrix}
\]
Challenges: From LeNet to AlexNet

- Need much more data: ImageNet
- A lot more computation burdens: GPU

- Overfitting prevention
  - Dropout regularization

- Stable initialization and training
  - Explosive/vanishing gradient problems
  - Requires careful tuning of initialization and data normalization

1.2 million images with 1000 categories
ReLU Unit

- ReLU $y = \max(x, 0)$

- Why ReLU?
  - Cheap to compute
  - It is roughly linear..
Dropout Regularization

- Randomly zero out neurons with probability 0.5
- During prediction, use expectation value (keep all neurons but scale output by 0.5)
Vanishing and Explosive Value Problem

- Imagine each layer multiplies its input by same weight matrix
  - $W > 1$: exponential explosion
  - $W < 1$: exponential vanishing

- In ConvNets, the weight are not tied, but their magnitude matters
  - Deep nets training was initialization sensitive
Batch Normalization: Stabilize the Magnitude

- Subtract mean
- Divide by standard deviation
- Output is invariant to input scale!
  - Scale input by a constant
  - Output of BN remains the same
- Impact
  - Easy to tune learning rate
  - Less sensitive initialization

---

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1...m\}$

Parameters to be learned: $\gamma, \beta$

**Output:** $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$
\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{// mini-batch mean}
$$

$$
\sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \quad \text{// mini-batch variance}
$$

$$
\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{// normalize}
$$

$$
y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \quad \text{// scale and shift}
$$

**Algorithm 1:** Batch Normalizing Transform, applied to activation $x$ over a mini-batch.
## Setting Hyperparameters

Your Dataset

### Idea #4: Cross-Validation

Split data into **folds**, try each fold as validation and average the results.

<table>
<thead>
<tr>
<th>fold 1</th>
<th>fold 2</th>
<th>fold 3</th>
<th>fold 4</th>
<th>fold 5</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fold 1</th>
<th>fold 2</th>
<th>fold 3</th>
<th>fold 4</th>
<th>fold 5</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fold 1</th>
<th>fold 2</th>
<th>fold 3</th>
<th>fold 4</th>
<th>fold 5</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Useful for small datasets, but not used too frequently in deep learning.
Understanding what a neural network has learned
What’s going on inside ConvNets?

Input Image: 3 x 224 x 224

What are the intermediate features looking for?

Class Scores: 1000 numbers
First Layer: Visualize Filters

AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7

Huang et al, “Densely Connected Convolutional Networks”, CVPR 2017
Last Layer: Dimensionality Reduction

Visualize the “space” of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions

Simple algorithm: Principal Component Analysis (PCA)

More complex: t-SNE
Maximally Activating Patches

Pick a layer and a channel; e.g. conv5 is 128 x 13 x 13, pick channel 17/128

Run many images through the network, record values of chosen channel

Visualize image patches that correspond to maximal activations

Springenberg et al, “Striving for Simplicity: The All Convolutional Net”, ICLR Workshop 2015. Figure copyright Joel Toomas Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller, 2015; reproduced with permission.
Which pixels matter: Saliency vs Occlusion

Mask part of the image before feeding to CNN, check how much predicted probabilities change

**P(elephant) = 0.95**

**P(elephant) = 0.75**

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014
Which pixels matter: Saliency vs Occlusion

Mask part of the image before feeding to CNN, check how much predicted probabilities change

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014
Transfer learning
Transfer learning (fixed feature extractor)

Too costly to train your own CNN (2-3 weeks on GPUs)

Use a CNN pretrained on ImageNet and adapt it to your own dataset

Final layer of CNN is a ‘dense’ layer with # of nodes == # of classes (1000 for ImageNet)

Remove final layer, replace with Dense layer with your # of nodes, with a softmax activation
Transfer learning (fine-tuning)

Fine-tune the weights of the n-1th convolution layer

Earlier layers encode most abstract features, lines, edges etc.

Later layers are biased towards correctly classifying the ImageNet dataset
Model Zoo

https://github.com/BVLC/caffe/wiki/Model-Zoo

Table of Contents

- Berkeley-trained models
- Network in Network model
- Models from the BMVC-2014 paper “Return of the Devil in the Details: Delving Deep into Convolutional Nets”
- Models used by the VGG team in ILSVRC-2014
- Places-CNN model from MIT.
- GoogLeNet GPU implementation from Princeton.
- Fully Convolutional Networks for Semantic Segmentation (FCNs)
- CaffeNet fine-tuned for Oxford flowers dataset
- CNN Models for Salient Object Subtitizing.
- Deep Learning of Binary Hash Codes for Fast Image Retrieval
- Places_CND5_models on Scene Recognition
- Models for Age and Gender Classification.
- GoogLeNet_cars on car model classification
- ParseNet: Looking wider to see better
- SegNet and Bayesian SegNet
- Conditional Random Fields as Recurrent Neural Networks
- Holistically-Nested Edge Detection
- CCNN: Constrained Convolutional Neural Networks for Weakly Supervised Segmentation
- Emotion Recognition in the Wild via Convolutional Neural Networks and Mapped Binary Patterns
- Facial Landmark Detection with Tweaked Convolutional Neural Networks
- Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks
<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
<th>Top-1 Accuracy</th>
<th>Top-5 Accuracy</th>
<th>Parameters</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xception</td>
<td>88 MB</td>
<td>0.790</td>
<td>0.945</td>
<td>22,910,480</td>
<td>126</td>
</tr>
<tr>
<td>VGG16</td>
<td>528 MB</td>
<td>0.713</td>
<td>0.901</td>
<td>138,357,544</td>
<td>23</td>
</tr>
<tr>
<td>VGG19</td>
<td>549 MB</td>
<td>0.713</td>
<td>0.900</td>
<td>143,667,240</td>
<td>26</td>
</tr>
<tr>
<td>ResNet50</td>
<td>99 MB</td>
<td>0.749</td>
<td>0.921</td>
<td>25,636,712</td>
<td>168</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>92 MB</td>
<td>0.779</td>
<td>0.937</td>
<td>23,851,784</td>
<td>159</td>
</tr>
<tr>
<td>InceptionResNetV2</td>
<td>215 MB</td>
<td>0.803</td>
<td>0.953</td>
<td>55,873,736</td>
<td>572</td>
</tr>
<tr>
<td>MobileNet</td>
<td>16 MB</td>
<td>0.704</td>
<td>0.895</td>
<td>4,253,864</td>
<td>88</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>14 MB</td>
<td>0.713</td>
<td>0.901</td>
<td>3,538,984</td>
<td>88</td>
</tr>
<tr>
<td>DenseNet121</td>
<td>33 MB</td>
<td>0.750</td>
<td>0.923</td>
<td>8,062,504</td>
<td>121</td>
</tr>
<tr>
<td>DenseNet169</td>
<td>57 MB</td>
<td>0.762</td>
<td>0.932</td>
<td>14,307,880</td>
<td>169</td>
</tr>
<tr>
<td>DenseNet201</td>
<td>80 MB</td>
<td>0.773</td>
<td>0.936</td>
<td>20,242,984</td>
<td>201</td>
</tr>
<tr>
<td>NASNetMobile</td>
<td>23 MB</td>
<td>0.744</td>
<td>0.919</td>
<td>5,326,716</td>
<td>-</td>
</tr>
<tr>
<td>NASNetLarge</td>
<td>343 MB</td>
<td>0.825</td>
<td>0.960</td>
<td>88,949,818</td>
<td>-</td>
</tr>
</tbody>
</table>
Health applications
Dermatologist-level classification of skin cancer with deep neural...
https://www.nature.com › letters
by A Esteva - 2017 - Cited by 1211 - Related articles
Jan 25, 2017 - a, The **deep learning** CNN outperforms the average of the dermatologists at skin cancer classification using photographic and dermoscopic images. Our CNN is tested against at least 21 dermatologists at keratinocyte carcinoma and melanoma recognition. ... The CNN outputs a malignancy probability P per image.
<table>
<thead>
<tr>
<th>Ground truth</th>
<th>hemangioma</th>
<th>pyogenic</th>
<th>venous+glomerulovenous</th>
<th>capillary+sturge weber</th>
<th>spider angioma</th>
<th>lymphatic</th>
<th>atopic</th>
<th>milia</th>
<th>nevus</th>
</tr>
</thead>
<tbody>
<tr>
<td>hemangioma</td>
<td>0.68</td>
<td>0.05</td>
<td>0.06</td>
<td>0.13</td>
<td>0.04</td>
<td>0.01</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>pyogenic</td>
<td>0.00</td>
<td>0.96</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>venous+glomerulovenous</td>
<td>0.05</td>
<td>0.01</td>
<td>0.84</td>
<td>0.06</td>
<td>0.01</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>capillary+sturge weber</td>
<td>0.09</td>
<td>0.00</td>
<td>0.04</td>
<td>0.83</td>
<td>0.01</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>spider angioma</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.91</td>
<td>0.01</td>
<td>0.04</td>
<td>0.02</td>
<td>0.00</td>
</tr>
<tr>
<td>lymphatic</td>
<td>0.02</td>
<td>0.01</td>
<td>0.06</td>
<td>0.01</td>
<td>0.01</td>
<td>0.84</td>
<td>0.06</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>atopic</td>
<td>0.01</td>
<td>0.00</td>
<td>0.01</td>
<td>0.04</td>
<td>0.05</td>
<td>0.01</td>
<td>0.85</td>
<td>0.03</td>
<td>0.00</td>
</tr>
<tr>
<td>milia</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>0.02</td>
<td>0.97</td>
<td>0.00</td>
</tr>
<tr>
<td>nevus</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.97</td>
</tr>
</tbody>
</table>

Predicted
Deep learning for wireless networks
Lora, Sigfox, NB-IoT

- **Long range + can be decoded BELOW the noise floor**
- **Lora range: > 10 miles**
- **No MAC protocols yet!**
- **Lora, Sigfox and Z-wave are proprietary (unlike Wi-Fi)**
Dimensionality reduction

SNR=-10

SNR=-20

SNR=-30
Machine learning underneath the noise floor
Keras tutorial
Inputs

Grayscale image: (200,100)

RGB image: (200,100,3)

Batch of RGB images: (256,200,100,3)
batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
    activation='relu',
    input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
    optimizer=keras.optimizers.Adadelta(),
    metrics=['accuracy'])

epochs=1000
for e in range(epochs):
    model.fit(xtrain, ytrain,
        shuffle=False, epochs=1,
        batch_size=256,
        validation_data=(xval,yval))

# checkpointing
if e%10==0:
    model.save('traintosh-' + str(e) + '-' + str(e) + '.h5')
```python
xtrain = np.load(direc+'bottleneck-train-')+str(experimentNumber)+str(validationFold)+'-0.npy'
ytrain = np.load(direc+'ytrain-')+str(experimentNumber)+str(validationFold)+'-0.npy'

i = Input(shape=xtrain.shape[1:])
a = Flatten(name='a1')(i)
a = Dense(256, activation='relu', name='a2')(a)
a = Dropout(0.6, name='a3')(a)
o = Dense(ytrain.shape[1], activation='softmax', name='a4')(a)
model = Model(inputs=i, outputs=o)

opt=keras.optimizers.RMSprop(lr=0.00001, rho=0.9, epsilon=None, decay=0.0)
model.compile(optimizer=opt,
              loss='categorical_crossentropy',
              metrics=['categorical_accuracy'])

print(model.summary())
```
epochs=1000
for e in range(epochs):
    for batch_counter in range(trains):
        batch_num=batch_counter
        fname=direct+'bottleneck-train-1'+str(experimentNumber)+'-1'+str(validationFold)+'-1'+str(batch_num)+'.npy
        fname2=direct+'ytrain-1'+str(experimentNumber)+'-1'+str(validationFold)+'-1'+str(batch_num)+'.npy

        xtrain = np.load(fname)
        ytrain = np.load(fname2)
        print (str(batch_counter)+'/'+str(trains-1))
        print (xtrain.shape,ytrain.shape)
        model.fit(xtrain, ytrain,batch_size=128)

    if e%10==0:
        model.save(direct+'train-top-1'+str(experimentNumber)+'-1'+str(validationFold)+'-1'+str(e)+'.h5')
Deploying to phone
3 99% it's Hemangioma

4 99% it's Nevus

5 99% it's Hemangioma

1 80% it's Capillary Malformation
Load in the `.h5` model file

```python
from keras.applications.inception_v3 import InceptionV3
from keras.layers import Dense, Flatten, Dropout
from keras.models import Model

base_model = InceptionV3(include_top=False, weights='imagenet', input_shape=(299, 299, 3))

i = base_model.output
a = Flatten(name='a1')(i)
a = Dense(256, activation='relu', name='a2')(a)
a = Dropout(.6, name='a3')(a)
o = Dense(9, activation='softmax', name='a4')(a)
model = Model(inputs=base_model.input, outputs=o)

model.load_weights('traintop-4-1-80.h5', by_name=True)
```
Convert to .mlmodel file

class_labels=['Hemangioma','Pyogenic Granuloma','Venous Malformation','Capillary Malformation','Spider Angioma','Lymphatic Malformation','Atopic Dermatitis','Milia','Nevus']

import coremltools

coreml_model = coremltools.converters.keras.convert(
    model,
    input_names="image",
    image_input_names="image",
    image_scale=1/127.5,
    red_bias=-1.0,
    green_bias=-1.0,
    blue_bias=-1.0,
    class_labels=class_labels,
)

coreml_model.save('VA.mlmodel')
func classify(image: CIImage) {
    let model: VNCoreMLModel = getmodel()

    let request = VNCoreMLRequest(model: model) {
        [weak self] request, error in
        guard let results = request.results as? [VNClassificationObservation],
        let topResult = results.first else {
            fatalError("unexpected result type from VNCoreMLRequest")
        }

        for i in 0..<9 {
            print("\(results[i].identifier) \(results[i].confidence)")
        }

        DispatchQueue.main.async {
            [weak self] in
            let t = (self?.counter)!
            self?.answerLabel.text = "\(t)\n(\(Int(topResult.confidence * 100))% it's \(topResult.identifier)"
        }
    }

    request.imageCropAndScaleOption = .centerCrop

    let handler = VNImageRequestHandler(ciImage: image)
    DispatchQueue.global(qos: .userInteractive).async {
        do {
            try handler.perform([request])
        } catch {
            print(error)
        }
    }
}
func reading() -> CIIImage? {
    let filePath = Bundle.main.resourcePath!+"/tree.png"

    do {
        let image = UIImage(contentsOfFile: filePath);
        guard let ciImage = CIIImage(image: image!) else {
            fatalError("couldn't convert UIImage to CIIImage")
        }
        return ciImage
    }
}