Lecture 4:
Medical Images -- Classification
Announcements

- A0 due 11:59pm tonight
- A1 will also be released tonight, due Wed Jan 29
Last time: deep learning fundamentals (part 2)

- Neural network architectures
- Loss functions
- Optimization methods
- Preparing data for deep learning
- Training models in practice
- Model inference
“Deep learning”

Can continue to stack more layers to get deeper models!

“Hidden” layers - will see lots of diversity in size (# neurons), type (linear, convolutional, etc.), and activation function (sigmoid, ReLU, etc.)

Input layer

Output layer - will differ for different types of tasks (e.g. regression, N-way classification, bounding box detection, etc.). Should match with loss function.

Vanilla fully-connected neural networks (MLPs) usually pretty shallow -- otherwise too many parameters! ~2-3 layers. Can have wide range in size of layers (16, 64, 256, 1000, etc.) depending on how much data you have.

Will see different classes of neural networks that leverage structure in data to reduce parameters + increase network depth
Common activation functions

You will see these extensively, typically after linear or convolutional layers. They add nonlinearity to allow the model to express complex nonlinear functions.

Typical in modern CNNs and MLPs

Sigmoid

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

Tanh

\[ \tanh(x) \]

ReLU

\[ \text{max}(0, x) \]

Leaky ReLU

\[ \text{max}(0.1x, x) \]

You can find these in Keras:
https://keras.io/layers/advanced-activations/
Common loss functions

**Regression**
Minimize squared difference between prediction output and target.

\[ L_{\text{regression}} = \frac{1}{M} \sum_{i} (\hat{y}^i - y^i)^2 \]

Label is a continuous value.

**Binary Cross-Entropy**

\[ L_{\text{BCE}} = \frac{1}{M} \sum_{i} - (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)) \]

Label is binary in \{0,1\}. Prediction in (0,1) is probably of label being a 1, and usually the output of a sigmoid operation after the final layer. Since we minimize loss functions, neg. sign translates to maximizing the log-probability of the true class, which is what we want.

Equivalent to negative log-probability of the prediction, for the true class. Think about what the expression looks like when \( y_i = 1 \) vs. \( 0 \).

**Softmax**

Negative log of the probability of the true class \( y_i \), as with the BCE loss.

\[ L_{\text{Softmax}} = \frac{1}{M} \sum_{i} - \log\left( \frac{e^{s_{y_i}}}{\sum_{j} e^{s_j}} \right) \]

Label is 1 of K classes in \{0, …, K\}. Extension of binary cross-entropy loss to multiple classes. \( s_j \) corresponds to the score (e.g. output of final layer) for each class; the fraction in the log provides a normalized probability for each class.

**SVM**

\[ L_{\text{SVM}} = \frac{1}{M} \sum_{i} \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \]

Label is 1 of K classes in \{0, …, K\}. Same use case as softmax, but different way of encouraging the model to produce outputs that we “like”. In practice, softmax is more popular and provides a nice probabilistic interpretation.

Label is 1 of K classes in \{0, …, K\}. Incurs loss of 0 (what we want) if the score for the true class \( y_i \) is greater than the score for each incorrect class \( j \) by a margin of 1.
Adam (full form): RMSProp + momentum

```python
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)

    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx

    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)

    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that first and second moment estimates start at zero

Adam with beta1 = 0.9, beta2 = 0.999, and learning_rate = 1e-3 or 5e-4 is a great starting point for many models!


Slide credit: CS231n
Learning rate decay

Step: Reduce learning rate at a few fixed points. E.g. for ResNets, multiply LR by 0.1 after epochs 30, 60, and 90.

Cosine: \[ \alpha_t = \frac{1}{2} \alpha_0 \left(1 + \cos\left(\frac{t\pi}{T}\right)\right) \]

\( \alpha_0 \): Initial learning rate
\( \alpha_t \): Learning rate at epoch \( t \)
\( T \): Total number of epochs

Loshchilov and Hutter, “SGDR: Stochastic Gradient Descent with Warm Restarts”, ICLR 2017
Radford et al, “Improving Language Understanding by Generative Pre-Training”, 2018
Feichtenhofer et al, “SlowFast Networks for Video Recognition”, arXiv 2018

Slide credit: CS231n
Transfer learning: amplifying training data

1. Train on big dataset (e.g. ImageNet)

2. Small Dataset (C classes)

   - Freeze these
   - Reinitialize this and keep training on target dataset

3. Bigger dataset

   - With bigger dataset, train more layers
   - Train these
   - Freeze these

   - Lower learning rate when finetuning; 1/10 of original LR is good starting point

---

Razavian et al., “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

Slide credit: CS231n
“Xavier” (or “Glorot”) Initialization

```python
inputs = [4096] * 7
hs = []
x = np.random.randn(16, inputs[0])
for Din, Dout in zip(inputs[:-1], inputs[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
x = np.tanh(x.dot(W))
hs.append(x)
```

“Just right”: Activations are nicely scaled for all layers!

Glorot and Bengio, “Understanding the difficulty of training deep feedforward neural networks”, AISTAT 2010

Slide credit: CS231n
Overfitting vs. underfitting

**Overfitting**
- Training loss much better than validation.
- Training loss may continue to get better while validation plateaus or gets worse.

Model is “overfitting” to the training data. Best strategy: Increase data or regularize model. Second strategy: decrease model capacity (make simpler).

**Underfitting**
- Small or no gap between training and validation loss.
- May have relatively higher loss overall (model not learning sufficiently).

Model is not able to sufficiently learn to fit the data well. Best strategy: Increase complexity (e.g. size) of the model. Second strategy: make problem simpler (easier task, cleaner data).
More debugging

Loss decreasing but slowly -> try higher learning rate

If you further decay learning rate too early, may look like this -> inefficient learning vs. keeping higher learning rate longer

Final metric is still improving -> keep training!

Healthy loss curve plateaus -> try further learning rate decay at plateau point
A few more things to finish up...
Design choices: network architectures

Major design choices:
- Architecture type (ResNet, DenseNet, etc. for CNNs)
- Depth (number of layers)
- For MLPs, number of neurons in each layer (hidden layer size)
- For CNNs, number of filters, filter size, filter stride in each layer
- Look at argument options in Tensorflow when defining network layers
Major design choices:
- Architecture type (ResNet, DenseNet, etc. for CNNs)
- Depth (# layers)
- For MLPs, # neurons in each layer (hidden layer size)
- For CNNs, # filters, filter size, filter stride in each layer
- Look at argument options in Tensorflow when defining network layers

If trying to make network bigger (when underfitting) or smaller (when overfitting), network depth and hidden layer size best to adjust first. Don't waste too much early time early on fiddling with choices that only minorly change architecture.
Design choices: optimization

- SGD
- SGD+Momentum
- RMSProp
- Adam

Slide credit: CS231n
Design choices: optimization

Major hyperparameters: learning rate and learning rate decay!

- SGD
- SGD+Momentum
- RMSProp
- Adam

Slide credit: CS231n
Design choices: regularization (loss term)

Remember optimizing loss functions, which express how well model fit training data, e.g.:

$$L_{\text{regression}} = \frac{1}{M} \sum_i (\hat{y}^i - y^i)^2$$
Design choices: regularization (loss term)

Remember optimizing loss functions, which express how well model fit training data, e.g.:

\[ L_{\text{regression}} = \frac{1}{M} \sum_i (\hat{y}^i - y^i)^2 \]

Regularization adds a term to this, to express preferences on the weights (that prevent it from fitting too well to the training data). Used to combat overfitting:

\[ L = \frac{1}{M} \sum_i (\hat{y}^i - y^i)^2 + \lambda R(W) \]

- Data loss
- Regularization loss
- importance of reg. term
Design choices: regularization (loss term)

Remember optimizing loss functions, which express how well model fit training data, e.g.:

$$L_{\text{regression}} = \frac{1}{M} \sum_i (\hat{y}^i - y^i)^2$$

Regularization adds a term to this, to express preferences on the weights (that prevent it from fitting too well to the training data). Used to combat overfitting:

$$L = \frac{1}{M} \sum_i (\hat{y}^i - y^i)^2 + \lambda R(W)$$

**Examples**

- L2 regularization:  
  $$R(W) = \sum_k \sum_l W_{k,l}^2$$  
  (weight decay)

- L1 regularization:  
  $$R(W) = \sum_k \sum_l |W_{k,l}|$$

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

https://www.tensorflow.org/api_docs/python/tf/keras/regularizers
Design choices: regularization (loss term)

Remember optimizing loss functions, which express how well model fit training data, e.g.:

\[ L_{\text{regression}} = \frac{1}{M} \sum_i (\hat{y}^i - y^i)^2 \]

Regularization adds a term to this, to express preferences on the weights (that prevent it from fitting too well to the training data). Used to combat overfitting:

\[ L = \frac{1}{M} \sum_i (\hat{y}^i - y^i)^2 + \lambda R(W) \]

Examples:

L2 regularization: \( R(W) = \sum_k \sum_l W_{k,l}^2 \) (weight decay)

L1 regularization: \( R(W) = \sum_k \sum_l |W_{k,l}| \)

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

https://www.tensorflow.org/api_docs/python/tf/keras/regularizers
Design choices: regularization (loss term)

Remember optimizing loss functions, which express how well model fit training data, e.g.:

\[ L_{\text{regression}} = \frac{1}{M} \sum_{i} (\hat{y}^i - y^i)^2 \]

Regularization adds a term to this, to express preferences on the weights (that prevent it from fitting too well to the training data). Used to combat overfitting:

\[ L = \frac{1}{M} \sum_{i} (\hat{y}^i - y^i)^2 + \lambda R(W) \]

**Examples**

- **L2 regularization**: \[ R(W) = \sum_{k} \sum_{l} W_{k,l}^2 \] (weight decay)
- **L1 regularization**: \[ R(W) = \sum_{k} \sum_{l} |W_{k,l}| \]
- **Elastic net (L1 + L2)**: \[ R(W) = \sum_{k} \sum_{l} \beta W_{k,l}^2 + |W_{k,l}| \]

L2 most popular: low loss when all weights are relatively small. More strongly penalizes large weights vs L1. Expresses preference for simple models (need large coefficients to fit a function to extreme outlier values).

Next: implicit regularizers that do not add an explicit term; instead do something implicit in network to prevent it from fitting too well to training data.

https://www.tensorflow.org/api_docs/python/tf/keras/regularizers
Design choices: regularization (dropout)

First example of an implicit regularizer. During training, at each iteration of forward pass randomly set some neurons to zero (i.e., change network architecture such that paths to some neurons are removed).

During testing, all neurons are active. But scale neuron outputs by dropout probability $p$, such that expected output during training and testing match.

Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014. Figure credit: CS231n.
Design choices: regularization (dropout)

First example of an implicit regularizer. During training, at each iteration of forward pass randomly set some neurons to zero (i.e., change network architecture such that paths to some neurons are removed).

During testing, all neurons are active. But scale neuron outputs by dropout probability $p$, such that expected output during training and testing match.

Probability of “dropping out” each neuron at a forward pass is hyperparameter $p$. 0.5 and 0.9 are common (high!).

Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014. Figure credit: CS231n.
Design choices: regularization (dropout)

First example of an implicit regularizer. During training, at each iteration of forward pass randomly set some neurons to zero (i.e., change network architecture such that paths to some neurons are removed).

Probability of “dropping out” each neuron at a forward pass is hyperparameter $p$. 0.5 and 0.9 are common (high!).

Intuition: dropout is equivalent to training a large ensemble of different models that share parameters.

During testing, all neurons are active. But scale neuron outputs by dropout probability $p$, such that expected output during training and testing match.

Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014. Figure credit: CS231n.
Design choices: regularization (batch normalization)

Another example of an implicit regularizer. Insert BN layers after FC or conv layers, before activation function.
During training, at each iteration of forward pass normalize neuron activations by mean and variance of minibatch. Also learn scale and shift parameter to get final output.

\[
\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}
\]
\[
\sigma_{j}^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2
\]
\[
\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_{j}^2 + \varepsilon}}
\]
\[
y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j
\]

During testing, normalize by a fixed mean and variance computed from the entire training set. Use learned scale and shift parameters.
Design choices: regularization (batch normalization)

Another example of an implicit regularizer. Insert BN layers after FC or conv layers, before activation function.
During training, at each iteration of forward pass normalize neuron activations by mean and variance of minibatch. Also learn scale and shift parameter to get final output.

\[
\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}
\]

\[
\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2
\]

\[
\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}
\]

\[
y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j
\]

Intuition: batch normalization allows keeping the weights in a healthy range. Also some randomness at training due to different effect from each minibatch sampling -> regularization!

During testing, normalize by a fixed mean and variance computed from the entire training set. Use learned scale and shift parameters.
Design choices: regularization (batch normalization)

Another example of an implicit regularizer. Insert BN layers after FC or conv layers, before activation function. During training, at each iteration of forward pass normalize neuron activations by mean and variance of minibatch. Also learn scale and shift parameter to get final output.

\[
\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}
\]

\[
\sigma^2_j = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2
\]

\[
\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma^2_j + \varepsilon}}
\]

\[
y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j
\]

Intuition: batch normalization allows keeping the weights in a healthy range. Also some randomness at training due to different effect from each minibatch sampling -> regularization!

There are many types of batch normalization worth trying: instance norm, layer norm, etc.

During testing, normalize by a fixed mean and variance computed from the entire training set. Use learned scale and shift parameters.
Design choices: data augmentation

Augment effective training data size by simulating more diversity from existing data. Random combinations of:

- Translation and scaling
- Flipping and rotation
- Distortion
- Image color adjustment
- Etc.
Design choices: data augmentation

Augment effective training data size by simulating more diversity from existing data. Random combinations of:
- Translation and scaling
- Flipping and rotation
- Distortion
- Image color adjustment
- Etc.

Think about the domain of your data: what makes sense as realistic augmentation operations?
Hyperparameter search

Step 1: Find LR that makes loss go down
Step 2: Define coarse grid of hyperparameter options, train for ~1-5 epochs
Step 3: Refine grid, train longer
Step 4: Look at loss curves
Step 5: GOTO step 3
Hyperparameter search

**Step 1**: Find LR that makes loss go down

**Step 2**: Define coarse grid of hyperparameter options, train for ~1-5 epochs

**Step 3**: Refine grid, train longer

**Step 4**: Look at loss curves

**Step 5**: GOTO step 3

Useful debugging / sanity check: restrict to a very small dataset first (e.g. 1 or 2 minibatches). You should be able to severely overfit and drive the loss to 0.
Hyperparameter search

Step 1: Find LR that makes loss go down
Step 2: Define coarse grid of hyperparameter options, train for ~1-5 epochs
Step 3: Refine grid, train longer
Step 4: Look at loss curves
Step 5: GOTO step 3

Useful debugging / sanity check: restrict to a very small dataset first (e.g. 1 or 2 minibatches). You should be able to severely overfit and drive the loss to 0.

Common pitfall: making grid too small. Sample a wide range of values to make sure you’ve explored the space. (e.g. LRs from 1e0 to 1e-5.)
Hyperparameter search

Step 1: Find LR that makes loss go down
Step 2: Define coarse grid of hyperparameter options, train for ~1-5 epochs
Step 3: Refine grid, train longer
Step 4: Look at loss curves
Step 5: GOTO step 3

Useful debugging / sanity check: restrict to a very small dataset first (e.g. 1 or 2 minibatches). You should be able to severely overfit and drive the loss to 0.

Common pitfall: making grid too small. Sample a wide range of values to make sure you’ve explored the space. (e.g. LRs from 1e0 to 1e-5.)

Aside: For LR, should sample e^x for x in Uniform [-5, 0]!
Random search vs. grid search

Grid Layout

Random Layout

Illustration of Bergstra et al., 2012 by Shayne Longpre, copyright CS231n 2017

Random Search for Hyper-Parameter Optimization
Bergstra and Bengio, 2012
Model inference
Maximizing test-time performance: apply data augmentation operations

Main idea: apply model on multiple variants of a data example, and then take average or max of scores

Can use data augmentation operations we saw during training! E.g.:

- Evaluate at different translations and scales
- Common approach for images: evaluate image crops at 4 corners and center, + horizontally flipped versions -> average 10 scores
Model ensembles

1. Train multiple independent models
2. At test time average their results
   (Take average of predicted probability distributions, then choose argmax)

Enjoy 2% extra performance
Model ensembles: tips and tricks

Instead of training independent models, use multiple snapshots of a single model during training!

Huang et al., “Snapshot ensembles: train 1, get M for free”, ICLR 2017
Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.
Model ensembles: tips and tricks

Instead of training independent models, use multiple snapshots of a single model during training!

Huang et al. “Snapshot ensembles: train 1, get M for free”, ICLR 2017
Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.

Cyclic learning rate schedules can make this work even better!

Lecture 4 - 40
Model ensembles: tips and tricks

Instead of using actual parameter vector, keep a moving average of the parameter vector and use that at test time (Polyak averaging)

```python
while True:
    data_batch = dataset.sample_data_batch()
    loss = network.forward(data_batch)
    dx = network.backward()
    x += -learning_rate * dx
    x_test = 0.995*x_test + 0.005*x  # use for test set
```

Conclusion

Further depth on:

- Neural network architectures
- Loss functions
- Optimization methods
- Preparing data for deep learning
- Training models in practice
- Model inference

Will see these come up in many examples as we begin discussing deep learning models for different types of biomedical data
Next: Medical Images and classification

- A first example
- Convolutional layers and CNN architectures
- Case studies of CNNs for medical image classification
Medical image data

E.g.:

X-rays (invented 1895).

CT (invented 1972).

MRI (invented 1977).
A first example: Gulshan et al. 2016

- **Task**: Binary classification of referable diabetic retinopathy from retinal fundus photographs (moderate and worse diabetic retinopathy, referable diabetic macular edema, or both)
- **Input**: Retinal fundus photographs
- **Output**: Binary classification of referable diabetic retinopathy ($y$ in $\{0,1\}$)
  - Defined as moderate and worse diabetic retinopathy, referable diabetic macular edema, or both

A first example: Gulshan et al. 2016

- **Dataset:**
  - 128,175 images, each graded by 3-7 ophthalmologists.
  - 54 total graders, each paid to grade between 20 to 62508 images.

- **Data preprocessing:**
  - Circular mask of each image was detected and rescaled to be 299 pixels wide

- **Model:**
  - Inception-v3 CNN, with ImageNet pre-training
  - Multiple BCE losses corresponding to different binary prediction problems, which were then used for final determination of referable diabetic retinopathy

A first example: Gulshan et al. 2016

- **Dataset:**
  - 128,175 images, each graded by 3-7 ophthalmologists.
  - 54 total graders, each paid to grade between 20 to 62508 images.

- **Data preprocessing:**
  - Circular mask of each image was detected and rescaled to be 299 pixels wide.

- **Model:**
  - Inception-v3 CNN, with ImageNet pre-training
  - Multiple BCE losses corresponding to different binary prediction problems, which were then used for final determination of referable diabetic retinopathy

A first example: Gulshan et al. 2016

- **Results:**
  - Evaluated using ROC curves, AUC, sensitivity and specificity analysis

### Evaluation metrics

#### Confusion matrix

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TN</td>
<td>FP</td>
</tr>
<tr>
<td>1</td>
<td>FN</td>
<td>TP</td>
</tr>
</tbody>
</table>

**Prediction**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Accuracy:** \( \frac{TP + TN}{total} \)

**Q:** When might evaluating purely accuracy be problematic?
### Evaluation metrics

#### Confusion matrix

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Accuracy: \( \frac{TP + TN}{\text{total}} \)

Q: When might evaluating purely accuracy be problematic?

A: Imbalanced datasets.
Evaluation metrics

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ground Truth</strong></td>
<td><strong>Prediction</strong></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>TN</td>
<td>FP</td>
</tr>
<tr>
<td>1</td>
<td>FN</td>
<td>TP</td>
</tr>
</tbody>
</table>

**Confusion matrix**

**Accuracy**: \(\frac{TP + TN}{total}\)

**Sensitivity / Recall** (true positive rate):
\(\frac{TP}{total\, positives}\)

**Specificity** (true negative rate):
\(\frac{TN}{total\, negatives}\)

**Precision** (positive predictive value):
\(\frac{TP}{total\, predicted\, positives}\)

**Negative predictive value**:
\(\frac{TN}{total\, predicted\, negatives}\)
Evaluation metrics

Confusion matrix

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>FN</td>
</tr>
</tbody>
</table>

We can trade-off different values of these metrics as we value our classifier’s score threshold to predict a positive.

**Accuracy**: \( \frac{(TP + TN)}{\text{total}} \)

**Sensitivity / Recall** (true positive rate): \( \frac{TP}{\text{total positives}} \)

**Specificity** (true negative rate): \( \frac{TN}{\text{total negatives}} \)

**Precision** (positive predictive value): \( \frac{TP}{\text{total predicted positives}} \)

**Negative predictive value**: \( \frac{TN}{\text{total predicted negatives}} \)
Evaluation metrics

Confusion matrix

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground Truth</td>
<td>TN</td>
<td>FP</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>FN</td>
<td>TP</td>
</tr>
</tbody>
</table>

Accuracy: \( \frac{\text{TP} + \text{TN}}{\text{total}} \)

Sensitivity / Recall (true positive rate): \( \frac{\text{TP}}{\text{total positives}} \)

Specificity (true negative rate): \( \frac{\text{TN}}{\text{total negatives}} \)

Precision (positive predictive value): \( \frac{\text{TP}}{\text{total predicted positives}} \)

Negative predictive value: \( \frac{\text{TN}}{\text{total predicted negatives}} \)

Q: As prediction threshold increases, how does that generally affect sensitivity? Specificity?
**Evaluation metrics**

**Confusion matrix**

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TN</td>
</tr>
<tr>
<td>1</td>
<td>FP</td>
</tr>
<tr>
<td>0</td>
<td>FN</td>
</tr>
<tr>
<td>1</td>
<td>TP</td>
</tr>
</tbody>
</table>

**Accuracy:** \( \frac{TP + TN}{\text{total}} \)

**Sensitivity / Recall** (true positive rate):
\( \frac{TP}{\text{total positives}} \)

**Specificity** (true negative rate):
\( \frac{TN}{\text{total negatives}} \)

**Precision** (positive predictive value):
\( \frac{TP}{\text{total predicted positives}} \)

**Negative predictive value:**
\( \frac{TN}{\text{total predicted negatives}} \)

**Q:** As prediction threshold increases, how does that generally affect sensitivity? Specificity?

**A:** Sensitivity goes down, specificity up
Evaluation metrics

- **Receiver Operating Characteristic (ROC) curve:**
  - Plots sensitivity and specificity (specifically, 1 - specificity) as prediction threshold is varied
  - Gives trade-off between sensitivity and specificity
  - Also report summary statistic AUC (area under the curve)
Evaluation metrics

- Receiver Operating Characteristic (ROC) curve:
  - Plots sensitivity and specificity (specifically, 1 - specificity) as prediction threshold is varied
  - Gives trade-off between sensitivity and specificity
  - Also report summary statistic AUC (area under the curve)

Perfect AUC = 1. AUC has an interpretation of given a randomly chosen true positive and true negative example, the probability that the classifier will score the TP higher than the TN example.
Evaluation metrics

- Sometimes also see **precision recall curve**
  - More informative when dataset is heavily imbalanced (sensitivity = true negative rate less meaningful in this case)

Figure credit: https://3qepr26caki16dnhd19sv6by6v-wpengine.netdna-ssl.com/wp-content/uploads/2018/08/Precision-Recall-Plot-for-a-No-Skill-Classifier-and-a-Logistic-Regression-Model4.png
Evaluation metrics

- Selecting optimal trade-off points
  - Youden’s Index
    - J = sensitivity + specificity - 1
    - Gives equal weight to optimizing true positives and true negatives

Figure credit: https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png
Evaluation metrics
- Selecting optimal trade-off points
  - **Youden’s Index**
    - $J = \text{sensitivity} + \text{specificity} - 1$
    - Gives equal weight to optimizing true positives and true negatives

Also equal to distance above chance line for a balanced dataset: $\text{sensitivity} - (1 - \text{specificity}) = \text{sensitivity} + \text{specificity} - 1$

Figure credit: [https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png](https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png)
Evaluation metrics

- Selecting optimal trade-off points
  - **Youden’s Index**
    - \( J = \text{sensitivity} + \text{specificity} - 1 \)
    - Gives equal weight to optimizing true positives and true negatives
  - Sometimes also see F-measure (or F1 score)
    - \( F1 = \frac{2 \times (\text{precision} \times \text{recall})}{\text{precision} + \text{recall}} \)
    - Harmonic mean of precision and recall

Also equal to distance above chance line for a balanced dataset: sensitivity - (1 - specificity) = sensitivity + specificity - 1

But selected trade-off points could also depend on application

Figure credit: https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png
Gulshan et al. 2016

AUC = 0.991

Looked at different operating points
- High-specificity point approximated ophthalmologist specificity for comparison. Should also use high-specificity to make decisions about high-risk actions.
- High-sensitivity point should be used for screening applications.
Convolutional layers and CNNs
Reminder: Fully connected layer

input

1

3072

1

$Wx$

10 x 3072 weights

activation

1

10

Slide credit: CS231n
Convolutional layer

32x32x3 image -> preserve spatial structure

32  height
32  width
3    depth

Slide credit: CS231n
Convolutional layer

32x32x3 image

5x5x3 filter (weights)

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolutional layer

32x32x3 image

5x5x3 filter (weights)

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Filters always extend the full depth of the input volume

Slide credit: CS231n
Convolutional layer

32x32x3 image
5x5x3 filter $w$

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. $5*5*3 = 75$-dimensional dot product + bias)

$$w^T x + b$$

Slide credit: CS231n
Convolutional layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map

Slide credit: CS231n
Consider a second, green filter.

Convolutional layer

32x32x3 image
5x5x3 filter

Convolve (slide) over all spatial locations

Activation maps

Slide credit: CS231n
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
Preview: ConvNet (or CNN) is a sequence of Convolution Layers, interspersed with activation functions.
**Preview:** ConvNet (or CNN) is a sequence of Convolution Layers, interspersed with activation functions.

![Diagram showing sequence of Convolution Layers](slide.png)
A closer look at spatial dimensions:

- 32x32x3 image
- 5x5x3 filter
- convolve (slide) over all spatial locations

Slide credit: CS231n
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)  
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter

=> 5x5 output

Slide credit: CS231n
A closer look at spatial dimensions:

- 7x7 input (spatially)
- Assume 3x3 filter
- Applied with stride 2

Slide credit: CS231n
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2

Slide credit: CS231n
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
applied with stride 2
=> 3x3 output!

Slide credit: CS231n
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?

doesn’t fit! cannot apply 3x3 filter on 7x7 input with stride 3.
Output size: 
\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7, F = 3:\)
- \(\text{stride 1} \Rightarrow (7 - 3)/1 + 1 = 5\)
- \(\text{stride 2} \Rightarrow (7 - 3)/2 + 1 = 3\)
- \(\text{stride 3} \Rightarrow (7 - 3)/3 + 1 = 2.33\)
In practice: Common to zero pad the border

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

E.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

(recall:)

\[
\frac{N - F}{\text{stride}} + 1
\]
In practice: Common to zero pad the border

e.g. input 7x7
3x3 filter, applied with \textit{stride} 1
\textit{pad with 1 pixel} border => what is the output?

7x7 output!
In practice: Common to zero pad the border

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

7x7 output!

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with \((F-1)/2\). (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Slide credit: CS231n
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.

Slide credit: CS231n
Common settings:

- \( K = (\text{powers of 2, e.g. } 32, 64, 128, 512) \)
- \( F = 3, S = 1, P = 1 \)
- \( F = 5, S = 1, P = 2 \)
- \( F = 5, S = 2, P = ? \) (whatever fits)
- \( F = 1, S = 1, P = 0 \)
In Keras

**Conv2D**

```python
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=\text{None}, \text{dilations=(1, 1)}, \text{activation=\text{None}}, \text{use_bias=True, \text{input_shape=\text{None}}})
```

2D convolution layer (e.g. spatial convolution over images).
In Keras

Padding options: ‘valid’ does not pad, use ‘same’ to pad such that input and output spatial dimensions are the same size.

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, name=None, dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, kernel_constraint=None, bias_constraint=None)
```

2D convolution layer (e.g. spatial convolution over images).
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Preview: can visualize learned features

[Zeiler and Fergus 2013]

Low-level features → Mid-level features → High-level features → Linearly separable classifier

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].
We call the layer convolutional because it is related to convolution of two signals:

\[ f[x,y] \ast g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1,y-n_2] \]

Elementwise multiplication and sum of a filter and the signal (image)
Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:
Max pooling

Single depth slice

max pool with 2x2 filters and stride 2

Slide credit: CS231n
Pooling layer: practical implementation

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent $F$,
  - the stride $S$,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

In Keras:

```python
MaxPooling2D,

keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)
```

Slide credit: CS231n
Pooling layer: practical implementation

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent $F$,
  - the stride $S$,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

**Common settings:**

- $F = 2$, $S = 2$
- $F = 3$, $S = 2$

**In Keras:**

```
MaxPooling2D

keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)
```

Slide credit: CS231n
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-FC-FC]
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

First CNN-based winner

- 2010: Lin et al
- 2011: Sanchez & Perronnin
- 2012: Krizhevsky et al (AlexNet)
- 2013: Zeiler & Fergus
- 2014: Simonyan & Zisserman (VGG)
- 2014: Szegedy et al (GoogLeNet)
- 2015: He et al (ResNet)
- 2016: Shao et al
- 2017: Hu et al (SENet)
- Human

Layers:
- 8 layers
- 19 layers
- 22 layers
- 152 layers

Slide credit: CS231n
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)

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Slide credit: CS231n
AlexNet
[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

**INPUT**

- **CONV1**: 96 11x11 filters at stride 4, pad 0
- **MAX POOL1**: 3x3 filters at stride 2
- **NORM1**: Normalization layer
- **CONV2**: 256 5x5 filters at stride 1, pad 2
- **MAX POOL2**: 3x3 filters at stride 2
- **NORM2**: Normalization layer
- **CONV3**: 384 3x3 filters at stride 1, pad 1
- **CONV4**: 384 3x3 filters at stride 1, pad 1
- **CONV5**: 256 3x3 filters at stride 1, pad 1
- **MAX POOL3**: 3x3 filters at stride 2

**FC6**: 4096 neurons
**FC7**: 4096 neurons
**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%

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Slide credit: CS231n
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)

Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.
Slide credit: CS231n
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- 2010: Lin et al
- 2011: Sanchez & Perronnin
- 2012: Krizhevsky et al (AlexNet)
- 2013: Zeiler & Fergus
- 2014: Simonyan & Zisserman (VGG)
- 2014: Szegedy et al (GoogLeNet)
- 2015: He et al (ResNet)
- 2016: Shao et al
- 2017: Hu et al (SENet)
- Human

Deeper Networks
- 152 layers
- 152 layers
- 152 layers

Shallow
- 8 layers
- 8 layers

Slide credit: CS231n
Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC’13 (ZFNet)
-> 7.3% top 5 error in ILSVRC’14

[Simonyan and Zisserman, 2014]
INPUT: [224x224x3]  memory: 224*224*3=150K  params: 0  (not counting biases)

CONV3-64: [224x224x64] memory: 224*224*64=3.2M  params: (3*3*3)*64 = 1,728

CONV3-64: [224x224x64] memory: 224*224*64=3.2M  params: (3*3*64)*64 = 36,864

POOL2: [112x112x64] memory: 112*112*64=800K  params: 0

CONV3-128: [112x112x128] memory: 112*112*128=1.6M  params: (3*3*64)*128 = 73,728

CONV3-128: [112x112x128] memory: 112*112*128=1.6M  params: (3*3*128)*128 = 147,456

POOL2: [56x56x128] memory: 56*56*128=400K  params: 0

CONV3-256: [56x56x256] memory: 56*56*256=800K  params: (3*3*128)*256 = 294,912

CONV3-256: [56x56x256] memory: 56*56*256=800K  params: (3*3*256)*256 = 589,824

CONV3-256: [56x56x256] memory: 56*56*256=800K  params: (3*3*256)*256 = 589,824

POOL2: [28x28x256] memory: 28*28*256=200K  params: 0

CONV3-512: [28x28x512] memory: 28*28*512=400K  params: (3*3*256)*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28*28*512=400K  params: (3*3*512)*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28*28*512=400K  params: (3*3*512)*512 = 2,359,296

POOL2: [14x14x512] memory: 14*14*512=100K  params: 0

CONV3-512: [14x14x512] memory: 14*14*512=100K  params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K  params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K  params: (3*3*512)*512 = 2,359,296

POOL2: [7x7x512] memory: 7*7*512=25K  params: 0

FC: [1x1x4096] memory: 4096  params: 7*7*512*4096 = 102,760,448

FC: [1x1x4096] memory: 4096  params: 4096*4096 = 16,777,216

FC: [1x1x1000] memory: 1000  params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
INPUT: [224x224x3] memory: 224*224*3=150K params: 0

CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 1,728

CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*64)*64 = 36,864

POOL2: [112x112x64] memory: 112*112*64=800K params: 0

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*128)*128 = 147,456

POOL2: [56x56x128] memory: 56*56*128=400K params: 0

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*256)*256 = 589,824

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*256)*256 = 589,824

POOL2: [28x28x256] memory: 28*28*256=400K params: 0

CONV3-512: [28x28x512] memory: 28*28*512=1M params: (3*3*256)*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28*28*512=1M params: (3*512)*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28*28*512=1M params: (3*512)*512 = 2,359,296

POOL2: [14x14x512] memory: 14*14*512=100K params: 0

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*512)*512 = 2,359,296

POOL2: [7x7x512] memory: 7*7*512=25K params: 0

FC: [1x1x4096] memory: 4096 params: 7*512*4096 = 102,760,448

FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216

FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

**TOTAL memory: 24M * 4 bytes ~ = 96MB / image (only forward! ~*2 for bwd)**

**TOTAL params: 138M parameters**
“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other
GoogLeNet
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- Avoids expensive FC layers
- 12x less params than AlexNet
- ILSVRC'14 classification winner (6.7% top 5 error)
The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners have shown a "Revolution of Depth" in their network architectures:

- **2010**: Lin et al. with 28.2% accuracy
- **2011**: Sanchez & Perronnin with 25.8% accuracy
- **2012**: Krizhevsky et al. (AlexNet) with 16.4% accuracy
- **2013**: Zeiler & Fergus with 11.7% accuracy
- **2014**: Simonyan & Zisserman (VGG) with 7.3% accuracy
- **2014**: Szegedy et al. (GoogLeNet) with 6.7% accuracy

Over time, the depth of the networks has increased significantly:
- **2015**: He et al. (ResNet) with 3.6 layers
- **2016**: Shao et al. with 3 layers
- **2017**: Hu et al. (SENet) with 2.3 layers

Finally, human performance is represented with 5.1% accuracy.
ResNet
[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC’15 classification winner (3.57% top 5 error)
- Won all major classification and detection benchmark challenges in 2015
ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

Q: What’s strange about these training and test curves?  
[Hint: look at the order of the curves]
ResNet
[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!
ResNet

[He et al., 2015]

Hypothesis: the problem is an optimization problem, deeper models are harder to optimize
ResNet

[He et al., 2015]

Hypothesis: the problem is an optimization problem, deeper models are harder to optimize

The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.
ResNet
[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

![Diagram of ResNet](Slide credit: CS231n)
ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

\[ H(x) = F(x) + x \]

Use layers to fit residual \( F(x) = H(x) - x \) instead of \( H(x) \) directly

Slide credit: CS231n
ResNet

[He et al., 2015]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers

Slide credit: CS231n
ResNet

[He et al., 2015]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
ResNet

[He et al., 2015]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
ResNet
[He et al., 2015]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

Residual block

Slide credit: CS231n
ResNet

[He et al., 2015]

Total depths of 34, 50, 101, or 152 layers for ImageNet
For deeper networks (ResNet-50+), use 1x1 "bottleneck" layer to improve efficiency (also used in GoogLeNet)

ResNet
[He et al., 2015]
CNNs for Medical Imaging Classification
Early steps of deep learning in medical imaging: using ImageNet CNN features

Bar et al. 2015

- Input: Chest x-ray images
- Output: Several binary classification tasks
  - Right pleural effusion or not
  - Enlarged heart or not
  - Healthy or abnormal
- Very small dataset: 93 frontal chest x-ray images

Early steps of deep learning in medical imaging: using ImageNet CNN features

Bar et al. 2015

- Input: Chest x-ray images
- Output: Several binary classification tasks
  - Right pleural effusion or not
  - Enlarged heart or not
  - Healthy or abnormal
- Very small dataset: 93 frontal chest x-ray images

Early steps of deep learning in medical imaging: using ImageNet CNN features

Bar et al. 2015

- Input: Chest x-ray images
- Output: Several binary classification tasks
  - Right pleural effusion or not
  - Enlarged heart or not
  - Healthy or abnormal
- Very small dataset: 93 frontal chest x-ray images

Q: How might we approach this problem?

Bar et al. 2015

- Did not train a deep learning model on the medical data
- Instead, extracted features an AlexNet trained on ImageNet
  - 5th, 6th, and 7th layers
- Used features with an SVM (support vector machine) to perform classification
- Performed zero-mean unit-variance normalization of all features
- Evaluated combination with other hand-crafted image features (LBP, GIST, PiCoDes)
Bar et al. 2015

Q: How might we interpret the AUC vs. CNN feature trends?

Table 1. Right Pleural Effusion Condition.

<table>
<thead>
<tr>
<th></th>
<th>Low Level</th>
<th>High Level</th>
<th>Deep</th>
<th>Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LBP</td>
<td>GIST</td>
<td>PiCoDes</td>
<td>Decaf L5</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>0.71</td>
<td>0.79</td>
<td>0.79</td>
<td>0.93</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.77</td>
<td>0.92</td>
<td>0.91</td>
<td>0.84</td>
</tr>
<tr>
<td>AUC</td>
<td>0.75</td>
<td>0.93</td>
<td>0.91</td>
<td>0.92</td>
</tr>
</tbody>
</table>

Table 2. Healthy vs. Pathology.

<table>
<thead>
<tr>
<th></th>
<th>Low Level</th>
<th>High Level</th>
<th>Deep</th>
<th>Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LBP</td>
<td>GIST</td>
<td>PiCoDes</td>
<td>Decaf L5</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>0.65</td>
<td>0.68</td>
<td>0.59</td>
<td>0.73</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.61</td>
<td>0.66</td>
<td>0.79</td>
<td>0.80</td>
</tr>
<tr>
<td>AUC</td>
<td>0.63</td>
<td>0.72</td>
<td>0.72</td>
<td>0.78</td>
</tr>
</tbody>
</table>

Table 3. Enlarged Heart Condition.

<table>
<thead>
<tr>
<th></th>
<th>Low Level</th>
<th>High Level</th>
<th>Deep</th>
<th>Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LBP</td>
<td>GIST</td>
<td>PiCoDes</td>
<td>Decaf L5</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>0.75</td>
<td>0.79</td>
<td>0.79</td>
<td>0.88</td>
</tr>
<tr>
<td>Specificity</td>
<td>0.78</td>
<td>0.81</td>
<td>0.84</td>
<td>0.78</td>
</tr>
<tr>
<td>AUC</td>
<td>0.80</td>
<td>0.82</td>
<td>0.87</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Ciompi et al. 2015

- Task: classification of lung nodules in 3D CT scans as peri-fissural nodules (PFN, likely to be benign) or not
- Dataset: 568 nodules from 1729 scans at a single institution. (65 typical PFNs, 19 atypical PFNs, 484 non-PFNs).
- Data pre-processing: prescaling from CT hounsfield units (HU) into [0,255]. Replicate 3x across R,G,B channels to match input dimensions of ImageNet-trained CNNs.

Ciompi et al. 2015

- Also extracted features from a deep learning model trained on ImageNet
  - Overfeat feature extractor (similar to AlexNet, but trained using additional losses for localization and detection)
  - To capture 3D information, extracted features from 3 different 2D views of each nodule, then input into 2-stage classifier (independent predictions on each view first, then outputs combined into second classifier).
Lakhani and Sundaram 2017
- Binary classification of pulmonary tuberculosis from x-rays
- Four de-identified datasets
- 1007 chest x-rays (68% train, 17.1% validation, 14.9% test)
- Now: training CNNs from scratch as well as fine-tuning from ImageNet

![AUC Test Dataset Table]

Note.—Data in parentheses are 95% confidence interval.

* Additional augmentation of 90, 180, 270 rotations, and Contrast Limited Adaptive Histogram Equalization processing.
Lakhani and Sundaram 2017

- Binary classification of pulmonary tuberculosis from x-rays
- Four de-identified datasets
- 1007 chest x-rays (68% train, 17.1% validation, 14.9% test)
- Now: training CNNs from scratch as well as fine-tuning from ImageNet

<table>
<thead>
<tr>
<th>AUC Test Dataset</th>
<th>Untrained</th>
<th>Pretrained</th>
<th>Untrained with Augmentation*</th>
<th>Pretrained with Augmentation*</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>0.90 (0.84, 0.95)</td>
<td>0.98 (0.95, 1.00)</td>
<td>0.95 (0.90, 0.98)</td>
<td>0.98 (0.94, 0.99)</td>
</tr>
<tr>
<td>GoogeNet</td>
<td>0.88 (0.81, 0.92)</td>
<td>0.97 (0.93, 0.99)</td>
<td>0.94 (0.89, 0.97)</td>
<td>0.98 (0.94, 1.00)</td>
</tr>
<tr>
<td>Ensemble</td>
<td></td>
<td></td>
<td></td>
<td>0.99 (0.96, 1.00)</td>
</tr>
</tbody>
</table>

Note.—Data in parentheses are 95% confidence interval.
* Additional augmentation of 90, 180, 270 rotations, and Contrast Limited Adaptive Histogram Equalization processing.

All training images were resized to 256x256 and underwent base data augmentation of random 227x227 cropping and mirror images. Additional data augmentation experiments in results table.

Lakhani and Sundaram 2017

- Binary classification of pulmonary tuberculosis from x-rays
- Four de-identified datasets
- 1007 chest x-rays (68% train, 17.1% validation, 14.9% test)
- Now: training CNNs from scratch as well as fine-tuning from ImageNet

All training images were resized to 256x256 and underwent base data augmentation of random 227x227 cropping and mirror images. Additional data augmentation experiments in results table.

Lakhani and Sundaram 2017

Performed further analysis at optimal threshold determined by the Youden Index.

Gulshan et al. 2016

- Binary classification of referable diabetic retinopathy from retinal fundus photographs (moderate and worse diabetic retinopathy, referable diabetic macular edema, or both)
- Huge data curation effort! 128,175 images, each graded by 3-7 ophthalmologists.
  - 54 total graders, each paid to grade between 20 to 62508 images.
- Inception-v3 (GoogLeNet) CNN with ImageNet pre-training

Gulshan et al. 2016

Looked at different operating points
- High-specificity point approximated ophthalmologist specificity for comparison. Should also use high-specificity to make decisions about high-risk actions.
- High-sensitivity point should be used for screening applications.

Gulshan et al. 2016

Gulshan et al. 2016

Q: What could explain the difference in trends for reducing # grades / image on training set vs. tuning set, on tuning set performance?

Summary

Today we saw:

- Convolutional and pooling layers
- Concrete formulations of widely used CNN architectures
- Some case studies of CNNs for medical image classification

Next time:

- A few more examples of CNNs for medical image classification
- Medical image detection and segmentation