Artificial Intelligence, Machine Learning, and Deep Learning

ECE 5470 Lecture 23

Anthony P. Reeves
Vision and Image Analysis Group
School of Electrical and Computer Engineering
Cornell University

Artificial Intelligence, Machine Learning, Deep Learning

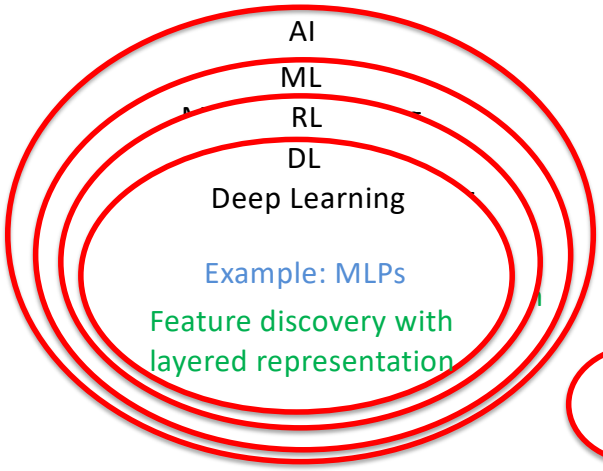
Artificial intelligence (**AI**), sometimes called **machine intelligence**, is intelligence demonstrated by machines, in contrast to the **natural intelligence** displayed by humans and other animals. ??

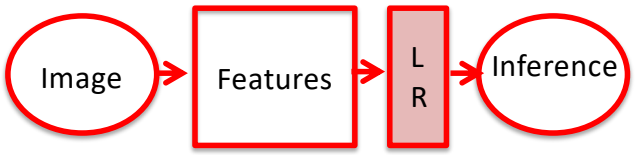
*APR -- **Artificial intelligence:** Computer algorithms that give the appearance of intelligent behavior in decision making.*

Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) with data, **without being explicitly programmed**.[2]

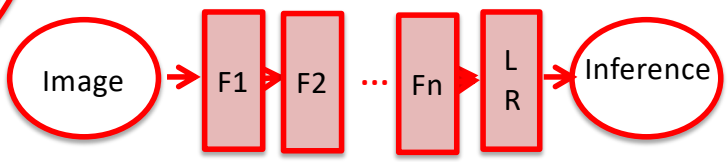
Wikipedia

Venn Diagram: Machine Learning





ML Logistic Regression

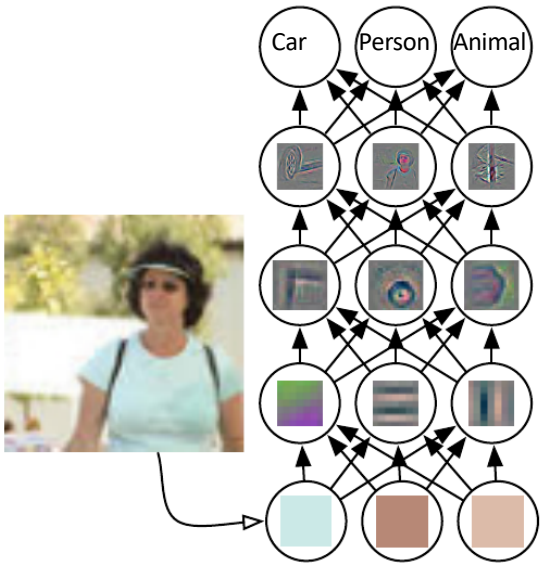


ML MLP Deep Learning

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Cornell University Vision and Image Analysis Group

Biologically Inspired Image Analysis Model?



Output (Object Identity)

3rd hidden layer (object parts

2nd hidden layer (corners and contours)

1st hidden layer (edges)

Visible layer (input pixels)

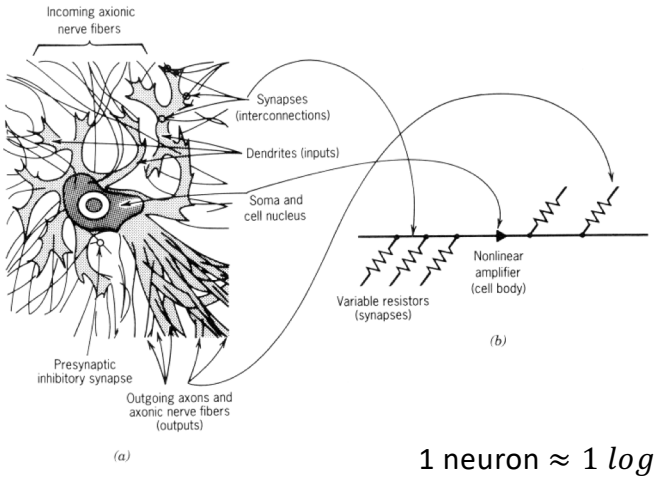
For Computer Vision

1 – 100+ layers

Neural Networks: The Neuron

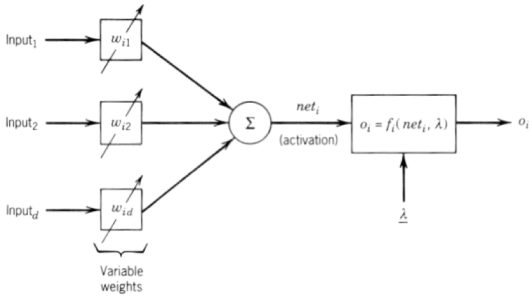
The mechanism for Deep Learning

Biological Neuron



Electronic Neuron

Artificial Neuron (Perceptron)



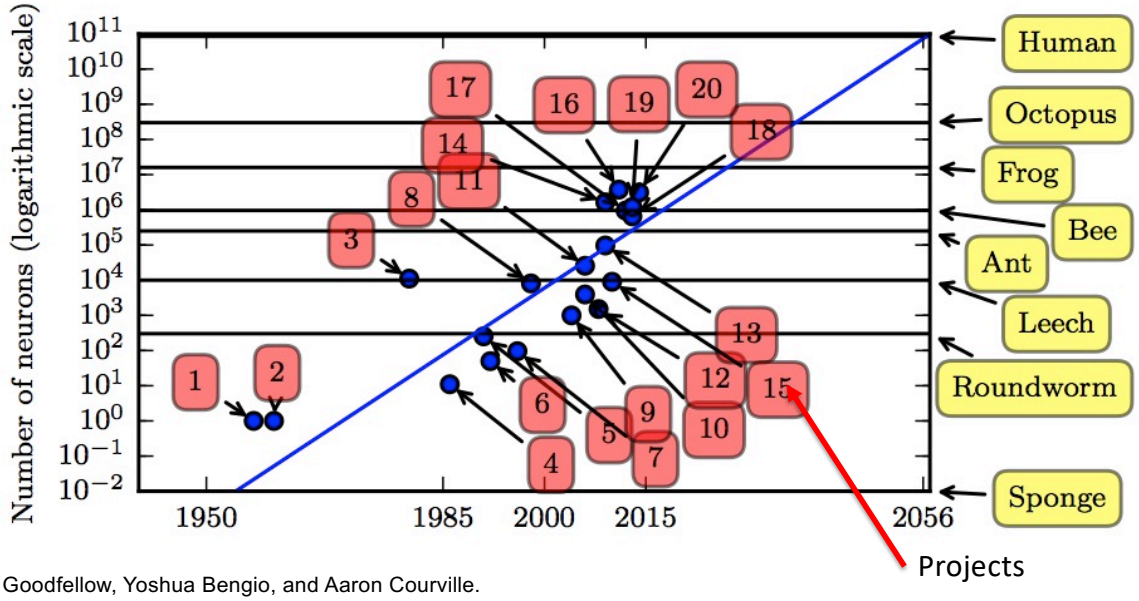
Matrix product + non-linear function

1 neuron ≈ 1 *logistic regression*

~10,000 dendrites

1 layer of n neurons ≈ *multinomial logistic regression*

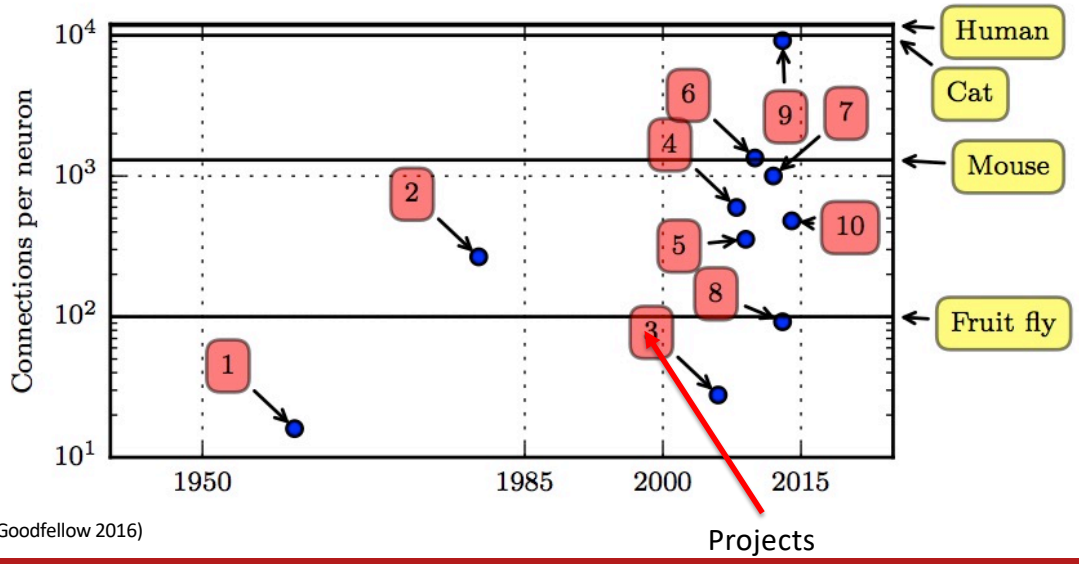
Historical Trends: Number of Neurons



Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

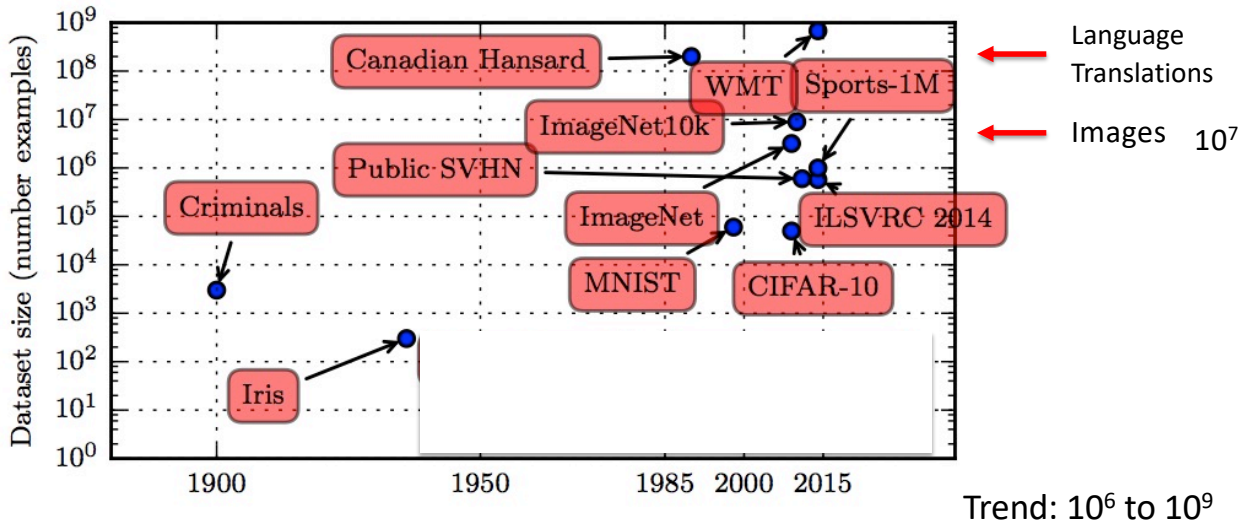
Size double every 2.4 years

Historical Trends: Connections per Neuron



(Goodfellow 2016)

Historical Trends: Growing Datasets



Trend: 10^6 to 10^9

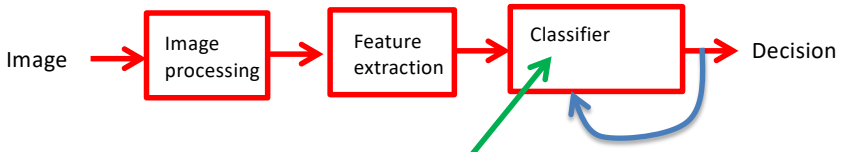
Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

(Goodfellow 2016)

Machine Learning (AI):

Traditional machine learning

Image processing + feature extraction + classification (ML)



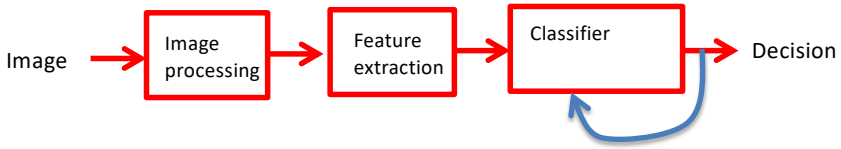
Machine Learning Methods

1. Logistic Regression
2. Parametric Classifiers (Gaussian)
3. Neural Networks
4. Support Vector Machines
5. Random forest of trees
6. …

Iterative Learning (optimization)

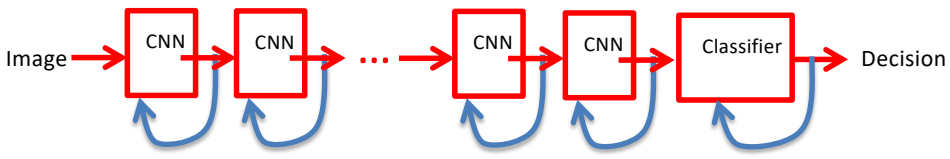
Deep Learning

Traditional machine learning



Added ingredient:

Deep Learning (Convolutional Neural Networks (CNN))



3 – 100+ stages

Convolutional Neural Networks and Deep Learning for Computer Vision

Anthony P. Reeves
Vision and Image Analysis Group
School of Electrical and Computer Engineering
Cornell University

Almost all slides are taken from presentations found on the web.
Attribution is given where possible

Convolutional Networks

- Concept: preform the low-level convolution and feature extraction stages of computer vision using neural networks.
- Components
  - Convolution layer: each neuron behaves as an individual convolution filter with a non-linear output function
  - Pooling layer: outputs from the previous layer are combined by average or max operation
  - Special purpose layers: L2 pooling, local contrast enhancement…

Image Convolution Example

Computer Vision: Algorithms and Applications (September 3, 2010 draft)

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

f(x,y)

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

g(x,y)

=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

h(x,y)

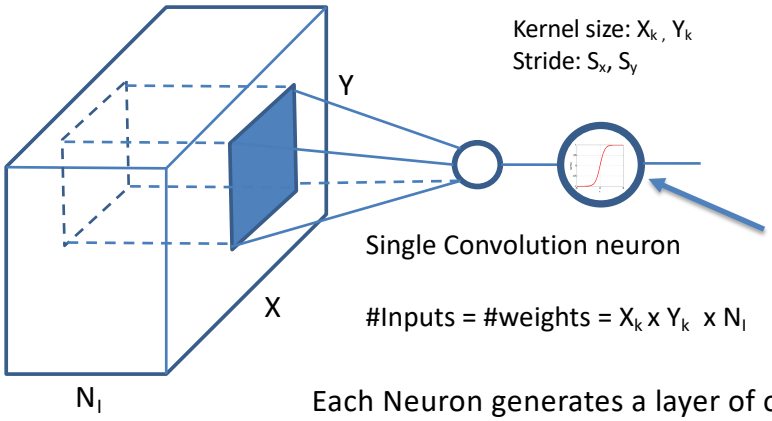
Kernel weights often sum to 1

Richard Szeliski

Convolutional Neural Network

Parameters

1. Kernel size
2. Stride
3. Padding
4. # Output layers
5. Activation function



#Inputs = #weights = X_k x Y_k x N_I

Each Neuron generates a layer of outputs

Layer size = $\lceil (X - X_k + 1)/S_x \rceil$ * $\lceil (Y - Y_k + 1)/S_y \rceil$

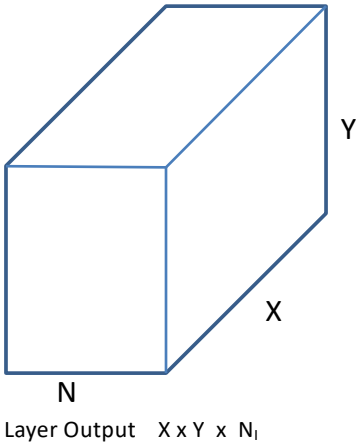
N_k neurons => N_k output layers

Input X x Y x N_I N_I = input layers

Example: color image = 3 layers

$\lfloor (X - X_k)/S_x \rfloor + 1$

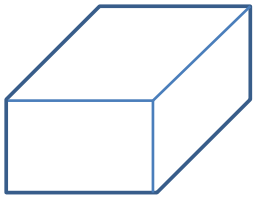
Convolution Neural Network Feature Size Reduction



Number of outputs = X * Y * N

Reduction Methods

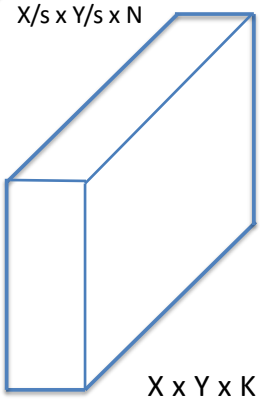
1. Use a stride. S > 1
   Output dimensions ≈ (X/S) * (Y/S) * N



2. Use pooling of local outputs (S x S)
   Output dimensions = (X/S) * (Y/S) * N
   Pooling functions: max, mean …

X/s x Y/s x N

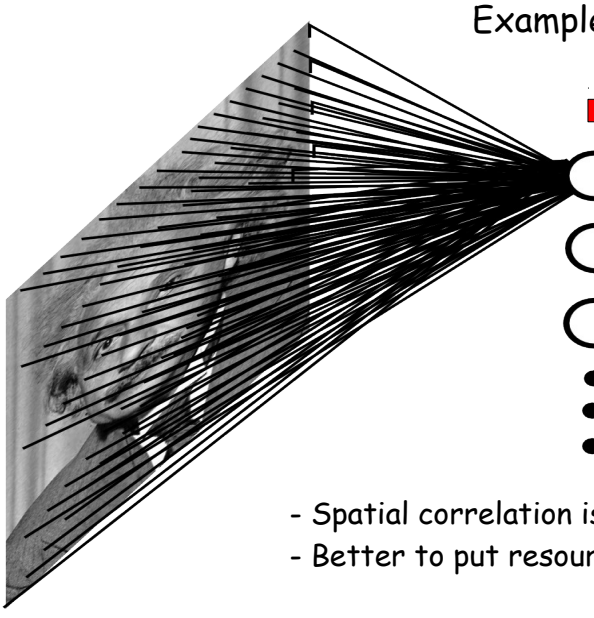
3. Use 1 x 1 conv network layer of size K < N
   Output dimensions =. X * Y * K
   Requires K N-input Neurons
   -- 2 –level CNN



X x Y x K

More neurons (N) => more power but makes next layer more complex
How to manage/reduce all the outputs?

FULLY CONNECTED NEURAL NET



Example: 1000x1000 image
1M hidden units
10^12 parameters!!!

- Spatial correlation is local
- Better to put resources elsewhere!

59

Ranzato

NEURAL NETS FOR VISION

A standard neural net applied to images:

-scales quadratically with the size of the input

- does not leverage stationarity

Solution:

- connect each hidden unit to a small patch of the input

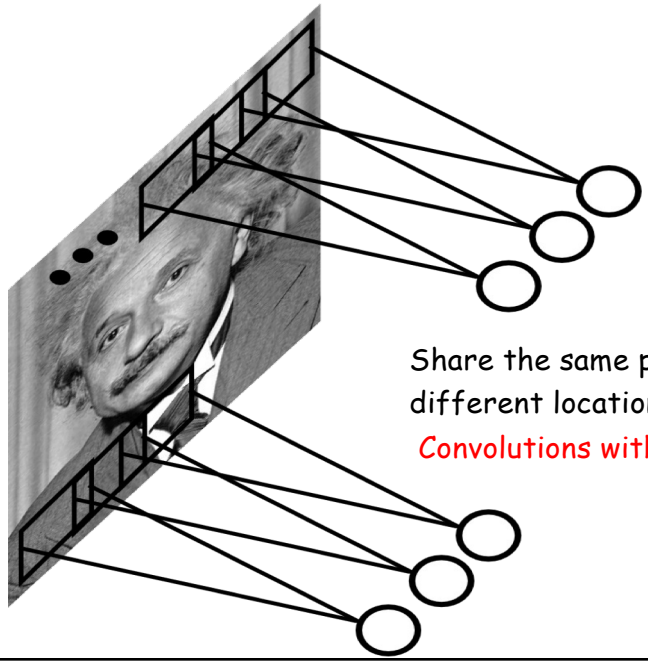
-share the weight across hidden units

This is called: **convolutional network.**

LeCun et al. "Gradient-based learning applied to document recognition" IEEE 1998

17 Ranzato

CONVOLUTIONAL NET

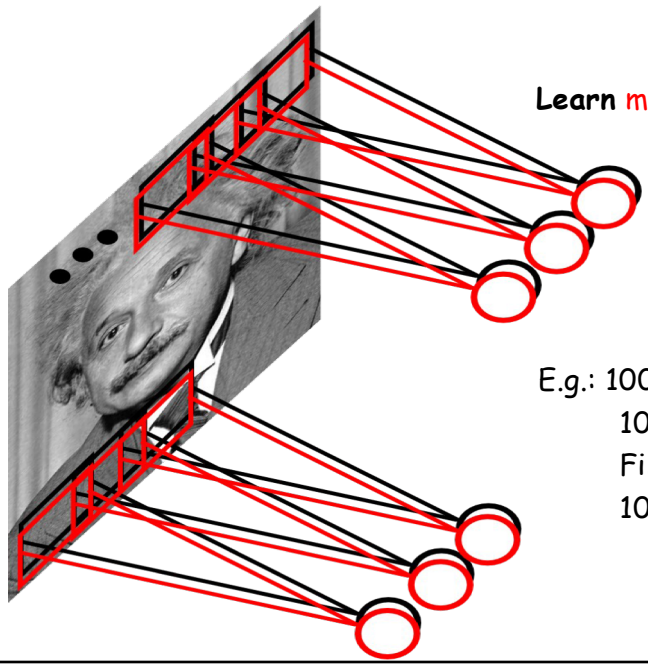


Share the same parameters across different locations:

Convolutions with learned kernels

18 Ranzato

CONVOLUTIONAL NET



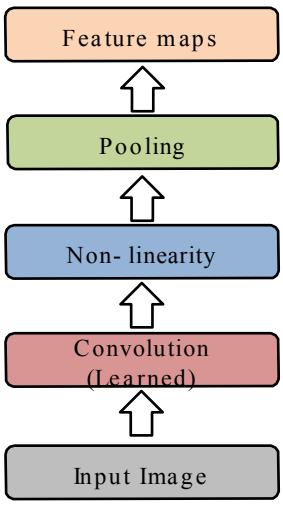
Learn multiple filters.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

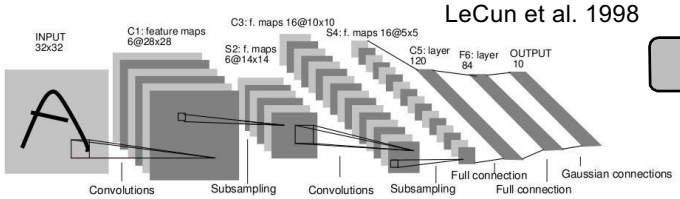
19
Ranzato

Convolutional Neural Networks

- Feed-forward:
  - Convolve input
  - Non-linearity (rectified linear)
  - Pooling (local max)
- Supervised
- Train convolutional filters by back-propagating classification error

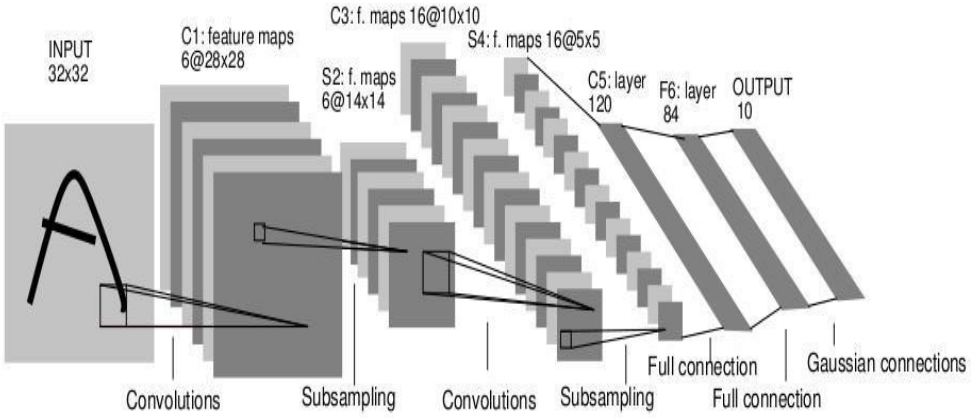


LeCun et al. 1998



Slide: R. Fergus

Convolutional Neural Networks



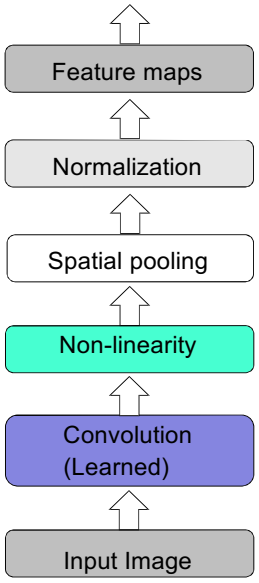
LeNet. 1998

Slide: R. Fergus

Convolutional Neural Networks (CNN, Convnet)

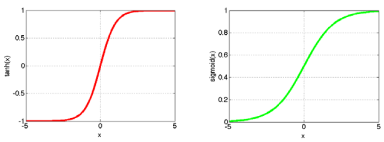
Circa. 2012

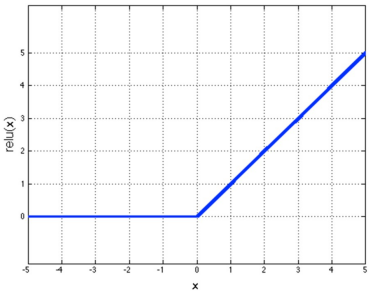
- Feed-forward feature extraction:
  1. Convolve input with learned filters
  2. Non-linearity
  3. Spatial pooling
  4. Normalization
- Supervised training of convolutional filters by back-propagating classification error



2. Non-Linearity

- Per-element (independent)
- Options:
  - Tanh
  - Sigmoid: 1/(1+exp(-x))
  - Rectified linear unit (ReLU)
    - Simplifies backpropagation
    - Makes learning faster
    - Avoids saturation issues
      → Preferred option

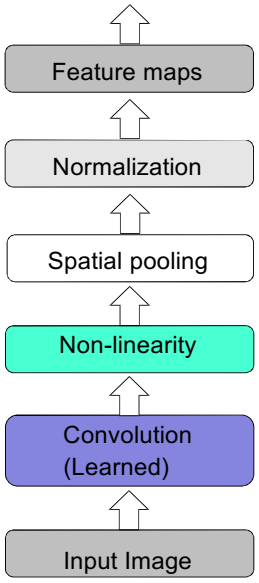




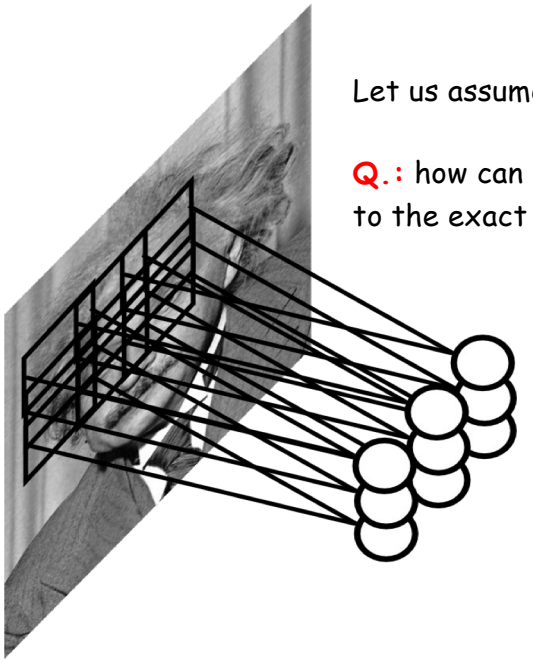
Convolutional Neural Networks (CNN, Convnet)

Circa. 2012

- Feed-forward feature extraction:
  1. Convolve input with learned filters
  2. Non-linearity
  3. Spatial pooling
  4. Normalization
- Supervised training of convolutional filters by back-propagating classification error



CONVOLUTIONAL NET



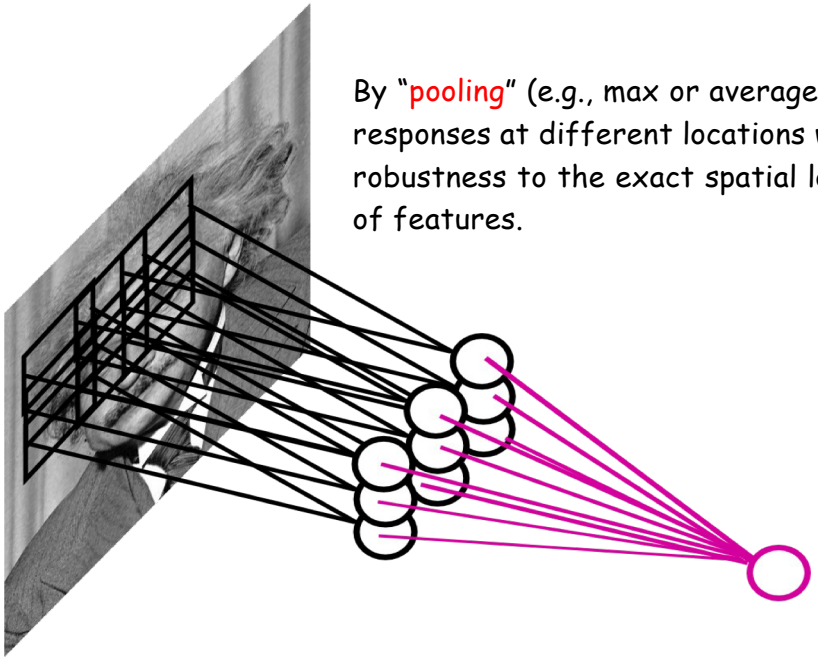
Let us assume filter is an "eye" detector.

Q.: how can we make the detection robust to the exact location of the eye?

25

Ranzato

CONVOLUTIONAL NET



By "pooling" (e.g., max or average) filter responses at different locations we gain robustness to the exact spatial location of features.

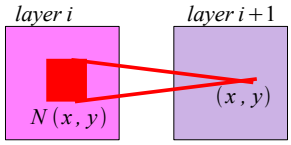
26

Ranzato

CONV NETS: EXTENSIONS

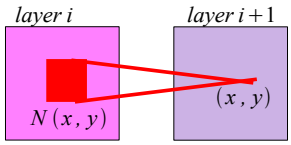
Over the years, some new modules have proven to be very effective when plugged into conv-nets:

- **L2 Pooling**



$$h_{i+1,x,y} = \sqrt{\sum_{(j,k)\in N(x,y)} h_{i,j,k}^2}$$

- **Local Contrast Normalization**



$$h_{i+1,x,y} = \frac{h_{i,x,y} - m_{i,N(x,y)}}{\sigma_{i,N(x,y)}}$$

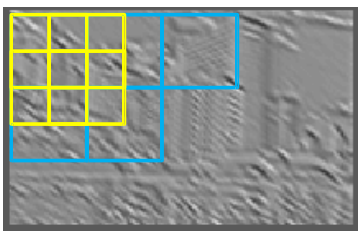
Jarrett et al. "What is the best multi-stage architecture for object recognition?" ICCV 2009

68

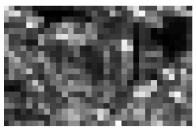
Ranzato

3. Spatial Pooling

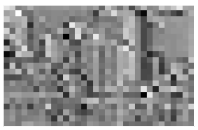
- Sum or max
- Non-overlapping / overlapping regions
- Role of pooling:
  - Invariance to small transformations
  - Larger receptive fields (see more of input)



Max

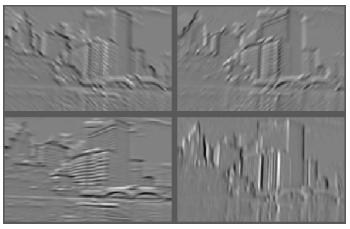


Sum

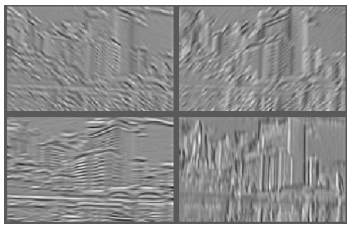


4. Normalization

- Within or across feature maps
- Before or after spatial pooling



Feature Maps



Feature Maps
After Contrast Normalization

Convolution Network Summary

- Dependencies are local
- Translation invariance
- Few parameters (filter weights)
- Stride can be greater than 1 (faster, less memory)

Input

Feature Map

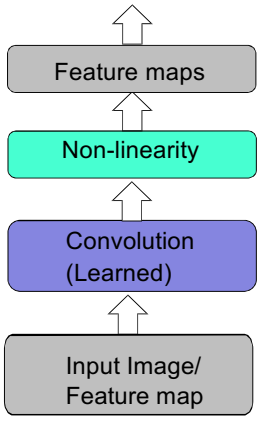
Topics

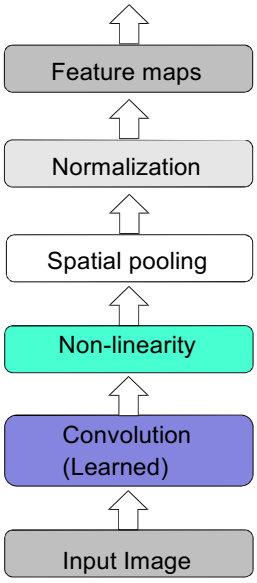
- Recent CNN models
- Determining Hyperparameter Values
- Programming CNN models

Convolutional Neural Networks (CNN, Convnet)

Building blocks

Circa. 2012





Not Common

Optional

CNN Recent Developments: Batch normalization

- Standardize outcome of convolution *before* activation function:
  1. Convolve input with learned filters
  2. Batch Normalization
  3. Non-linearity
  4. Spatial pooling

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

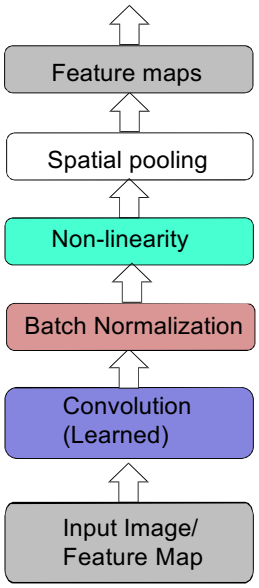
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

Concept: standardize each output of the convolution by the mean and standard deviation for that mini-batch.

For testing, accumulate mean and standard deviation from selected training batches



GoogleNet Inception Module

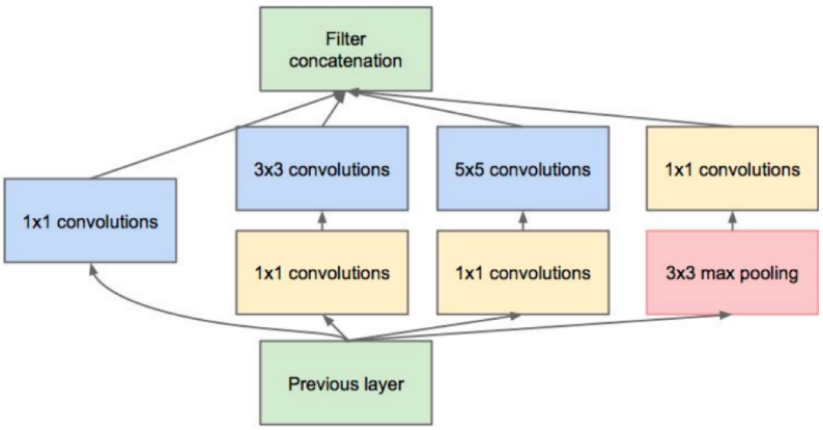


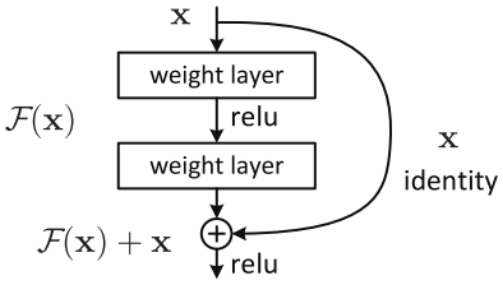
Fig. 6 - Inception module

GoogleNet/Inception(2014)

22 layers. 4 million parameters

Inception modules

ResNet



Concept

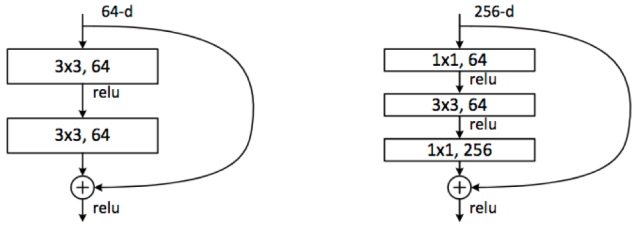


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

Examples

Deep Learning Training

How to find the best hyperparameter settings for a Model?

Hyperparameters:

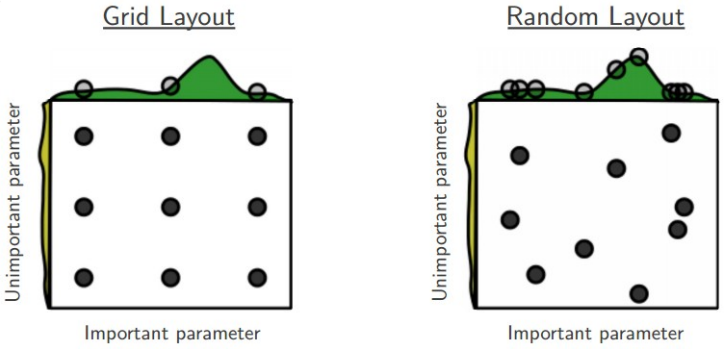
Hyperparameter Search Strategy
Number of epochs
Size of minibatch (power of 2, fit into memory, typically 32, 64, 128…)
Strategy for learning rate decrease (see examples)
Drop out strategy for training (probability 0.2 for input, 0.5 for hidden neurons)
---Weight initialization
Optimization Gradient Descent Algorithm
Data Augmentation

Model Parameters:

- Number and sizes of layers
- Layer architecture
- Regularization (type and amount)
- Neuron functions

Hyperparameter search: Grid or Random

Prefer random search to grid search. As argued by Bergstra and Bengio in Random Search for Hyper-Parameter Optimization, "randomly chosen trials are more efficient for hyper-parameter optimization than trials on a grid". As it turns out, this is also usually easier to implement.



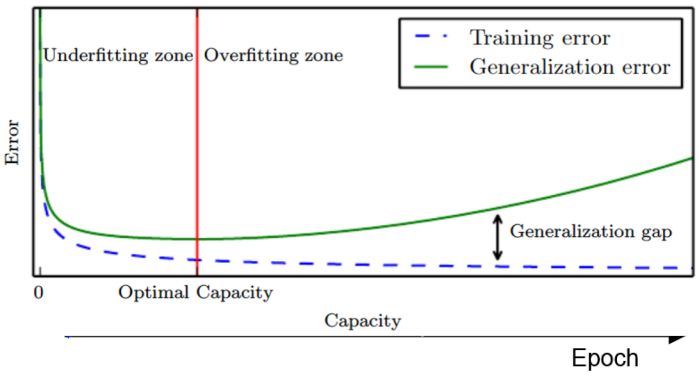
Bayesian Hyperparameter Optimization is a whole area of research devoted to coming up with algorithms that try to more efficiently navigate the space of hyperparameters.

Early Stopping

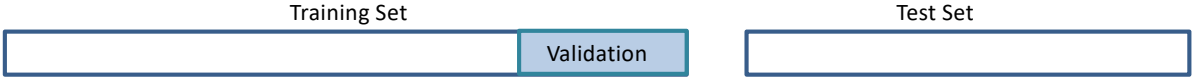
DL4J

How many full passes of the data set (epochs) should be used?
If we use too few epochs, we might underfit (i.e., not learn everything we can from the training data);
if we use too many epochs, we might overfit (i.e., fit the 'noise' in the training data, and not the signal).

1. Split data into training and test sets
2. At the end of each epoch (or, every N epochs):
   a) evaluate the network performance or the test set
   b) if the network outperforms the previous best model: save a copy of tl network at the current epoch
3. Take as our final model the model that ha the best test set performance



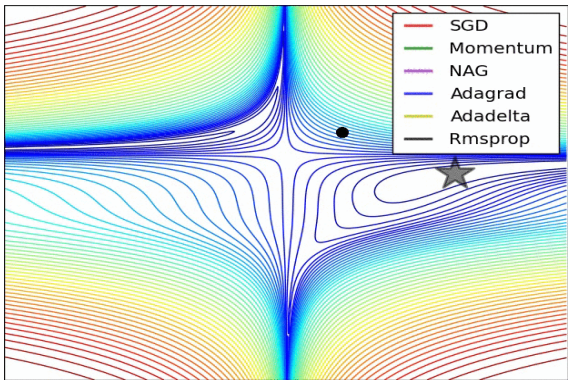
Note, you cannot use the real test set!
Select part of the training set as a validation set.

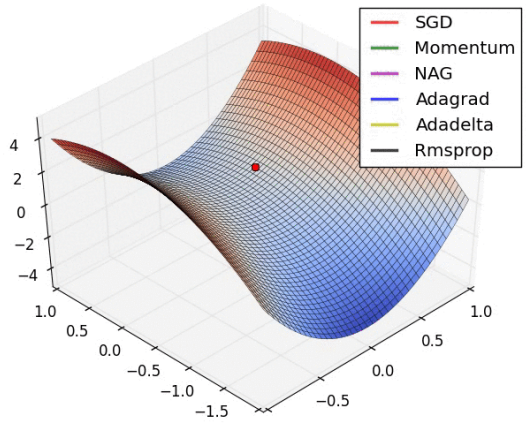


Gradient descent optimization algorithms

Sebastian Ruder. Sebastian is a PhD student in Natural Language Processing and a research scientist at AYLIEN.

https://www.datasciencecentral.com/profiles/blogs/an-overview-of-gradient-descent-optimization-algorithms

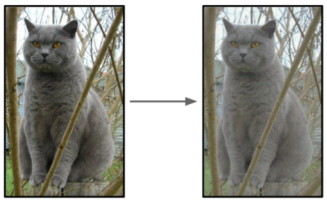




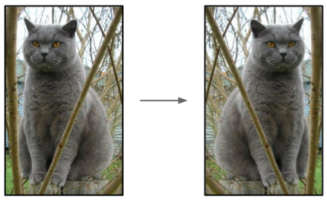
Deep Learning Training: Data Augmentation

Random mix/combinations of :
- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

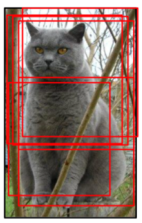
Randomly jitter contrast



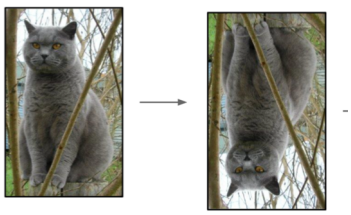
Horizontal Flip



Random crops/scales



Vertical Flip?



Weight Initialization

- Weights must be initially set to some non-zero values
- Typically small random values are used
- Selecting an appropriate range of values may improve the speed or learning.

Transfer learning

- For deep learning systems, training from scratch may be very computationally expensive. In transfer learning, weights from a previously trained task may be used to initialize a network for a new task.
- For some tasks the early stages of the network may not need to be retrained or further adapted.

Convolutional Neural Networks: Implementations

Anthony P. Reeves
Vision and Image Analysis Group
School of Electrical and Computer Engineering
Cornell University

Scikit-Learn Models

K-Nearset-Neighbor Classifier:

```
KNN = KNeighborsClassifier(n_neighbors=3)
```

MLP with two hidden layers:

```
mlp_2 = MLPClassifier(hidden_layer_sizes=(50,50), max_iter=10, alpha=1e-4,
                      solver='sgd', verbose=10, tol=1e-4, random_state=1,
                      learning_rate_init=.1)

mlp_2.fit(X_train, y_train)
```

```
print("Test set score: %f" % mlp_2.score(X_test, y_test))
```

Keras Model for CNN

```
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'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
```

PyTorch Model for CNN

```
# Convolutional neural network (two convolutional layers)
class ConvNet(nn.Module):
    def __init__(self, num_classes=10):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(7*7*32, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out
```

PyTorch Model

```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

PyTorch Model

```
# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

PyTorch Model

```
# Test the model
model.eval()  # eval mode (batchnorm uses moving mean/variance instead of
mini-batch mean/variance)
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Test Accuracy of the model on the 10000 test images: {} %'.format(1
00 * correct / total))
```

```
Test Accuracy of the model on the 10000 test images: 98.94 %
```