Lab 5: More on CNN

Outline

Part 1: Image Databases for ML

Part 2: Applications of CNNs

Part 3: CNN Architectures:

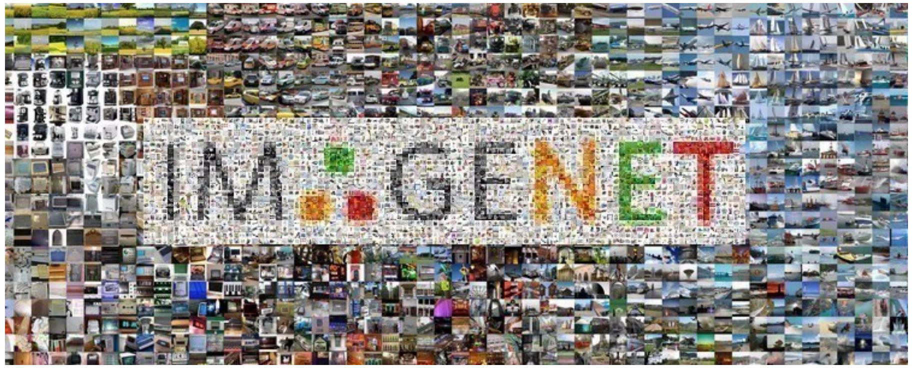
- AlexNet (2012)
- VGG-Net (2014)
- Google-Net (2014)
- Residual-Net (2015)

Part 4: Image Segmentation Example with Fully Convolutional Network

Part 5: Lab Assignment

Part 1: Image Databases for ML

Image Databases

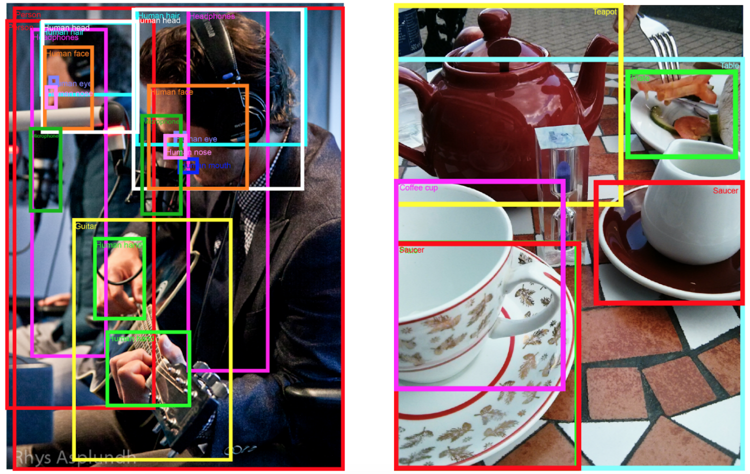


ImageNet

COCO

COCO
Common Objects in Context

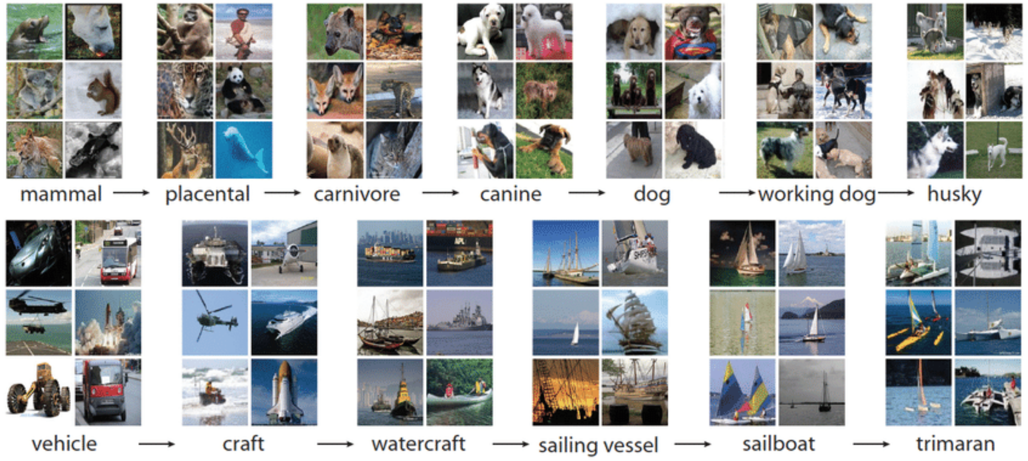
Google Open Images



ImageNet

Large visual database for the visual recognition research

- More than >14 million hand-annotated images
- More than >21k categories
- ImageNet Large Scale Visual Recognition Challenge (ILSVRC) for algorithm evaluation

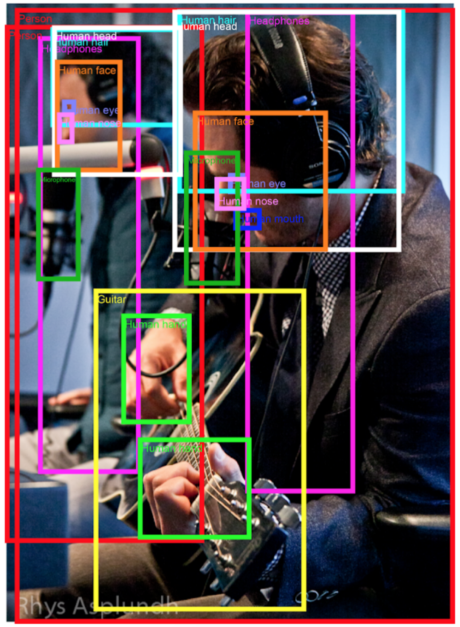


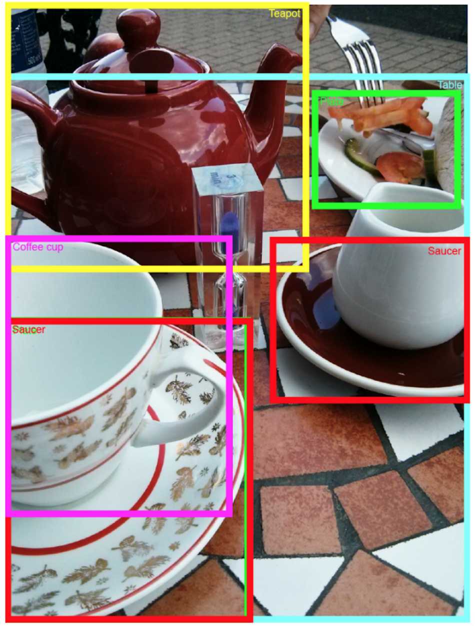
Website: http://www.image-net.org/

Google Open Images

Large-scale object detection, segmentation, and captioning dataset

- >9 mil images annotated
- >16 mil bounding boxes for 600 classes
- Visual relationship annotations (e.g. woman playing guitar)
- Image segmentation for some images





Website: https://storage.googleapis.com/openimages/web/index.html

COCO (Common Objects in Context)

Large-scale object detection, segmentation, and captioning dataset

- 330k images (>200k labeled)
- 1.5 mil object instances
- Object segmentation for all images
- 5 visual relation annotations/image

Website: https://cocodataset.org/#home

Part 2: Applications of CNNs

Semantic Segmentation

- Image analysis procedure of assigning each pixel into a class
- Human brain is capable of high levels of semantic segmentation
- CNNs can be used for semantic segmentation

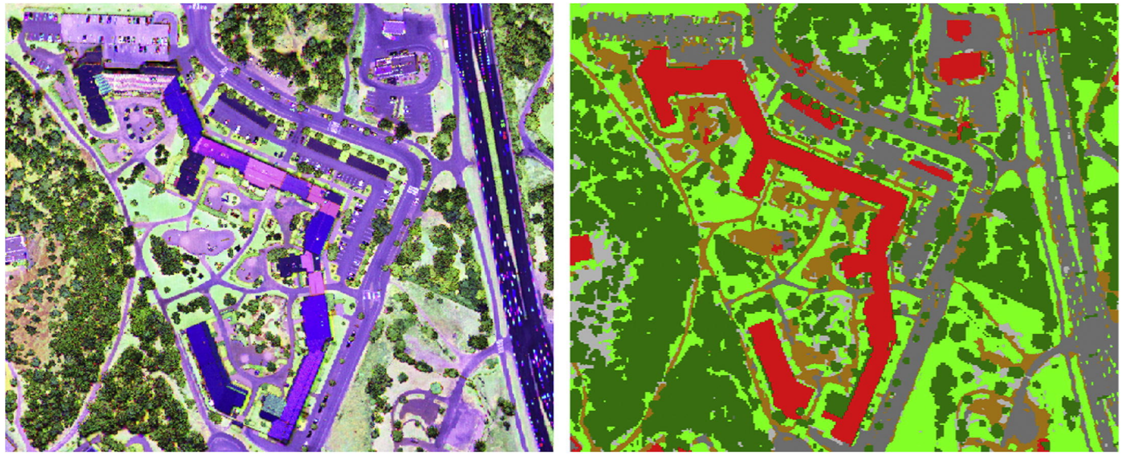
Semantic segmentation →

Image credit: https://learnopencv.com/pytorch-for-beginners-semantic-segmentation-using-torchvision/

Applications of Semantic Segmentations

Facial Segmentation

Autonomous Driving



Geo Land Sensing

Image credit: https://learnopencv.com/pytorch-for-beginners-semantic-segmentation-using-torchvision/

Visual Recognition

- Visual recognition assigns an image into a class

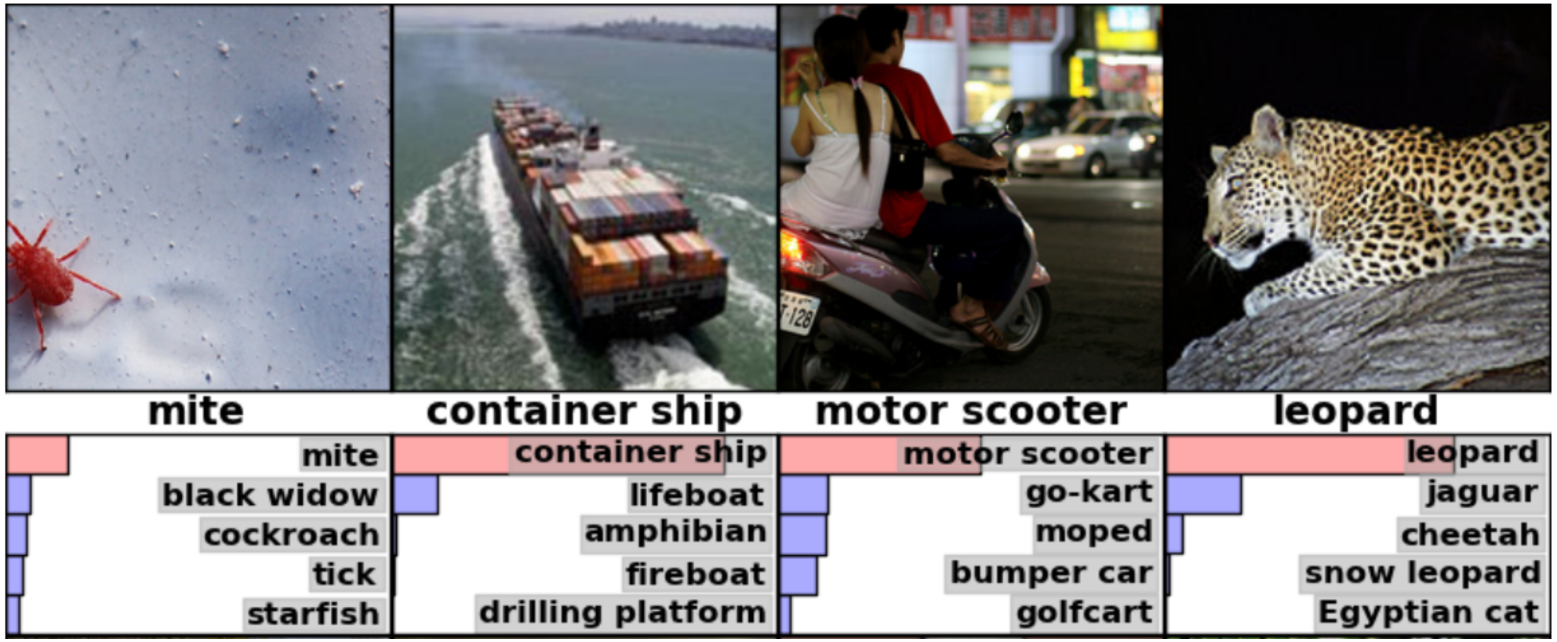
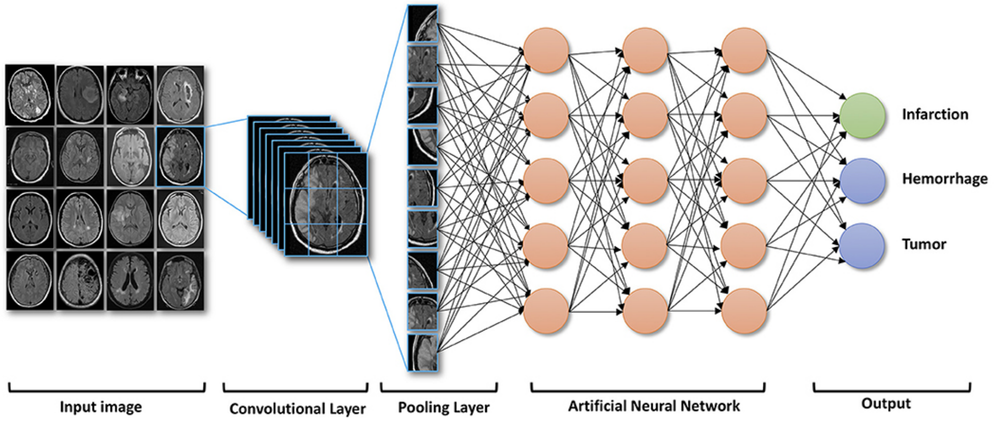
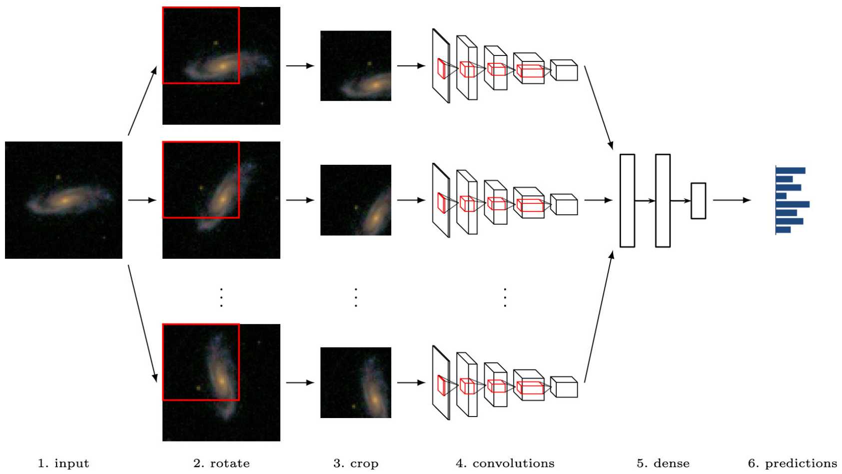


Image credit:
https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

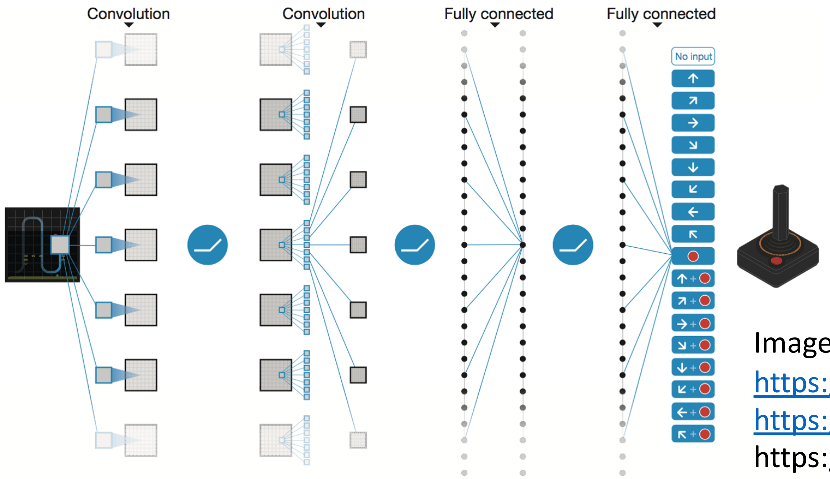
Applications of Visual Recognition



Medical Diagnosis



Astronomical Image Analysis

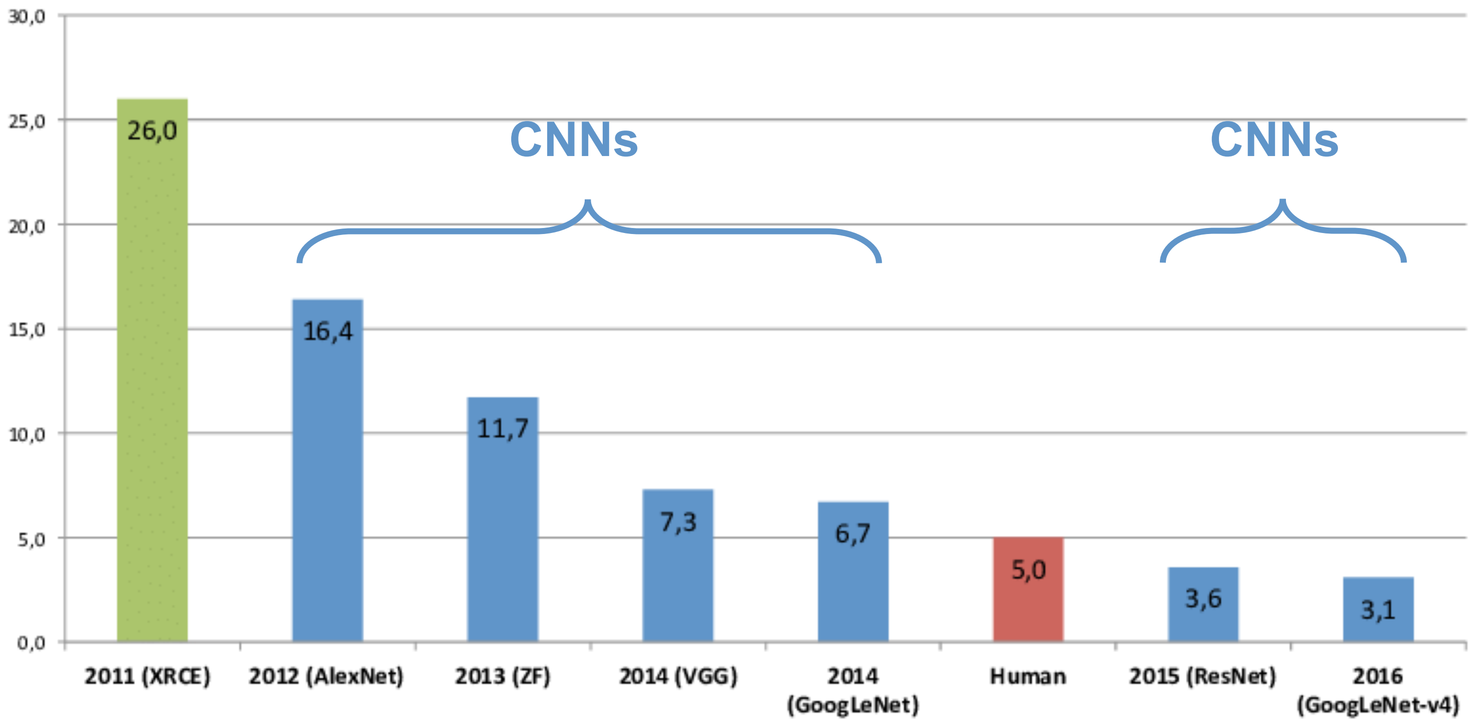


Application in AI Plays Games

Image credits:
https://www.frontiersin.org/articles/10.3389/fneur.2019.00869/full
https://jwuphysics.github.io/blog/galaxies/astrophysics/
https://www.nature.com/articles/nature14236

CNNs are leading algorithm for visual recognition

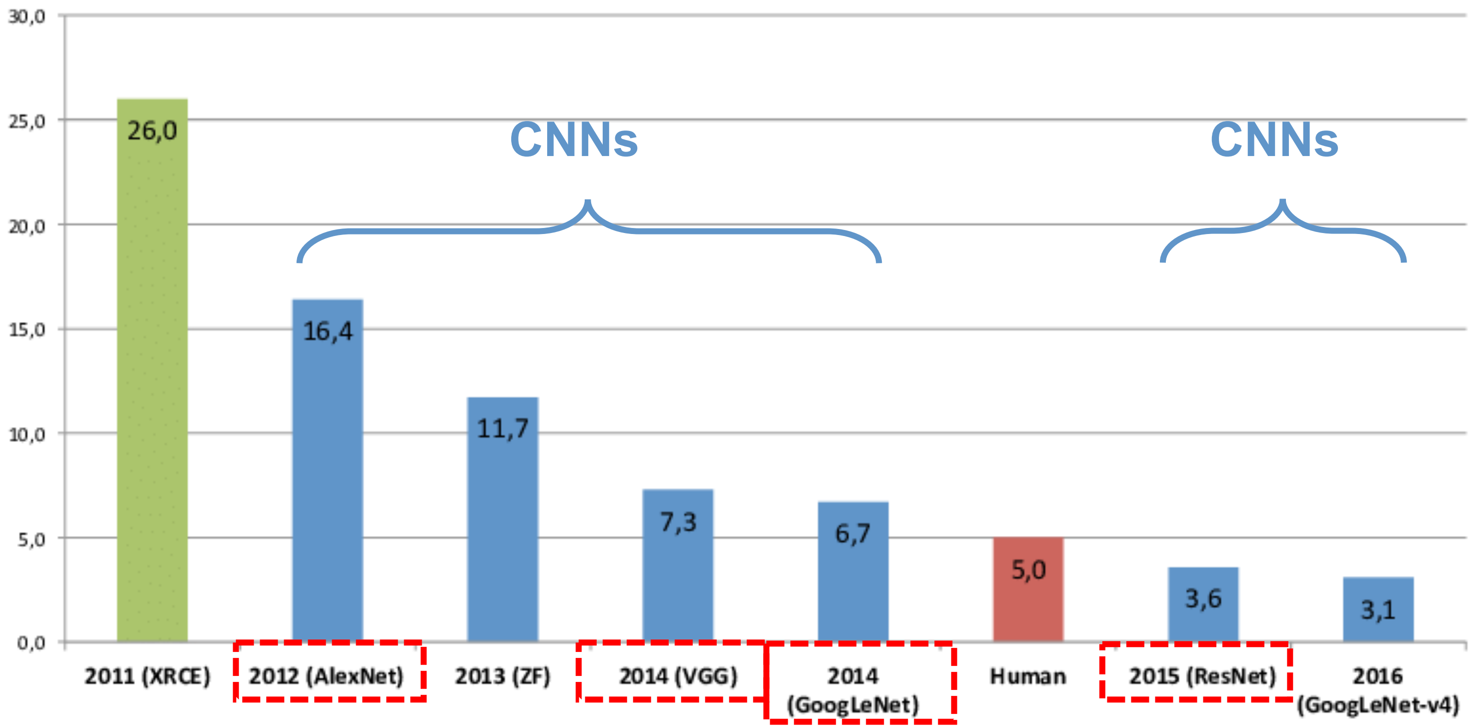
Image credit: Edge ai + Vision Alliance



ILSVRC winners over time

CNNs are leading algorithm for visual recognition

Image credit: Edge ai + Vision Alliance



ILSVRC winners over time

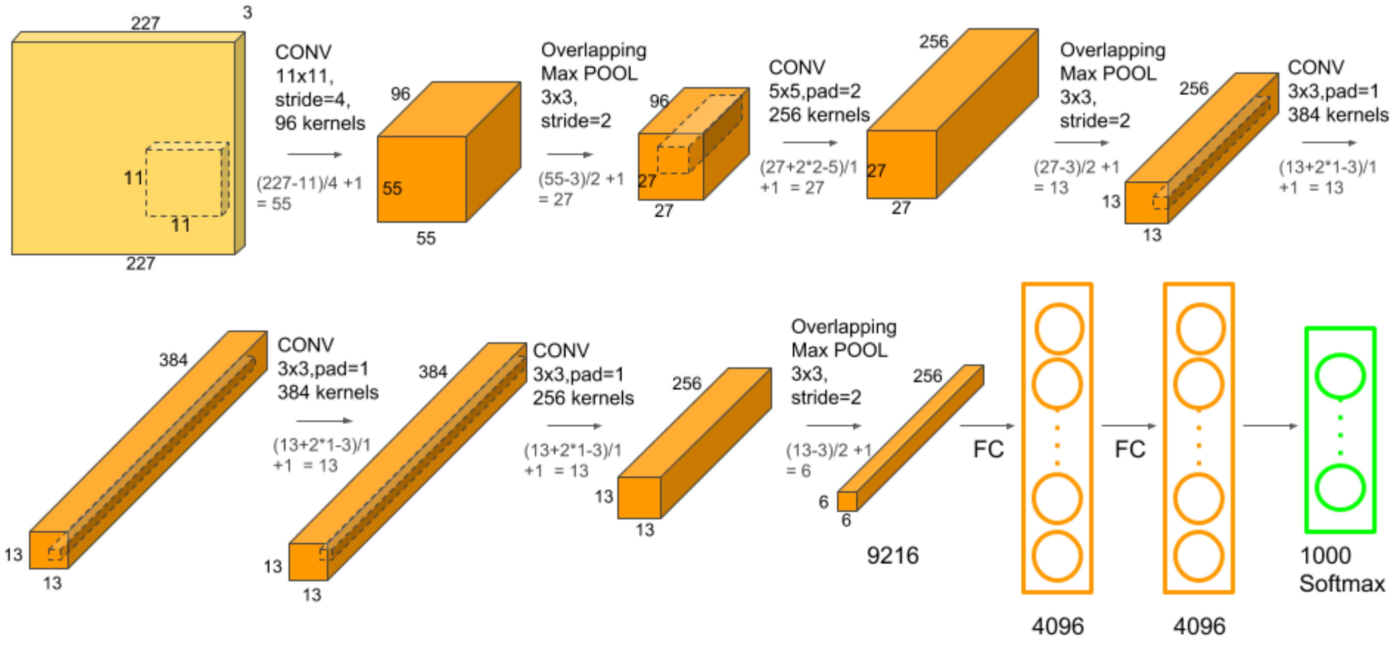
Part 3: CNN Architectures

AlexNet (2012)

- Designed by Alex Krizhevsky (University of Toronto)
- Winner of ILSVRC 2012 (Top-5 error of 15.3%, 10% improvement from 2011)
- Depth of the model as the essential component for high performance
- Utilized GPUs during training for faster computation

- Link to the original paper - https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

AlexNet Architecture



Input layer: 227 * 227 * 3
Output layer: 1000 categories

8-layers in total
- 5 CV layers
- 3 MP layers
- 3 FC layers

First 2 CV layers are connected to **Max Pooling** layers

Uses **ReLU** activation function

Uses **Dropout layers** for first two FC

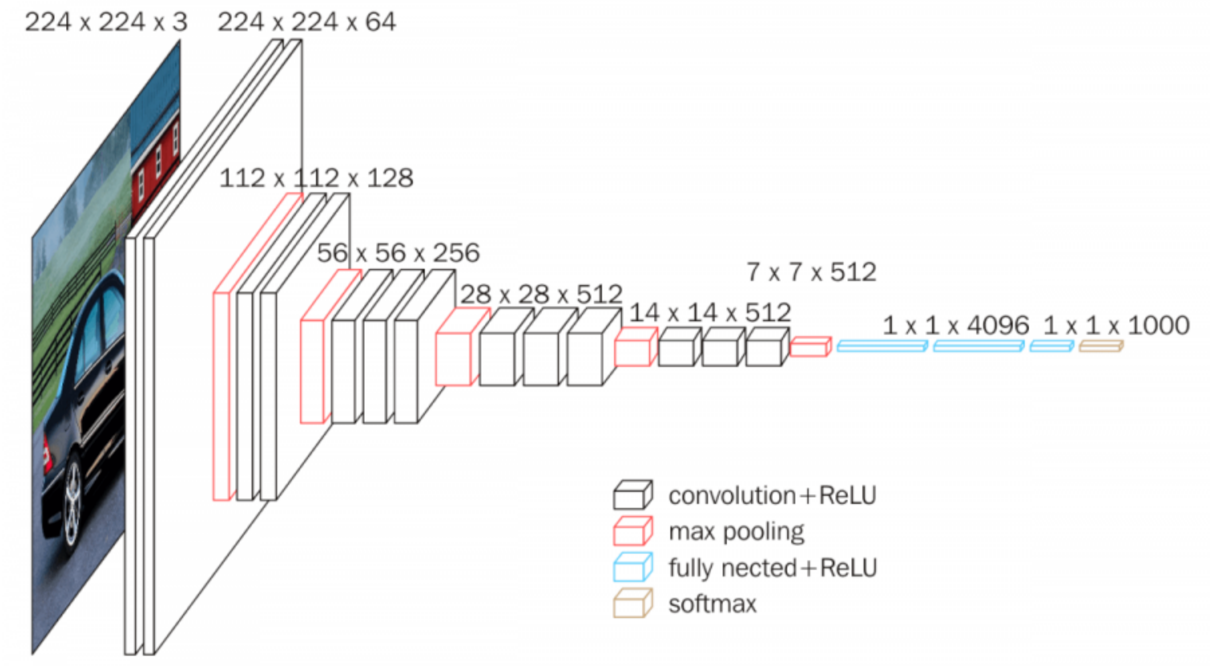
Total # of parameters: >60M

VGG Network (2014)

- Designed by Visual Geometry Group in University of Oxford
- 2nd highest accuracy in ILSVRC 2014 (Top-5 error of 7.3%)
- Uses very small receptive fields throughout the network (3x3 with a stride of 1)
- Introduces convolution block layers (sequence of convolutions -> max pooling)
- Effective architecture for feature extractions in images

- Link to the original paper - https://arxiv.org/abs/1409.1556

VGG16 Architecture



PyTorch Implementation:
https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py

Input layer: 224 * 224 * 3
Output layer: 1000 categories

8-layers in total
- 13 CV layers
- 5 MP layers
- 3 FC layers

Consists of **5 VGG blocks**

Uses **ReLU** activation function

Uses **Dropout layers** for first two FC

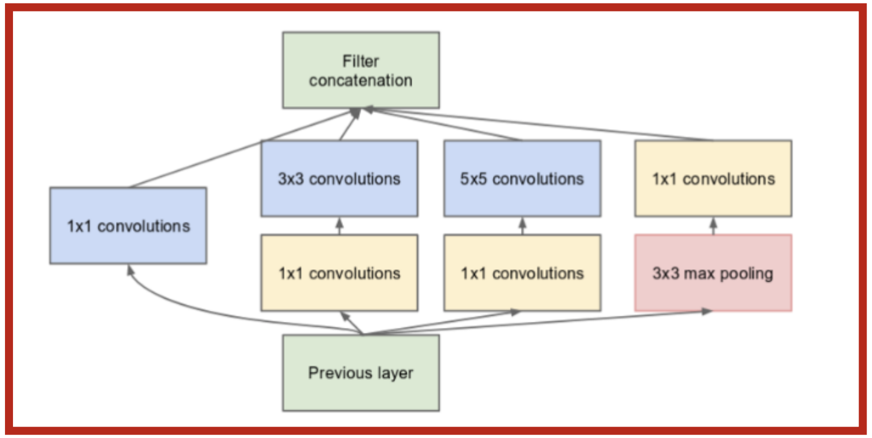
Total # of parameters: > 138M

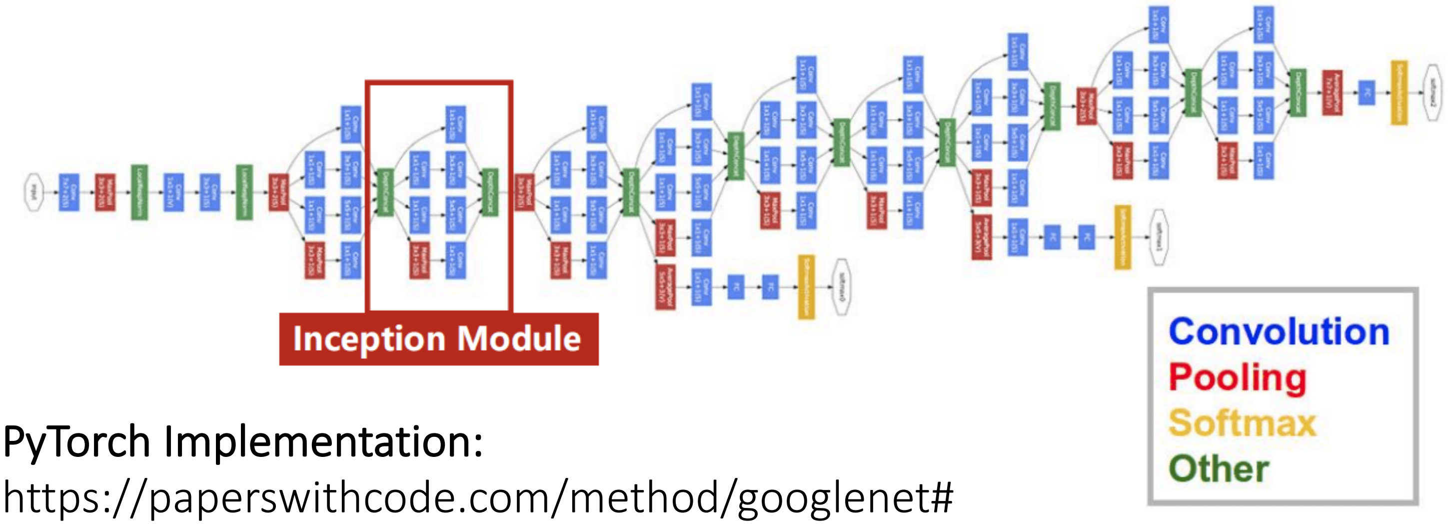
Google-Net (2014)

- Designed by Google
- Also known as Inception Net
- Winner of ILSVRC 2014 (Top-5 error of 6.7%)
- Introduces multiple convolution filters acting on same level (inception module)
- Inception module decreases the number of parameters and alleviates overfitting

- Link to the original paper - https://arxiv.org/abs/1409.4842

Google-Net Architecture





Input layer: 224 * 224 * 3
Output layer: 1000 categories

Total 22 layers with **27 pooling layers**

9 inception modules

Global pooling layer at the end

Dropout layer for FC

2 auxiliary classifiers to improve convergence during training

Total # of parameters: >7M

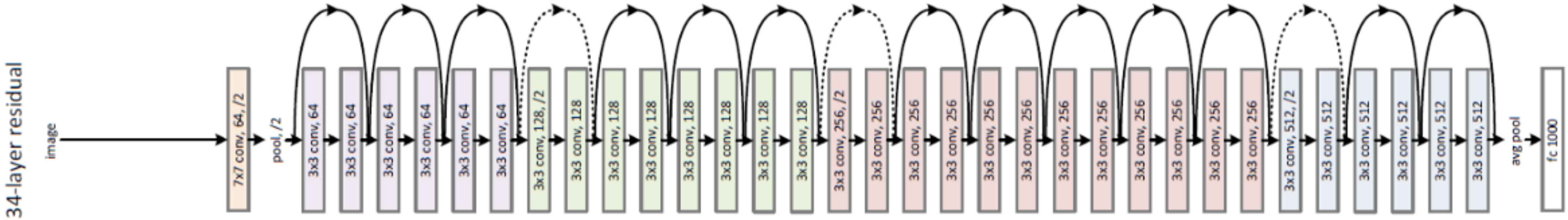
PyTorch Implementation:
https://paperswithcode.com/method/googlenet#

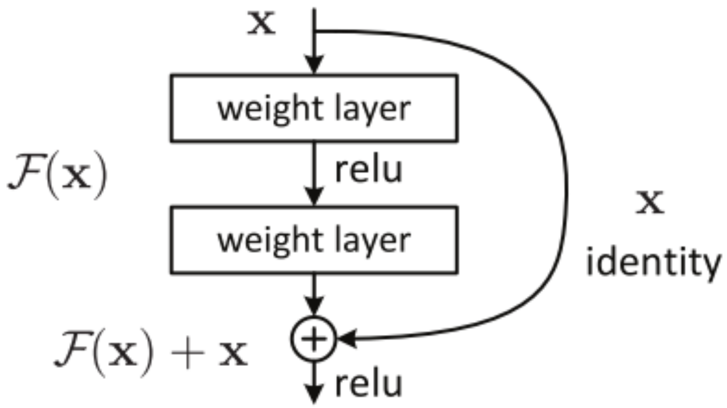
Residual-Net (2015)

- Designed by Kaiming He at Facebook
- Also known as ResNet
- Winner of ILSVRC 2015 (Top-5 error of 3.6%, overcoming Human 5.0%)
- Introduces skip connection concept
- Utilizes heavy batch normalization
- Allows much deeper network architecture without vanishing/exploding gradients
- Currently a start-of-the-art CNN architecture

- Link to the original paper - https://arxiv.org/abs/1512.03385

ResNet-34 Architecture





Input layer: 224 * 224 * 3
Output layer: 1000 categories

Total 6 layers:
- 1 CV layer
- 4 ResNet layers
- 1 FC layer (includes dropout)

Each ResNet layer performs **3 x 3 convolution** with fixed feature dimension (64, 128, 256, 512)

Bypass connection every 2 convolutions.

Weight layers learn **residual function f(x)** to be added to **x-identity**

Total # of parameters: >21M

PyTorch Implementation:
https://pytorch.org/hub/pytorch_vision_resnet/

Part 4: Example: Image Segmentation with Fully Convolutional Network (FCN)

Fully Convolutional Neural Network (FCN) with ResNet Backbone

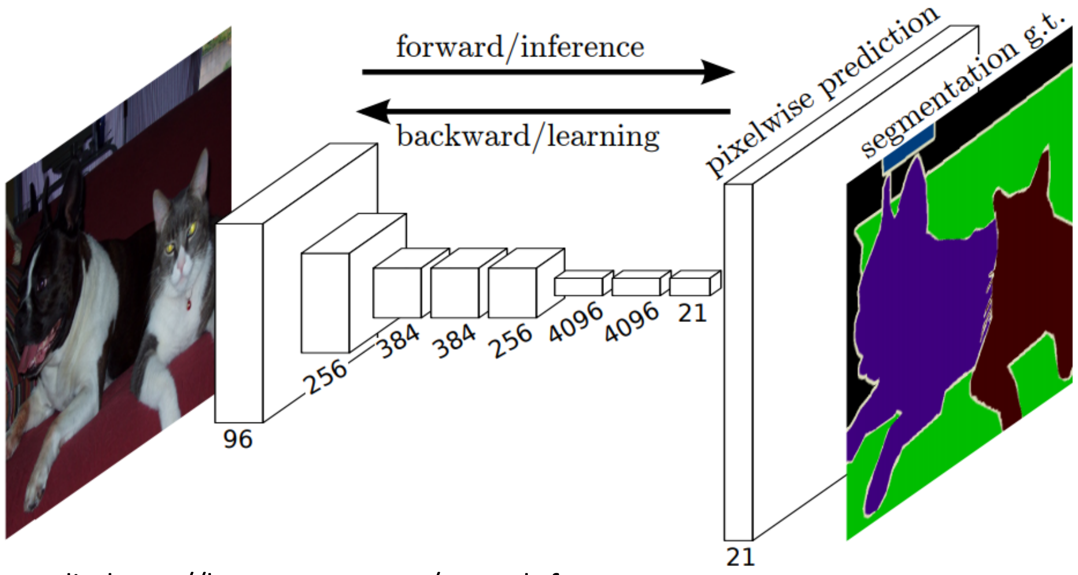
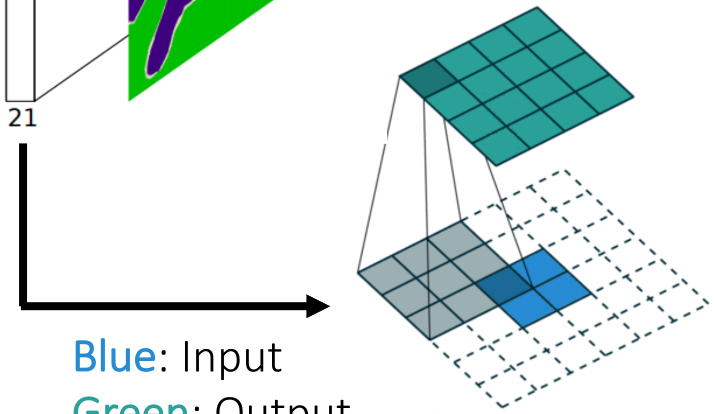


Image credit: https://learnopencv.com/pytorch-for-beginners-semantic-segmentation-using-torchvision/



Blue: Input
Green: Output

Input layer: 224 * 224 * 3
Output layer: 21 * 224 * 224

Replaces **FC** into **convolutional layers**

FCN uses convolutional layers to **classify each pixel** in the image

FCN outputs **# of classes * H * W**

Uses **De-convolution** to **up-sample** the max-pooled convolutional layers

Implementation of FCN with TorchVision

```
from torchvision import models
fcn = models.segmentation.fcn_resnet101(pretrained=True).eval()
```

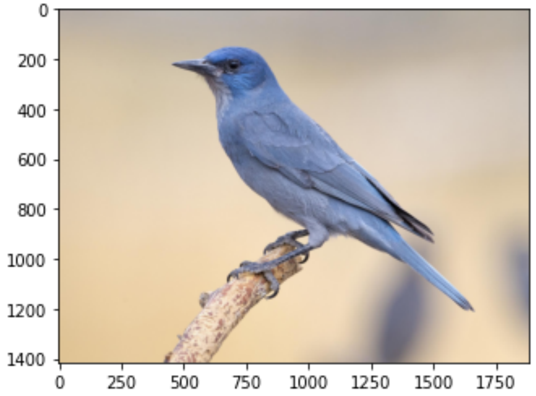
Load the model

```
Downloading: "https://download.pytorch.org/models/fcn_resnet101_coco-7ecb50ca.pth" to /root/.cache/torch/hub/checkpoints/fcn_resnet101_coco-7ecb50ca.pth
100% 208M/208M [00:01<00:00, 153MB/s]
```

```
from PIL import Image
import matplotlib.pyplot as plt
import torch

!wget -nv https://static.independent.co.uk/s3fs-public/thumbnails/image/2018/04/10/19/pinyon-jay-bird.jpg -O bird.png
img = Image.open('./bird.png')
plt.imshow(img); plt.show()
```

Download and Load the image to be segmented

```
2021-03-22 06:37:41 URL:https://static.independent.co.uk/s3fs-public/thumbnails/image/2018/04/10/19/pinyon-jay-bird.jpg [182965/182965] -> "bird.png" [1]
```



More info: https://learnopencv.com/pytorch-for-beginners-semantic-segmentation-using-torchvision/

Implementation of FCN with TorchVision

```python
# Apply the transformations needed
import torchvision.transforms as T
trf = T.Compose([T.Resize(256),
                 T.CenterCrop(224),
                 T.ToTensor(),
                 T.Normalize(mean = [0.485, 0.456, 0.406],
                             std = [0.229, 0.224, 0.225])])
inp = trf(img).unsqueeze(0)
```

Transform image

```python
# Pass the input through the net
out = fcn(inp)['out']
print (out.shape)
om = torch.argmax(out.squeeze(), dim=0).detach().cpu().numpy()
print (om.shape)
```

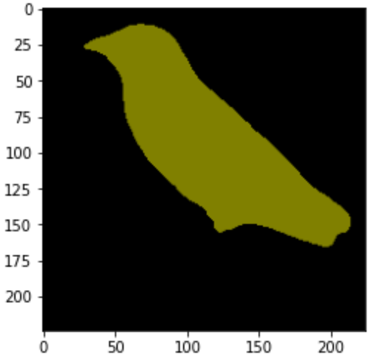
Forward pass

Extract the best class

```
torch.Size([1, 21, 224, 224])
(224, 224)
```

```python
rgb = decode_segmap(om)
plt.imshow(rgb); plt.show()
```

Decode the output & Display the Result




```python
# Define the helper function
def decode_segmap(image, nc=21):

  label_colors = np.array([(0, 0, 0),  # 0=background
               # 1=aeroplane, 2=bicycle, 3=bird, 4=boat, 5=bottle
               (128, 0, 0), (0, 128, 0), (128, 128, 0), (0, 0, 128), (128, 0, 128),
               # 6=bus, 7=car, 8=cat, 9=chair, 10=cow
               (0, 128, 128), (128, 128, 128), (64, 0, 0), (192, 0, 0), (64, 128, 0),
               # 11=dining table, 12=dog, 13=horse, 14=motorbike, 15=person
               (192, 128, 0), (64, 0, 128), (192, 0, 128), (64, 128, 128), (192, 128, 128),
               # 16=potted plant, 17=sheep, 18=sofa, 19=train, 20=tv/monitor
               (0, 64, 0), (128, 64, 0), (0, 192, 0), (128, 192, 0), (0, 64, 128)])

  r = np.zeros_like(image).astype(np.uint8)
  g = np.zeros_like(image).astype(np.uint8)
  b = np.zeros_like(image).astype(np.uint8)

  for l in range(0, nc):
    idx = image == l
    r[idx] = label_colors[l, 0]
    g[idx] = label_colors[l, 1]
    b[idx] = label_colors[l, 2]

  rgb = np.stack([r, g, b], axis=2)
  return rgb
```

Function for reconstructing RGB image

Implementation of FCN with TorchVision

```python
# Apply the transformations needed
import torchvision.transforms as T
trf = T.Compose([T.Resize(256),
                 T.CenterCrop(224),
                 T.ToTensor(),
                 T.Normalize(mean = [0.485, 0.456, 0.406],
                             std = [0.229, 0.224, 0.225])])
inp = trf(img).unsqueeze(0)
```

Transform image

```python
# Pass the input through the net
out = fcn(inp)['out']
print (out.shape)
om = torch.argmax(out.squeeze(), dim=0).detach().cpu().numpy()
print (om.shape)
```

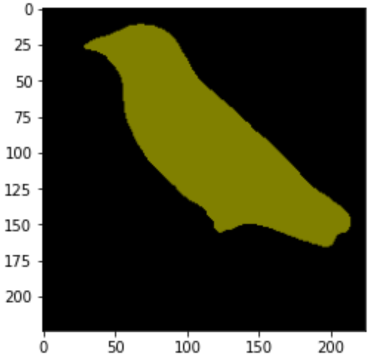
Forward pass

Extract the best class

```
torch.Size([1, 21, 224, 224])
(224, 224)
```

```python
rgb = decode_segmap(om)
plt.imshow(rgb); plt.show()
```

Decode the output & Display the Result



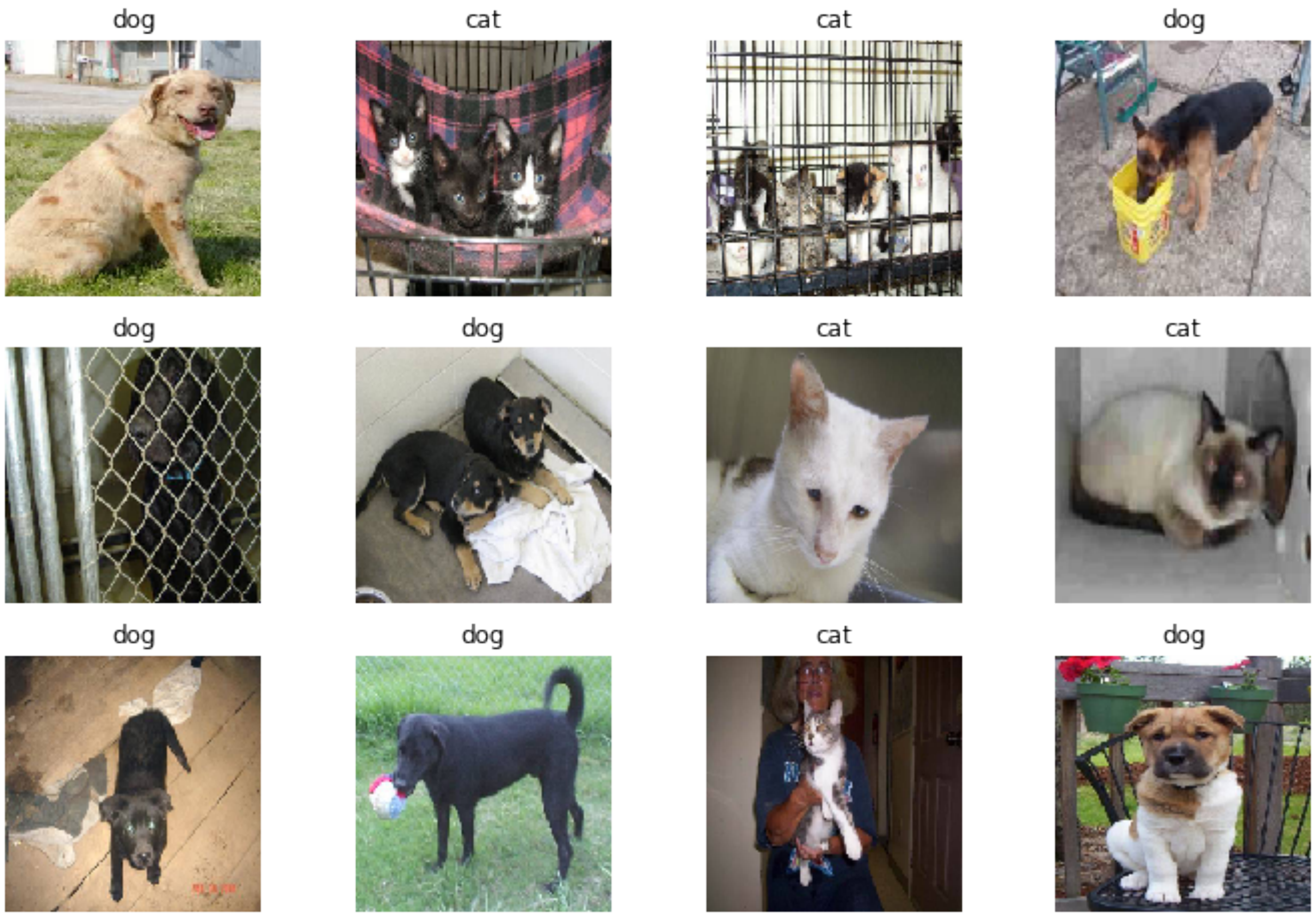
You can modify the pre-trained network to perform other tasks

```python
# Define the helper function
def decode_segmap(image, nc=21):

  label_colors = np.array([(0, 0, 0),  # 0=background
               # 1=aeroplane, 2=bicycle, 3=bird, 4=boat, 5=bottle
               (128, 0, 0), (0, 128, 0), (128, 128, 0), (0, 0, 128), (128, 0, 128),
               # 6=bus, 7=car, 8=cat, 9=chair, 10=cow
               (0, 128, 128), (128, 128, 128), (64, 0, 0), (192, 0, 0), (64, 128, 0),
               # 11=dining table, 12=dog, 13=horse, 14=motorbike, 15=person
               (192, 128, 0), (64, 0, 128), (192, 0, 128), (64, 128, 128), (192, 128, 128),
               # 16=potted plant, 17=sheep, 18=sofa, 19=train, 20=tv/monitor
               (0, 64, 0), (128, 64, 0), (0, 192, 0), (128, 192, 0), (0, 64, 128)])

  r = np.zeros_like(image).astype(np.uint8)
  g = np.zeros_like(image).astype(np.uint8)
  b = np.zeros_like(image).astype(np.uint8)

  for l in range(0, nc):
    idx = image == l
    r[idx] = label_colors[l, 0]
    g[idx] = label_colors[l, 1]
    b[idx] = label_colors[l, 2]

  rgb = np.stack([r, g, b], axis=2)
  return rgb
```

Function for reconstructing RGB image

Lab Assignment:
Dogs and Cats Classification using AlexNet

Dogs vs Cats Dataset



Train data – 25k images of dogs and cats
Test data – 12.5k images

Download Link

https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/overview

Performance goal:

>85% validation accuracy within 10 epochs

Test the model on 10 test images

AlexNet Implementation Details

Define AlexNet

```
class AlexNet(nn.Module):
    """
    Neural network model consisting of layers propsed by AlexNet paper.
    """
    def __init__(self, num_classes=):
        """
        Define and allocate layers for this neural net.
        Args:
            num_classes (int): number of classes to predict with this model
        """
        super().__init__()

        # Define the layers

        self.net =

        self.classifier =

        # initialize bias

        self.init_bias()

    def init_bias(self):
        # Initialize weights according to original paper
        for layer in self.net:
            if isinstance(layer, nn.Conv2d):
                nn.init.normal_(layer.weight, mean=0, std=0.01)
                nn.init.constant_(layer.bias, 0)
        nn.init.constant_(self.net[4].bias, 1)
        nn.init.constant_(self.net[10].bias, 1)
        nn.init.constant_(self.net[12].bias, 1)

    def forward(self, x):

        x = self.net(x)

        # reduce the dimensions for linear layer input

        return self.classifier(x)
```

Training Code

```
# Set the seed value
seed = torch.initial_seed()

# create model

# create dataset and data loader

# Define loss function

# create optimizer

# start training
```

Details of each layer

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax