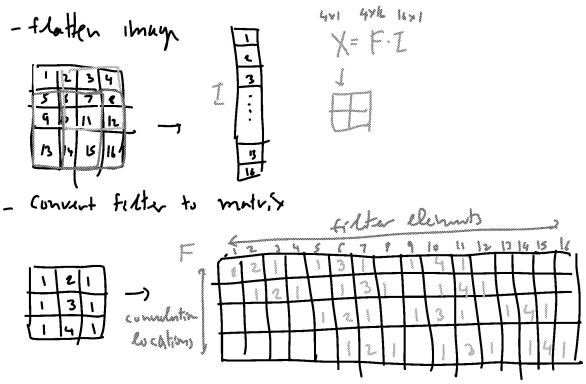
Transpose convolution

- Expressing 2D convolution as matrix multiplication

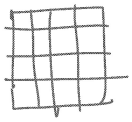


Transpose convolution

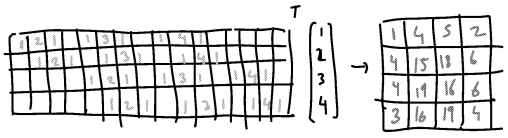
Transpose convolution:

$$I' = F^T X$$

16×1 6×4 4×1

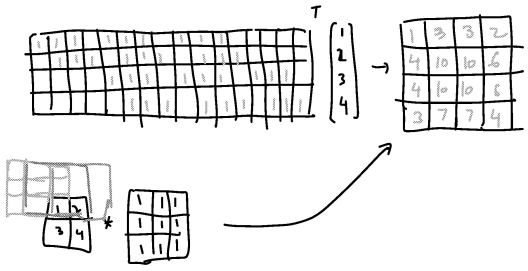


Example:



Transpose convolution

Example:



Transpose convolution example

```
Original Matrix (8x8):
[[ 1.  2.  3.  4.  5.  6.  7.  8.]
 [ 9. 10. 11. 12. 13. 14. 15. 16.]
 [17. 18. 19. 20. 21. 22. 23. 24.]
 [25. 26. 27. 28. 29. 30. 31. 32.]
 [33. 34. 35. 36. 37. 38. 39. 40.]
 [41. 42. 43. 44. 45. 46. 47. 48.]
 [49. 50. 51. 52. 53. 54. 55. 56.]
 [57. 58. 59. 60. 61. 62. 63. 64.]]

Convolution Kernel Used (2x2):
[[0.25 0.25]
 [0.25 0.25]]

Matrix-based Convolution Matrix:
[[0.25 0.25 0.   ... 0.   0.   0.  ]
 [0.   0.   0.25 ... 0.   0.   0.  ]
 [0.   0.   0.   ... 0.   0.   0.  ]
 ...
 [0.   0.   0.   ... 0.   0.   0.  ]
 [0.   0.   0.   ... 0.25 0.   0.  ]
 [0.   0.   0.   ... 0.   0.25 0.25]]

Matrix-based Convolution Output (5x5):
[[ 5.5  7.5  9.5 11.5]
 [21.5 23.5 25.5 27.5]
 [37.5 39.5 41.5 43.5]
 [53.5 55.5 57.5 59.5]]
```

Transpose convolution example

```
Matrix-based Transpose Convolution Matrix:
[[1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]]

Matrix-based Restored (Deconvolved) Output ([redacted]):
[[ 5.5  5.5  7.5  7.5  9.5  9.5 11.5 11.5]
 [ 5.5  5.5  7.5  7.5  9.5  9.5 11.5 11.5]
 [21.5 21.5 23.5 23.5 25.5 25.5 27.5 27.5]
 [21.5 21.5 23.5 23.5 25.5 25.5 27.5 27.5]
 [37.5 37.5 39.5 39.5 41.5 41.5 43.5 43.5]
 [37.5 37.5 39.5 39.5 41.5 41.5 43.5 43.5]
 [53.5 53.5 55.5 55.5 57.5 57.5 59.5 59.5]
 [53.5 53.5 55.5 55.5 57.5 57.5 59.5 59.5]]
```

8x8

Dataset

- Oxford-IIIT Pets dataset (http://www.robots.ox.ac.uk/~vgg/data/pets/)
- Contains 7,390 pictures of various breeds of cats and dogs, together with foreground-background segmentation masks for each picture.
- There are overall 37 categories with roughly 200 images for each class.
- Segmentation mask: an image the same size as the input image, with a single color channel where each integer value corresponds to the class of the corresponding pixel in the input image.
- Segmentation mask values:
  - 1 (foreground)
  - 2 (background)
  - 3 (contour)
- Data folders:
  - images: The input pictures are stored as JPG files in the images/ folder (such as images/Abys- sinian_1.jpg),
  - labels: The corresponding segmentation mask is stored as a PNG file with the same name in the annotations/trimaps/ folder (such as annotations/trimaps/ Abyssinian_1.png).

Dataset

Download dataset:

```
!wget http://www.robots.ox.ac.uk/~vgg/data/pets/data/images.tar.gz
!wget http://www.robots.ox.ac.uk/~vgg/data/pets/data/annotations.tar.gz
!tar -xf images.tar.gz
!tar -xf annotations.tar.gz
```

Fixes to the dataset: The original dataset contains a few damaged images. We fixed them and uploaded into this repository. These images are mentioned below, as well as the steps of fixing them: (https://github.com/ml4py/dataset-iiit-pet). Problem images:

```
Abyssinian_34.jpg
Egyptian_Mau_139.jpg
Egyptian_Mau_145.jpg
Egyptian_Mau_167.jpg
Egyptian_Mau_177.jpg
beagle_116.jpg
chihuahua_121.jpg
```

Prepare the data

Prepare list of input files with corresponding path mask files:

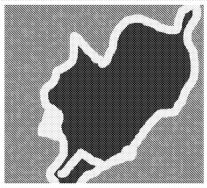
```
import os input_dir = "images/"
target_dir = "annotations/trimaps/"

# get image paths and sort by name (breed)
input_img_paths = sorted(
    [os.path.join(input_dir, fname)
     for fname in os.listdir(input_dir)
     if fname.endswith(".jpg")])
target_paths = sorted(
    [os.path.join(target_dir, fname)
     for fname in os.listdir(target_dir)
     if fname.endswith(".png") and not fname.startswith(".")])
```

Display data:

The target image (segmentation map) is grayscale. The original labels are 1, 2, and 3. Subtract 1 so that the labels range from 0 to 2,then multiply by 127 so that the labels become 0 (black), 127 (gray), 254 (near-white).

Prepare the data

```
import matplotlib.pyplot as plt
from tensorflow.keras.utils import load_img, img_to_array
plt.axis("off")
plt.imshow(load_img(input_img_paths[9])) # plot image number 9

def display_target(target_array):
    normalized_array = (target_array.astype("uint8") - 1) * 127
    plt.axis("off")
    plt.imshow(normalized_array[:, :, 0])

img = img_to_array(load_img(target_paths[9], color_mode="grayscale"))
display_target(img)
```



Prepare the data

Split data into training and validation sets:

```
import numpy as np
import random
img_size = (200, 200)           # resize images to 200x200
num_imgs = len(input_img_paths) # total number of images

# use the same random seed to make sure images and segmentation maps match
random.Random(1337).shuffle(input_img_paths)
random.Random(1337).shuffle(target_paths)

# load image function
def path_to_input_image(path):
    return img_to_array(load_img(path, target_size=img_size))

# load segmentation map function
def path_to_target(path):
    img = img_to_array(load_img(path,
                        target_size=img_size,
                        color_mode="grayscale"))
    img = img.astype("uint8") - 1 # subtract 1 so for 0,1,2 image labels
```

Prepare the data

```
# load all images
input_imgs = np.zeros((num_imgs,) + img_size + (3,), dtype="float32") # RGB
targets = np.zeros((num_imgs,) + img_size + (1,), dtype="uint8")      # gray
for i in range(num_imgs):
    input_imgs[i] = path_to_input_image(input_img_paths[i])
    targets[i] = path_to_target(target_paths[i])

# split training and validation sets
num_val_samples = 1000
train_input_imgs = input_imgs[:-num_val_samples]
train_targets = targets[:-num_val_samples]
val_input_imgs = input_imgs[-num_val_samples:]
val_targets = targets[-num_val_samples:]
```

Define the network

Encoder-decoder architecture:

- Learn to encode the image and then decode the segmentation map.
- In the encoder a sequence of convolutions with down-sampling up to size $(25, 25, 256)$. Each encoded spatial location (pixel) contains information about a spatial region of the original image. Down-sampling is done with strides (instead of max-pooling) to preserve spatial location so that we can produce per-pixel segmentation labels.
- In the decoder a sequence of transpose convolutions (`Conv2DTranspose` layers) restore the original resolution by up-sampling. They convert the $(25, 25, 256)$ encoding back to $(200, 200, 3)$ thus producing a probability map for each of the 3 classes. Up-sampling is done with strides.

Transpose convolution:

- The purpose to deconvolution is to learn to up-sample a down-sampled image.
- Passing an input of $(100, 100, 64)$ through a `Conv2D(128, 3, strides=2, padding="same")` layer gives a $(50, 50, 128)$ down-sampled output.
- Passing this output through a `Conv2DTranspose(64, 3, strides=2, padding="same")`, gives back an up-sampled output of shape $(100, 100, 64)$. Unlike the down-sampling, The up-sampling (transpose convolution) is learned.

Define the network

Define model:

- In the encoder alternate between convolutions with and without strides. Only the convolutions with strides cause dimensionality reduction (down-sampling).
- In the decoder alternate between transposed convolutions without and with strides. Only the transposed convolutions with strides cause dimensionality increase (up-sampling).

```
from tensorflow import keras
from tensorflow.keras import layers

def get_model(img_size, num_classes):
    inputs = keras.Input(shape=img_size + (3,))
    x = layers.Rescaling(1./255)(inputs)

    # encoder
    x = layers.Conv2D(64, 3, strides=2, activation="relu", padding="same")(x)
    x = layers.Conv2D(64, 3, activation="relu", padding="same")(x)
    x = layers.Conv2D(128, 3, strides=2, activation="relu", padding="same")(x)
    x = layers.Conv2D(128, 3, activation="relu", padding="same")(x)
    x = layers.Conv2D(256, 3, strides=2, padding="same", activation="relu")(x)
    x = layers.Conv2D(256, 3, activation="relu", padding="same")(x)
```

Define the network

```
    # decoder
    x = layers.Conv2DTranspose(256, 3, activation="relu", padding="same")(x)
    x = layers.Conv2DTranspose(
        256, 3, activation="relu", padding="same", strides=2)(x)
    x = layers.Conv2DTranspose(128, 3, activation="relu", padding="same")(x)
    x = layers.Conv2DTranspose(
        128, 3, activation="relu", padding="same", strides=2)(x)
    x = layers.Conv2DTranspose(64, 3, activation="relu", padding="same")(x)
    x = layers.Conv2DTranspose(
        64, 3, activation="relu", padding="same", strides=2)(x)

    # use softmax to produce output probability for each of the 3 classes
    outputs = layers.Conv2D(num_classes, 3, activation="softmax",
        padding="same")(x)

    model = keras.Model(inputs, outputs)
    return model

model = get_model(img_size=img_size, num_classes=3)
model.summary()
```

Define the network

Model summary:

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 200, 200, 3)]     0
_________________________________________________________________
rescaling (Rescaling)        (None, 200, 200, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 100, 100, 64)      1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 100, 100, 64)      36928
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 50, 50, 128)       73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 50, 50, 128)       147584
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 25, 25, 256)       295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 25, 25, 256)       590080
```

Define the network

```
conv2d_transpose (Conv2DTran (None, 25, 25, 256)       590080
conv2d_transpose_1 (Conv2DTr (None, 50, 50, 256)       590080
conv2d_transpose_2 (Conv2DTr (None, 50, 50, 128)       295040
conv2d_transpose_3 (Conv2DTr (None, 100, 100, 128)     147584
conv2d_transpose_4 (Conv2DTr (None, 100, 100, 64)      73792
conv2d_transpose_5 (Conv2DTr (None, 200, 200, 64)      36928
conv2d_6 (Conv2D)            (None, 200, 200, 3)       1731
=================================================================
Total params: 2,880,643
Trainable params: 2,880,643
Non-trainable params: 0
```

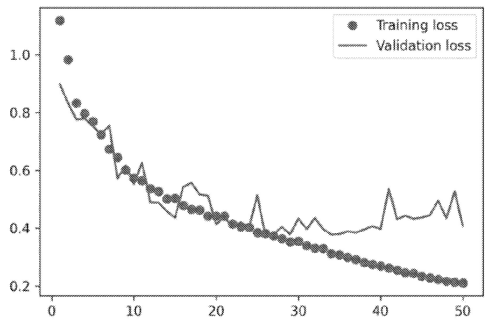
why do we do conv at the end for softmax instead of FC?

1. The i/p to that layer need not be fixed, it can be of any size and FC needs to have a fixed i/p
2. Less number of parameters needed for conv than FC

Compile and train

```
model.compile(optimizer="rmsprop", loss="sparse_categorical_crossentropy")

callbacks = [
    keras.callbacks.ModelCheckpoint("oxford_segmentation.keras",
                                    save_best_only=True)
]

history = model.fit(train_input_imgs, train_targets,
                    epochs=50,
                    callbacks=callbacks,
                    batch_size=64,
                    validation_data=(val_input_imgs, val_targets))

# plot
epochs = range(1, len(history.history["loss"]) + 1)
loss = history.history["loss"]
val_loss = history.history["val_loss"]
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
```

Compile and train



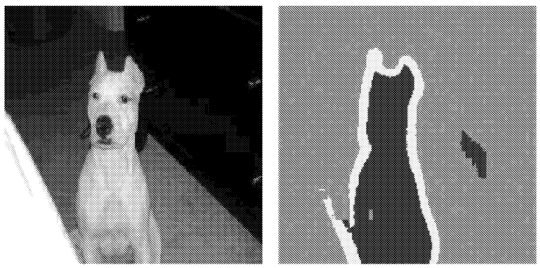
Overfitting starts around epoch 25.

Inference

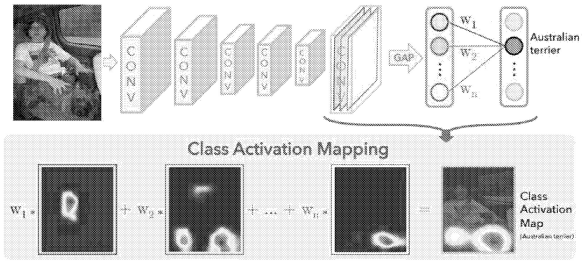
Load the best performing model (according to validation loss), and use it to predict a segmentation mask.

```
from tensorflow.keras.utils import array_to_img

model = keras.models.load_model("oxford_segmentation.keras")
i = 4
test_image = val_input_imgs[i]
plt.axis("off")
plt.imshow(array_to_img(test_image))

# example of how to get the mask for class 0
mask = model.predict(np.expand_dims(test_image, 0))[0]

# Utility to display a model's prediction
def display_mask(pred):
    mask = np.argmax(pred, axis=-1)
    mask *= 127
    plt.axis("off")
    plt.imshow(mask)

# display the model's prediction
display_mask(mask)
```

Inference

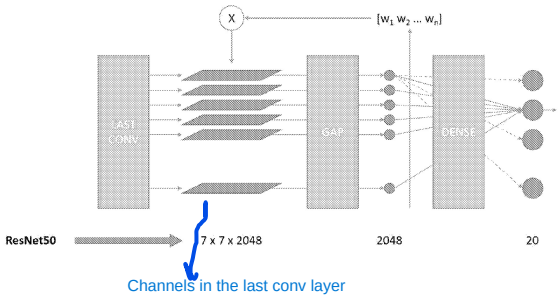


Class activation maps



On what basis, did the model classify this as that exactly??we wanted to know what the model is looking at and if it look at the right thing to classify. Because a model may have wrongly learnt something that accidentally happened to be the same in all the training images of the dataset, like the background was always for people may be. We dont want the model to look at the background and say it is a person, it should look at the relevant features of th eperson

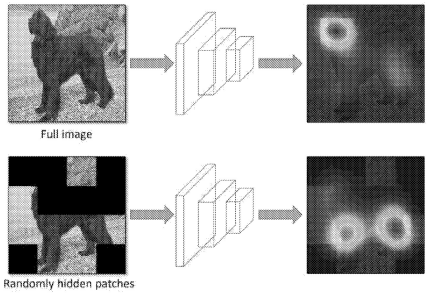
Class activation maps



weightsxchannels-->output

Class activation maps

- Hide and seek (HaS)

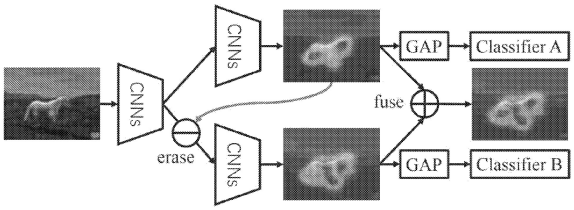


This helps to check if the classifr really looked at all the relevant parts of the object or only a portion like head/tail. So, we hide some patches and classify and then check based on what the classification was made

Here, the patch is hidden randomly

Class activation maps

- Adversarial Complementary Learning (ACoL)



Here, the relevant observed patch is hidden or deleted to see what it does to calssify f

Class activation maps

- Excitation Backprop

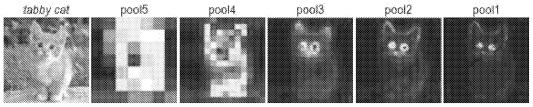


Fig. 3. Example Marginal Winning Probability (MWP) maps computed via Excitation Backprop from different layers of the public `VGG16` model [24] trained on ImageNet. The input image is shown on the right. The MWP maps are generated for the category `tabby cat`. Neurons at higher-level layers have larger receptive fields and strides. Thus, they can capture larger areas but with lower spatial accuracy. Neurons at lower layers tend to more precisely localize features at smaller scale.

Class activation maps

- Excitation Backprop

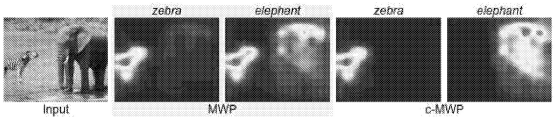


Fig. 4. Marginal Winning Probability (MWP) *vs.* contrastive MWP (c-MWP). The input image is resized to 224×224, and we use `GoogleNet` pretrained on ImageNet to generate the MWP maps and c-MWP maps for `zebra` and `elephant`. The MWP map for `elephant` does not successfully suppress the zebra. In contrast, by cancelling out common winner neurons for `elephant` and `non-elephant`, the c-MWP map more effectively highlights the elephant.

Image captioning with attention

- Not all pixels are equally important

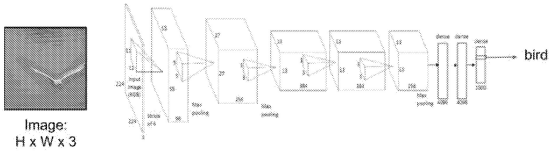


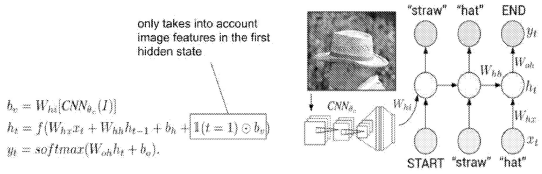
Image captioning with attention

- Attend to different parts of the image. E.g., image captioning

A bird flying over a body of water

Image captioning with attention

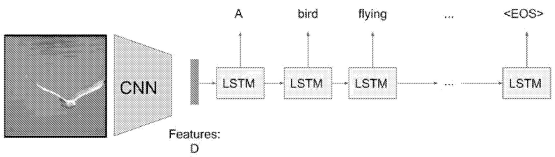
- Image captioning using RNN without attention:



Karpathy and Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." CVPR 2015

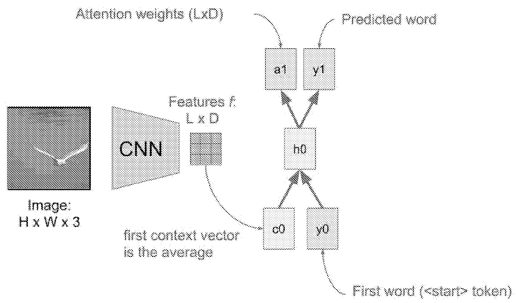
Image captioning with attention

- Decoding is based on a final summarized state



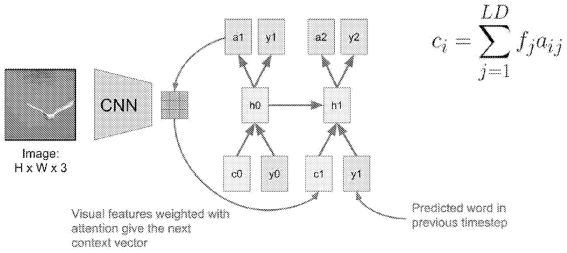
Vinyals et al. Show and tell: A neural image caption generator. CVPR 2015

Image captioning with attention



Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

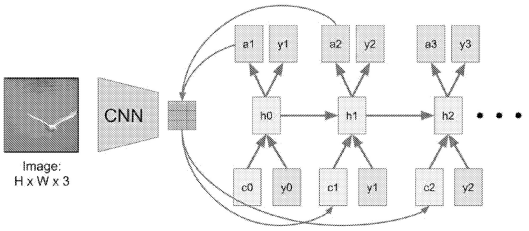
Image captioning with attention



$$c_i = \sum_{j=1}^{LD} f_j a_{ij}$$

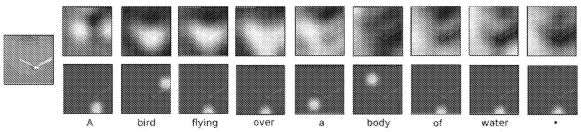
Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

Image captioning with attention



Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

Image captioning with attention



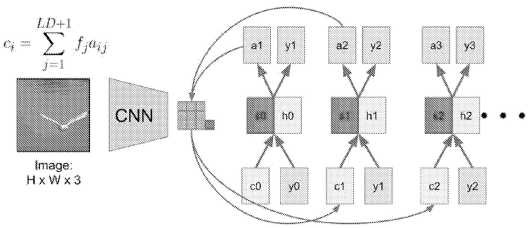
Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

Image captioning with visual sentinel

- Produce an additional "sentinel" feature to decide when to look at the image

$$f = [f; s_i]$$

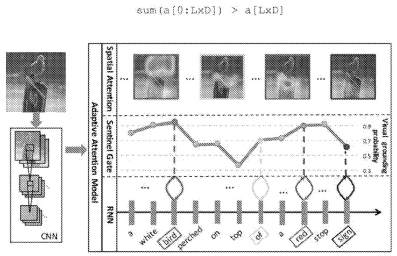
$$c_i = \sum_{j=1}^{LD+1} f_j a_{ij}$$



Lu et al. Knowing When to Look: Adaptive Attention via a Visual Sentinel for Image Captioning, CVPR 2017

Image captioning with visual sentinel

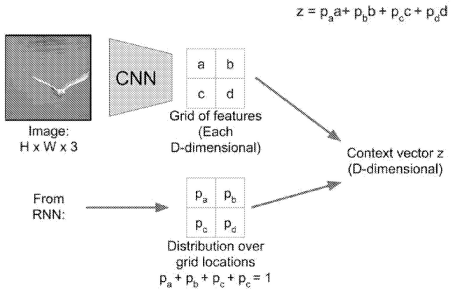
- The sentinel feature decides when to rely on RNN *state* instead of the attentive image



Lu et al. Knowing When to Look: Adaptive Attention via a Visual Sentinel for Image Captioning, CVPR 2017

Soft spatial attention

- Summarize all locations



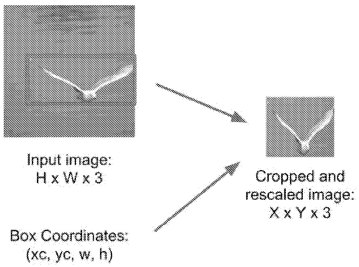
weights

weigjts given for alll anf multiplied to highligt the relvant parts

Xu et al. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

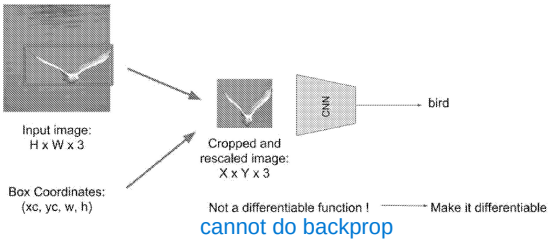
Hard spatial attention

- Crop part of the image (not differentiable and cannot be trained)



Spatial transformer networks

- Want:

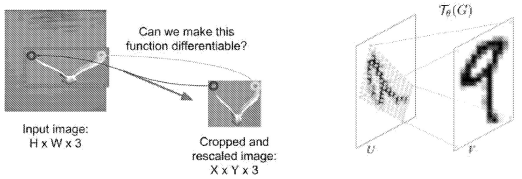


cannot do backprop

Jaderberg et al. Spatial Transformer Networks, NIPS 2015

Spatial transformer networks

- Use affine mapping instead of cropping (learn affine parameters)



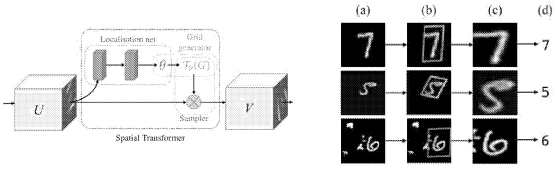
$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

Jaderberg et al. Spatial Transformer Networks, NIPS 2015

learn the 6 parameters such the instad of cropping the object expands and fills the ima

Spatial transformer networks

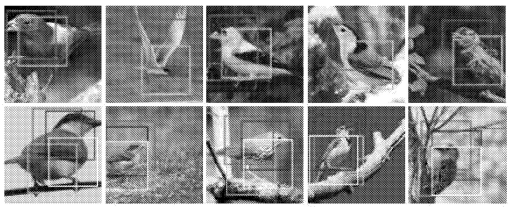
- Use a spatial transformer block to learn to attend to and transform the input



Jaderberg et al. Spatial Transformer Networks, NIPS 2015

Spatial transformer networks

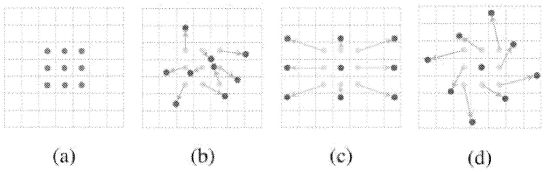
- Can be used for proposals in detection and segmentation networks. Results of 2 and 4 spatial transformers in parallel



Jaderberg et al. Spatial Transformer Networks, NIPS 2015

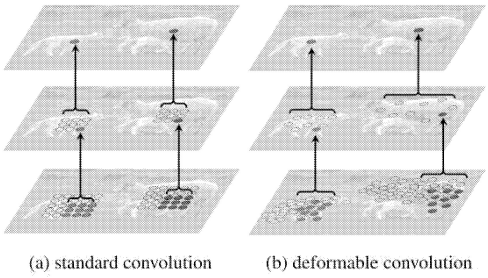
Deformable convolution

- Dynamic and learnable receptive field



Dai, Qi, Xiong, Li, Zhang et al. Deformable Convolutional Networks. arXiv Mar 2017

Deformable convolution



Dai, Qi, Xiong, Li, Zhang et al. Deformable Convolutional Networks. arXiv Mar 2017

Deformable convolution

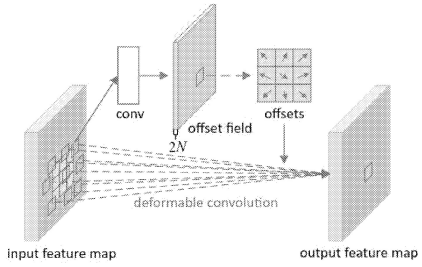


Figure 2: Illustration of 3×3 deformable convolution.

Dai, Qi, Xiong, Li, Zhang et al. Deformable Convolutional Networks. arXiv Mar 2017

learn to get the offset by which the filter should be pushed so as to maximize the output

Deformable convolution

Figure 6: Each image triplet shows the sampling locations ($9^3 = 729$ red points in each image) in three levels of 3×3 deformable filters (see Figure 5 as a reference) for three activation units (green points) on the background (left), a small object (middle), and a large object (right), respectively.

Dai, Qi, Xiong, Li, Zhang et al. Deformable Convolutional Networks, arXiv Mar 2017

Channel attention

- Model the importance of each feature channel

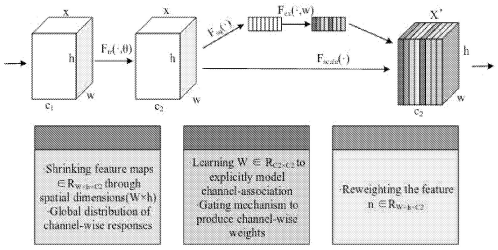


Fig. 3 Principle of SENet

Mixed attention

- Combine multiple attention mechanisms (e.g., CBAM – spatial + channel attention)

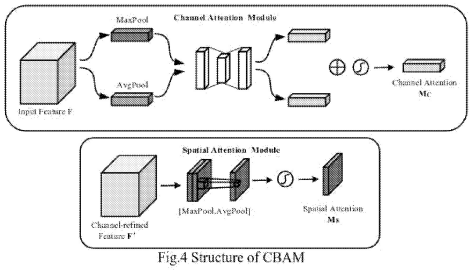


Fig.4 Structure of CBAM