CIS680 Final Project - Fall 20

November 1, 2020

1 Overview

We have five options in this final project as:

1. Faster RCNN to Mask RCNN extension
2. GAN and VAE to modelMultimodal Distribution extension
3. Real-word Industry Challenge
4. Activity Recognition Competition
5. Open-ended Project

The final project could be an individual or group project. The maximum group size is two for options 1 and 2, three for options 3 to 5. If you wish to do an individual project, we recommend you pick option 5.

2 Submission Detail

1. **11/09** Final Project Proposal Due (for open-ended question)
2. **12/02** Milestone 1 (for open-ended question)
3. **12/10** Final Project Code Due
4. **12/17** Final Project Report Due

3 Code template availability

Notice that, for options 1 & 2, There will be available code templates. We will release those code template very soon.

Final Project Option-1: FasterRCNN to MaskRCNN extension

1 Overview

In this project you will implement MaskRCNN [1], an algorithm that addresses the task of instance segmentation, which combines object detection and semantic segmentation into a per-pixel object detection framework. Parts of this model were also implemented in HW4. The difference is that the goal of this project is to implement the full architecture without the simplifications that were added in HW4.

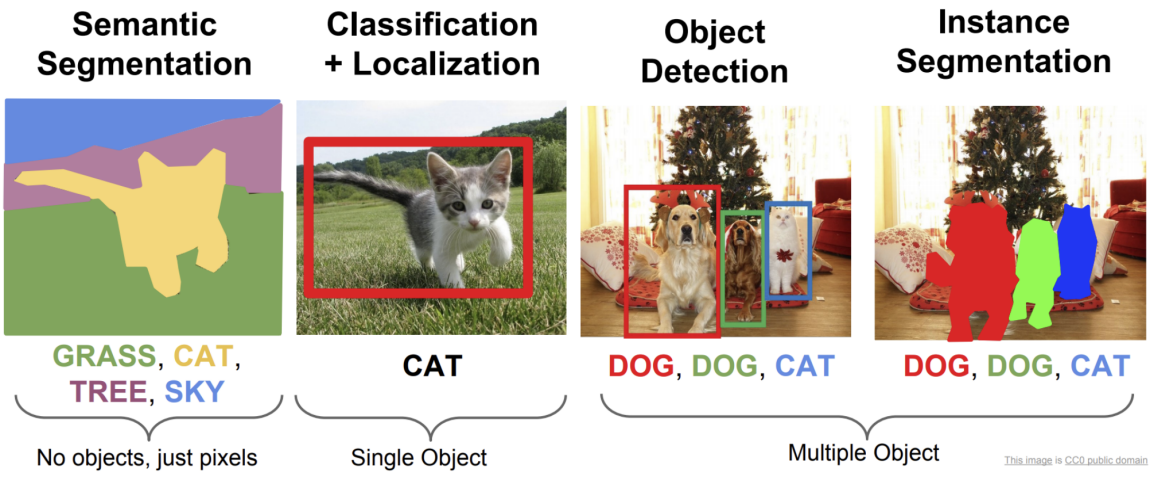


Figure 1: Instance Segmentation

In figure (2) we can see a diagram of the full system. It contains:

- A FPN [4] that plays the role of a shared backbone that will be used by the other parts of the system as a share feature extractor.
- A Regional Proposal Network [2] that produces Region of Interest Proposals that are used as a fast estimation of the regions in the image that contain objects.
- A Box Head [3] that refines and classifies the boxes produced by the RPN network
- A Mask Head [1] that predicts a mask of the object that is contained in the refined boxes of the Box Head

You are strongly encouraged to read the papers. This way you will get a better understanding of the various choices that result in the Mask RCNN architecture.

Although we will provide a detailed description of the system, feel free to make adjustments and change any part that you feel that can help your performance. For example the losses that we present are the ones proposed in the papers, but you may want to experiment with newer ideas such as the Focal Loss that were introduced after these papers were published.

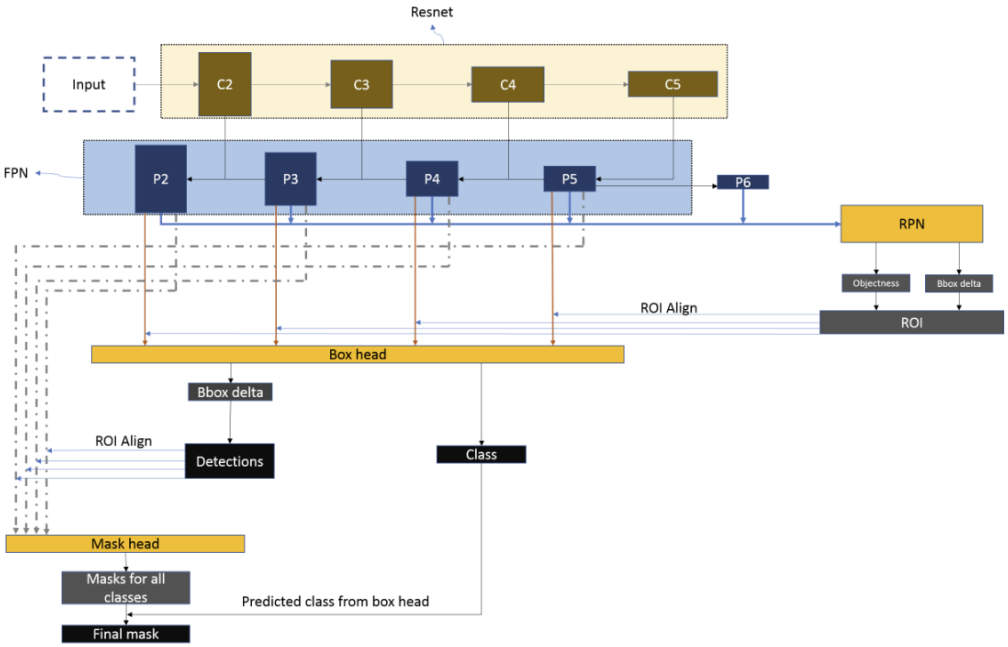


Figure 2: MaskRCNN

2 Dataset Description and Preprocessing (implemented in HW4)

Your task is to detect 3 types of objects: Vehicles, People and Animals. The dataset that will be used can be found here. It consists of:

1. a numpy array of all the RGB Images ($3 \times 300 \times 400$)
2. a numpy array of all the masks (300×400)
3. list of ground truth labels per image.
4. list of ground truth bounding boxes per image. The four numbers are the upper left and lower right coordinates.

You should use the label list to figure out which mask belongs to which image.

After the grouping of the masks into each image, we also want to preprocess our data as follows

- normalize image pixel value to [0,1]
- rescale the image to 800x1066 (we want to keep the aspect ratio the same)
- normalize each channel with mean: [0.485,0.456,0.406], std:[0.229,0.224,0.225]
- add zero padding to get an image of size 800x1088

Do not forget to also apply the rescaling to the bounding boxes and the masks. Check the pre processing by visualizing the images of the dataset and the corresponding bounding boxes, masks. (similar to figure (3))

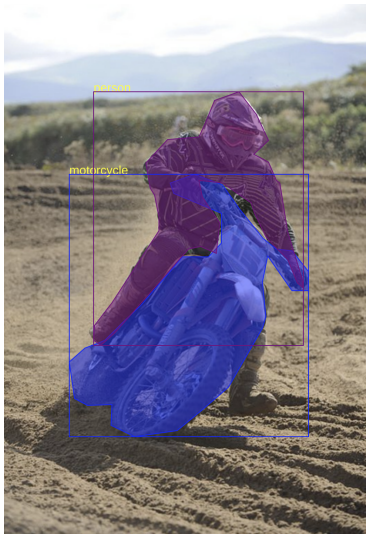


Figure 3: Datapoint

3 Backbone

For the backnone we will use a pretrained resnet50 FPN backbone from **torchvision.models.detection.maskrcnn_resnet50_fpn**.

4 Region Proposal Network

Region Proposal Networks (RPNs) are "attention mechanisms" for the object detection task, performing a crude but inexpensive first estimation of where the bounding boxes of the objects should be. They were first proposed in [2] as a way to address the issue of expensive greedy algorithms like Selective Search [3], opening new avenues to end-to-end object detection tasks. They work through classifying the initial anchor boxes into object/background and refine the coordinates for the boxes with objects. Later, these boxes will be further refined and tightened by the instance segmentation heads as well as classified in their corresponding classes. The architecture for the RPN and the later refinement of the proposals is shown below:

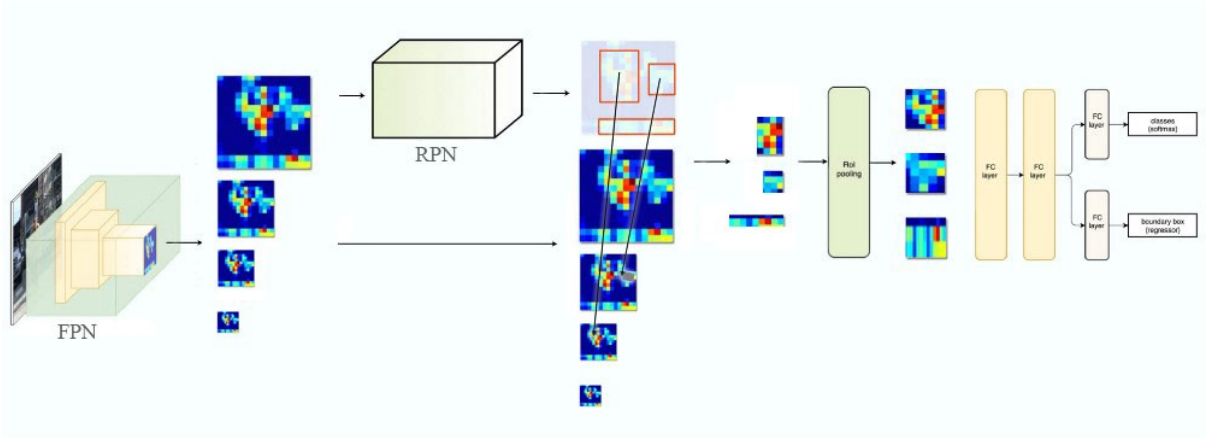


Figure 4: FasterRCNN that uses FPN as a backbone

As you can see the FPN in the beginning serves as a shared backbone. Its output is the input to the RPN, which in turn, outputs the objectness and bounding boxes for each anchor box, for each feature of each feature map .

4.1 Anchor Box Creation

You learned in class that the anchor boxes are reference boxes, relative to which, we parametrize the outputs of the RPN.

Given an anchor box with width w and height h we define:

$$\text{aspect ratio} = \frac{w}{h}, \quad \text{scale} = \sqrt{wh}$$

Anchor Boxes for a single level of the FPN

The i^{th} level of the FPN produces a feature map with spatial dimensions $S_{y,i} \times S_{x,i}$. So it segments the input image into a $S_{y,i} \times S_{x,i}$ grid, and each point of the feature map corresponds to one of these grid cells. In a single level of the FPN, for each grid cell we will assign 3 anchor boxes with the same scale and aspect ratios of 1:2, 1:1, 2:1. Also their centers will be the center of their assigned grid cell.

As a result if we have a FPN level with a $S_{y,i} \times S_{x,i}$ feature map we will get $3 * S_{y,i} * S_{x,i}$ anchor boxes.

Anchor boxes for all the levels of the FPN:

To assign the anchor boxes across all the levels of the FPN we just have to repeat the process of the single level anchor box assignment for all the levels. Notice that the scale of the anchor boxes varies between the FPN levels. For the FPN levels with feature maps P_2, P_3, P_4, P_5, P_6, the corresponding anchor boxes have scales $32, 64, 128, 256, 512$ respectively.

So when we have 5 FPN feature maps, the total number of anchor boxes that we will assign will be $\{\text{Total number of Anchor Boxes}\} = \sum_{i=1}^{5} 3 \cdot S_{y,i} \cdot S_{x,i}$.

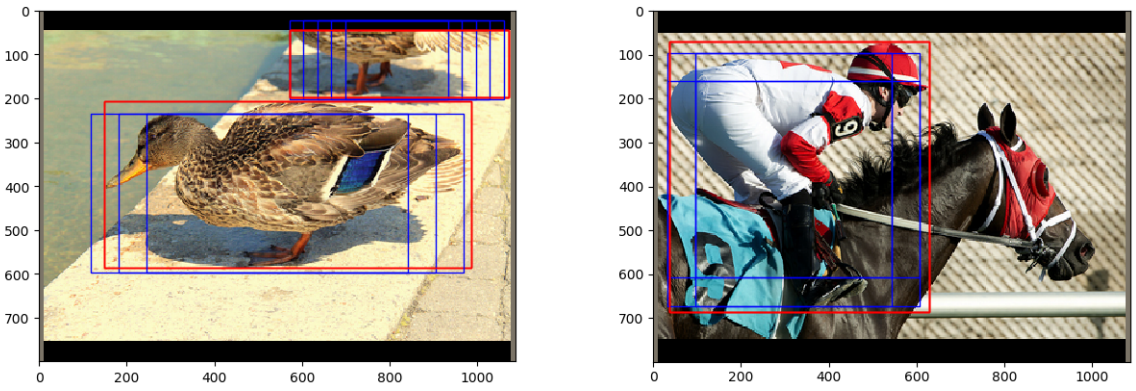
4.2 Ground truth creation

We will assign a binary label p^* (object or not) to each anchor. Firstly, cross-boundrary anchors are removed. Out of the remaining:

- positive labels ($p^* = 1$) will get the anchors that have either: (1) highest IOU with a ground truth box, or (2) $IOU > 0.7$ with ANY ground truth box. Note that a single ground truth box may assign positive labels to multiple anchors.
- Negative labels ($p^* = 0$) will get the anchors that (1) are non-positive and (2) have $IOU < 0.3$ with EVERY ground truth box.

Anchors with neither positive or negative labels are excluded from training both for the classification and the regression branch. After this assignment each anchor with a positive label is assigned a ground truth box, which is the box that the anchor has the highest IOU with. (a ground truth box can be assigned to multiple anchors, one anchor is assigned to a single ground truth box)

In figure (5) we can see some examples of anchors (blue) with positive labels and the corresponding ground truth boxes (red). To make sure that your ground truth assignment is correct, visualize the anchor box assignment similar to (5).



(a) (b)

Figure 5: Positive anchors boxes (blue) and the corresponding ground truth bounding boxes (red)

4.3 RPN Heads

Intermediate Layer: Define a standard convolutional layer with kernel size (3,3,256) with same padding (followed by BatchNorm and ReLU). The input of the intermediate layer is the feature maps produced by the FPN. (one feature map at a time)

Classifier Head: Define a convolutional layer with kernel size (1,1,1*3) and same padding, followed by a Sigmoid function. The input of the classifier head is the output of the intermediate layer. Because we use same padding if we run the intermediate layer and the classifier head on a feature map of size $S_y \times S_x$ with 256 channels, the output will be a tensor of size $(3, S_y, S_x)$. So when we pass all the FPN feature maps through the RPN Heads, the classifier outputs one objectness score for each anchor for each grid cell of all the feauture maps. This means that in our architecture each channel of the classifier's output corresponds to the objectness score of one of the 3 anchor boxes in the specific grid cell.

Regressor Head: Define a convolutional layer with kernel size (1,1,4*3) and same padding. Similar to the classifier's head, its input is the output of the intermediate layer. Also if we pass a feature map of size $S_y \times S_x$ with 256 channels through the intermediate layer and the regressor the output will be a tensor of size $(12, S_y, S_x)$. So the regressor outputs one bounding box prediction for each anchor for each grid cell of all the feature maps.
The proposals of the regressor for one anchor are not the raw box coordinates but coordinates relative to the anchor. In particular for a single anchor, the regressor outputs a vector $\mathbf{t} = (t_x, t_y, t_w, t_h)$, which encodes the raw box proposal as follows:

$$t_x = (x - x_a)/w_a \quad t_y = (y - y_a)/h_a \quad t_w = \log(w/w_a) \quad t_h = \log(h/h_a)$$

where x, y, w, h denote the proposed box's center coordinates and its width,height. Also x_a, y_a, w_a, h_a denote the anchor box's center coordinates and its width, height.
This means that in our architecture the output of the regressor which has 12 channels is splitted into 3 groups of 4 channels. Each one of the 3 groups corresponds to the encoded proposed coordinates for one of the 3 anchor boxes of the specific grid cell.

Remark: For all the feature maps of the FPN we use the same intermediate layer, classifier and regressor head.

4.4 Losses

Classifier's Loss:
For the classifier's loss, we compute the binary cross entropy between the predicted objectness p and the ground truth objectness p^*.
We take the binary cross entropy loss across all the anchors with positive and negative ground truth

labels in the sampled mini batch. (see the remark in the end of the section for details about the sampling process)

Regressor's Loss:
For the regressor's loss we use the Smooth L1 loss between $\mathbf{t}, \mathbf{t}^*$, which is defined as:

$$L_{reg}(\mathbf{t}) = \frac{1}{N_{reg}} \sum_{i=1}^{N_{reg}} p_i^* \sum_{j=1}^{4} smooth_{L1}(\mathbf{t}_i[j] - \mathbf{t}_i^*[j])$$

where,

$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & |x| \leq 1 \\ |x| - 0.5 & else \end{cases}$$

and N_{reg} is the number of anchor locations in the sampled mini-batch. Note that the regressor loss is computed only on the positive anchors in the sampled mini-batch (where p_i^* is 1).

Remark: In [2], the authors propose an "image-centric" sampling strategy, where each mini-batch arises from a single image that contains many positive and negative example anchors. In order, to minimize the bias towards the negative classes they subsample the positive and negative anchors in a ratio as close to 1:1 as possible. Also they set a constant size M for the mini-batch and if the positive anchors are not enough to get M total samples and keep the 1:1 ratio, they complete the batch with negative anchors.

We will follow the same approach but we will not limit the sampling to only the anchors of one image, but we will use the anchors from a small batch of images (e.g. batch size is 1 to 4 images). So the sampling process is the following

- Set a constant size M for the mini-batch
- Randomly choose M/2 anchors with positive ground truth labels. If there are not enough positive anchors choose as much as possible.
- Fill the rest of the batch with negative anchors so you will have a mini batch with M anchors
- For the mini batch that you have constructed compute the losses.

4.5 Training

Train the whole network with *equally* weighted losses from the proposal classifier and the regressor. Mind that the losses in Pytorch are usually normalized (reduction='mean'), but maybe they are normalized differently between classification and regression (eg. over batch versus over anchor boxes) in your implementation. If this is the case find the multiplication factor in one of them, so that the losses are normalized equally.

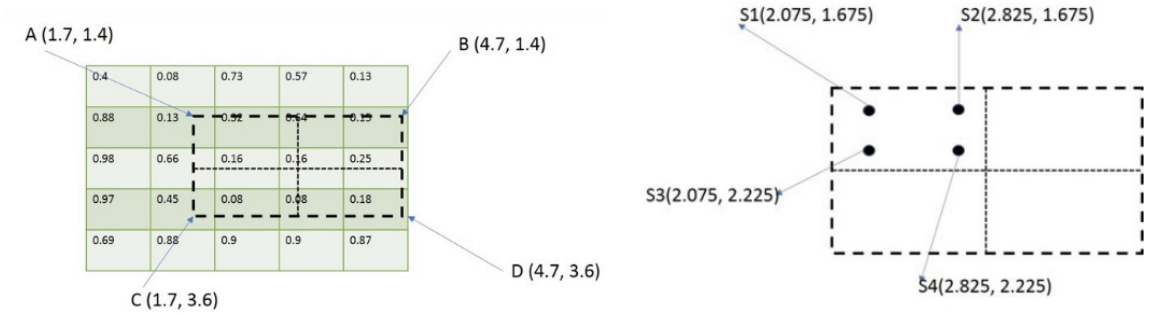
4.6 Post-Processing

After decoding the boxes, clip the boundary crossing ones during testing and remove them during training. From them keep the top K boxes (the boxes with the top objectness scores from the classifier). Then apply NMS and then keep the top N boxes. For training the rest of the network you should tune K,N so after the post-processing we get around 1000 proposal boxes.

5 Preparation of the RPN outputs for the second stage

5.1 ROI Align

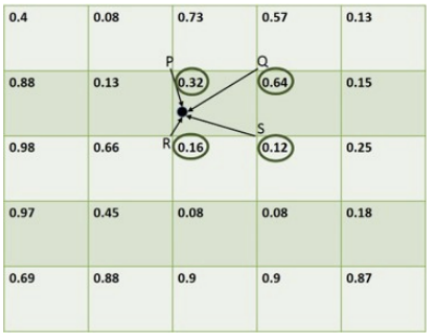
The outputs of the Region Proposal Network may not be integers, which means that the corresponding predicted boxes do not align perfectly with the image. Moreover, we need to transform the region that each box surrounds into a fixed $P \times P$ matrix to train the network further.



(a) (b)

Figure 6: (a) Proposed Bounding Box from RPN,(b) Region Sampling

In order to avoid quantization both for the alignment and the value assignment RoI Align was introduced in [1]. To implement a RoI align you will have to divide your boxes into P by P regions. To assign values into those P by P cells you sample (regularly or randomly) 4 points inside each cell (their coordinates not their value).
The value of each cell will be the average values of $\{S_1, S_2, S_3, S_4\}$. The value of each S_i, on the other hand, can be found through bi-linear interpolation. The assumption is that the underlying image is continuous inside each pixel and it scales linearly in both directions x,y until we hit the next pixel. Each one of the S_i's is inside one cell of the feature map. So, we can use the values of the neighbouring pixels to reconstruct its value, as shown in figure (8).
We apply the ROI Align for each channel of the feature map independently. So if we apply ROI Align for one bounding box in a feature map of the FPN, the output will be a map with the same number of channels so it will be a map represented by a tensor of size (256,P,P)



$$f(x,y) = \frac{1}{(x2-x1)(y2-y1)} \begin{bmatrix} x2-x & x-x1 \end{bmatrix} \begin{bmatrix} f(x1,y1) & f(x1,y2) \\ f(x2,y1) & f(x2,y2) \end{bmatrix} \begin{bmatrix} y2-y \\ y-y1 \end{bmatrix}$$
$$f(S1) = 0.21778 = \frac{1}{(1)(1)} \begin{bmatrix} 9.25 & 0.75 \end{bmatrix} \begin{bmatrix} 0.32 & 0.16 \\ 0.64 & 0.12 \end{bmatrix} \begin{bmatrix} 3.25 \\ 6.75 \end{bmatrix}$$

Figure 7: Bilinear Interpolation

5.2 Choose the appropriate FPN level to pool features (implemented in HW4)

As shown in figure (2), for each proposal we will use one of the feature maps $P2, P3, P4, P5$ of the FPN to pool features that will be used from the later stages.
Given a proposal box with width w, height h we will pool features from the feature map P_k where:

$$k = \lfloor 4 + log_2 \left(\frac{\sqrt{wh}}{224} \right) \rfloor$$

Of course k has a range limited from 2 to 5.

After finding the feature map P_k, we must find the region on the feature map that corresponds to the proposal box (the region of the proposal box is given in the image coordinates, so we have to rescale them to the coordinates of the feature map).

Finally having found the correct region in the feature map, we use ROI Align to pool a feature map with 256 channels and spatial dimensions $P \times P$. (we will use P=7 for the preparation of the features of the Box Head). For each proposal we flatten the (256,P,P) output of the ROI Align to get a feature vector of size $256 * P^2$. This feature vector will be the input of the box head that will refine the proposal box.

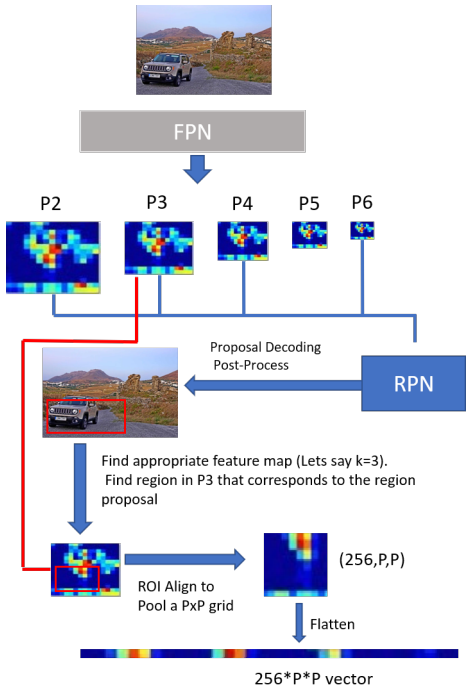


Figure 8: Preparation of feature vector for the 2nd stage of Mask RCNN

6 Box Head (implemented in HW4)

This part of the architecture is responsible for refining and classifying the $N \approx 1000$ proposals produced by the RPN. Similar to RPN it consists of an intermediate layer a classifier and a regressor. Its input is the feature vectors with size $256 * P^2$ that was described in Section 5.

Remark 1: In order to simplify the notation we will use the same symbols p, p^*, t, t^*, that were used in Section 4 to denote the outputs and the ground truths of the classifier and the regressor of the Box Head. So in this section when we are using these symbols we are referring to the outputs and ground truths of the classifier and the regressor of the Box Head, not the RPN.

Remark 2: The Box Head processes each proposal independently but in your implementation you can stack them together in a single batch, this way you can benefit from the batch parallel computation of PyTorch. So for a single image the Box Head input is a tensor of size $(N, 256 * P^2)$

6.1 Creation of the Ground Truth (implemented in HW4)

For the ground truth creation of the Box Head each proposal produced by the RPN is either assigned to a ground truth box or to the background.
Assuming we have C number of classes (in the given dataset C=3), we will give each proposal a ground truth class label p^* that assigns the proposal to one of the $C + 1$ possible classes (C classes plus background). So $p^* \in \{0, \dots, C\}$, where we will use 0 as the background class.

A proposal is assigned to a ground truth box and its class, if it has an $IOU > 0.4$ with that box. If a proposal has $IOU > 0.4$ with multiple ground truth boxes then it is assigned to the ground truth box that it has the highest IOU with. Finally if a proposal doesn't have $IOU > 0.4$ with any ground truth box then it is assigned to the background class, so it gets $p^* = 0$.

6.2 Box Head Network (implemented in HW4)

6.2.1 Intermediate Layer

For the intermediate layer define a simple 2 layer MLP (2 linear layers in Pytorch) with ReLU activation functions. The size of the output of both of the 2 layers are 1024. The input of the intermediate layer is the flattened feature vector of dimension $256 * P^2$ that was produced in Section 5, after the ROI Align.

6.2.2 Classifier

For the classifier define another fully connected layer (Linear Layer) with output size of $C + 1$, followed by a softmax layer. The input of the classifier is the output of the intermediate layer for one proposal box.
The output p of the classifier is a vector with size $C+1$, which represents the predicted class probabilities for the proposal box.

6.2.3 Regressor

For the regressor define another fully conntected layer with output size of $4*C$. The input of the regressor is the output of the intermediate layer.

In figure (2) you can see that the regressor predicts the final bounding boxes in parallel with the classifier that predicts the classes of the bounding boxes. For this reason for each proposals it predicts one bounding box for each class. During testing we keep the bounding box that corresponds to the class predicted by the classifier. Because for class i the regressor outputs a vector of encoded coordinates $\mathbf{t^{(i)}}$ with 4 elements, the total output of the regressor has the form:

$$\mathbf{t}_{\text{total}} = [t_x^{(1)}, t_y^{(1)}, t_w^{(1)}, t_h^{(1),}, \ldots, t_x^{(C)}, t_y^{(C)}, t_w^{(C)}, t_h^{(C)}]$$

and as a result this is the reason the output has size $4 * C$.

Notice that similar to the RPN, the output of this regressor uses the encoded coordinates of the bounding boxes, not the raw ones. Here the coordinates are relative to the proposal box (in the RPN they were relative to the anchor box). So

$$t_x = (x - x_p)/w_p \quad t_y = (y - y_p)/h_a \quad t_w = \log(w/w_p) \quad t_h = \log(h/h_p)$$

where x, y, w, h are the center coordinates and the width, height of the predicted bounding box. And x_p, y_p, w_p, h_p are the center coordinates and the width, height of the corresponding proposed box from the RPN. Similarly the ground truth for the regressor will be parameterized as:

$$t_x^* = (x^* - x_p)/w_p \quad t_y^* = (y^* - y_p)/h_p \quad t_w^* = \log(w^*/w_p) \quad t_h^* = \log(h^*/h_p)$$

where x^*, y^*, w^*, h^* corresponds to the center coordinates and the width, height of the ground truth box that the proposal was assigned to.

6.3 Losses (implemented in HW4)

Classifier's Loss:
For the classifier's loss, we compute the cross entropy loss between the predicted class probability vector p and the ground truth class p^*. It is the same with the loss in the RPN with the only difference that now we don't have binary cross entropy because we have $C + 1$ classes.
We take the cross entropy loss across all the proposals in the sampled mini batch.

Regressor's Loss:
For the regressor loss for each proposal we use the the Smooth L1 loss between the $\mathbf{t^{(p^*)}}$ and t^*. This

means that we compare the encoded coordinates of the ground truth bounding box with the specific prediction of the regressor that corresponds to the ground truth class. If the ground truth class is the background then we do not compute the Smooth L1 loss for this proposal. We can write the previous description as the following formula:

$$L_{reg}(\mathbf{t}) = \frac{1}{N_{reg}} \sum_{i=1}^{N_{reg}} \left(\sum_{c=1}^{C} \mathbb{1}(p_i^* = c) \sum_{j=1}^{4} smooth_{L1}(\mathbf{t}_i[j] - \mathbf{t}_i^*[j]) \right)$$

where N_{reg} is the number of proposals of the sampled minibatch, and $\mathbb{1}$ is an indicator function

For sampling the mini batch we can use a similar policy with the RPN but now we sample proposals with no-background ground truth and proposals with background ground truth with a ratio as close to 3:1 as possible. Again you should set a constant size for the mini-batch.

6.4 Training (implemented in HW4)

You should train the whole Box Head using the appropriate weighting so that the classifier's and regressor's loss have similar values (what we want to avoid is one of the losses having much lower value than the other one, so it is practicaly ignored during training).

6.5 PostProcessing (implemented in HW4)

During Training of the Mask Head:
Remove the cross boundary boxes. Choose the bounding boxes that have an $IOU > 0.5$ with some ground truth bounding box and keep the top K' boxes. Apply NMS for each class independently and then keep the top N' boxes. For the training of the mask head you should aim to keep around 100 boxes so you should tune K',N' appropriately.

During Testing:
Crop the cross boundary boxes and remove the ones where the background class has the highest predicted probability. Then keep the top K' boxes, apply NMS for each class independently and from the remaining keep the top N' boxes. You should tune K',N' to find which values give you the best results during testing.

7 Mask Head

At this point for each image, the box head has produced $D \approx 100$ refined bounding boxes.

7.1 Prepare the Input for Mask Head

Again similar with 5.2, for each refined bounding box we find the appropriate FPN feature map and we apply the RoI Align with $P = 14$. So we end up with a tensor `[D,256,14,14]`, where `D` < 100.

7.2 Mask Head Network

We pass this tensor `[D,256,14,14]`, from a 6 layer CNN.

- the first four layers have kernel size (3,3,256) with same padding and ReLU activation
- the last two layers are:
    1. A deconvolution layer which doubles the region's size and keeps the channels to 256 (with ReLU activation)
    2. A convolutional layer with kernel (1,1,C).

So the output for one image is a tensor of size (D,C,28,28). This means that the mask head produces one mask for each class, for each detection from the regressor. Since this is a pixelwise binary classification problem, as always we will use sigmoid to suppress the output in $[0, 1]$ range.

7.3 Loss Computation

From all the masks that are predicted for one bounding box, we will use the mask associated with the ground truth class to train the mask branch. The mask target is the intersection between the bounding box produced by the Box Head and its associated ground-truth mask. This branch is trained with binary cross entropy (BCE) loss pixelwise.

$$L_{mask} = -\frac{1}{m^2} \sum_{i,j} y_{ij} \log \hat{y}_{ij} + (1 - y_{ij}) \log(1 - \hat{y}_{ij})$$

Notice that the original mask target does not have the same dimensions with the output of the Mask head which has spatial dimensions of 28×28. So after finding the intersection between the bounding box and the ground-truth mask we need to rescale it to the same dimensions as the output of the Mask head.

7.4 Training

To train the Mask Head we are using the pixelwise binary cross entropy loss, averaged across all the D predicted masks.

7.5 PostProcessing

During testing choose the masks that correspond to the class that was predicted by the Box Head. After that, rescale the predicted masks back to the original image, using bilinear interpolation and rescale the boxes too. Choose a threshold (e.g. 0.5) to turn the network's output into a binary mask.

8 Overall Training

In the previous sections we describe how to train the different parts of the Mask RCNN independently, while we freeze the parameters of all the previous parts. To increase the performance you may consider after the independent training, to also train them all together to fine tune your network.

9 Requirements

9.1 Proposal

The proposal must contain:

- Names of your team member
- Brief introduction to the Mask RCNN architecture (4%)
- Related work and brief comparison of it, with the Mask RCNN architecture (3%)
- Description of the dataset (e.g statistics about the scales of the objects, the number of objects per image). Also a brief comment about how this statistics can affect the performance of your system (3%)
- Description of the evaluation methods that you plan to use to test and show the performance of you network (10%)

9.2 Final Submission

The final submission must contain:
Report (45%**):**

- Introduction: A brief introduction of the system. (You can use the same as the proposal if you haven't changed something major in your implemantation) (3%)

- Training :A description of your training procedure. You should also include the different learning curves that were produced during the training of the different parts of the system (15%)
- Results: Results from the experiments that showcase the performance of your system. This should include the evaluation metrics mentioned in the proposal, or a discussion about what new evaluation metrics you used (20%)
- Conclusion: What observations can you make about the results of your implementation. What suggestions you have that can help improve your performance (7%)

Code (15%**:)**
All the necessary files of your implementation must be submitted as a zip file. This file must also contain a README.md file with instructions about how to run your code.

Video (10%**:)**
A 5 minute video where you go through your implementation and your results.

References

[1] He, K., Gkioxari, G., Dollar, P., Girshick, R.: Mask R-CNN. In: ICCV. (2017)

[2] Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS. (2015)

[3] Girshick, R.: Fast R-CNN. In: ICCV. (2015)

[4] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie: Feature Pyramid Networks for Object Detection In: CVPR (2017)

Final Project Option-2: Extension of GAN and VAE to model Multimodal Distribution

1 Background

In many image conditional synthesis tasks, we are expanding the input image signals. For example, when we translate an edge image into an RGB image, we need to fill the missing color and texture, which could have multiple plausible hypotheses, such as figure 1 shown below.

Figure 1: A visual illustration of one-to-many mapping between an edge image and multiple RGB images. First column is the input edge image. Second column is the ground truth edge. Column 3-5 are a set of the generated plausible and diverse RGB images that satisfy the given edge constraint.

In the upcoming Project GAN, we will be implementing basic GAN and VAE models as well as a popular unsupervised image-to-image translation model CycleGAN.

The generative model, CycleGAN, can only be used to model one-to-one image mapping, which is ill-posed in many situations, such as edge2shoe translation. This extension project will implement a more advanced generator that can model one-to-many mapping or so-called multimodal image-to-image translation. This network aims to produce not only plausible but also diverse generated image outputs given the conditional image input.

2 Instruction

We will implement a more advanced generative model named BicycleGAN, an extension of CycleGAN. This model links GAN and VAE, and explicitly learns a latent distribution that encodes the uncertainty information in image-to-image translation. During inference, the latent code can be randomly sampled to generate multimodal outputs. This section will provide some relatively high-level instructions for how you could implement training and inference code for BicycleGAN. It's also strongly encouraged for you to read the original paper (https://arxiv.org/pdf/1711.11586.pdf), as you could get more in-depth insights and find more details. If you choose this project, please follow the code template that we provided for the model implementation.

2.1 Training

Denote $\mathcal{A} \subset \mathbb{R}^{H \times W \times 3}$ and $\mathcal{B} \subset \mathbb{R}^{H \times W \times 3}$ are the two domains of images, during training, we are given a dataset of paired instances from these domains, $\{(\mathbf{A} \in \mathcal{A}, \mathbf{B} \in \mathcal{B})\}$, which is representative of a joint distribution $p(\mathbf{A}, \mathbf{B})$. G as generator, and E as encoder. The goal is to translate images from domain $\mathcal{A}$ to domain $\mathcal{B}$ with multimodal distribution $p(\mathbf{B} \mid \mathbf{A})$. For the loss term, "$L_1$" stands for L1 loss, "$D$" stands for discriminator loss, and "KL" stands for KL divergence.

The training phase of BicycleGAN model consists of two major modules: cVAE-GAN ($\mathbf{B} \rightarrow \mathbf{z} \rightarrow \widehat{\mathbf{B}}$) and cLR-GAN ($\mathbf{z} \rightarrow \widehat{\mathbf{B}} \rightarrow \widehat{\mathbf{z}}$). First, cVAE-GAN starts from a ground truth target image B and encode it into the latent space represented by distribution $Q(z \mid B)$. The generator then attempts to map the input image A along with a sampled z back into the original image B. Second, cLR-GAN randomly samples a latent code from a known prior distribution, usually is $\mathcal{N}(0, 1)$, uses it to map A into the output $\hat{B}$, and then tries to reconstruct the latent code from the output.

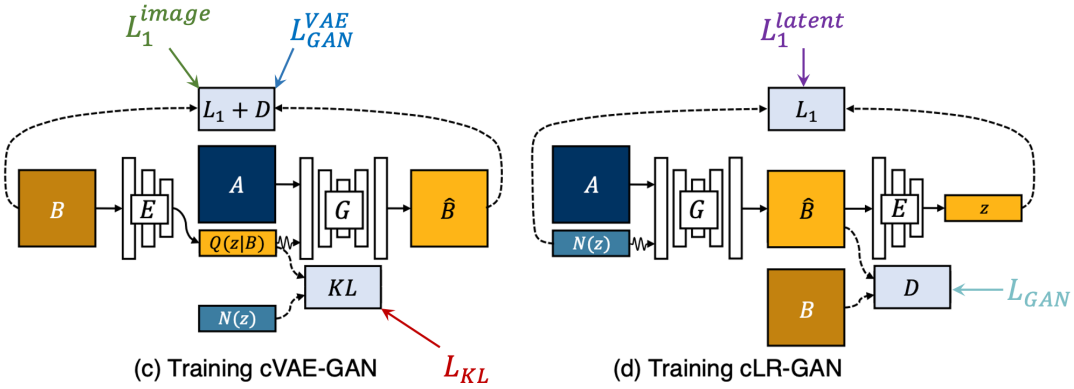


Figure 2: An overview of BicycleGAN training scheme.

To train cVAE-GAN module, you will need to implement the following loss functions. First, we need a simple L1 distance to constraint that the transformed image $G(A, z)$ should be similar to target image B.

$$\mathcal{L}_1^{image}(G) = \mathbb{E}_{A,B \sim p_{(A,B)}, z \sim E(B)} ||B - G(A, z)||_1 \tag{1}$$

Further, the adversarial loss is applied to encourage the generated image to be photo-realistic:

$$\mathcal{L}_{GAN}^{VAE} = \mathbb{E}_{A,B \sim p_{(A,B)}}[log(D(A, B))] + \mathbb{E}_{A,B \sim p_{(A,B)}, z \sim E(B)}[log(1 - D(A, G(A, z)))] \tag{2}$$

Also, we need to enforce the latent space to be a compact Gaussian distribution, where we can sample from during inference time. Thus, we need a KL divergence to enforce such property, similar to what you have done in the VAE project.

$$\mathcal{L}_{KL} = \mathbb{E}_{B \sim p_{(B)}}[D_{KL}(E(B)||\mathcal{N}(0, 1))] \tag{3}$$

In the cLR-GAN module, we want to regularize that the generated image $\hat{B}$ can be encoded back into the learned latent space of B, which is a normal distribution after KL divergence is optimized in cVAE-GAN. Thus, the implementation is simply placing a L_1 loss between the encoded latent vector of $\hat{B}$ and prior distribution p(z).

$$\mathcal{L}_1^{latent}(G, E) = \mathbb{E}_{A \sim p_{(A)}, z \sim p(z)} ||z - E(G(A, z))||_1 \tag{4}$$

We also enforce the generated image $\hat{B}$ to be as photo-realistic as possible in the cLR-GAN module, so the adversarial loss is applied again here. Although this adversarial loss is the same as $\mathcal{L}_{GAN}^{VAE}$, you should use two separate discriminator networks in cVAE-GAN and cLR-GAN for easier optimization purpose.

$$\mathcal{L}_{GAN} = \mathbb{E}_{A,B\sim p_{(A,B)}}[log(D(A,B))] + \mathbb{E}_{A\sim p_{(A)},z\sim p(z)}[log(1-D(A,G(A,z))] \quad (5)$$

The overall objective can be expressed as:

$$G^*, E^* = arg \min_{G,E} \max_{D} \mathcal{L}_{GAN}^{VAE} + \lambda \mathcal{L}_1^{image}(G) + \mathcal{L}_{GAN} + \lambda_{latent}\mathcal{L}_1^{latent} + \lambda_{KL}\mathcal{L}_{KL} \quad (6)$$

where λ = 10, λ_{latent} = 0.5, λ_{KL} = 0.01. You can use Adam optimizer with learning rate of 0.0002 to optimize the algorithm. The latent dimension of z could be chosen as 8. The latent code z is injected into the generator by concatenating with the image or intermediate layers after adjusting its spatial dimensions.

Since this is a final project, we leave some freedom to design your generator and discriminator architectures. You may try out different types of generator, such as U-Net, and different types of discriminator, such as PatchGAN discriminator. You are also encouraged to explore other tricks to stabilize GAN training or improve image quality, such as instance normalization, spectral normalized GAN, etc. To further enforce diversity, you may investigate MSGAN (https://arxiv.org/abs/1903.05628) for a simple but effective trick to encourage sampling diversity in this type of multimodal generation models. Note that most of these tricks can be done in a few code lines, as there are many Pytorch built-in implementations. So, don't be afraid to try them out!

Our code template consists of three files: "datasets.py", "models.py", "train.py". You can directly download the dataset and use "datasets.py" for you dataloader without any modifications. Your task is to fill in the missing parts in "models.py" and "train.py", where the detailed requirements are in the code.

2.2 Inference

The inference process of multimodal prediction is slightly different than the earlier projects. For a given input image A, we aim to sample a distribution of plausible outputs. This is achieved by sampling random variable in the VAE latent space, and together with the conditional input A, we can decode different plausible images $\hat{B}$. Basically, if you sample random variables $z \sim \mathcal{N}(0,1)$ ten times, you should theoretically generate ten different image outputs. (if no severe mode collapse happens!)

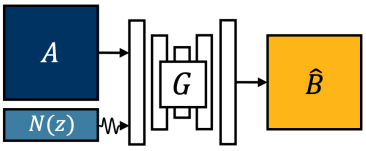


Figure 3: An overview of BicycleGAN inference scheme.

You need to create a new file named "inference.py" that can generates transformed images from your pretrained models, and saves generated images into local directory for further evaluation.

3 Evaluation Metrics

- **Visualization of Training:** Please plot all the losses over time, and save multiple intermediate generated images with a fixed set of inputs at every epoch. You may want to try at least 20 epochs.
- **Quantitative Evaluations:** (1). To evaluate the photo-realistic quality, please use FID score between 200 generated images and real images in the validation set. You can use the code provided here (https://github.com/mseitzer/pytorch-fid) to do this. (2). To quantify the diversity of your multimodal prediction, you need to compute the pairwise distance between ten samples for each conditional inputs. The distance metrics can be LPIPS. Please refer LPIPS github (https://github.com/richzhang/PerceptualSimilarity) for computing this LPIPS score. Once you figure out how to use LPIPS, you need to write a simple script to compute the averaged pairwise distance across the whole validation set.
- **Qualitative Evaluations:** For each of the validation image, please demo three randomly sampled results together with the input. To demonstrate this clearly, you can stitch the input image at the first column, and the

other four images at column 2-4. "torchvision.utils.save_image" maybe helpful to do this, but feel free to use anything you like. Please includes at least 10 examples in your submission.

(Note: Please use the provided validation set for your evaluations.)

4 Final Deliverable

Please write a README file to describe how to run your inference code, and zip all of your code into a file for submission. Please also put all of the evaluation results from section 1.3 into a single PDF file.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In Advances in Neural Information Processing Systems 26, pages 2672–2680, 2014.

[2] Thalles Silva. An Intuitive Introduction to Generative Adversarial Networks (GANs). FreeCodeCamp.org, Jan. 2018, www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394/.

[3] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In Proceedings of 4th International Conference on Learning Representations, 2016.

[4] Mehdi Mirza, and Simon Osindero. Conditional Generative Adversarial Nets. arXiv preprint arXiv:1411.1784, 2014.

Final Project Option-3: Real-word Industrial Challenge

1 Background

One of the long-standing unsolved problems in food inspection is seal verification. Think of all of the food, beverage, and pharma products you see in sealed packages. In many of these applications, a broken seal means product spoilage or a big mess! The traditional method to find these defects is that operators manually squeeze the pouch to check whether there is any leak.

Fortunately, many seal defects can now be detected with high-speed, high-resolution x-ray systems. There are three critical assessments in checking if a seal is good. We show three examples below.

1. Does the sealing area contain any foreign material that would undermine seal strength? In Fig.[1a], there is a triangle-shape food region in the corner.

2. Are there any micro-leaks in the sealing area (these are folds in the packaging material)? In Fig.[1b], there is a long channel in the bottom seal. This micro-leak connects the food with the outside air, making the food oxidized.

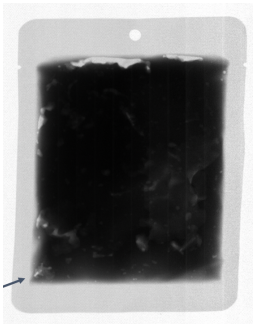
3. Are the seal width consistent on each side? In Fig.[1c], you can find the left side seal is uneven because of a bad manufacturing cut.

In this scenario, our task is to identify the defected pouch images, which includes at least one of the above defected.

(a) Food in the seal region

(b) Micro-leak in the seal region

(c) Uneven and narrow seal width



(d) Good product without any defects

Figure 1: Different defect types and good product without defects

2 Problem formulation

For industrial problems, a combination of the traditional method (i.e., image processing using OpenCV) and machine learning method (i.e., Neural Networks) could help tackle the problem. We show an example in Fig.[2], with a flat pouch with several micro-leaks, scotch tape, and folds. A micro-leak can form a channel between the outside of the package and the inner product. The left image is a raw greyscale image. The middle image is the edge detection result. The right image shows the red lines highlight the micro-channel leak, part of the food edges, and part of the pouch edges. Fig.[3] shows the defect labels in details.

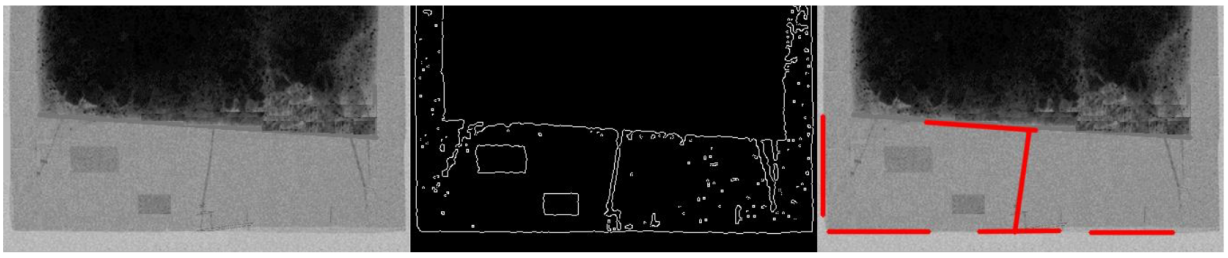


Figure 2: Example image that includes the micro leak

The first thought that comes to mind would be using the deep learning segmentation schema we learned in class. But such methods usually involve a large amount of manual labeling. In this particular industry, defected objects are rare ($\frac{Pouch\ with\ Defects}{All\ pouches} = 1:10000 - 1:1000$). Currently, we only have 14 labeled images, which makes the supervised learning method nearly infeasible.

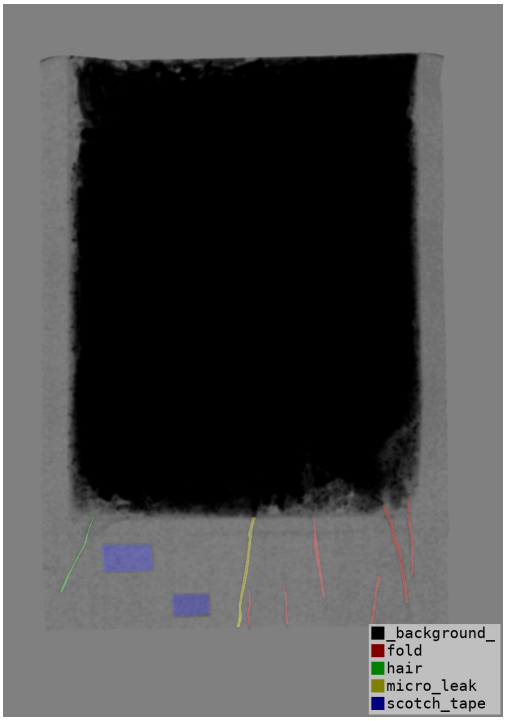


Figure 3: Different defect segmentation in a single pouch product

2.1 Possible solution using generative model

To overcome the lack of labeled images, we will treat this problem as an abnormality detection. Since we have near infinite images of 'normal' food packages, it might be feasible to create a generative model that can recreate the non-defect images. We can then detect 'abnormality' by comparing the recreated image vs. the original image.

The CycleGAN (Generative Adversarial Network)[2] is designed to generate images across two different image domains. Assuming $\mathcal{A} \subset \mathbb{R}^{H \times W \times 3}$ and $\mathcal{B} \subset \mathbb{R}^{H \times W \times 3}$ are two image domains. $\mathcal{A}$ here is the normal package image domain and $\mathcal{B}$ is the defected package image domain.

One of the Generator network in CycleGAN will learn a mapping from defected domain $\mathcal{B}$ to normal image domain $\mathcal{A}$: given a conditional image $B \in \mathcal{B} \subset \mathbb{R}^{H \times W \times 3}$, generator model G could synthesized image A in domain $\mathcal{A}$.

One can think of this network performs a 'defect' removal function. By comparing the original image with the defected removed image, we can detect an abnormality.

A critical feature of CycleGAN is that it can learn from unpaired distribution. Our application means we don't need to specify how a 'normal' food image becomes defected. The network will learn by observing many 'normal' and 'defected' images. For a generator G mapping from domain $\mathcal{B}$ to $\mathcal{A}$, in inference, we want to find such a distance metric Df which could capture follow information:

1. when input image is $A \in \mathcal{A}$, output is $A^{'}$, $Df(A - A^{'}) \leq \sigma_1$
2. when input image is $B \in \mathcal{B}$, output is A, $Df(B - A) > \sigma_2$

Here $\sigma_1 << \sigma_2$ and you could use a threshold to separate them easily and identify the defected image.

To produce a mask over the defect image region, we could do the following. For detected image B, we could divide the picture into several patches. Then go through each image patch pair (p_A, p_B), if $Df(p_A, p_B) > \sigma_3$, then we could claim this patch as a defected patch. Noted here Df for patch distance computation could be different to the Df we define for image distance computation.

Since this problem is a real industry challenge, we give the most freedom in the design choice. You could utilize any method to solve this problem, as long as the model could detect the defected image. There is also one method you should look at, namely CAM[1]. CAM creates a heat map over the original image, indicating the important image areas for a learned network.

3 Evaluation Metric

We would give a test set for this project, including the normal package image and defected package image. You will use your model to classify each image and produce a mask to indicate the region's defect.

1. The classification accuracy will be computed using mAP.
2. Since original data set itself will not have bounding box annotation, so we will evaluate the output defected area by visual results.

4 Final Deliverable

Please write a README file to describe how to run your inference code, and zip all of your code into a file for submission. Please also put all of the generated results from section 3 into a folder and zip it.

References

[1] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE international conference on computer vision, pages 618–626, 2017. 3

[2] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE international conference on computer vision, pages 2223–2232, 2017. 3

Final Project Option-4: Human Action classification

1 Problem formulation

1.1 Background

Human action recognition has a wide range of applications: surveillance, human-computer interaction, video-storage retrieval, sports video analysis, etc.

It is a computer vision problem towards an automated understanding of videos. There are several challenges in the problem, such as large changes in the video's appearances arising from occlusion, viewpoint changes. Furthermore, there are issues related to ill defined action labeling vocabulary.

We shall be tackling the problem of action classification, categorizing the segmented input videos into one of the activity labels. It is different than action detection, where the problem is to annotate the start and end frame of an activity correctly.

1.2 Action Classification Architectures

Existing video action recognition architectures rely on either 2D or 3D kernels or their fusion. The frameworks are also designed to extract the long-range spatiotemporal context in videos. For example, LSTMs or feature aggregation can be performed on 2D Convnet models. Besides RGB frames, additional information from the video clips such as optical flow or human pose estimation can be combined in a multistream architecture to enhance classification. Figure 1 provides a table of some of these existing architectures.

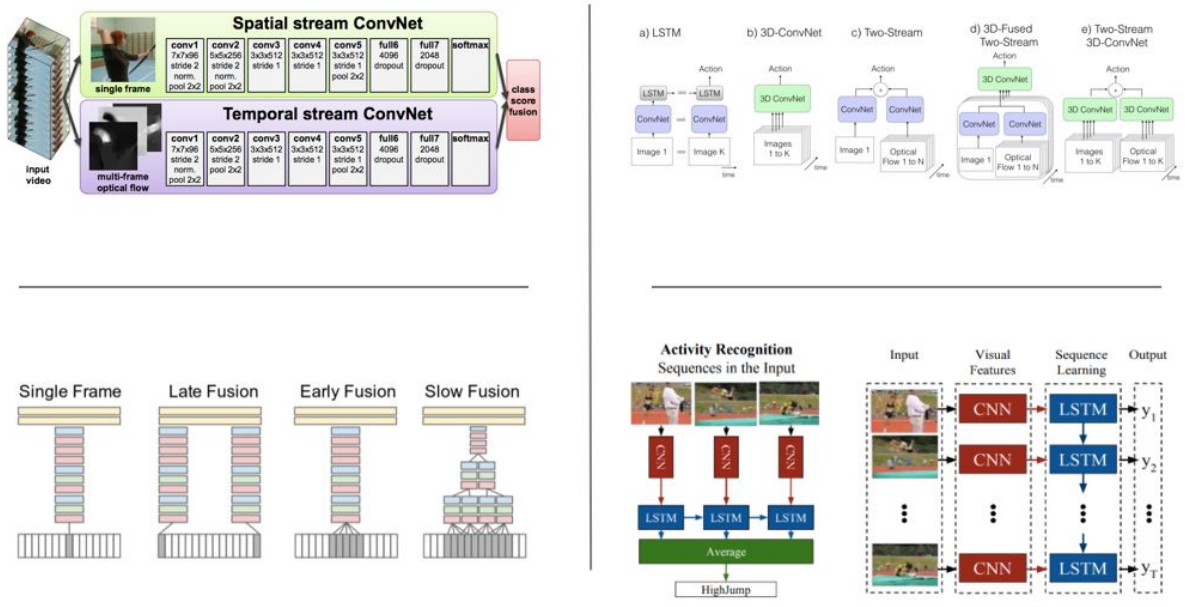


Figure 1: Different action classification architectures. From top clockwise, [1], [2], [3], [4]

2 Data-Set

We will be using the 'HMDB:A Large Video Database for Human Motion Recognition dataset'. It can be downloaded here - HMDB-Dataset. The paper detailing the dataset can be found here - paper.
Download the video database and the three split folder.

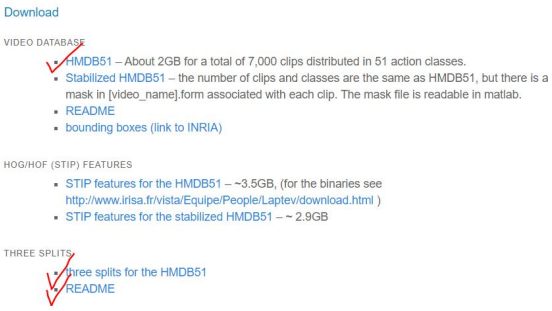


Figure 2: Items to download

In the split folder you should have a total of 153 **[action]_test_split[1-3].txt** files corresponding to all the three splits.
The format of each file is [video_name] [id].

- video is included in the training set if id is 1
- video is included in the testing set if id is 2
- video is not included for training/testing if id is 0

In each text file there are 70 videos with id 1, that is 70 videos for training and 30 videos with id 2, that is 30 videos for testing.
The three splits of the data are generated to ensure that clips from the same video are not used for both training and testing for 3-fold cross validation. In our case we will be using only 1 split - **split 1**.
Since we are using only split 1, only use the videos from **[action]_test_split[1].txt** for training and evaluation.

3 Evaluation

Use only the **split 1** of the dataset, the ids for train, validation and test sets are 1,0,2 respectively. Use train set for training, and validation for evaluation of different models during the training process. Finally report the classification accuracy on the test set which will be the ranking criteria for the leader-board.
Grading (100%) + Extra (10% Novelty)

1. Proposal (20%)
   (a) Include names of your team members
   (b) Literature review
   (c) Propose a baseline (to be delivered on first milestone) and final method plan
   (d) Explain the expected challenges
   (e) Include a timeline
   (f) References
2. Milestone 1 (10%)
   (a) Show the performance of baseline model

(b) Propose and show progress on next steps towards the final model.

3. Final Report (45%)
   (a) Abstract and Introduction [3%]
   (b) Related Work [5%]
   (c) Method [12%]
   (d) Experiments [15%]
      i. Include any observations, challenges and ablation studies. This is also your chance to show the efforts that didn't fetch good results.
      ii. You can optimize the network size, i.e. compare with smaller networks, implement pruning, quantization, etc. This is an important step in practical scenarios and so, you are encouraged to optimize the inference model for size and speed. **5% score out of the 15% is reserved for this.**
   (e) Results, Conclusion and Future Work [10%]
   (f) References
4. Code (10%)
   (a) Document the code well.
   (b) Try to make it modular and readable.
5. Ranking on the leaderboard (5%)
   (a) Race to learn and improve your method, not for numbers.
6. Video (10%)
   (a) Make a video presentation of about 5 minutes explaining your approach, results and learnings.

4 Important Dates

1. **11/09** Final Project Proposal Due
2. **12/02** Milestone 1
3. **12/10** Final Project Code Due
4. **12/17** Final Project Report Due

5 Final Deliverable

Please write a README file to describe how to run your inference code, and zip all of your code into a file for submission. Also submit the report and video presentation.

6 Additional Notes

- Baseline method need not be sophisticated, it can be as simple as you want it to be. It's mainly to get you to understand the format of data, formulate the problem and sense the difficulty of the problem.
- You may refer to existing methods but cannot use them as they are. Do cite any resources used or referred.
- Be ethical and do not use testing data for training. Violators will be heavily penalized.

Final Project Option-5: Open-ended proposal

1 Overview

As this is an open-ended project option, TAs will keep track of your progress and give our help during the milestone meeting. We will assign one TA to assist the team according to your topic, so please keep in touch with your TA to ensure the project is optimal.

The final report should be **4 pages of content** and complete references. The content should include a literature review of approaches for addressing the challenges.

The final report should include a detailed description of the approach you developed and a summary of the experiments you ran, including failed attempts. You need to submit a video presentation of 5 minutes long. Your submitted code should include a README with detailed instructions on how to run the code and reproduce your results.

2 Important Dates

1. **11/09** Final Project Proposal Due
2. **12/02** Milestone 1
3. **12/10** Final Project Code Due (Allow changes until 12/17)
4. **12/17** Final Project Report Due

3 Proposal Requirements (20% Due: Nov 9 23:59 PM)

The Final Project Proposal should be composed of the following sections:

3.1 Names of your team members

3.2 Abstract

Your first deliverable will be a short (300 word maximum) abstract on the project idea you would like to work on for this course. To develop a project idea, we would like you to survey existing applications and literature on your topic. Use your abstract to convince why this could be an exciting topic.

Requirements: In your abstract, you need to provide:

1. Motivation of this project (i.e., why this is a useful/exciting project)
2. Previous approaches and their drawbacks; What's your plan to do differently? How your method solve existing deficiencies?
3. A summary of your potential method, novel points of your method
4. Potential performance you want to achieve

3.3 Related works

List papers of your chosen topic and related topics and a brief comparison and Conclusion of their findings. Why your approach better?

3.4 Dataset Description

A description of datasets you want to use, with some examples of the data (images, labels, masks, or 3d mesh). If there is no exist dataset, what's your plan to generate your own dataset? (Annotation, Unsupervised labeling, etc.)

3.5 Your hypothesis

A description of your hypothesis (i.e. what do you expect to happen?)

3.6 Timeline

A description of the team plan and individual plan of each team member, and results you expected to present or discuss in the milestone meeting and final report.

3.7 Expected Method

Describe your potential method as detailed as possible. It should include possible architectures, main ideas, design of your losses. We understand things will change throughout the process.

3.8 Experiments

Describe the design of metrics to illustrate the performance of your model and ways to prove your hypothesis. Please provide a short explanation of the metrics.

4 Project Submission + Requirements (70%)

Code Due: Dec 10 23:59 PM, video & report Due: Dec 17 23:59 PM. Your final project submission should have three parts, and overall scores:

1. Report(45%)
2. Code(15%)
3. Video(10%)

We understand projects will change from the initial proposal. **You won't be penalized for this**. Please reach out to your TA to get guidance on this.

4.1 Report(45%)

The final project report should be written in LaTeX. The Final Project Report should be composed of the **following sections**:

Abstract: Same requirement as in your proposal, but this time should be based on your final model and the results.

Introduction: In the Introduction section, you should give a high-level overview of the topic that your paper will be exploring.

Related Work: Here you will discuss papers relevant to your group's final project: list papers and summarize their findings. Comparing with related works on your topic, why your approach better? We recommend you list your main contribution by comparing it with previous methods at the end of this section.

Methods: In this section, you will detail your deep learning methods for this final project. Use mathematics symbols and equations to describe your network. It should contain the following parts:

1. An overview figure to illustrate your network and entire computation process
2. Pre-processing details
3. The inputs information to your network and expected outputs
4. Network design and detailed architecture(layers, special designs, parameters, training methods)
5. Loss function

Experiments or Analysis: Design at least two experiments to demonstrate the performance of your network. Compare the performance of your model vs. a baseline method or previous benchmark. Experiments could be quantitative or qualitative. Typical experiment designs include ablation study, metrics analysis, user study, etc. It should contain at least the following parts:

1. Introduction of your metrics and experiments
2. The testing dataset format, samples of the dataset
3. quantitative results of your metrics and a comparison with other benchmarks
4. Conclusion of your experiment results and how it will prove your hypothesis

We understand the performance of your network may be worse than the current state-of-art. We care more about the whole design of your method.

Discussion/Conclusion: Based on your results, what further work do you think should be done? What are other approaches would you try if you had more time? What conclusions do you think you can draw from your results?

References: Be familiar with the bibliography system and use references in your report.

4.2 Code Requirements(15%)

Code submission can either be a jupyter/colab notebook or a python project. We recommend you to write a demo on colab to show your inference results but train it on local GPUs. The code should include a README with detailed instructions on how to run the code and reproduce your results.

4.3 Video Requirements(10%)

The video presentation should be 5 minutes long, introducing your ideas and show your exciting results.

5 Extra points: Novelty (10%)

We encourage students to develop novel ideas for your projects. It is a great chance to do exciting research with help from TAs. However, **do consider the feasibility of finishing the project on time**. Please feel free to contact your TA/Professor for advice.