Chapter 1

Background

1.1 Machine Learning

1.1.1 Introduction

Machine Learning (ML) is a field which is raised out of Artificial Intelligence (AI). Applying AI, we wanted to build better and intelligent machines. But except for mere tasks such as finding the shortest path between point A and B, we were unable to program more complex and constantly evolving challenges. There was a realization that the only way to be able to achieve this task was to let machines learn from themselves. This sounds similar to a child learning from its self. So machine learning was developed as a new capability for computers. And now machine learning is present in so many segments of technology, that we don't even realize it while using it.

Finding patterns in data on planet earth is possible only for human brains. Data being very massive and time taken to compute them made Machine Learning take action, in order to help people exploit them in minimum time.
There are three kinds of Machine Learning Algorithms :

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

Supervised Learning

A majority of practical machine learning uses supervised learning. In supervised learning, the system tries to learn from the previous examples that are given. Speaking mathematically, supervised learning is where you have both input variables (x) and output variables (Y) and can use an algorithm to derive the mapping function from the input to the output. The mapping function is expressed as $Y = f(x)$.

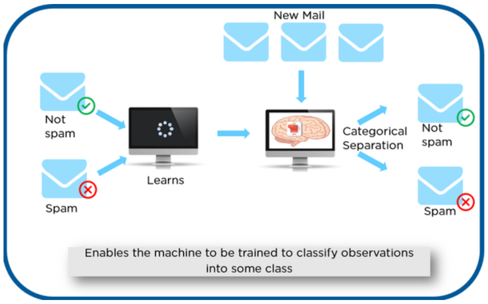

New Mail
Not spam
Spam
Learns
Categorical Separation
Not spam
Spam
Enables the machine to be trained to classify observations into some class

Figure 1.1: Example of supervised Learning

As shown in Figure 1.1, we have initially taken some data and marked them as 'Spam' or 'Not Spam'. This labeled data is used by the training supervised model, in order to train the model. Once it is trained, we can test our model by testing it with some new mails and checking if the model is able to predict the right output.

Supervised learning problems can be further divided into two parts, namely **classification**, and **regression**.

Classification : A classification problem is when the output variable is a category or a group, such as "black" or "white" or "spam" and "no spam".

Regression : A regression problem is when the output variable is a real value, such as "Rupees" or "height."

Some Supervised learning algorithms include:

- Decision trees
- Support-vector machine
- Naive Bayes classifier
- k-nearest neighbors
- linear regression

Unsupervised Learning

In unsupervised learning, the algorithms are left to themselves to discover interesting structures in the data. Mathematically, unsupervised learning is when you only have input data (X) and no corresponding output variables. This is called unsupervised learning because unlike supervised learning above, there are no given correct answers and the machine itself finds the answers. In Figure 1.2, we have given some characters to our model which are 'Ducks' and 'Not Ducks'. In our training data, we don't provide any label to the corresponding data. The unsupervised model is able to separate both the characters by looking

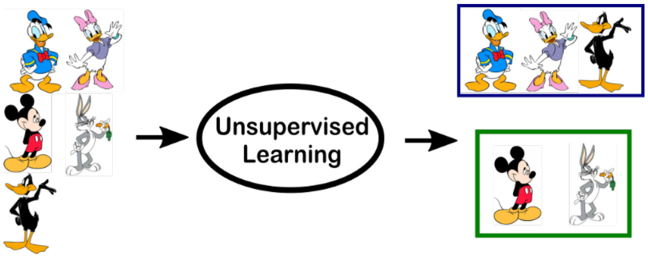


Figure 1.2: Example of unsupervised Learning

at the type of data and models the underlying structure or distribution in the data in order to learn more about it. Unsupervised learning problems can be further divided into **association** and **clustering** problems.

Association : An association rule learning problem is where you want to discover rules that describe large portions of your data, such as "people that buy X also tend to buy Y".

Clustering : A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.

Reinforcement Learning

A computer program will interact with a dynamic environment in which it must perform a particular goal (such as playing a game with an opponent or driving a car). The program is provided feedback in terms of rewards and punishments as it navigates its problem space. Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it continuously trains itself using trial and error method. In Figure 1.3, we can see that the agent is given 2 options i.e. a path with water or a path with fire. A reinforcement algorithm works on reward a system i.e. if

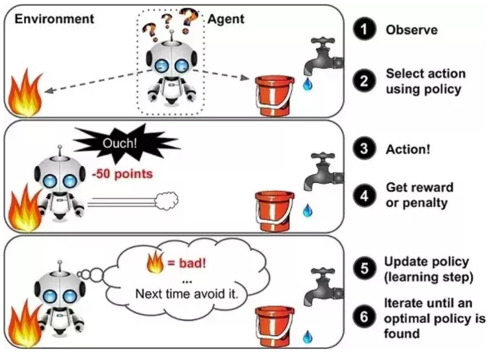

Environment
Agent
1 Observe
2 Select action using policy
Ouch!
-50 points
3 Action!
4 Get reward or penalty
= bad!
...
Next time avoid it.
5 Update policy (learning step)
6 Iterate until an optimal policy is found

Figure 1.3: Example of Reinforcement Learning

the agent uses the fire path then the rewards are subtracted and agent tries to learn that it should avoid the fire path. If it had chosen the water path or the safe path then some points would have been added to the reward points, the agent then would try to learn what path is safe and what path isn't

1.1.2 Neural Networks

Neural Networks are a class of models within the general machine learning literature. Neural networks are a specific set of algorithms that have revolutionized the field of machine learning. They are inspired by biological neural networks and the current so called deep neural networks have proven to work quite very well. Neural Networks are themselves general function approximations, that is why they can be applied to literally almost any machine learning problem where the problem is about learning a complex mapping from the input to the output space.

1.1.3 A single Neuron

The basic unit of computation in a neural network is the neuron, often called a **node** or **unit**. It receives input from some other nodes, or from an external source and computes an output. In purely mathematical terms, a neuron in the machine learning world is a placeholder for a mathematical function, and its only job is to provide an output by applying the function on the inputs provided. Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function *f (defined below)* to the weighted sum of its inputs as shown in Figure 1.4. The network takes numerical inputs *X1* and *X2* and has weights *w1* and *w2* associated with those inputs. Additionally, there is another *input 1* with weight b (called *Bias*) associated with it. The main function of Bias is to provide every

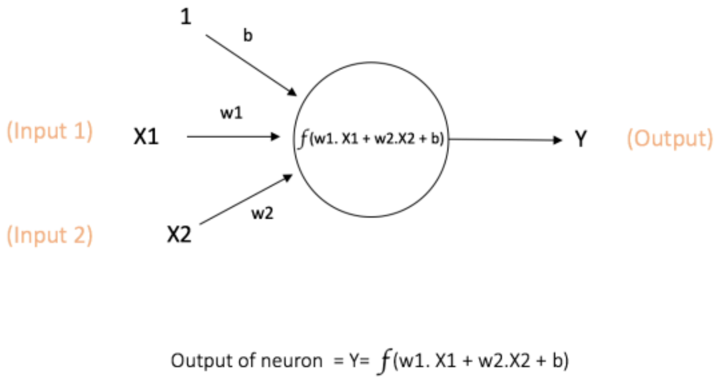


Figure 1.4: An example of a single Neuron

node with a trainable constant value (in addition to the normal inputs that the node receives). The output Y from the neuron is computed as shown in the Figure 1.4. The function f is non-linear and is called **Activation Function**. The purpose of the activation function is to introduce non-linearity into the output of a neuron. This is important because most real world data are non linear and we want neurons to learn these non-linear representations.

Activation Functions

Every activation function (or non-linearity) takes a single number and performs a certain fixed mathematical operation on it. There are several activation functions:

Sigmoid : takes a real-valued input and squashes it to range between 0 and 1. Its formula is:

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

It is easy to understand and apply but it has major reasons which have made it fall out of popularity:

- Vanishing gradient problem
- Its output isn't zero centered. It makes the gradient updates go too far in different directions.
- Sigmoids saturate and kill gradients.
- Sigmoids have slow convergence.

Tanh : takes a real-valued input and squashes it to the range [-1, 1]. Its formula is:

$$tanh(x) = 2\sigma(2x) - 1$$

Now it's output is zero centered because its range in between -1 to 1. Hence optimization is easier in this method and in practice it is always preferred over Sigmoid function . But still it suffers from Vanishing gradient problem.

Re-LU : Re-LU stands for *Rectified Linear Unit.* It takes a real-valued input and thresholds it at zero (replaces negative values with zero). So its formula is:

$$f(x) = max(0, x)$$

It has become very popular in the past couple of years. It was recently proved that it had 6 times improvement in convergence from Tanh function. Seeing the mathematical form of this function we can see that it is very simple and efficient . A lot of times in Machine learning and computer science we notice that most simple and consistent techniques and methods are only preferred and are best. Hence it avoids and rectifies vanishing gradient problem . Almost all deep learning Models use ReLu nowadays.

Figure 1.5 show each of the above activation functions.

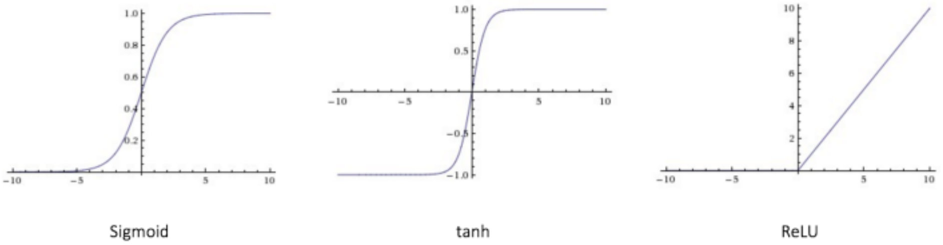


Figure 1.5: Plots of Activation functions

Feed-forward Neural Network

Till now we have covered neuron and activation functions which together for the basic building blocks of any neural network. The feedforward neural network was the first and simplest type of artificial neural network devised. It contains multiple neurons (nodes) arranged in layers. A layer is nothing but a collection of neurons which take in an input and provide an output. Inputs to each of these neurons are processed through the activation functions assigned to the neurons. Nodes from adjacent layers have connections or edges between them. All these connections have weights associated with them. An example of a feedforward neural network is shown in Figure 1.6. A feedforward neural network can consist of three types of nodes:

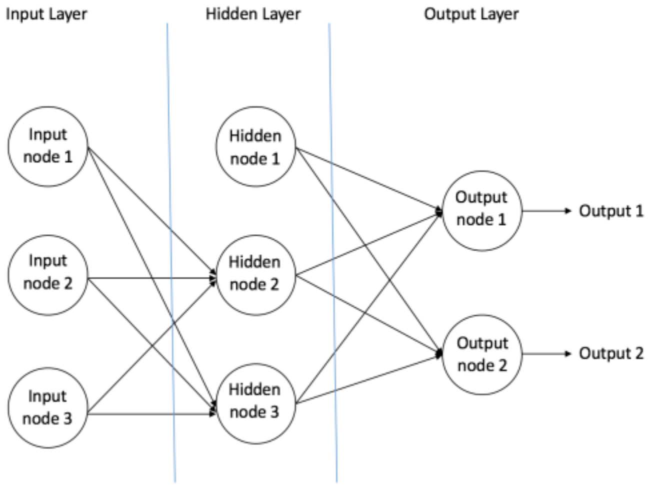


Figure 1.6: An example of a Feedforward Neural Network

Input Nodes The Input nodes provide information from the outside world to the network and are together referred to as the "Input Layer". No computation is performed in any of the Input nodes – they just pass on the information to the hidden nodes.

Hidden Nodes The Hidden nodes have no direct connection with the outside world (hence the name "hidden"). They perform computations and transfer information from the input nodes to the output nodes. A collection of hidden nodes forms a "Hidden Layer". While a feedforward network will only have a single input layer and a single output layer, it can have zero or multiple Hidden Layers.

Output Nodes The Output nodes are collectively referred to as the "Output Layer" and are responsible for computations and transferring information from the network to the outside world.

In a feedforward network, the information moves in only one direction – forward – from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network (this property of feed forward networks is different from Recurrent Neural Networks in which the connections between the nodes form a cycle). Another important point to note here is that each of the hidden layers can have a different activation function, for instance, hidden layer1 may use a sigmoid function and hidden layer2 may use a ReLU, followed by a Tanh in hidden layer3 all in the same neural network. Choice of the activation function to be used again depends on the problem in question and the type of data being used.

1.1.4 2D Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is one of the variants of neural networks used heavily in the field of Computer Vision. It derives its name from the type of hidden layers it consists of. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers, and normalization layers. Here it simply means that instead of using the normal activation functions defined above, convolution and pooling functions are used as activation functions. It can take in an input image, assigning importance (learning weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to the other classification algorithms. While in primitive method filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the structure of the Visual Cortex. However, most ConvNets consist mainly in 2 parts:

- **Feature extractor** :
  This part of the network takes as input the image and extract features that are meaningful for its classification. It amplifies aspects of the input that are important for discrimination and suppresses irrelevant variations.

Usually, the feature extractor consists of several layers. For instance, an image which could be seen as an array of pixel values. The first layer often learns representation that represent the presence or absence of edges at particular orientations and locations in the image. The second layer typically detects motifs by spotting particular arrangements of edges, regardless of small variations in the edge positions. Finally, the third may assemble motifs into larger combinations that correspond to paths of familiar objects, and subsequent layers would detect objects as combinations of these parts.

- **Classifier** :

  This part of the network takes as input the previously computed features and use them to predict the correct label.

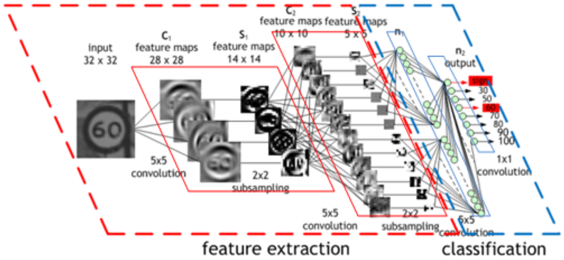


Figure 1.7: Typical structure of a ConvNet

Convolutional Layers In order to extract such features, ConvNets use 2D convolution operations. These operations take place in convolutional layers. Convolutional layers consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of input. During forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position (as Figure 1.8 shows). The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.

Pooling Layers Pooling Layers are also referred as downsampling layers and are used to reduce the spatial dimensions, but not depth, on a convolution neural network. The intuitive reasoning behind this layer is that once we know that a

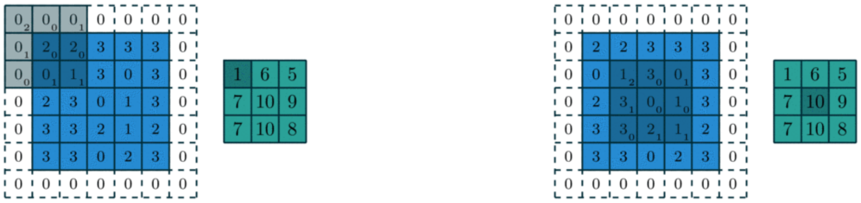


Figure 1.8: Convolution with kernel of 3, stride of 2 and padding of 1

specific feature is in the original input volume (there will be a high activation value), its exact location is not as important as its relative location to the other features. The main advantages of pooling layer are:

- We gain computation performance since the amount of parameters is reduce.
- Less parameters also means we deal with overfitting situations.

The pooling operation is specified, rather than learned. Two common functions used in the pooling operation are:

Average Pooling Calculate the average value for each patch on the feature map.

Maximum Pooling (or Max Pooling) Calculate the maximum value for each patch of the feature map.

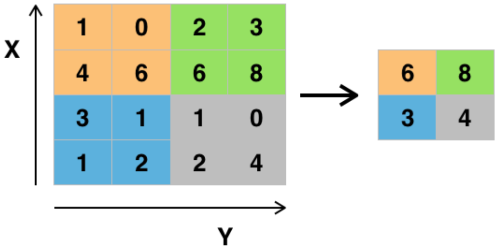


Figure 1.9: Example of Max pooling operation with a 2x2 filter and stride of 2

1.1.5 3D Convolutional Neural Network

Traditionally, ConvNets are targeting RGB images (3 channels). The goal of 3D CNN is to take as input a video and extract features from it. When ConvNets extract the graphical characteristics of a single image and put them in a vector (a low-level representation), 3D ConvNets extract the graphical characteristics of a set of images. 3D CNNs takes in to account a temporal dimension (the order of the images in the video). From a set of images, 3D CNNs find a low-level representation of a set of images, and this representation is useful to find the right label of the video (a given action is performed). In order to extract such features, 3D ConvNets use 3D convolution operations, whose kernel shape for a 3D Convolution is specified along 3 dimensions. When thinking about the convolution operation in terms of a kernel sliding across a multidimensional input array, for a 3D Convolution, the kernel slides in 3 directions Their output shape is a 3 dimentional volume space such as cube or cuboid.

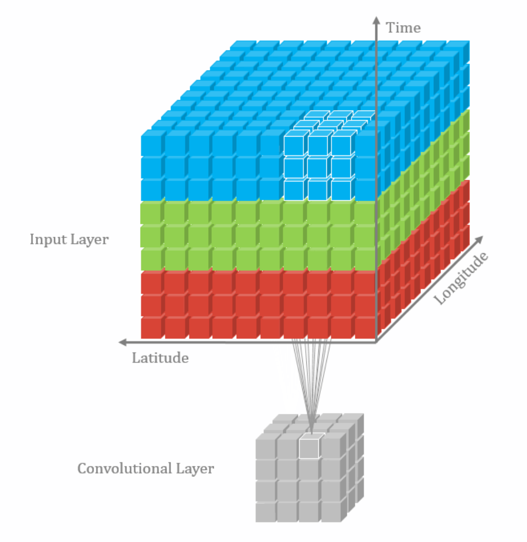


Figure 1.10: 3D Convolution operation

Also, such 3D relationship is important for some applications, such as in 3D segmentations / reconstructions of biomedical imagining, e.g. CT and MRI where objects such as blood vessels meander around in the 3D space.

1.2 Object Detection

Within the field of Deep Learning, the sub-discipline called “Object Detection” involves processes such as identifying the objects through a picture, video or a webcam feed. The challenge of detecting all objects existing in image in counterpart of action localization in videos and a lot of object detection techniques are used in action localization architectures, so it is worth presenting it. Object Detection methods are used almost everywhere these days. The use cases are endless such as Tracking objects, Video surveillance, Pedestrian detection etc. An object detection model is trained to detect the presence and location of multiple classes of objects. For example, a model might be trained with images that contain various pieces of fruit, along with a label that specifies the class of fruit they represent (e.g. an apple, a banana, or a strawberry), and data specifying where each object appears in the image.

The main process followed by most of CNN for Object Detection is:

1. Firstly, we do feature extraction using as backbone network, the first Convolutional Layers of a known pre-trained CNN such as AlexNet, VGG, ResNet etc.
2. Then, we propose regions of interest (ROI) in the image. These regions contain possibly an object, which we are looking for.
3. Finally, we classify each proposed ROI.

1.2.1 Region Proposal Network

From the 3 above steps, the 2nd step is considered to be very important. That is because, in this step, we should choose regions of the image, which will be classified. Poor choice of ROIs means that the CNN will pass by some object that are located in the image, because, they were not be proposed to be classified.

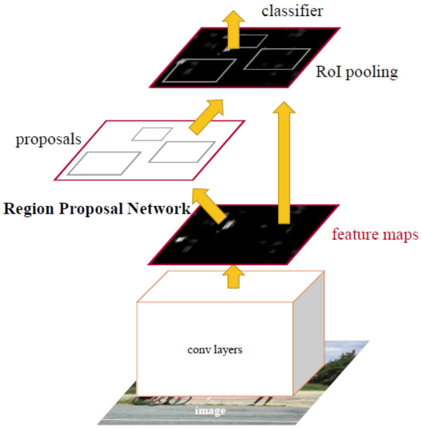


Figure 1.11: Region Proposal Network's structure

The first Object-Detection CNNs use several algorithms for proposing ROIs. For example, R-CNN(Girshick et al. 2013), and Fast R-CNN(Girshick 2015) used Selective Search Algorithm for extracting ROIs. One of novelties introduced by the Faster R-CNN(Ren et al. 2015) is **Region Proposal Network** (RPN). Its function is to propose ROIs and its structure can be shown in 1.11. As we can see, RPN is consisted of:

- 1 2D Convolutional Layer
- 1 score layer
- 1 regression layer

Before describing RPN's function, we introduce another basic element of RPN which is its **anchors**. Anchors are predefined boxes used for extracting ROIs. In figure 1.12 is depicted an example of some anchors

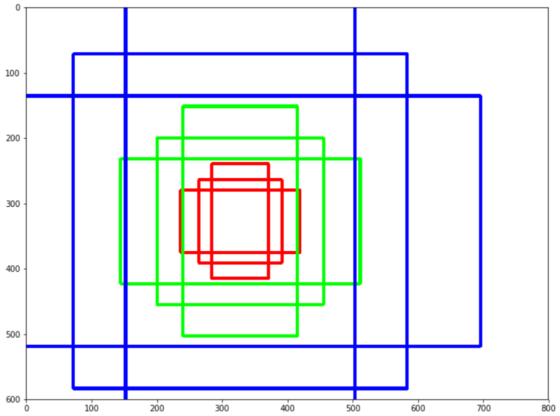


Figure 1.12: Anchors for pixel (320,320) of an image (600,800)

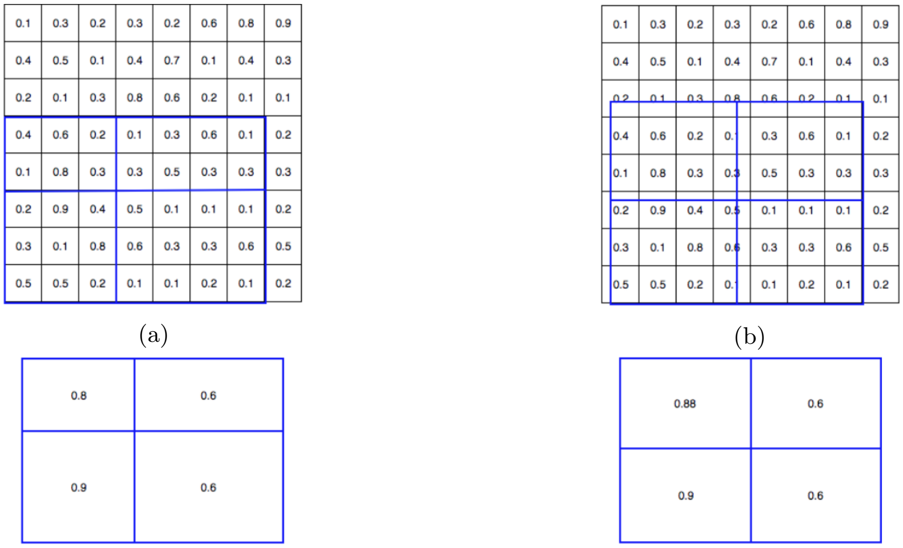
For each feature map's pixel corresponds **k (k=9)** anchors (3 different scales and 3 different ratios 1:1, 1:2, 2:1).

So, RPN's procedure is:

1. RPN gets as input feature maps extracted from the backbone CNN.
2. Then it performs 2D convolution over this input and passes the output to its scoring layer and regression layer.
3. Scoring layer produces confidence score of existing an object in each anchor's area. On the other hand, regression layer outputs 4k displacements, 4 for each anchor. Finally, we keep as output only the *n-best scoring* anchors.

1.2.2 Roi Align

The biggest problem facing Object Detection Networks is the need for fixed input size. Classification networks require a fixed input size, which is easy for image classification because it is handled by resizing the input image. However, in object recognition architectures, each proposal has a different size and shape. This creates the need for converting all proposals to a fixed shape. At Fast-RCNN(Girshick 2015) and Faster-RCNN(Ren et al. 2015) methods, this operation happens by applying Roi Pooling. However, this wrapping is digitalized because the cell boundaries of the target feature map are forced to realign with the boundary of the input feature maps as shown in Figure 1.13a (the top left diagram). As a result, each target cells may not be in the same size (Figure 1.13c).



0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

(a)

0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

(b)

0.8	0.6
0.9	0.6

(c)

0.88	0.6
0.9	0.6

(d)

Figure 1.13: Roi Pooling and Roi Align examples

On the other hand, Mask-RCNN (He et al. 2017b) introduced Roi Align operation. Roi Align avoids digitalizing the boundary of the cells as shown in Figure 1.13b, and achieves to make every target cell to have the same size according to Figure 1.13d. In order to calculate feature maps values, Roi Align uses bi-linear interpolation as shown in Figure 1.14. This means that we calculate the value of the desired bins according to their neighbors'.

1.2.3 Non-maximum suppression (NMS) algorithm

Another problem that object detection networks face is that neighbour bounding boxes have similar scores to some extent. Most object detection systems employ a sliding window or a Region Proposal Network for proposing areas in the image that is likely to contain an object. These techniques, which return several areas in the images, achieve high recall performance. However, in these approaches, more that 1 proposal may be related with only one ground-truth object coordinates. This situation creates the need for choosing the best proposals, because, alternatively, hundreds of unnecessary proposals will be classified. For that reason, Non-Maximum Suppression (NMS) algorithm was proposed for

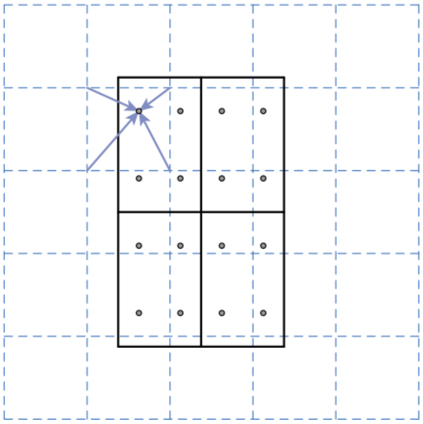


Figure 1.14: Example of bi-linear interpolation for calculation Roi Align's final feature map

filtering these proposals base on some criteria. NMS gets as input a list of proposal bounding boxes B, their corresponding confidence score S and an overlap threshold N and return as output a list of the final filtered proposals D. NMS algorithm's steps are:

1. Initialize an empty list D. Select the proposal with the highest confidence score, remove it from B and add it to D.
2. Calculate the overlap score between this proposal and all the other proposals. For all the proposals that their overlap score is bigger than N, remove from B.
3. From the remaining proposals, picked again the one with the highest score and remove it from B.
4. Repeat steps 2 and 3 until no more proposals are left in list B.

The aforementioned algorithm shows that the whole process depends mostly on a single threshold value. So that makes the selection of threshold a crucial factor for the performance of the model. In some situations, bad choice of the threshold may make the network to remove bounding boxes with good confidence score, if there are side by side. Figure 1.15 shows a situation like this, where red and blue boxes will be removed because of the presence of the black box.

Soft NMS

A simple and efficient way to deal the aforementioned situation is to use Soft-NMS algorithm, which is presented in Bodla et al. 2017. Soft-NMS algorithm is based on the idea of reducing confidence score of the proposals proportional to their overlap score, instead of completely removing them. The score

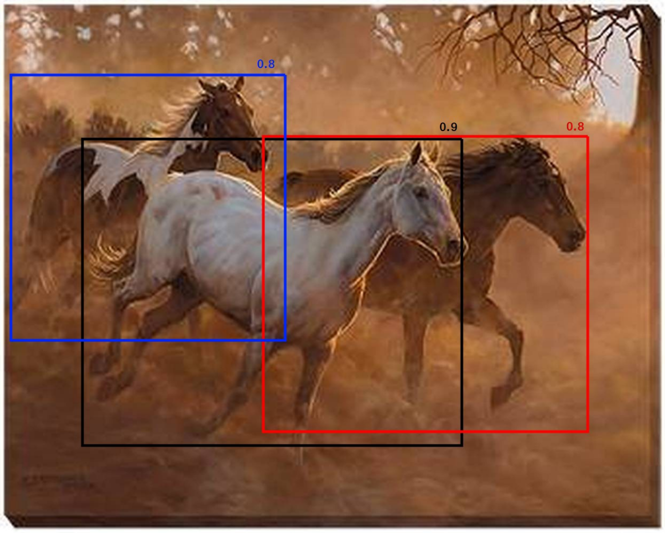


Figure 1.15: Example of a situation where NMS algorithm will remove good proposals

calculation follows the formula

$$s_i = \begin{cases} s_i, & overlapscore(M, b_i) < N_t \\ s_i(1 - overlapscore(M, b_i)), & overlapscore(M, b_i) \geq N_t \end{cases}$$

where s_i is the score of proposal i, b_i is the box coordinates of proposal i, M is the coordinates of the box with the maximum confidence and N_t is the overlap threshold. Let's, again, consider as input a list of proposal bounding-boxes B, their corresponding confidence score S and as output a list of proposals D. Soft-NMS algorithm includes the following steps:

1. Select the proposal with the highest confidence score, remove it from B and add it to D.
2. Calculate the overlap score between this proposal and all the other proposals. For all the proposals that their overlap score is bigger than N, recalculate their confidence score according to previous formula.
3. From the remaining proposals, picked again the one with the highest score and remove it from B.
4. Repeat steps 2 and 3 until no more proposals are left in list B.

1.3 Losses and Metrics

In order to train our model and check its performance, we use some known Loss functions and Metrics used in Object Detection systems.

1.3.1 Losses

For training our network, we use **Cross Entropy Loss** for classification layers and **smooth L1-loss** for bounding box regression in each frame and their diagram is show at Figure 1.16.

Cross Entropy Loss

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1.

Entropy is the measure of uncertainty associated with a given distribution *q(y)* and its formula is:

$$H = -\sum_{i=1}^{n} p_i \cdot \log p_i$$

Intuitively, entropy tells us how “surprised” we are when some event E happened. When we are sure about an event E to happened ($p_E = 1$) we have 0 entropy (we are not surprised) and vise versa.

On top of that, let’s assume that we have 2 distributions, one known (our network’s distribution) *p(y)* and one unknown (the actual data’s distribution) *q(y)*. Cross-entropy tells us how accurate is our known distribution in predicting the unknown distribution’s results. Respectively, Cross-entropy measures how accurate is our model in predicting the test data. Its formula is:

$$H_p(q) = -\sum_{c=1}^{C} q(y_c) \cdot log(p(y_c))$$

Smooth L1-loss

Smooth L1-loss can be interpreted as a combination of L1-loss and L2-loss. It behaves as L1-loss when the absolute value of the argument is high, and it behaves like L2-loss when the absolute value of the argument is close to zero. It is usually used for doing box regression on some object detection systems like Fast-RCNN(Girshick 2015), Faster-RCNN(Ren et al. 2015) and it is less sensitive to outliers according according to Girshick 2015. As shown in Girshick 2015, its formula is:

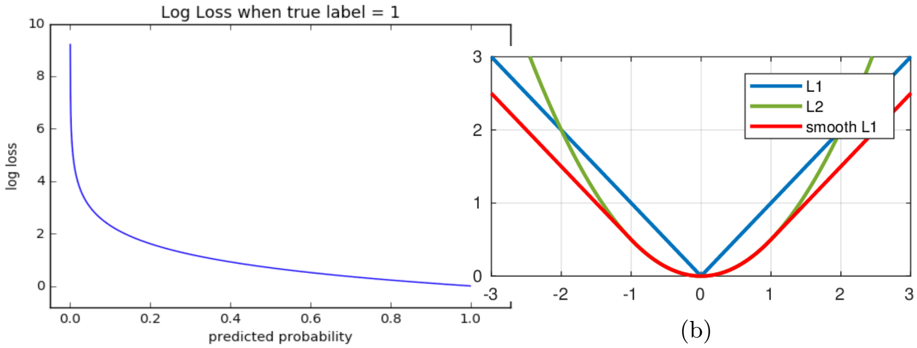
$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } x < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

It is similar to Huber loss whose formula is:

$$L_\delta(x) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$

if we set δ parameter equal 1.

Smooth L1-loss combines the advantages of L1-loss (steady gradients for large values of x) and L2-loss (less oscillations during updates when x is small). Figure 1.16b shows a comparison between L1-norm, L2-norm and smooth-L1 .



(b)

(a)

Figure 1.16: (a) and (b) show the behavior of cross-entropy loss and smooth-L1 respectively.

1.3.2 Metrics

Evaluating our machine learning algorithm is an essential part of any project. The way we choose our metrics influences how the performance of machine learning algorithms is measured and compared. They influence how to weight the importance of different characteristics in the results and finally, the ultimate choice of classification algorithm. Most of the times we use classification accuracy to measure the performance of our model, however it is not enough to truly judge our model.

At first, we introduce some basic evaluation metrics in order,then, to present those we use for our assessment.

Intersection over Union

The first and most important metric that we use is Intersection over Union (IoU). IoU measures the overlap between 2 boundaries. It is usually used in Object Recognition Networks in order to define how good overlap a predicted bounding box with the actual bounding box as shown in Figure 1.17. We predefine an IoU threshold (say 0.5) in classifying whether the prediction is a true positive or a false positive.

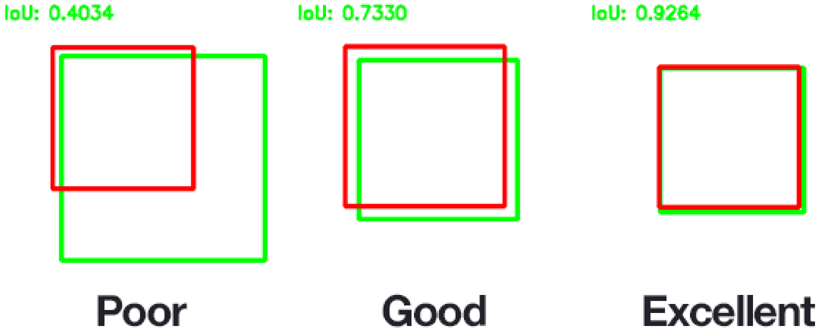


Figure 1.17: Example of IoU scoring policy

Intersection over Union is defined as:

$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$

In Figure 1.17, spatial IoU between 2 bounding boxes, (x_1, y_1, x_2, y_2) and (x_3, y_3, x_4, y_4),is presented, which means IoU metric is implemented for x-y dimensions. Area of overlap and area of union can be defined as:

$$\begin{aligned}\text{Area of overlap } &= \|(min(x_2, x_4) - max(x_1, x_3), min(y_2, y_4) - max(y_1, y_3))\| \\ \text{Area of union } &= \|(max(x_2, x_4) - min(x_1, x_3), max(y_2, y_4) - min(y_1, y_3))\|\end{aligned}$$

On top of that, we can implement IoU for 1 dimension and for 3 dimensions.

1D IoU We can name 1D IoU as temporal overlap. Let's consider 2 temporal segments (t_1, t_2) and (t_3, t_4), between which we want to estimate their overlap score. Their IoU can be described as:

$$\begin{aligned}\text{Length of overlap } &= \|(min(t_2, t_4) - max(t_1, t_3))\| \\ \text{Length of union } &= \|(max(t_2, t_4) - min(t_1, t_3))\|\end{aligned}$$

3D IoU 3-dimensional Intersection over Union which can, also, be named as spatiotemporal IoU, can be defined by 2 ways:

3D boxes are cuboids In this case, 3D boxes can be written as (x, y, z, x', y', z'). So, the IoU overlap between 2 boxes, $(x_1, y_1, z_1, x_2, y_2, z_2)$ and $(x_3, y_3, z_3, x_4, y_4, z_4)$, is defined as:

$$\begin{aligned}\text{Volume of overlap } &= \|(min(x_2, x_4) - max(x_1, x_3), \\ min(y_2, y_4) - max(y_1, y_3), &min(z_2, z_4) - max(z_1, z_3))\| \\ \text{Volume of union } &= \|(max(x_2, x_4) - min(x_1, x_3), \\ max(y_2, y_4) - min(y_1, y_3), &max(z_2, z_4) - min(z_1, z_3)\end{aligned}$$

x-y are continuous and z discrete In this case 3D boxes is defined as a sequence of 2D boxes (x, y, x', y'). For this definition, z-dimension is discrete, and IoU can be defined with 2 ways, which both result in the same overlapping score. Let's consider 2 sequences of boxes, with temporal limits (t_1, t_2) and (t_3, t_4). We calculate their IoU following one of the following methods:

1. IoU is the product between temporal-IoU and the average spatial-IoU between 2D boxes in the overlapping temporal area and it is described as:

$$IoU = IoU((t_1, t_2), (t_3, t_4)) \cdot \frac{1}{K_2 - K_1} \sum_{i=K_1}^{K_2} IoU(X_1^i, X_2^i)$$

   where

   - $K_1 = min(t_2, t_4)$
   - $K_2 = max(t_1, t_3)$
   - $X_1^i = (x_1^i, y_1^i, x_2^i, y_2^i)$ and $X_2^i = (x_3^i, y_3^i, x_4^i, y_4^i)$

2. IoU is the average spatial-IoU if we consider 2D boxes as $(0, 0, 0, 0)$ if $t \notin [t_{start}, t_{finish}]$ and it is written as:

$$IoU = \frac{1}{K} \sum_{i=min(t_1,t_3}^{max(t_2,t_4)} IoU(X_1^i, X_2^i)$$

   - $K = max(t_2, t_4) - min(t_1, t_3)$
   - $X_1 = (x_1, y_1, x_2, y_2)$ if $i \in [t_1, t_2]$ or $(0, 0, 0, 0)$ if $i \notin [t_1, t_2]$
   - $X_2 = (x_3, y_3, x_4, y_4)$ if $i \in [t_3, t_4]$ or $(0, 0, 0, 0)$ if $i \notin [t_3, t_4]$

From above implementations, we are involve mostly with temporal and spatiotemporal IoU.

Precision & Recall

In order to describe **precision** and **recall** metrics, we will use an example. Let's consider a group of people in which, some of them are sick and the others are not. We use a network which, given some data as input, is able to predict if a person is sick or not.

Precision measures how accurate are our model's predictions, i.e. the percentage of predictions that are correct. In our case, how accurate is our model when it predicts that a person is sick.

Recall measures how well we found all the sick people. In our case, how many of the actual sick people we managed to find.

Their definitions are:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

where

- TP = True positive, which means that we predict a person to be sick and he is actually sick.
- TN = True negative, we predict that a person isn't sick and he isn't.
- FP = False positive, we predict a person to be sick but he isn't actually.
- FN = False negative, we predict a person not to be sick but he actually is.

From these 2 metrics we use recall metric in order to evaluate our networks performance, and more specifically, its performance on finding good action tube proposals. We consider a groundtruth action as true positive when there is at least 1 proposed action tube that its IoU overlap score is bigger that a predefined threshold. If there is no such action tube, then we consider this groundtruth action tube as false negative.

mAP

Precision and recall are single-value metrics based on the whole list of predictions. By looking their formulas, we can see that there is a trade-off between precision and recall performance. This trade-off can be adjusted by the softmax threshold, used in model's final layer. In order to have high precision performance, we need to decrease the number of FP. But this will lead to decrease recall performance and vice-versa.

As a result, these metrics fail to determine if a model is performing well in object detection tasks as well as action detection tasks. For that reason, we use mean Average Precision (mAP) metric, which for videos is named video-AP as introduced by Gkioxari and Malik 2014.

AP (Average precision) Before defining mAP metric, we will define Average Precision metric (AP). AP is a popular metric in measuring the accuracy of object detectors like Faster R-CNN, SSD, etc. Average precision computes the average precision value for recall value over 0 to 1. As mentioned before, during classification stage, our prediction results in True positive(TP), False positive(FP), True Negative(TN) or False Negative(FN). For object recognition and action localization networks, we don't care about TN. We consider a prediction as True positive when our prediction (an bounding box for object detection network or a sequence of bounding boxes in action localization networks) overlaps with the groundtruth bounding box/action tube over a predefined threshold, and our predicted class is the same as groundtruth's. In addition, we consider a False Negative when either no detection overlaps with the groundtruth bounding box/action tube or our prediction's class label was incorrect. We consider a prediction as False positive, when more that one predictions overlap with the groundtruth. In this situation, we consider the prediction with the biggest confidence score as TP and the rest as FP.

For a class, we need to calculate, first, its precision and recall scores in order to calculate its AP score. We sort our predictions according to their confidence score and for each new prediction we calculate precision and recall values. An example, for a class containing 4 TP and 8 predictions is shown at Table 1.1. Precision and recall are calculated according to the number of elements that are above in the order. So, for rank #3, Precision is calculated as the proportion of TP = 2/3 = 0.67 and Recall as the proportion of TP out of all the possible TP = 2/4 = 0.5.

Rank	Prediction	**Precision**	**Recall**
1	Correct	1.0	0.25
2	Correct	1.0	0.5
3	False	0.67	0.5
4	False	0.5	0.5
5	False	0.4	0.5
6	Correct	0.5	0.75
7	False	0.42	0.75
8	Correct	0.5	1

Table 1.1: Ordered by confidence predictions and their precision and recall values

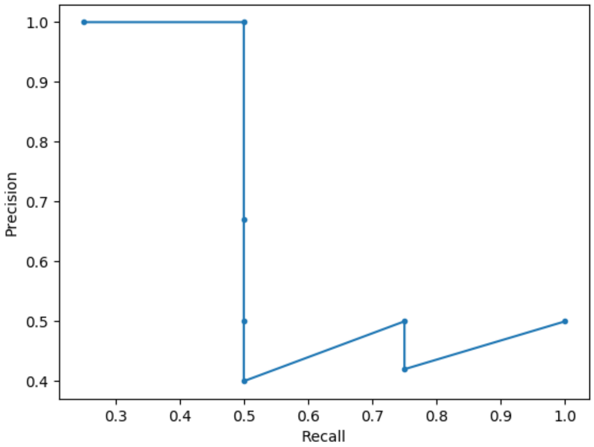


Figure 1.18: Precision/Recall curve

We plot Precision against Recall and their curve is shown in Figure 1.18. The general definition for Average Precision(AP) is finding the area under the precision-recall curve and its formula is:

$$AP = \int_0^1 p(r)dr$$

Precision and Recall values $\in [0, 1]$, so AP $\in [0, 1]$, too. This integral can be replaced with a finite, as we have a finite number of predictions. So its formula is:

$$AP = \sum_{k=1}^{n} P(k)\Delta r(k)$$

where $P(k)$ is the precision until prediction k and Δr is the change in recall from $k - 1$ to k.

Interpolated Precision As we can see at Figure 1.18, P-R curve has a zigzag pattern as it goes down with false predictions, and goes up with correct. So, before calculation AP, we need to smooth out this zigzag pattern using Interpolated precision, as introduced in Everingham et al. 2010. Interpolated precision is calculated at each recall level r by taking the maximum precision measured for that r and it is defined as:

$$p_{interp}(r) = \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r})$$

where $p(\tilde{r})$ is the measured precision at recall $\tilde{r}$. Graphically, at each recall level, we replace each precision value with the maximum precision value to the right of that recall level. At Figure 1.19 are shown both P-R curves. The previous P-R curve has blue color and the interpolated has red color.

In order to calculate AP, we sample the curve at all unique recalls values, whenever the maximum precision drops. On top of that, we define mean Average Precision (mAP) the mean of the AP for each class. So, AP and mAP are defined as

$$AP = \sum (rn + 1 - r_n) p_{interp}(r_{n+1})$$

$$p_{interp}(r_{n+1}) = \max_{\tilde{r} \geq r_{n+1} p(\tilde{r})} p(\tilde{r})$$

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i$$

Mean Average Best Overlap - MABO

In order to evaluate the quality of our proposals, both during TPN and connecting tube stages, recall metric isn't enough. That's because recall metric tells us only for how many actual objects/action tubes, there was at least 1 proposal that satisfied the detection criterion. However, it doesn't tells us how

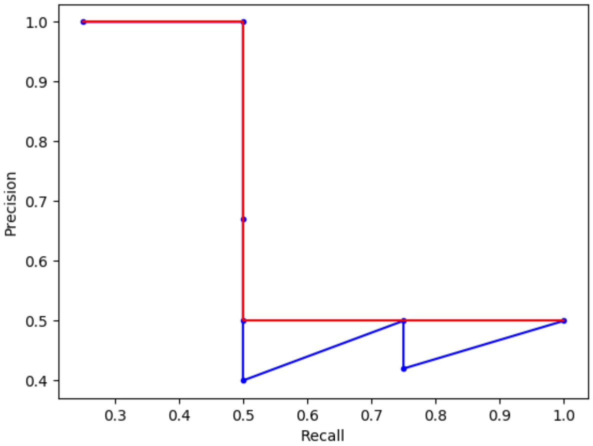


Figure 1.19: Both P-R curves. Interpolated P-R curve has red col-our.

close these proposals are to the actual objects/action tubes. In order to quantify this performance, Mean Average Best Overlap (MABO) was introduced by Winschel, Lienhart, and Eggert 2016. The importance of MABO can be clarified we consider figure 1.20.

As we can see, recall performance is almost perfect, but, MABO perfonmance, which tells us where most proposal overlap scores are gathered, is just fine and not perfect.

In order to define MABO, we need first to define Average Best Overlap. Let $c \in C$ denote a class c from the set of all classes C and G^c the set of ground truth annotations of this class in all images; let L be the set of all generated object proposals for all images. Average Best Overlap is defined as the average value of the maximum overlap score, (in our situation, we use intersection over union) of L with each groundtruth annotation $g \in G^c$. The Mean Average Best Overlap (MABO) is defined as the average value of all class ABO values :

$$MABO = \frac{1}{|C|}\sum_{c \in C}[\frac{1}{|G^C|}\sum_{g \in G^c}\max_{l \in L} IoU(g,l)]$$

In our situation, which we care only for the quality of our proposals, we consider only 1 class, foreground class. As a result, MABO's performance identifies with ABO's.

1.4 Related work

In this section, we present some of the most relevant methods to our work and others studied for designing this approach. These methods are divided into two sections: *action recognition* and *action localization*. The first part refers

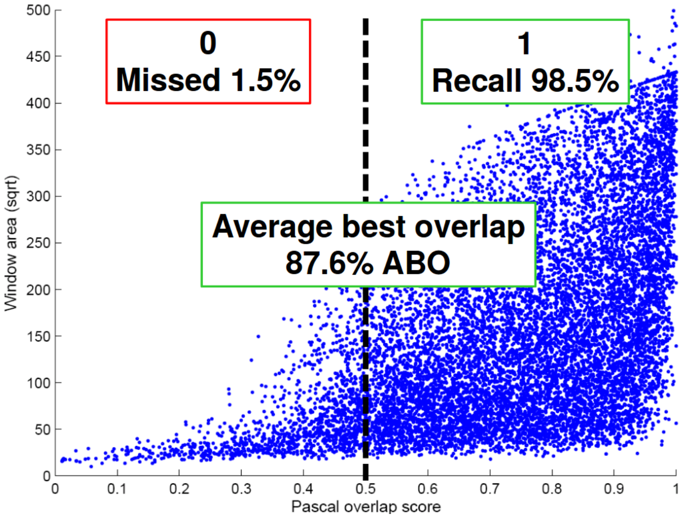


Figure 1.20: Recall versus MABO example

to classic action classification methods introduced until recently and the second part, respectively, to recent action localization methods.

1.4.1 Action Recognition

First approaches for action classification consisted of two steps a) compute complex handcrafted features from raw video frames such as SIFT, HOG, ORB features and b) train a classifier based on those features. These features can be separated into 3 categories: 1) space-time volume approaches, 2) trajectories and 3) space-time features. For space-time volume methods the approach is as follows. Based on the training videos, the system contracts a 3D space-time model, by concatenating 2D images (x-y dimension) along time (t or z dimension), in order to represent each action. When system is given an unlabeled video, it constructs a 3D space-time volume corresponding to this video. This new 3D volume, then, is compared with each activity model to measure the similarity in shape and appearance between these two volumes. The system extracts the class label of the unknown video by corresponding to the action with the highest similarity. Furthermore, there are several variations of space-time representations. Instead of volume representation, the system may represent the action as trajectories in space-time dimensions or even more, the action can be represented as a set of features extracted from the volume or the trajectories. Pure space-time volume representations include methods of comparing foreground regions of a person (i.e. silhouettes) like Bobick and Davis 2001 did, comparing volumes in terms of their patches like Shechtman and Irani 2005. Ke, Sukthankar, and Hebert 2007 method uses oversegmented volumes, automatically calculating a set of 3-D XYT volume segments that corresponds to a moving human. Rodriguez, Ahmed, and Shah 2008 proposed filters for capturing volume's characteristics, in order to match them more reliably and efficiently. From the other hand, trajectory-based approaches include representing an action as a set of 13 joint trajectories (Sheikh, Sheikh, and Shah 2005) or using a set of $XYZT$-dimensions joint trajectories obtained from moving cameras (Yilmaz and Shah 2005). Finally, several methods use local features extracted from 3-dimensional space-time volumes, like extracting local features at every frame and concatenate them temporally (Chomat and Crowley 1999; Zelnik-Manor and Irani 2001; Blank et al. 2005, extracting sparse spatio-temporal local interest points from 3D volumes (Laptev and Lindeberg 2003; Dollar et al. 2005; Niebles, Wang, and Li 2006; Alper Yilmaz and Mubarak Shah 2005; Ryoo and Aggarwal 2006) These approaches made the choice of features a significant factor for network's performance. That's because different action classes may appear dramatically different in terms of their appearance and motion patterns. Another problem was that most of those approaches make assumptions about the circumstances under which the video was taken due to problems such as cluttered background, camera viewpoint variations etc. A review of the techniques, used until 2011, is presented in Aggarwal and Ryoo 2011.

Recent results in deep architectures and especially in image classification motivated researchers to train CNN networks for the task of action recognition. The first significant attempt was made by Karpathy et al. 2014. They design their architecture based on the best-scoring CNN in the ImageNet competition. they explore several methods for fusion of spatio-temporal features using 2D operations mostly and 3D convolution only in slow fusion. Simonyan and Zisserman 2014 used a 2 CNNs, one for spatial information and one for optical flow and combined them using late fusion. They show that extracting spatial context from videos and motion context from optical flow can improve significantly action recognition accuracy. Feichtenhofer, Pinz, and Zisserman 2016 extend this approach by using early fusion at the end of convolutional layers, instead of late fusion which takes places at the last layer of the network. On top that, they used a second network for temporal context which they fuse with the other network using late fusion. Furthermore, Wang et al. 2016 based their method on Simonyan and Zisserman 2014, too. They deal with the problem of capturing long-range temporal context and training their network given limited training samples. Their approach, which they named Temporal Segment Network (TSN), separates the input video in K segments and a short snippet from each segment is chosen for analysis. Then they fuse the extracted spatio-temporal context, making, eventually, their prediction. Most recently, Zhang et al. 2016 and Zhu et al. 2017 used two-stream approach, too. Zhang et al. 2016 replace optical flow with motion vector which can be obtained directly from compressed videos without extra calculation and feeding it to . Zhu et al. 2017 trained a CNN for calculating optical flow, calling it MotionNet and use a temporal stream cnn for project motion information to action labels. Finally, they use late fusion through the weighted averaging of the prediction scores of the temporal and spatial streams. On the other hand, a novel approach was introduced by Girdhar and Ramanan 2017 incorporating attention maps to give significant improvement in action recognition performance

Some other methods included a RNN or LSTM network for classification like Donahue et al. 2014, Ng et al. 2015 and Ma et al. 2017. Donahue et al. 2014 address the challenge of variable lengths of input and output sequences, exploiting convolutional layers and long-range temporal recursions. They propose a Long-term Recurrent Convolutional Network (LRCN) which is capable of dealing with the tasks of action Recognition, image caption and video description. In order to classify a given sequence of frames, LRCN firstly gets as input a frame, and in particular its RGB channels and optical flow and predicts a class label. After that, it extracts video class by averaging label probabilities, choosing the most probable class. Ng et al. 2015 firstly explore several approaches for temporal feature pooling. These techniques include handling video frames individually by 2 CNN architectures: either AlexNet or GoogleNet, and consisted of early fusion, late fusion of a combination between them. Furthermore, they propose a recurrent neural Network architecture in order to consider video clips as a sequences of CNN activations. Proposed LSTM takes an input the output of the final CNN layer at each consequentive video frame and after five stacked LSTM layers using a Softmax classifier, it proposes a class label. For video classification, they return a label after last time step, max-pool the predictions over time, sum predictions over time and return the max or linearly weight the predictions over time by a factor g, sum them and return the max. They showed that all approaches are 1% different with a bias for using weighting predictions for supporting the idea that LSTM becomes progressively more informed. Last but not least, Ma et al. 2017 use a two-stream ConvNet for feature extraction and either a LSTM or convolutional layers over temporally-constructed feature matrices, for fusing spatial and temporal information. They use a ResNet-101 for extracting feature maps for both spatial and temporal streams. They divide video frames in several segments like Wang et al. 2016 did, and use a temporal pooling layer to extract distinguished features. Taken these features, LSTM extracts embedded features from all segments.

Additionally, Tran et al. 2014 explored 3D Convolutional Networks (Ji, Yang, and Yu 2013) and introduced C3D network which has 3D convolutional layers with kernels $3 \times 3 \times 3$. This network is able to model appearence and motion context simultaneously using 3D convolutions and it can be used as a feature extractor, too. Combining Two-stream architecture and 3D Convolutions, Carreira and Zisserman 2017 proposed I3D network. On top of that, the authors emphasize in the advantages of transfer learning for the task of action recognition by repeating 2D pre-trained weights in the 3rd dimension. Hara, Kataoka, and Satoh 2017 proposed a 3D ResNet Network for action recognition based on Residual Networks (ResNet) (He et al. 2015) and explore the effectiveness of ResNet with 3D Convolutional kernels. On the other hand, Diba et al. 2017 based their approach on DenseNets(Huang, Liu, and Weinberger 2016) and extend DenseNet architecture by using 3D filters and pooling kernels instead of 2D, naming this approach as DenseNet3D. Futhermore, they introduce Temporal Transition Layer (TTL), which concatenates temporal feature-maps extracted at different temporal depth ranges and replaces DenseNet's transition layer. On top of that Diba et al. 2018 introduced a new temporal layer that models variable temporal Convolution kernel depths. Last but not least, Tran et al. 2017 experiment with several residual network architectures using combinations of 2D and 3D convolutional layer. Their purpose is to show that a 2D spatial convolution followed by a 1D temporal convolution achieves state of the art classification performance, naming this type of convolution layer as R(2+1)D. Recently Guo et al. 2018 proposed a framework which can learn to recognize a previous unseen 3D action class with only a few examples by exploiting the inherent structure of 3D data through a graphical representation. A more detailed presentation for Action Recognition techniques used until 2018 is included in Kong and Fu 2018.

1.4.2 Action Localization

As mentioned before, Action Localization can be seen as an extention of the object detection problem. Instead of outputing 2D bounding boxes in a single image, the goal of action localization systems is to output action tubes which are sequences of bounding boxes that contain an performed action. So, there are several approaches including an object-detector network for single frame action proposal and a classifier.

First object detection approaches included extending a object proposal algorithm into 3-dimensions. Tian, Sukthankar, and Shah 2013 extended deformable part models (Felzenszwalb et al. 2010) by treating actions as spatiotemporal patterns and generate a deformable part for each action. Jain et al. 2014 introduced the concept of tubelets, aka sequences of bounding boxes and based their method on selective search algorithm (Uijlings et al. 2013), extending super-pixels to super-voxels for producing spatio-temporal shapes. On the other hand, Oneata et al. 2014 extend a randomized superpixel merging procedure which was used for object proposals as presented by Manen, Guillaumin, and Gool 2013. Yu and Yuan 2015 first propose bounding boxes for each frame using a human and motion detector and then by picking the best-scoring bounding boxes, they proposed a greedy linking algorithm by formulating liking task as a maximum set coverage problem. Gemert et al. 2015 generate spatiotemporal proposals directly from dense trajectories, which also used for classification. Chen and Corso 2015 create a spatio-temporal trajectory graph and select action proposals based only on intentional movement extracted from the graph. Soomro, Idrees, and Shah 2015 seperate the video segments into supervoxels and use their context as a spatial relation between supervoxels relative to foreground action. They create a graph for each video, where supervoxels form thenodes and directed edges capture the spatial relations between them.During testing, they perform a context walk where each step is guided by the context relations learned during training, resulting in a probability distribution of an action over all the supervoxels. Mettes, Gemert, and Snoek 2016, instead of annotating boxes in all frames, annotate points on a sparse subset of video frames, and use proposals obtained by an overlap measure between action proposals and points. Behl et al. 2017 deal with online action detection and localization by getting per-frame action proposal and proposing a linking algorithm which is able to construct and update action tubes at each frame.

The introduction of R-CNN (Girshick et al. 2013) achieve significant improvement in the performance of Object Detection Networks. This architecture, firstly, proposes regions in the image which are likely to contain an object and then it classifies them using a SVM classifier. Inspired by this architecture, Gkioxari and Malik 2014 design a 2-stream RCNN network in order to generate action proposals for each frame, one stream for frame level and one for optical flow. Then they connect them using the viterbi connection algorithm. Weinzaepfel, Harchaoui, and Schmid 2015 extend this approach, by performing frame-level proposals and using a tracker for connecting those proposals using both spatial and optical flow features. Also their method performs temporal localization using a sliding window over the tracked tubes. Most recently, Soomro and Shah 2017 tried to deal with the problem of unsupervised action detection and localization. Their approach included extracting supervoxel segmentation and then assigning a weight to each supervoxel. Using extracted supervoxels, they create a graph and then using a discriminate clustering approach a classifier is trained.

The introduction of Faster RCNN (Ren et al. 2015) contribute a lot to the improvement of the performance of Actiol Localization Networks Peng and Schmid 2016, Saha et al. 2016 and use Faster R-CNN instead of RCNN for frame-level proposals, using RPN for both RGB and optical flow images. After getting spatial and motion proposals,Peng and Schmid 2016 fuse them exploring and from each proposed ROI, generate 4 ROIs in order to focus in specific body parts of the actor. After that, they connect the proposal using Viterbi algorithm for each class and perform temporal localization by using a sliding window, with multiple temporal scales and stride using a maximum subarray method. From the other hand, Saha et al. 2016 perform, too, frame-level classification. After that, their method performs fusion based on a combination between the actioness scores of the appearence and motion based proposals and their overlap score. Finally, temporal localization takes place using dynamic programming. On the other hand, Weinzaepfel, Martin, and Schmid 2016 use Faster RCNN for extracting human tubes from videos focusing on weakly-supervised action localization problem. Then using dense trajectories and a multi-fold Multiple Instance Learning approach (Cinbis, Verbeek, and Schmid 2016) train a classifier. Mettes and Snoek 2017 introduced a method for zero-shot action localization. Their approach includes scoring proposed action tubes according to the interactions between actors and local objects. They used Faster-RCNN, in the first step, for detecting both actors and objects and then using spatial relations between them they link the proposed boxes over time based on zero-shot likelihood from the presence of actors, relevant objects around the actors and the expected spatial relations between objects and actors. Furthermore, He et al. 2017a proposed the Tube Proposal Network (TPN) for generating generic class-independent tubelet proposals, which uses Faster-RCNN for getting 2D region proposals and a linking algorithm for linking tubelets with these region proposals. Most recently, Girdhar et al. 2018 proposed a method for action Localization on the AVA dataset (Gu et al. 2017) combining I3D (Carreira and Zisserman 2017) and Faster-RCNN architectures. They use I3D blocks for getting video representation and Fast-RCNN's RPN for generation "person" proposals for the center frame.

On top of that, Singh et al. 2017 and Kalogeiton et al. 2017 design their networks based on the Single Shot Multibox Detector Liu et al. 2015). Singh et al. 2017 created an online real-time spatio-temporal network. In order their network to execute real-time, Singh et al. 2017 propose a novel and efficient algorithm by adding boxes in tubes in every frame if they overlap more than a threshold, or alternatively, terminate the action tube if for k-frames no box was added. Kalogeiton et al. 2017 designed a two-stream network, which they called ACT-detector, and introduced anchor cuboids. For K frames, for both networks, Kalogeiton et al. 2017 extract spatial features in frame-level, then they stack these features. Finally, using cuboid anchors, the network extracts tubelets, that is a sequence of boxes, with their corresponding classification scores and regression targets. For linking the tubelets, Kalogeiton et al. 2017 follow about the same steps as Singh et al. 2017 did. For temporal localization, they use a temporal smoothing approach.

Most recently, YOLO Network (Redmon et al. 2015) became the inspiration for Hu et al. 2019 and El-Nouby and Taylor 2018. In Hu et al. 2019, concepts of progression and progress rate were introduced. Except from proposing bounding boxes in frame level, they use YOLO together with a RNN classifier for extracting temporal information for the proposals. Based on this information, they create action tubes, seperated into classes. Some other approaches include pose estimation like Luvizon, Picard, and Tabia 2018 do. They proposed a method for calcualatin 2D and 3D poses and then they performed action classification. They use the diffferentiable Soft-argamax function for estimating 2D and 3D joints, because argmax function is not differentiable. Then, for T adjacent poses, they create an image representation with time and N_j joins as $x-y$ dimensions and having 2 channes for 2D poses and 3 channels for 3D poses. They use Convolutional Layers in order to produce action heats and then using max plus min pooling and a Softmax activation they perform action classification. Zolfaghari et al. 2017 proposed a three-stream architecture which includes 2D pose, optical flow and RGB information. These streams are integrated sequentially via a Markov chain model. In addition, Zhu, Vial, and Lu 2017 proposed an architecture using a temporal convolutional regression network, for capturin the long-term dependency and contexts among adjacent frames, and a spatial regression network, getting per-frame proposals. They use tracking methods and dynamic programming for generating action proposals.

Most of aforementioned networks use per-frame spatial proposals and extract their temporal infomation by calculating optical flow. On the other hand, Saha, Singh, and Cuzzolin 2017 and Hou, Chen, and Shah 2017 design an architecture which icludes proposal in video segment level, handling more that 1 frame simultaneously. Saha, Singh, and Cuzzolin 2017 proposed a 3D-RPN which is able to generate and classify 3D region proposals consisted of two successive frames. Also, they proposed a linking algorithm, modifying the one proposed by Saha et al. 2016. On top of that, Hou, Chen, and Shah 2017 design an architecture for generating action proposals for more than 2 frame, which they called Tube CNN (T-CNN). In their approach, video segment level means that the whole video is seperated into equal length video clips, and using a C3D for extracting features, it returns spatio-temporal proposals. After getting proposals, Hou, Chen, and Shah 2017 link the tube proposals by an algorithm based on tubes' actioness score and overlap. Finally, classification operation is performed for the linked video proposals.

Bibliography

[1] O. Chomat and J. L. Crowley. “Probabilistic recognition of activity using local appearance”. In: *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149).* Vol. 2. 1999, 104–109 Vol. 2. DOI: 10.1109/CVPR.1999.784616.

[2] Aaron Bobick and J.W. Davis. “The recognition of human movement using temporal templates”. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 23 (Apr. 2001), pp. 257 –267. DOI: 10.1109/34.910878.

[3] L. Zelnik-Manor and M. Irani. “Event-based analysis of video”. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.* Vol. 2. 2001, pp. II–II. DOI: 10.1109/CVPR.2001.990935.

[4] Laptev and Lindeberg. “Space-time interest points”. In: *Proceedings Ninth IEEE International Conference on Computer Vision.* 2003, 432–439 vol.1. DOI: 10.1109/ICCV.2003.1238378.

[5] Alper Yilmaz and Mubarak Shah. “Actions sketch: a novel action representation”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05).* Vol. 1. 2005, 984–989 vol. 1. DOI: 10.1109/CVPR.2005.58.

[6] M. Blank et al. “Actions as space-time shapes”. In: *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1.* Vol. 2. 2005, 1395–1402 Vol. 2. DOI: 10.1109/ICCV.2005.28.

[7] P. Dollar et al. “Behavior recognition via sparse spatio-temporal features”. In: *2005 IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance.* 2005, pp. 65–72. DOI: 10.1109/VSPETS.2005.1570899.

[8] E. Shechtman and M. Irani. “Space-time behavior based correlation”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05).* Vol. 1. 2005, 405–412 vol. 1. DOI: 10.1109/CVPR.2005.328.

[9] Y. Sheikh, M. Sheikh, and M. Shah. “Exploring the space of a human action”. In: *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1.* Vol. 1. 2005, 144–149 Vol. 1. DOI: 10.1109/ICCV.2005.90.

[10] A. Yilmaz and M. Shah. “Recognizing human actions in videos acquired by uncalibrated moving cameras”. In: *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1.* Vol. 1. 2005, 150–157 Vol. 1. DOI: 10.1109/ICCV.2005.201.

[11] Juan Carlos Niebles, Hongcheng Wang, and Fei Fei Li. “Unsupervised Learning of Human Action Categories Using Spatial-Temporal Words.” In: vol. 79. Sept. 2006, pp. 1249–1258.

[12] M.S. Ryoo and J.K. Aggarwal. “Semantic Understanding of Continued and Recursive Human Activities”. In: Jan. 2006, pp. 379 –378. DOI: 10.1109/ICPR.2006.1043.

[13] Y. Ke, R. Sukthankar, and M. Hebert. “Spatio-temporal Shape and Flow Correlation for Action Recognition”. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition.* 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383512.

[14] M. D. Rodriguez, J. Ahmed, and M. Shah. “Action MACH a spatio-temporal Maximum Average Correlation Height filter for action recognition”. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition.* 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587727.

[15] Mark Everingham et al. “The Pascal Visual Object Classes (VOC) Challenge”. In: *Int. J. Comput. Vision* 88.2 (June 2010), pp. 303–338. ISSN: 0920-5691. DOI: 10.1007/s11263-009-0275-4. URL: http://dx.doi.org/10.1007/s11263-009-0275-4.

[16] P. F. Felzenszwalb et al. “Object Detection with Discriminatively Trained Part-Based Models”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.9 (2010), pp. 1627–1645. DOI: 10.1109/TPAMI.2009.167.

[17] J.K. Aggarwal and M.S. Ryoo. “Human Activity Analysis: A Review”. In: *ACM Comput. Surv.* 43.3 (Apr. 2011), 16:1–16:43. ISSN: 0360-0300. DOI: 10.1145/1922649.1922653. URL: http://doi.acm.org/10.1145/1922649.1922653.

[18] Ross B. Girshick et al. “Rich feature hierarchies for accurate object detection and semantic segmentation”. In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[19] Shuiwang Ji, Ming Yang, and Kai Yu. “3D convolutional neural networks for human action recognition.” In: *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2013), pp. 221–31.

[20] Santiago Manen, Matthieu Guillaumin, and Luc Van Gool. "Prime Object Proposals with Randomized Prim's Algorithm". In: *Proceedings of the 2013 IEEE International Conference on Computer Vision*. ICCV '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 2536–2543. ISBN: 978-1-4799-2840-8. DOI: 10.1109/ICCV.2013.315. URL: http://dx.doi.org/10.1109/ICCV.2013.315.

[21] Y. Tian, R. Sukthankar, and M. Shah. "Spatiotemporal Deformable Part Models for Action Detection". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 2642–2649. DOI: 10.1109/CVPR.2013.341.

[22] J.R.R. Uijlings et al. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* (2013). DOI: 10.1007/s11263-013-0620-5. URL: http://www.huppelen.nl/publications/selectiveSearchDraft.pdf.

[23] Jeff Donahue et al. "Long-term Recurrent Convolutional Networks for Visual Recognition and Description". In: *CoRR* abs/1411.4389 (2014). arXiv: 1411.4389. URL: http://arxiv.org/abs/1411.4389.

[24] Georgia Gkioxari and Jitendra Malik. "Finding Action Tubes". In: *CoRR* abs/1411.6031 (2014). arXiv: 1411.6031. URL: http://arxiv.org/abs/1411.6031.

[25] M. Jain et al. "Action Localization with Tubelets from Motion". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 740–747. DOI: 10.1109/CVPR.2014.100.

[26] A. Karpathy et al. "Large-Scale Video Classification with Convolutional Neural Networks". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1725–1732. DOI: 10.1109/CVPR.2014.223.

[27] Dan Oneata et al. "Spatio-Temporal Object Detection Proposals". In: Sept. 2014. DOI: 10.1007/978-3-319-10578-9_48.

[28] Karen Simonyan and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos". In: *Advances in Neural Information Processing Systems*. 2014, pp. 568–576.

[29] Du Tran et al. "Learning Spatiotemporal Features with 3D Convolutional Networks". In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2014), pp. 4489–4497.

[30] W. Chen and J. J. Corso. "Action Detection by Implicit Intentional Motion Clustering". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 3298–3306. DOI: 10.1109/ICCV.2015.377.

[31] Jan C. van Gemert et al. "APT: Action localization proposals from dense trajectories". In: *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, 2015, pp. 177.1–177.12. ISBN: 1-901725-53-7. DOI: 10.5244/C.29.177. URL: https://dx.doi.org/10.5244/C.29.177.

[32] Ross Girshick. "Fast R-CNN". In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV).* ICCV '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 1440–1448. ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.169. URL: http://dx.doi.org/10.1109/ICCV.2015.169.

[33] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[34] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325. URL: http://arxiv.org/abs/1512.02325.

[35] Joe Yue-Hei Ng et al. "Beyond Short Snippets: Deep Networks for Video Classification". In: *CoRR* abs/1503.08909 (2015). arXiv: 1503.08909. URL: http://arxiv.org/abs/1503.08909.

[36] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[37] Shaoqing Ren et al. "Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1.* NIPS'15. Montreal, Canada: MIT Press, 2015, pp. 91–99. URL: http://dl.acm.org/citation.cfm?id=2969239.2969250.

[38] K. Soomro, H. Idrees, and M. Shah. "Action Localization in Videos through Context Walk". In: *2015 IEEE International Conference on Computer Vision (ICCV).* 2015, pp. 3280–3288. DOI: 10.1109/ICCV.2015.375.

[39] Philippe Weinzaepfel, Zaïd Harchaoui, and Cordelia Schmid. "Learning to track for spatio-temporal action localization". In: *CoRR* abs/1506.01929 (2015). arXiv: 1506.01929. URL: http://arxiv.org/abs/1506.01929.

[40] G. Yu and J. Yuan. "Fast action proposals for human action detection and search". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2015, pp. 1302–1311. DOI: 10.1109/CVPR.2015.7298735.

[41] R. G. Cinbis, J. Verbeek, and C. Schmid. "Weakly Supervised Object Localization with Multi-Fold Multiple Instance Learning". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.1 (2016), pp. 189–203. DOI: 10.1109/TPAMI.2016.2535231.

[42] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. "Convolutional Two-Stream Network Fusion for Video Action Recognition". In: *CoRR* abs/1604.06573 (2016). arXiv: 1604.06573. URL: http://arxiv.org/abs/1604.06573.

[43] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[44] Pascal Mettes, Jan C. van Gemert, and Cees G. M. Snoek. “Spot On: Action Localization from Pointly-Supervised Proposals”. In: *CoRR* abs/1604.07602 (2016). arXiv: 1604.07602. URL: http://arxiv.org/abs/1604.07602.

[45] Xiaojiang Peng and Cordelia Schmid. “Multi-region two-stream R-CNN for action detection”. In: *ECCV - European Conference on Computer Vision*. Vol. 9908. Lecture Notes in Computer Science. Amsterdam, Netherlands: Springer, Oct. 2016, pp. 744–759. DOI: 10.1007/978-3-319-46493-0_45. URL: https://hal.inria.fr/hal-01349107.

[46] Suman Saha et al. “Deep Learning for Detecting Multiple Space-Time Action Tubes in Videos”. In: *CoRR* abs/1608.01529 (2016). arXiv: 1608.01529. URL: http://arxiv.org/abs/1608.01529.

[47] Limin Wang et al. “Temporal Segment Networks: Towards Good Practices for Deep Action Recognition”. In: *CoRR* abs/1608.00859 (2016). arXiv: 1608.00859. URL: http://arxiv.org/abs/1608.00859.

[48] Philippe Weinzaepfel, Xavier Martin, and Cordelia Schmid. “Towards Weakly-Supervised Action Localization”. In: *CoRR* abs/1605.05197 (2016). arXiv: 1605.05197. URL: http://arxiv.org/abs/1605.05197.

[49] Anton Winschel, Rainer Lienhart, and Christian Eggert. “Diversity in Object Proposals”. In: *CoRR* abs/1603.04308 (2016). arXiv: 1603.04308. URL: http://arxiv.org/abs/1603.04308.

[50] Bowen Zhang et al. “Real-time Action Recognition with Enhanced Motion Vector CNNs”. In: *CoRR* abs/1604.07669 (2016). arXiv: 1604.07669. URL: http://arxiv.org/abs/1604.07669.

[51] Harkirat S. Behl et al. “Incremental Tube Construction for Human Action Detection”. In: *CoRR* abs/1704.01358 (2017). arXiv: 1704.01358. URL: http://arxiv.org/abs/1704.01358.

[52] Navaneeth Bodla et al. “Improving Object Detection With One Line of Code”. In: *CoRR* abs/1704.04503 (2017). arXiv: 1704.04503. URL: http://arxiv.org/abs/1704.04503.

[53] João Carreira and Andrew Zisserman. “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset”. In: *CoRR* abs/1705.07750 (2017). arXiv: 1705.07750. URL: http://arxiv.org/abs/1705.07750.

[54] Ali Diba et al. “Temporal 3D ConvNets: New Architecture and Transfer Learning for Video Classification”. In: *CoRR* abs/1711.08200 (2017). arXiv: 1711.08200. URL: http://arxiv.org/abs/1711.08200.

[55] Rohit Girdhar and Deva Ramanan. “Attentional Pooling for Action Recognition”. In: *CoRR* abs/1711.01467 (2017). arXiv: 1711.01467. URL: http://arxiv.org/abs/1711.01467.

[56] Chunhui Gu et al. “AVA: A Video Dataset of Spatio-temporally Localized Atomic Visual Actions”. In: *CoRR* abs/1705.08421 (2017). arXiv: 1705.08421. URL: http://arxiv.org/abs/1705.08421.

[57] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. “Learning Spatio-Temporal Features with 3D Residual Networks for Action Recognition”. In: *CoRR* abs/1708.07632 (2017). arXiv: `1708.07632`. URL: `http://arxiv.org/abs/1708.07632`.

[58] Jiawei He et al. “Generic Tubelet Proposals for Action Localization”. In: *CoRR* abs/1705.10861 (2017). arXiv: `1705.10861`. URL: `http://arxiv.org/abs/1705.10861`.

[59] Kaiming He et al. “Mask R-CNN”. In: *CoRR* abs/1703.06870 (2017). arXiv: `1703.06870`. URL: `http://arxiv.org/abs/1703.06870`.

[60] Rui Hou, Chen Chen, and Mubarak Shah. “Tube Convolutional Neural Network (T-CNN) for Action Detection in Videos”. In: *CoRR* abs/1703.10664 (2017). arXiv: `1703.10664`. URL: `http://arxiv.org/abs/1703.10664`.

[61] Vicky Kalogeiton et al. “Action Tubelet Detector for Spatio-Temporal Action Localization”. In: *ICCV 2017 - IEEE International Conference on Computer Vision*. Venice, Italy, Oct. 2017.

[62] Chih-Yao Ma et al. “TS-LSTM and Temporal-Inception: Exploiting Spatiotemporal Dynamics for Activity Recognition”. In: *CoRR* abs/1703.10667 (2017). arXiv: `1703.10667`. URL: `http://arxiv.org/abs/1703.10667`.

[63] Pascal Mettes and Cees G. M. Snoek. “Spatial-Aware Object Embeddings for Zero-Shot Localization and Classification of Actions”. In: *CoRR* abs/1707.09145 (2017). arXiv: `1707.09145`. URL: `http://arxiv.org/abs/1707.09145`.

[64] Suman Saha, Gurkirt Singh, and Fabio Cuzzolin. “AMTnet: Action-Micro-Tube regression by end-to-end trainable deep architecture”. In: *CoRR* abs/1704.04952 (2017). arXiv: `1704.04952`. URL: `http://arxiv.org/abs/1704.04952`.

[65] Gurkirt Singh et al. “Online Real time Multiple Spatiotemporal Action Localisation and Prediction”. In: 2017.

[66] K. Soomro and M. Shah. “Unsupervised Action Discovery and Localization in Videos”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 696–705. DOI: `10.1109/ICCV.2017.82`.

[67] Du Tran et al. “A Closer Look at Spatiotemporal Convolutions for Action Recognition”. In: *CoRR* abs/1711.11248 (2017). arXiv: `1711.11248`. URL: `http://arxiv.org/abs/1711.11248`.

[68] H. Zhu, R. Vial, and S. Lu. “TORNADO: A Spatio-Temporal Convolutional Regression Network for Video Action Proposal”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 5814–5822. DOI: `10.1109/ICCV.2017.619`.

[69] Yi Zhu et al. “Hidden Two-Stream Convolutional Networks for Action Recognition”. In: *CoRR* abs/1704.00389 (2017). arXiv: `1704.00389`. URL: `http://arxiv.org/abs/1704.00389`.

[70] Mohammadreza Zolfaghari et al. "Chained Multi-stream Networks Exploiting Pose, Motion, and Appearance for Action Classification and Detection". In: *CoRR* abs/1704.00616 (2017). arXiv: 1704.00616. URL: http://arxiv.org/abs/1704.00616.

[71] Ali Diba et al. "Temporal 3D ConvNets Using Temporal Transition Layer". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2018, Salt Lake City, UT, USA, June 18-22, 2018*. 2018, pp. 1117–1121. URL: http://openaccess.thecvf.com/content_cvpr_2018_workshops/w19/html/Diba_Temporal_3D_ConvNets_CVPR_2018_paper.html.

[72] Alaaeldin El-Nouby and Graham W. Taylor. "Real-Time End-to-End Action Detection with Two-Stream Networks". In: *CoRR* abs/1802.08362 (2018). arXiv: 1802.08362. URL: http://arxiv.org/abs/1802.08362.

[73] Rohit Girdhar et al. "A Better Baseline for AVA". In: *CoRR* abs/1807.10066 (2018). arXiv: 1807.10066. URL: http://arxiv.org/abs/1807.10066.

[74] Michelle Guo et al. "Neural Graph Matching Networks for Fewshot 3D Action Recognition". In: *The European Conference on Computer Vision (ECCV)*. 2018.

[75] Yu Kong and Yun Fu. "Human Action Recognition and Prediction: A Survey". In: *CoRR* abs/1806.11230 (2018). arXiv: 1806.11230. URL: http://arxiv.org/abs/1806.11230.

[76] Diogo C. Luvizon, David Picard, and Hedi Tabia. "2D/3D Pose Estimation and Action Recognition using Multitask Deep Learning". In: *CoRR* abs/1802.09232 (2018). arXiv: 1802.09232. URL: http://arxiv.org/abs/1802.09232.

[77] Bo Hu et al. "Progress Regression RNN for Online Spatial-Temporal Action Localization in Unconstrained Videos". In: *CoRR* abs/1903.00304 (2019). arXiv: 1903.00304. URL: http://arxiv.org/abs/1903.00304.