CIS 522 - University Of Pennsylvania, Spring 2020

Lecture 7: CNN Architectures and Applications

4 February 2020

Lecturer: Konrad Kording *Scribe: Kushagra, Dewang*

1 CNN Architectures

1.1 Neocognitron – Fukushima

Neocognitron was an early attempt at modelling the convolution in the visual cortex and was proposed by Kunihiko Fukushima in 1979. It was inspired from the Hubel Wiesel model which had 2 types of cells in the visual primary cortex : simple cell and complex cell. Neocognitron similarly has S-Cells and C-Cells. S-Cells are local feature extractors where as C-Cells handle local shift. This further motivated the development of Convolutional Neural Networks.

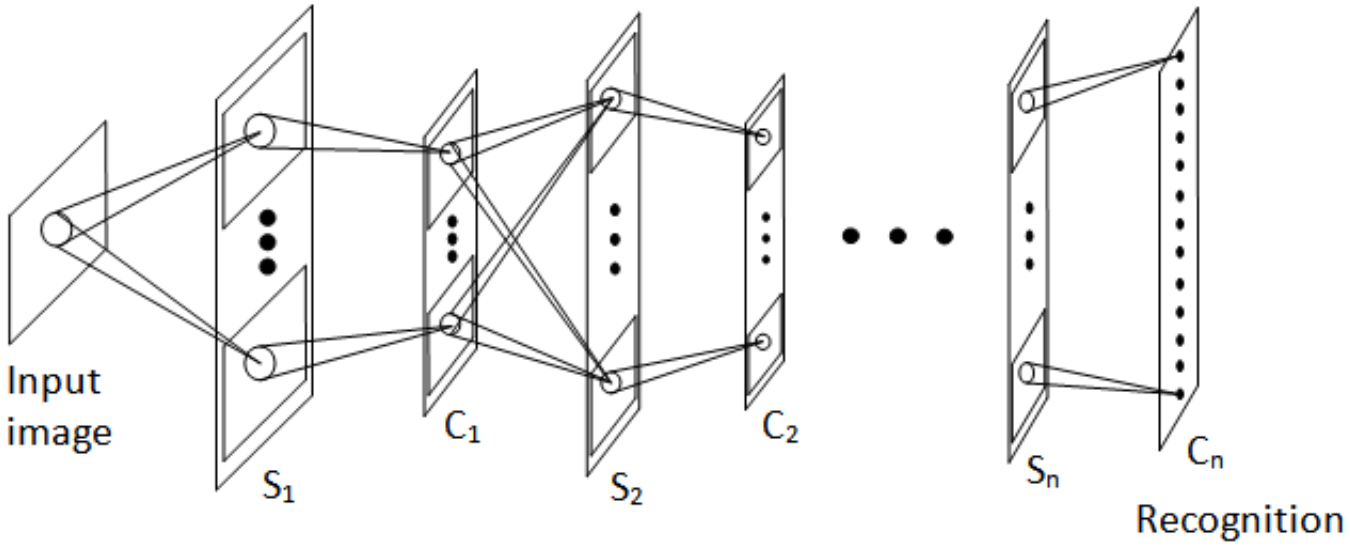

Figure 1: Neocognitron

1.2 LeNet

LeNet is one of the first successful applications of Convolutional neural networks developed by Yann LeCun in the 1990's. It was used to read zip codes, digits, etc. A LeNet consists of two parts

- A block of convolutional layers.
- A block of fully connected layers.

The basic units in the convolutional block are convolutional layer and a subsequent average pooling layer. The convolutional layer is used to recognize the spatial patterns in the image, such as lines and the parts of objects, and the subsequent average pooling layer is used to reduce the dimensionality. The convolutional layer block is composed of repeated stacks of these two basic units. Each convolutional layer uses a 5×5 kernel and processes each output with a sigmoid activation function.The first convolutional layer has 6 output channels, and second convolutional layer increases channel depth further to 16.However, coinciding with this increase in the number of channels, the height and width are shrunk considerably. Therefore, increasing the number of output channels makes the parameter sizes of the two convolutional layers similar. The two average pooling layers are of size 2×2 and take stride 2 (note that this means they are non-overlapping). In other words, the pooling layer downsamples the representation to be precisely one quarter the pre-pooling size.

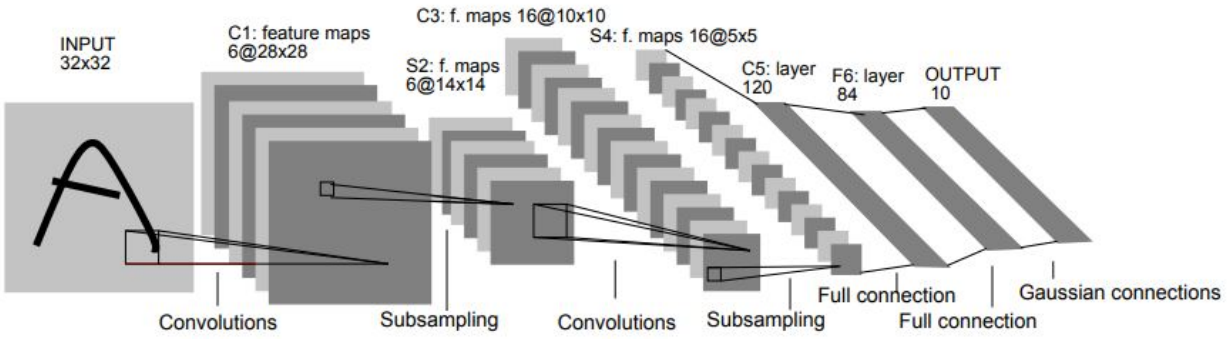

Figure 2: LeNet-5 Architecture

The convolutional block emits an output with size given by (batch size, channel, height, width). Before we can pass the convolutional block's output to the fully-connected block, we must flatten each example in the minibatch. In other words, we take this 4D input and transform it into the 2D input expected by fully-connected layers: as a reminder, the first dimension indexes the examples in the minibatch and the second gives the flat vector representation of each example. LeNet's fully-connected layer block has three fully-connected layers, with 120, 84, and 10 outputs, respectively. Because we are still performing classification, the 10 dimensional output layer corresponds to the number of possible output classes.

Figure 3: Compressed Notation for LeNet

1.3 AlexNet

The first work that popularized Convolutional Networks in Computer Vision was the AlexNet, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. The AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% compared to runner-up with 26% error).This network proved, for the first time, that the features obtained by learning can transcend manually-design features, breaking the previous paradigm in computer vision.The design philosophies of AlexNet and LeNet are very similar, but there are also significant differences. First, AlexNet is much deeper than the comparatively small LeNet5. AlexNet consists of eight layers: five convolutional layers, two fully-connected hidden layers, and one fully-connected output layer. Second, AlexNet used the ReLU instead of the sigmoid as its activation function.

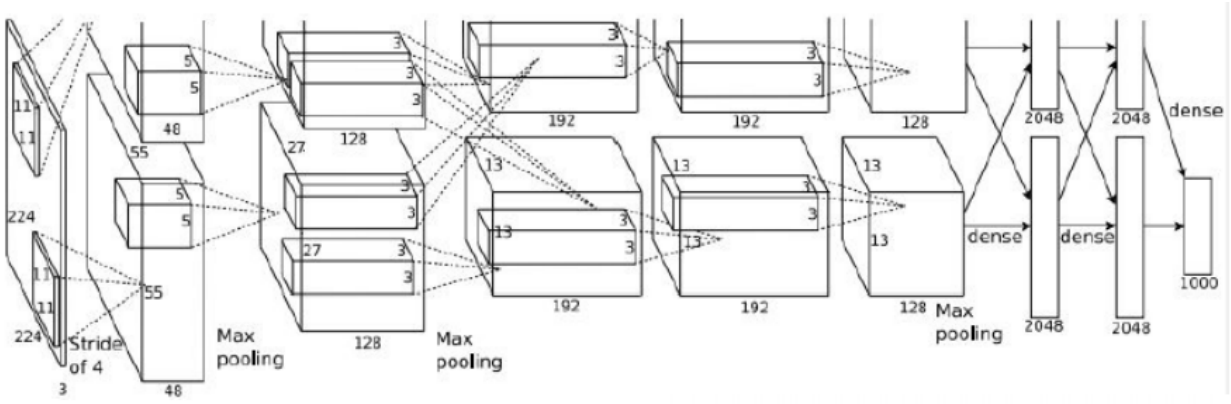

Figure 4: AlexNet Architecture

AlexNet contains eight layers; the first five are convolutional layers, some of them followed

by max-pooling layers, and the last three are fully connected layers. It uses the non-saturating ReLU activation function, which showed improved training performance over tanh and sigmoid.In AlexNet's first layer, the convolution window shape is 11×11 . Since most images in ImageNet are more than ten times higher and wider than the MNIST images, objects in ImageNet data tend to occupy more pixels. Consequently, a larger convolution window is needed to capture the object. The convolution window shape in the second layer is reduced to 5×5 , followed by 3×3 . In addition, after the first, second, and fifth convolutional layers, the network adds maximum pooling layers with a window shape of 3×3 and a stride of 2. Moreover, AlexNet has ten times more convolution channels than LeNet. After the last convolutional layer are two fully-connected layers with 4096 outputs. These two huge fully-connected layers produce model parameters of nearly 1 GB. Due to the limited memory in early GPUs, the original AlexNet used a dual data stream design, so that each of their two GPUs could be responsible for storing and computing only its half of the model.

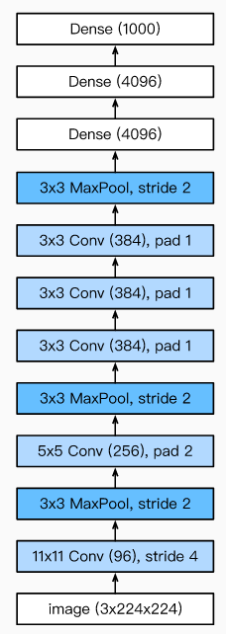

Figure 5: Compressed AlexNet Architecture

1.4 VGG

Alexnet proved that deep convolutional networks can give good results but it did not offer a general template to guide people in designing new networks. The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. Its main contribution was in showing that the depth of the network is a critical component for good performance. Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end. A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M). Most of these parameters are in the first fully connected layer, and it was since found that these FC layers can be removed with no performance downgrade, significantly reducing the number of necessary parameters.

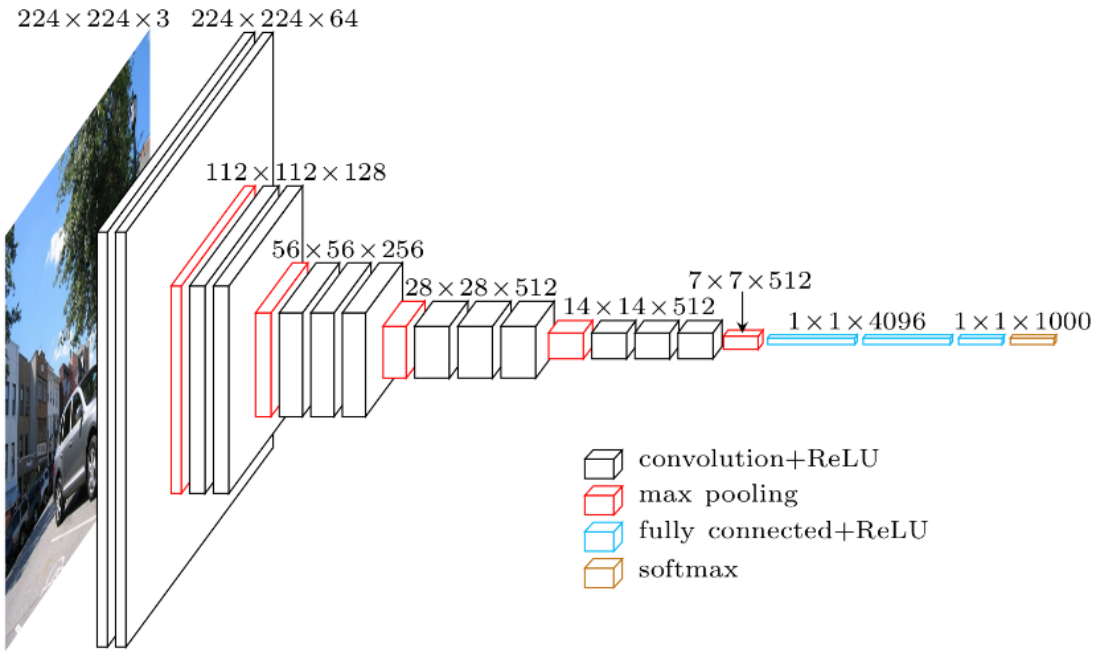

Figure 6: VGGNet Architecture

1.5 Inception/GoogleLeNet

1.5.1 Inception V1

Inception[8] was created in order to address several problems that single sized kernels had:

1. Salient parts in the image can have extremely large variation in size. That means that same object can occupy the whole image (example human portraits) or occupy tiny part in the image (example drone image of a crowd). Because of this, choosing the right kernel size becomes difficult as large kernels are good at capturing information globally while smaller kernels can capture information at more granular level.

2. Secondly, very deep networks are prone to overfitting and its hard to pass gradient updates through the entire network.

3. Naively stacking large convolution operations is computationally expensive.

The solution proposed by inception net was to essentially to use multiple sizes at the same level and thereby increasing the width of the network rather than the depth.

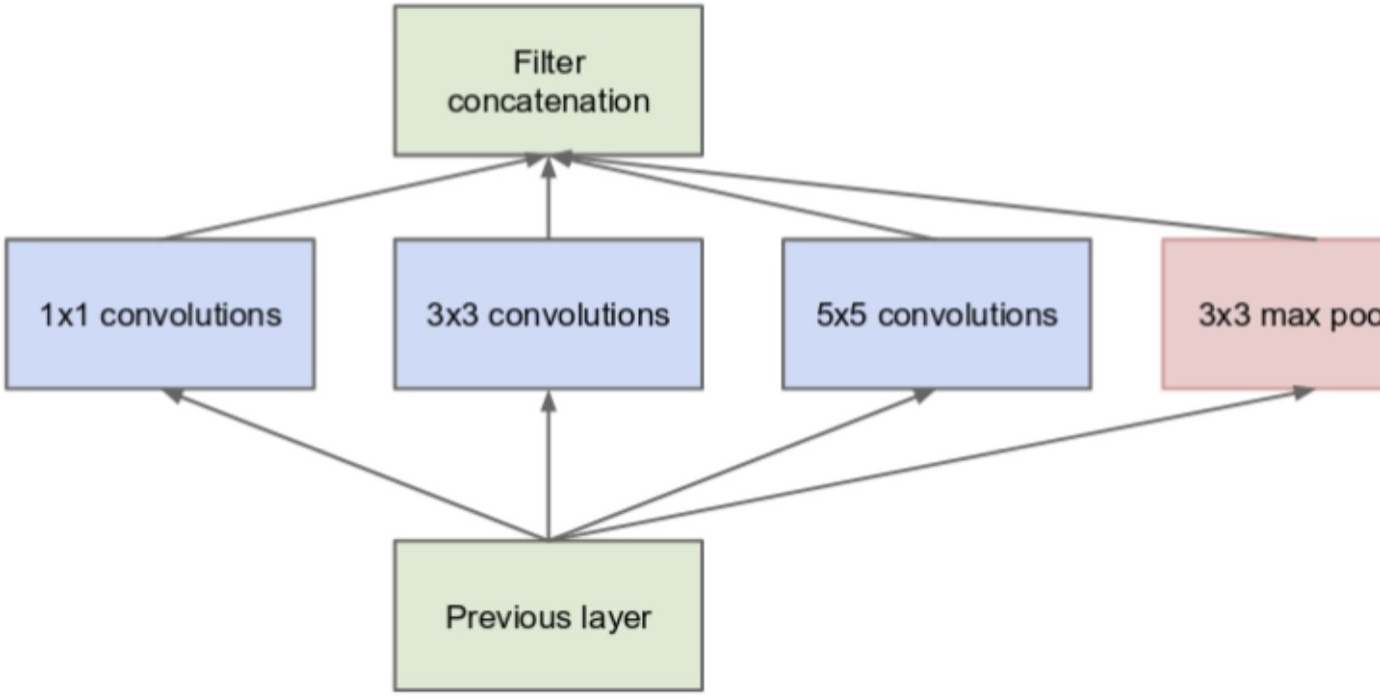

Figure 7: Inception Module, naive version

Since the input to these modules may have arbitrarily large number of channels, the authors limit the number of input channels to the main module by adding an extra 1×1 convolution before the bigger convolutions.

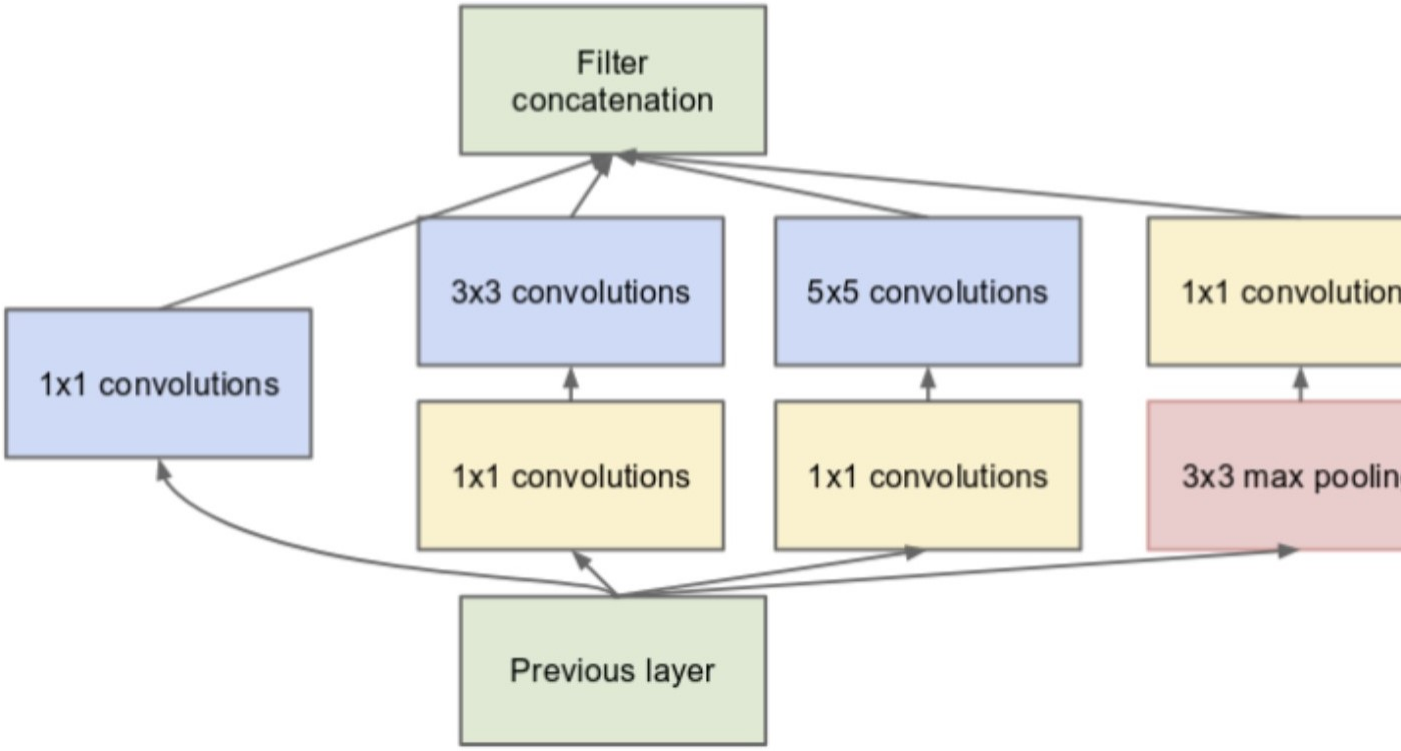

Figure 8: Inception module with demension reductions

This module was used in a popular architecture GoogLeNet (inception v1).

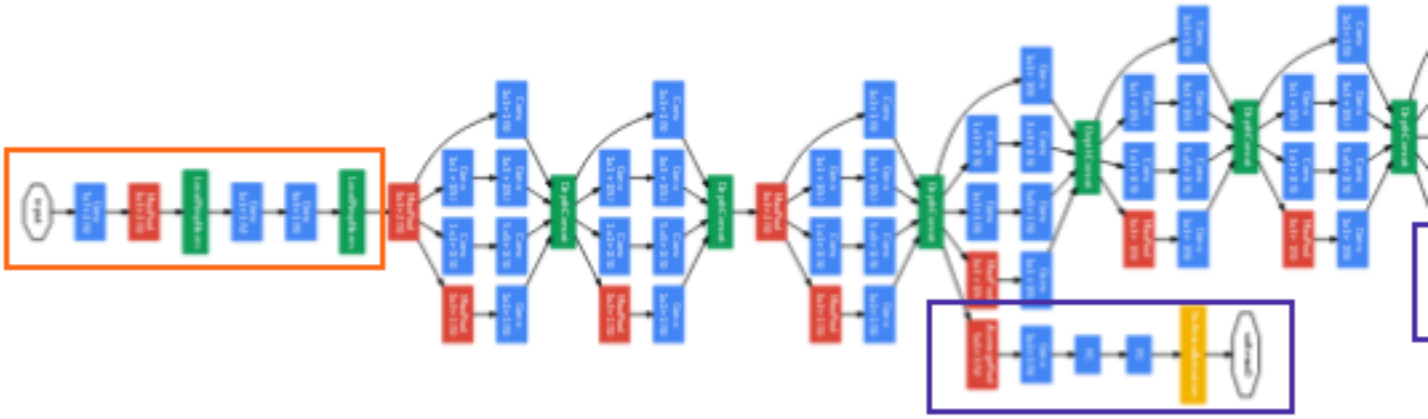

Figure 9: GoogLeNet. The orange box is the stem, which has some preliminary convolutions. The purple boxes are auxiliary classifiers. The wide parts are the inception modules.

GoogLeNet has 9 such inception modules stacked linearly. It is 22 layers deep (27, including the pooling layers). It uses global average pooling at the end of the last inception module. Needless to say, it is a pretty deep classifier. As with any very deep network, it is subject to the vanishing gradient problem. To prevent the middle part of the network from "dying out", the authors introduced two auxiliary classifiers (The purple boxes in the image). They essentially applied softmax to the outputs of two of the inception modules, and computed an auxiliary loss over the same labels. The total loss function is a weighted sum of the auxiliary loss and the real loss. The auxiliary loss is purely used for training purposes, and is ignored during inference.

1.5.2 Inception V2 and V3

Inception v2 and Inception v3 [9] were presented in the same paper. The authors proposed a number of upgrades which increased the accuracy and reduced the computational complexity. The include factorizing 55 convolutions to two 3×3 convolutions to improve speed and then factorizing the 3×3 convolutions into a series of 3×1 and 1×3 convolution to achieve even greater speed up. The filter banks in the modules were made wider instead of deeper to remove representational bottleneck as deeper module would result in excessive reduction in dimensions and therefore loss of information.

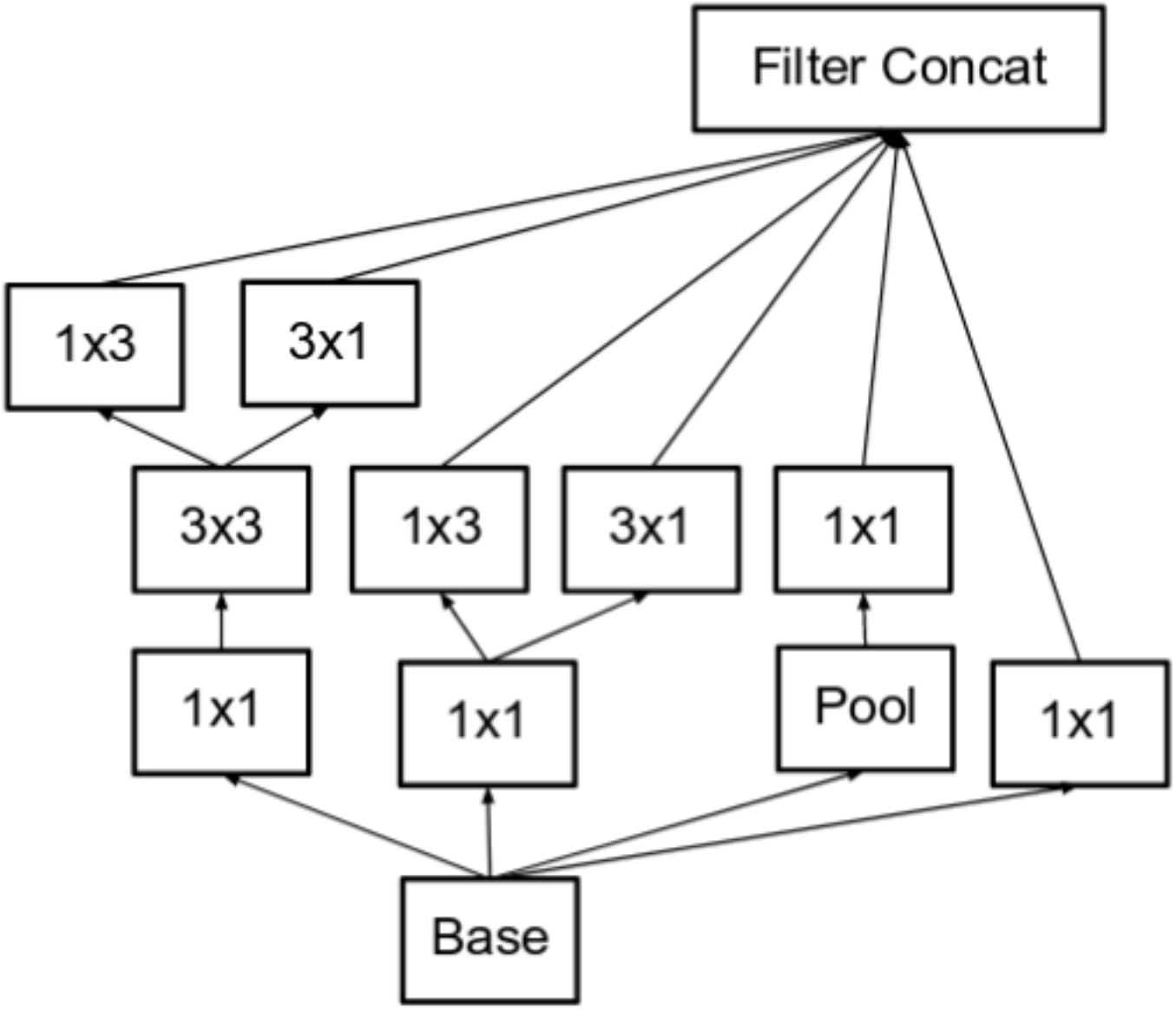

Figure 10: Inception v2

The paper also suggested using auxiliary classifiers on the top of the last 17×17 layer. Inception V3 incorporated all of the upgrades stated for inception v2 and added the following to it:

1. RMSProp Optimizer Factorized 7×7 convolutions.
2. BatchNorm in Auxiliary classifiers
3. Label Smoothing (A type of regularizing component added to the loss formula that prevents the network from becoming too confident about a class. Prevents over fitting)

[2]

1.6 ResNet

Very Deep learning networks are notoriously hard to train because of the vanishing gradient problems. And as the networks get deeper and deeper, the performance starts degrading rapidly. ResNet[4] were designed in order to address this problem. The core idea of ResNet is the "identity shortcut connection" that skips one or more layers as shown below :

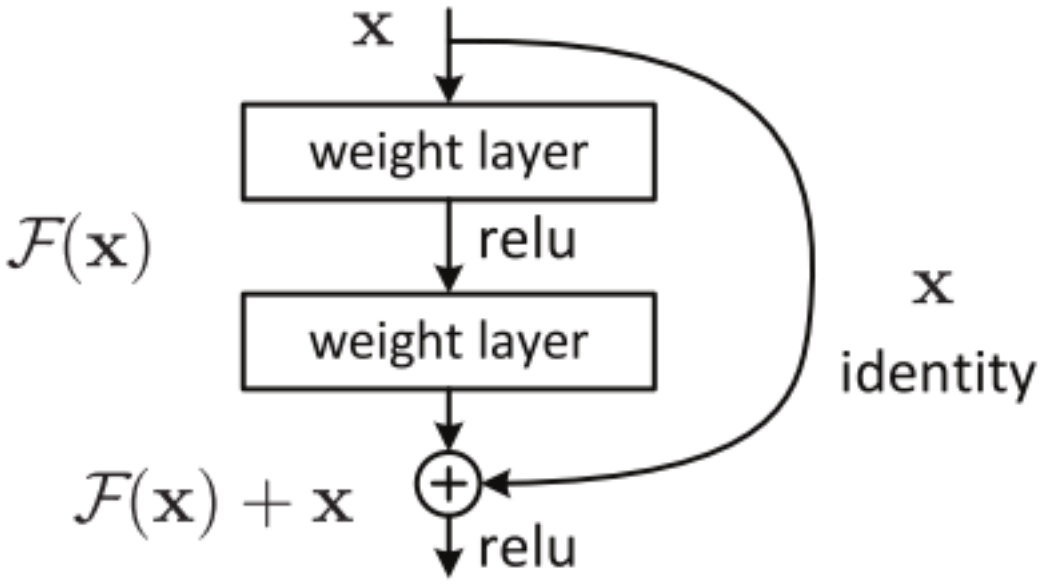

Figure 11: Residual Block

The authors claim that stacking layers shouldn't decrease the performance of the network as many of the layers might end up just performing an identity mapping and thereby not doing anything. This inherently suggests that residual connections makes it easier for the layers to fit the desired underlying mapping. There are several variations as to how residual connections can be arranged in a block as shown below with their classification accuracies on CIFAR-10:

case	Fig.	ResNet-110	ResNet-164
original Residual Unit [1]	Fig. 4(a)	6.61	5.93
BN after addition	Fig. 4(b)	8.17	6.50
ReLU before addition	Fig. 4(c)	7.84	6.14
ReLU-only pre-activation	Fig. 4(d)	6.71	5.91
full pre-activation	Fig. 4(e)	**6.37**	**5.46**

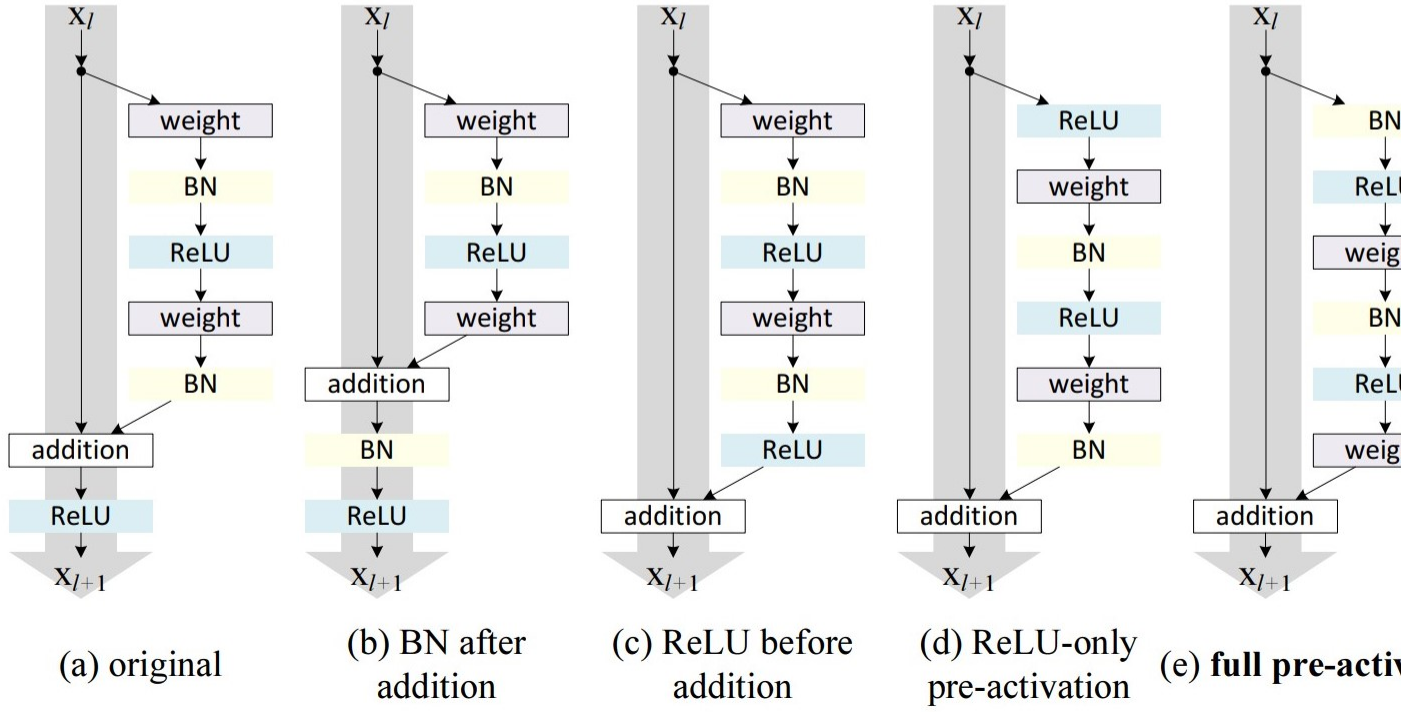

Figure 12: Different variations of Residual Activations

1.7 Combining Inception and ResNet - ResNeXt

Combining the inception and the resnet module, ResNeXt was proposed that had the following architecture:

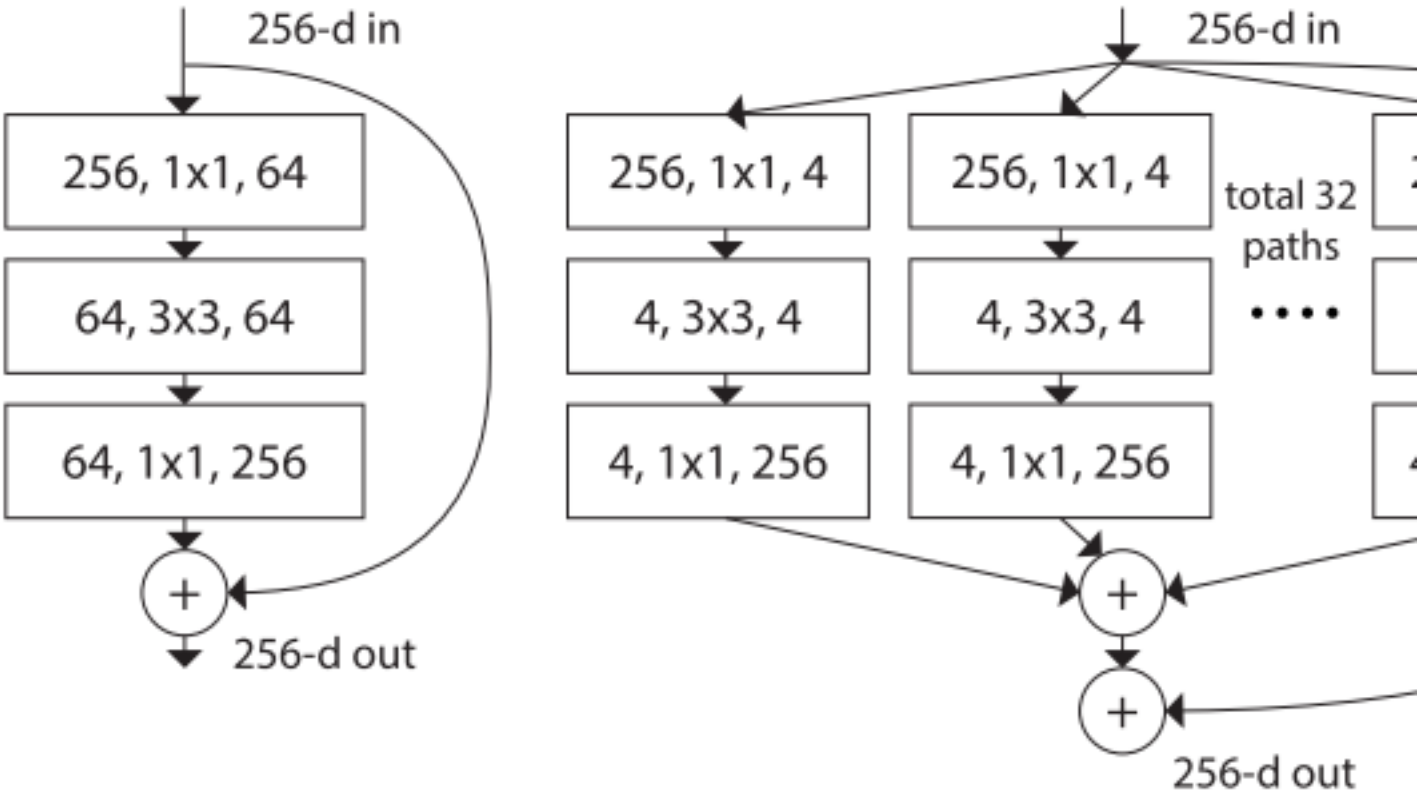

Figure 13: ResNeXt

Some of the key differences from the inception that need to be observed are

- Outputs of different paths are merged by adding them together instead of depth-concatenating them.
- Each path in this architecture has the same topology unlike inception in which each path had a different sized convolution.

The authors introduced a hyper-parameter called cardinality i.e. the number of independent paths, to provide a new way of adjusting the model capacity. Experiments show that accuracy can be gained more efficiently by increasing the cardinality than by going deeper or wider. The authors state that compared to Inception, this novel architecture is easier to adapt to new datasets/tasks, as it has a simple paradigm and only one hyper-parameter to be adjusted, while Inception has many hyper-parameters (like the kernel size of the convolutional layer of each path) to tune.

1.8 DenseNet

Taking the shortcut connections even further, we have DenseNets[5] in which all layers are directly connected with each other. In this novel architecture, the input of each layer consists of the feature maps of all earlier layer, and its output is passed to each subsequent layer. The feature maps are aggregated with depth-concatenation.

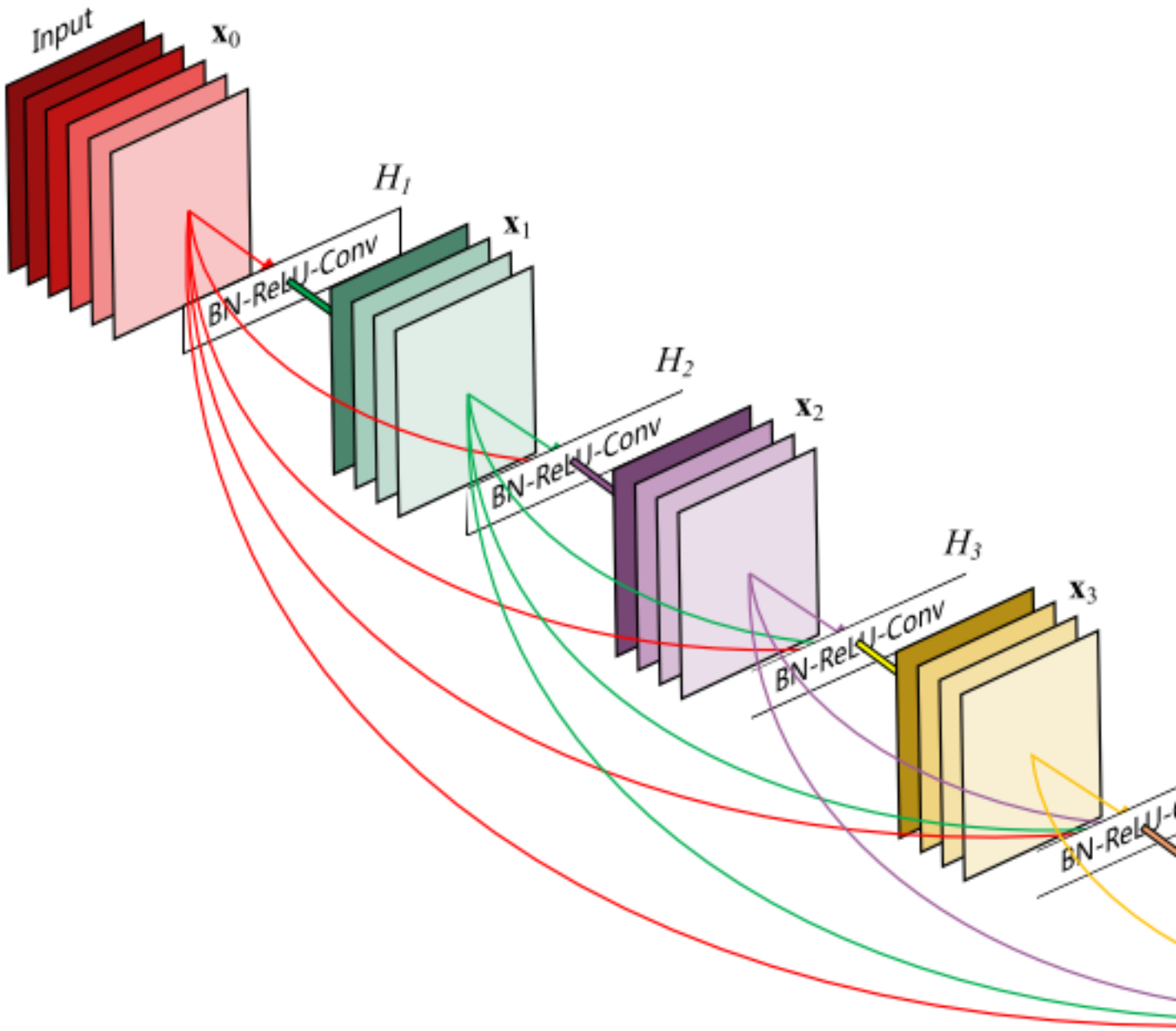

Figure 14: DensNet

Other than tackling the vanishing gradients problem, the authors claim that this architecture also encourages feature reuse, making the network highly parameter-efficient.

[1]

2 Applications of CNN

2.1 Style Transfer

Neural style transfer is the task of rendering semantic content of an image in a different style. The goal is to synthesise a texture from a source image while constraining texture synthesis in order to preserve the semantic content of a target image as shown below.

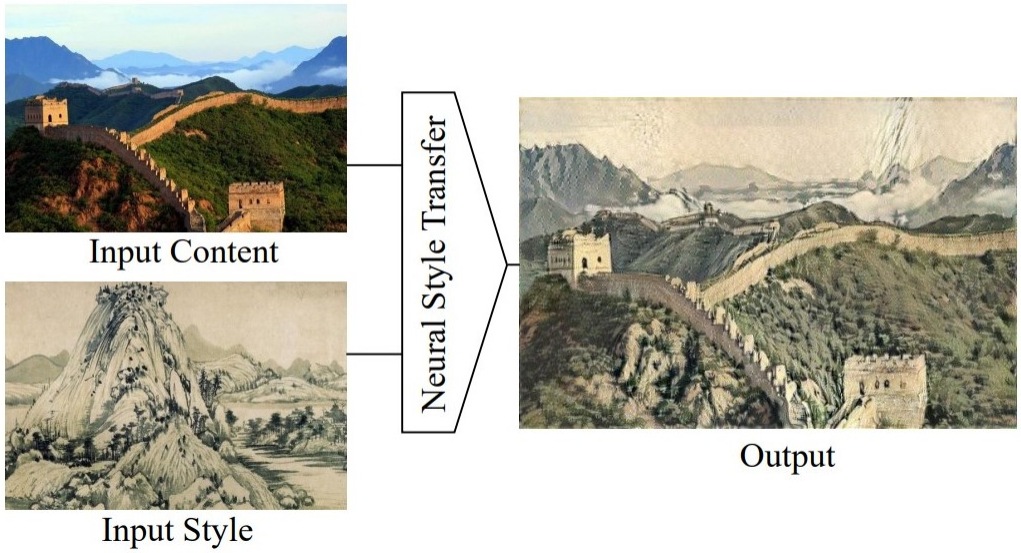

Figure 15: Example of NST algorithm to transfer the style of a Chinese painting onto a given photograph. The style image is named *Dwelling in the Fuchun Mountains* by Gongwang Huang.

Recent works leverage the power of Convolutional Neural Networks to extract high level semantic information from the target image and combine that with the style of the source image. A pre-trained model (like VGG network) which was trained to perform object recognition and localization is used. The weights are normalized such that the mean activation of each convolution filter over the images is one (which can be done as only the layers with Rectified Linear Units are used and none of the fully connected layers).

2.1.1 Content Representation

A given input image $\tilde{x}$ is encoded in each layer of the Convolutional Neural Network by the filter responses to that image. A layer with N_l distinct filters has N_l feature maps each of size M_l , where M_l is the height times the width of the feature map. So the responses in a layer l can be stored in a matrix $F\ l \in^{N_l \times M_l}$ where F_{lji} is the activation of the i^{th} filter at position j in layer l. To visualise the image information that is encoded at different layers of the hierarchy one can perform gradient descent on a white noise image to find another image that matches the feature responses of the original image.

When Convolutional Neural Networks are trained on object recognition, they develop a representation of the image that makes object information increasingly explicit along the processing

hierarchy [10]. Therefore, along the processing hierarchy of the network, the input image is transformed into representations that are increasingly sensitive to the actual content of the image, but become relatively invariant to its precise appearance. Thus, higher layers in the network capture the high-level content in terms of objects and their arrangement in the input image but do not constrain the exact pixel values of the reconstruction very much. In contrast, reconstructions from the lower layers simply reproduce the exact pixel values of the original image. We therefore refer to the feature responses in higher layers of the network as the content representation.

2.1.2 Style Representation

To obtain a representation of the style of an input image, we use a feature space designed to capture texture information. This feature space can be built on top of the filter responses in any layer of the network. It consists of the correlations between the different filter responses, where the expectation is taken over the spatial extent of the feature maps. These feature correlations are given by the Gram matrix $G_l \in R^{N_l \times N_l}$, where G^l_{ij} is the inner product between the vectorised feature maps i and j in layer l:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

We create images matching the styles of the style target by using gradient descent from a white noise image to minimise the mean-squared distance between the entries of the Gram matrices from the original image and the Gram matrices of the image to be generated.

2.1.3 Style Transfer

To transfer the style of an artwork $\tilde{a}$ onto a photograph $\tilde{p}$ we synthesise a new image that simultaneously matches the content representation of $\tilde{p}$ and the style representation of $\tilde{a}$. Thus we jointly minimise the distance of the feature representations of a white noise image from the content representation of the photograph in one layer and the style representation of the painting defined on a number of layers of the Convolutional Neural Network. The loss function we minimise

$$\mathcal{L}_{\text{total}}\left(\vec{p}, \vec{a}, \vec{x}\right) = \alpha \mathcal{L}_{\text{content}}\left(\vec{p}, \vec{x}\right) + \beta \mathcal{L}_{\text{style}}\left(\vec{a}, \vec{x}\right)$$

Figure 16: Images that combine the content of a photograph with the style of several well-known artworks. The images were created by finding an image that simultaneously matches the content representation of the photograph and the style representation of the artwork. The original photograph depicting the Neckarfront in Tubingen, Germany, is shown in **A** (Photo: Andreas Praefcke). The painting that provided the style for the respective generated image is shown in the bottom left corner of each panel. **B** *The Shipwreck of the Minotaur* by J.M.W. Turner, 1805. **C** *The Starry Night* by Vincent van Gogh, 1889. **D** *Der Schrei* by Edvard Munch, 1893. **E** *Femme nue assise* by Pablo Picasso, 1910. **F** *Composition VII* by Wassily Kandinsky, 1913.

2.2 Image Segmentation

Image segmentation [6] is an essential component in many visual understanding systems. It involves partitioning images (or video frames) into multiple segments or objects . Segmentation plays a central role in a broad range of applications , including medical image analysis (e.g., tumor boundary extraction and measurement of tissue volumes), autonomous vehicles (e.g., navigable surface and pedestrian detection), video surveillance, and augmented reality to count a few. Numerous image segmentation algorithms have been developed in the literature, from the earliest methods, such as thresholding, histogram-based bundling, region-growing, k-means clustering, watersheds, to more advanced algorithms such as active contours, graph cuts, conditional and Markov random fields, and sparsity-based methods. Over the past few years, however, deep learning networks have yielded a new generation of image segmentation models with remarkable performance improvements, often achieving the highest accuracy rates on popular benchmarks.

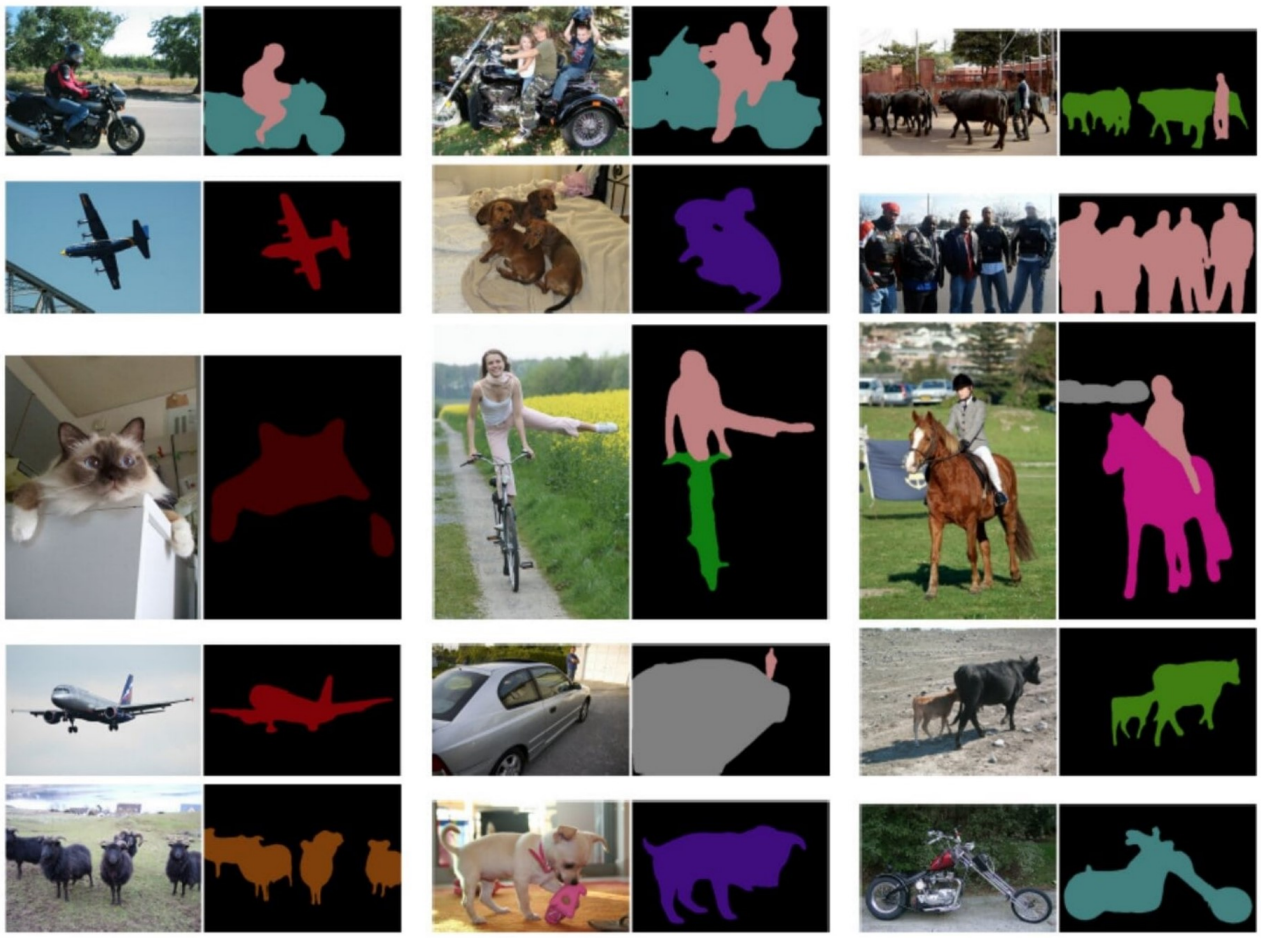

Figure 17: Segmentation result of deepLabv3

2.2.1 U-Net

Ronneberger et al.[7] proposed the U-Net for segmenting biological microscopy images. Their network and training strategy relies on the use of data augmentation to learn from the available annotated images more effectively. The U-Net architecture (Figure **??**) comprises two parts, a contracting path to capture context, and a symmetric expanding path that enables precise localization. The down-sampling or contracting part has a FCN-like architecture that extracts features with 3×3 convolutions. The up-sampling or expanding part uses up-convolution (or deconvolution), reducing the number of feature maps while increasing their dimensions. Feature maps from the down-sampling part of the network are copied to the up-sampling part to avoid losing pattern information. Finally, a 1×1 convolution processes the feature maps to generate a segmentation map that categorizes each pixel of the input image. U-Net was trained on 30 transmitted light microscopy images, and it won the ISBI cell tracking challenge 2015 by a large margin.

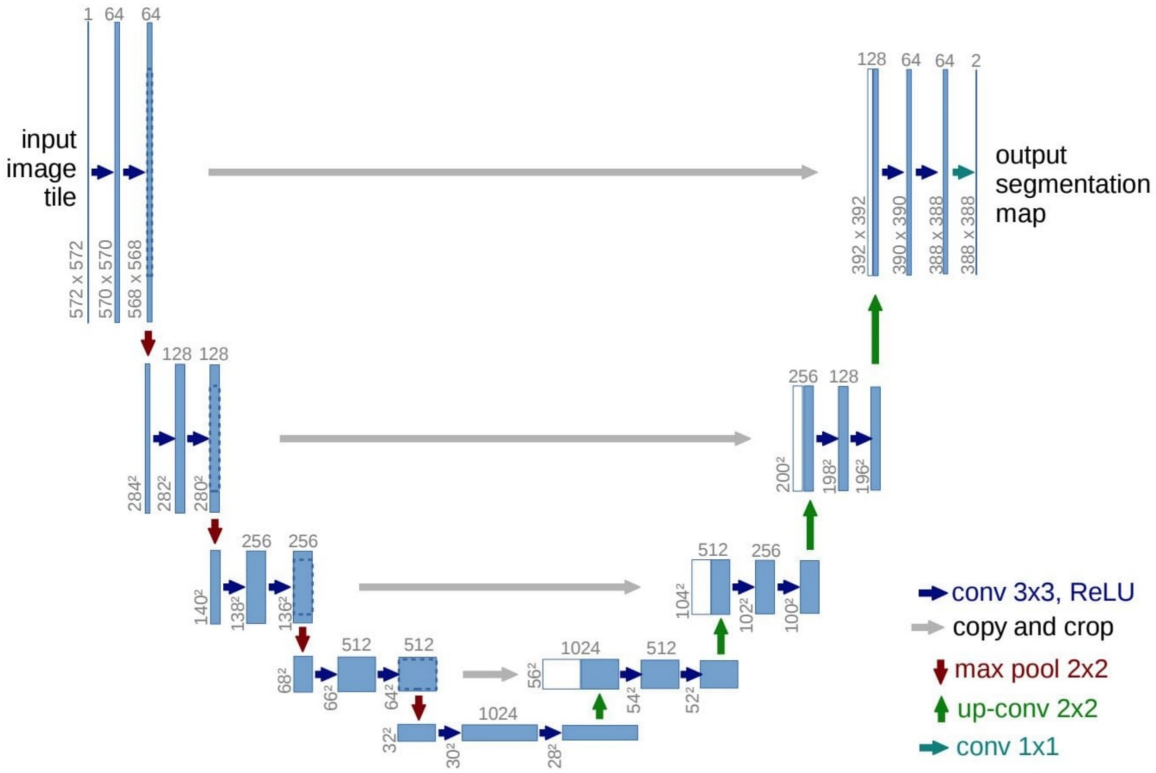

Figure 18: The U-net model. The blue boxes denote feature map blocks with their indicated shapes

2.2.2 DeepLab

DeepLabv1 and DeepLabv2 are among some of the most popular image segmentation approaches, developed by Chen et al.. The latter has three key features. First is the use of dilated convolution to address the decreasing resolution in the network (caused by max-pooling and striding). Second is Atrous Spatial Pyramid Pooling (ASPP), which probes an incoming convolutional feature layer

with filters at multiple sampling rates, thus capturing objects as well as image context at multiple scales to robustly segment objects at multiple scales. Third is improved localization of object boundaries by combining methods from deep CNNs and probabilistic graphical models. The best DeepLab (using a ResNet-101 as backbone) has reached a 79.7% mIoU score on the 2012 PASCAL VOC challenge, a 45.7% mIoU score on the PASCAL-Context challenge and a 70.4% mIoU score on the Cityscapes challenge. Figure 19 illustrates the Deeplab model, which is similar to, the main difference being the use of dilated convolution and ASPP.

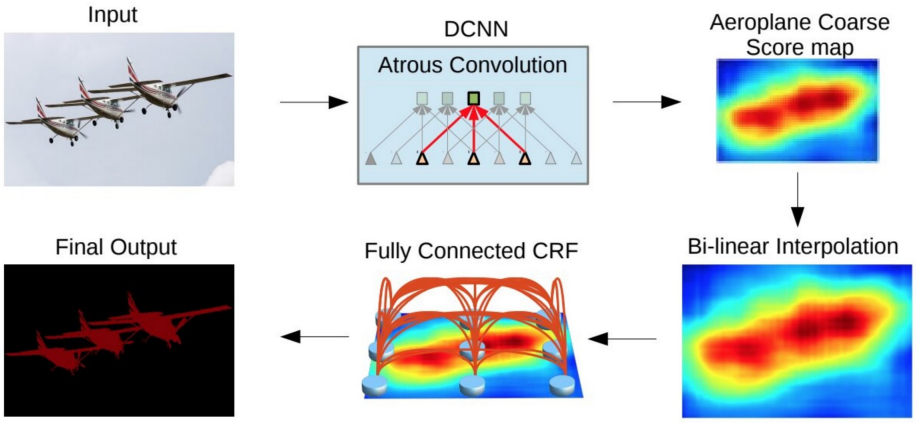

Figure 19: The DeepLab model. A CNN model such as VGG-16 or ResNet101 is employed in fully convolutional fashion, using dilated convolution. A bilinear interpolation stage enlarges the feature maps to the original image resolution. Finally, a fully connected CRF refines the segmentation result to better capture the object boundaries

Subsequently, Chen et al.[3] proposed DeepLabv3, which combines cascaded and parallel modules of dilated convolutions. The parallel convolution modules are grouped in the ASPP. A 1×1 convolution and batch normalisation are added in the ASPP. All the outputs are concatenated and processed by another 1×1 convolution to create the final output with logits for each pixel. In 2018, Chen et al. released Deeplabv3+, which uses an encoder-decoder architecture (Figure 20), including atrous separable convolution, composed of a depthwise convolution (spatial convolution for each channel of the input) and pointwise convolution (1×1 convolution with the depthwise convolution as input). They used the DeepLabv3 framework as encoder. The most relevant model has a modified Xception backbone with more layers, dilated depthwise separable convolutions instead of max pooling and batch normalization. The best DeepLabv3+ pretrained on the COCO and the JFT datasets has obtained a 89.0% mIoU score on the 2012 PASCAL VOC challenge.

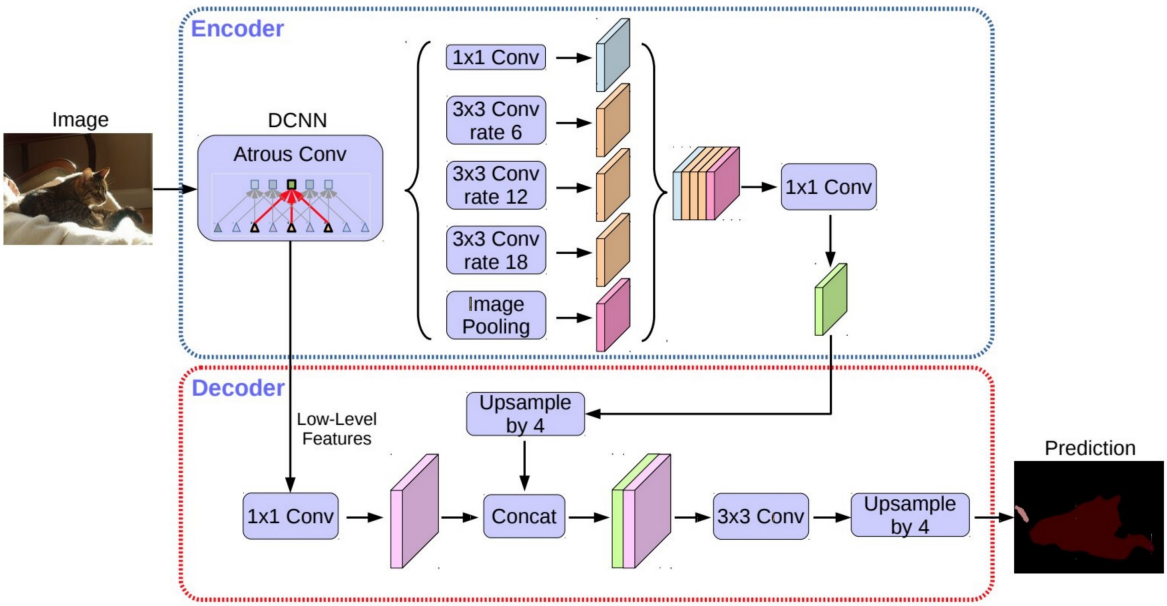

Figure 20: DeepLabV3+

2.3 Object Detection

Object detection, aiming at locating object instances from a large number of predefined categories in natural images, is one of the most fundamental and challenging problems in computer vision. The goal of object detection is to determine whether or not there are any instances of objects from some given categories (such as humans, cars, bicycles, dogs and cats) in some given image and, if present to return the spatial location and extent of each object instance(e.g. via a bounding box). Object detection has a wide range of applications in many areas of artificial intelligence and information technologies, including robot vision, consumer electronics, security, autonomous driving, human computer interaction, content based image retrieval, intelligent video surveillance, and augmented reality.

Typically, the spatial location and extent of an object can be defined coarsely using a bounding box, i.e., an axis-aligned rectangle tightly bounding the object, a precise pixel-wise segmentation mask, or a closed boundary , as illustrated in Fig.

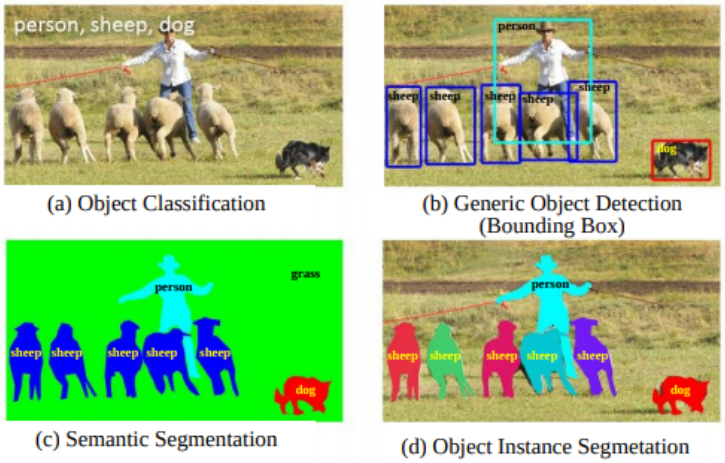

Figure 21: Recognition problems related to generic object detection

Generic object detection aims at localizing and recognizing a broad range of natural object categories. The ideal goal of generic object detection is to develop general-purpose object detection algorithms achieving two competing goals: high quality/accuracy and high efficiency High quality detection has to accurately localize and recognize objects in images or video frames, such that the large variety of object categories in the real world can be distinguished (i.e., high distinctiveness), and that object instances from the same category, subject to intraclass appearance variations, can be localized and recognized (i.e., high robustness). High efficiency requires the entire detection task to run at a sufficiently high frame rate with acceptable memory and storage usage. Despite several decades of research and significant progress, arguably the combined goals of accuracy and efficiency have not yet been met.

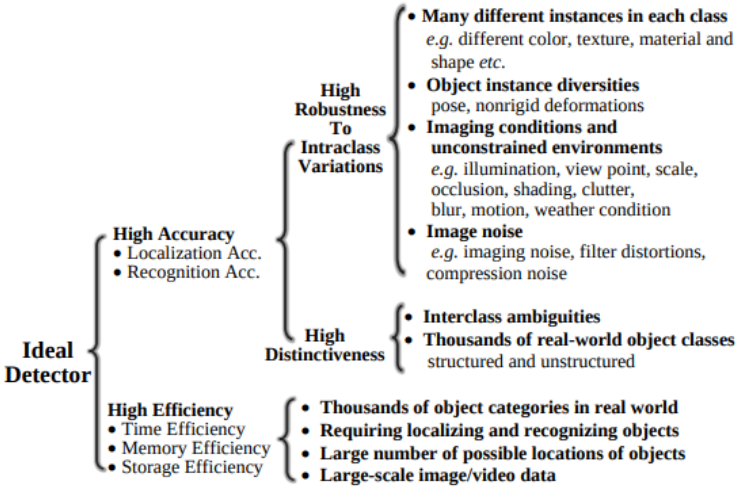

Figure 22: Summary of challenges in generic object detection

We will now briefly review all milestone detectors that have been used for object detection since deep learning entered the field. Nearly all detectors proposed over the last several years are

based on one of these milestone detectors, attempting to improve on one or more aspects. Broadly these detectors can be organized into two main categories:

1. Two stage detection framework, which includes a pre-processing step for region proposal, making the overall pipeline two stage.

2. One stage detection framework, or region proposal free framework, which is a single proposed method which does not separate detection proposal, making the overall pipeline single stage.

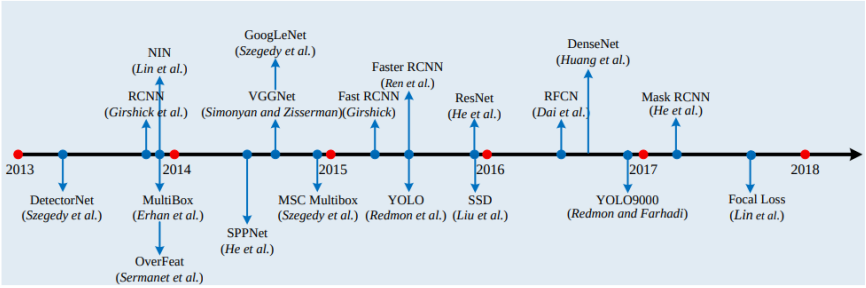

Figure 23: Milestones in generic object detection based on the point in time of the first arXiv version

2.3.1 Region Based Frameworks (Two staged)

In a region based framework, category-independent region proposals are generated from an image, CNN features are extracted from these regions, and then category-specific classifiers are used to determine the category labels of the proposals.

1. **RCNN** Inspired by the breakthrough image classification results obtained by CNN and the success of selective search in region proposal for hand-crafted features, Girshick et al. were among the first to explore CNN for generic object detection and developed RCNN which integrates AlexNet with the region proposal method selective search. Training in an RCNN framework consists of multistage pipelines:

   (a) Class-agnostic region proposals, which are candidate regions hat might contain objects, are obtained via selective search.

   (b) Region proposals, which are cropped from the image and warped into the same size, are used as the input for finetuning a CNN model pre-trained using large-scale dataset such as ImageNet.

   (c) A set of class specific linear SVM classifiers are trained using fixed length features extracted with CNN, replacing the softmax classifier learned by finetuning.

   (d) Bounding box regression is learned for each object class with CNN features.

   In spite of achieving high object detection quality, RCNN has notable drawbacks:

(a) Training is a multistage complex pipeline, which is inelegant, slow and hard to optimize because each individual stage must be trained separately.

(b) Numerous region proposals which provide only rough localization need to be externally detected.

(c) Training SVM classifiers and bounding box regression is expensive in both disk space and time, since CNN features are extracted independently from each region proposal in each image, posing great challenges for large-scale detection, especially very deep CNN networks such as AlexNet and VGG.

(d) Testing is slow, since CNN features are extracted per object.

2. **Fast RCNN**: Girshick proposed Fast RCNN that addresses some of the disadvantages of RCNN and while improving on their detection speed and quality. Fast RCNN enables end-to-end detector training (when ignoring the process of region proposal generation) by developing a streamlined training process that simultaneously learns a softmax classifier and class-specific bounding box regression using a multitask loss, rather than training a softmax classifier, SVMs, and BBRs in three separate stages as in RCNN/SPPnet. Fast RCNN employs the idea of sharing the computation of convolution across region proposals, and adds a Region of Interest (RoI) pooling layer between the last CONV layer and the first FC layer to extract a fixedlength feature for each region proposal (i.e. RoI). Essentially, RoI pooling uses warping at feature level for approximating warping at image level. The features after the RoI pooling layer are fed into a sequence of FC layers that finally branch into two sibling output layers: softmax probabilities for object category prediction and class-specific bounding box regression offsets for proposal refinement. Compared to RCNN, Fast RCNN improves the efficiency considerably – typically 3 times faster in training and 10 times faster in testing. In summary, Fast RCNN has attractive advantages of higher detection quality, a single-stage training process that updates all network layers, and no storage required for feature caching.

Figure 24: Illustration of the object detecting framework RCNN in great detail.

3. **Faster RCNN**: Although Fast RCNN significantly sped up the detection process, it still relies on external region proposals. Region proposal computation is exposed as the new bottleneck in Fast RCNN. Recent work has shown that CNNs have a remarkable ability to localize objects in CONV layers an ability which is weakened in the FC layers. Therefore, the selective search can be replaced by the CNN in producing region proposals. The Faster RCNN framework proposed by Ren et al. proposed an efficient and accurate Region Proposal Network (RPN) to generating region proposals. They utilize single network to accomplish the task of RPN for region proposal and Fast RCNN for region classification. In Faster RCNN, the RPN and fast RCNN share large number of convolutional layers. The features from the last shared convolutional layer are used for region proposal and region classification from separate branches. RPN first initializes k n $\times$ n reference boxes (i.e. the so called anchors) of different scales and aspect ratios at each CONV feature map location. Each n $\times$ n anchor is mapped to a lower dimensional vector (such as 256 for ZF and 512 for VGG), which is fed into two sibling FC layers — an object category classification layer and a box regression layer. Different from Fast RCNN, the features used for regression in RPN have the same size. RPN shares CONV features with Fast RCNN, thus enabling highly efficient region proposal computation. RPN is, in fact, a kind of Fully Convolutional Network (FCN) ; Faster

RCNN is thus a purely CNN based framework without using handcrafted features. For the very deep VGG16 model. Faster RCNN can test at 5fps (including all steps) on a GPU, while achieving state of the art object detection accuracy on PASCAL VOC 2007 using 300 proposals per image. The initial Faster RCNN in contains several alternating training steps. This was then simplified by one step joint training in .Concurrent with the development of Faster RCNN, Lenc and Vedaldi challenged the role of region proposal generation methods such as selective search, studied the role of region proposal generation in CNN based detectors, and found that CNNs contain sufficient geometric information for accurate object detection in the CONV rather than FC layers. They proved the possibility of building integrated, simpler, and faster object detectors that rely exclusively on CNNs, removing region proposal generation methods such as selective search.

4. **Mask RCNN**: Mask RCNN: Following the spirit of conceptual simplicity, efficiency, and flexibility, He et al. proposed Mask RCNN to tackle pixel-wise object instance segmentation by extending Faster RCNN. Mask RCNN adopts the same two stage pipeline, with an identical first stage (RPN). In the second stage, in parallel to predicting the class and box offset, Mask RCNN adds a branch which outputs a binary mask for each RoI. The new branch is a Fully Convolutional Network (FCN) on top of a CNN feature map. In order to avoid the misalignments caused by the original RoI pooling (RoIPool) layer, a RoIAlign layer was proposed to preserve the pixel level spatial correspondence. With a backbone network ResNeXt101-FPN, Mask RCNN achieved top results for the COCO object instance segmentation and bounding box object detection. It is simple to train, generalizes well, and adds only a small overhead to Faster RCNN, running at 5 FPS.

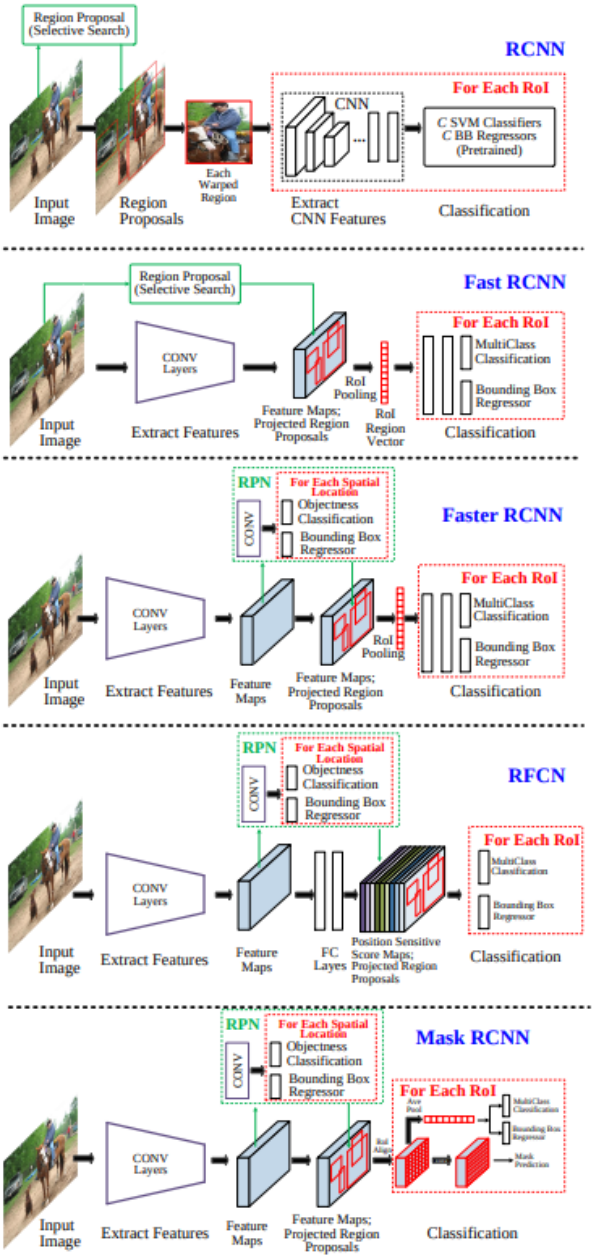

Figure 25: 8 High level diagrams of the leading region based frameworks for generic object detection.

2.3.2 Unified Pipeline (One Stage Pipeline)

The region-based pipeline strategies of have prevailed on detection benchmarks since RCNN. The significant efforts introduced in Section 3.1 have led to faster and more accurate detectors, and the current leading results on popular benchmark datasets are all based on Faster RCNN. In spite of that progress, region-based approaches could be computationally expensive for mobile/wearable devices, which have limited storage and computational capability. Therefore, instead of trying to optimize the individual components of a complex region-based pipeline, researchers have begun

to develop unified detection strategies. Unified pipelines refer broadly to architectures that directly predict class probabilities and bounding box offsets from full images with a single feed forward CNN network in a monolithic setting that does not involve region proposal generation or post classification. The approach is simple and elegant because it completely eliminates region proposal generation and subsequent pixel or feature resampling stages, encapsulating all computation in a single network. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

1. **YOLO (You Only Look Once):** Redmon et al. proposed YOLO, a unified detector casting object detection as a regression problem from image pixels to spatially separated bounding boxes and associated class probabilities. Since the region proposal generation stage is completely dropped, YOLO directly predicts detections using a small set of candidate regions. Unlike region-based approaches, e.g. Faster RCNN, that predict detections based on features from local region, YOLO uses the features from entire image globally. In particular, YOLO divides an image into a S $\times$ S grid. Each grid predicts C class probabilities, B bounding box locations and confidences scores for those boxes. These predictions are encoded as an S$\times$S$\times$(5B+C) tensor. By throwing out the region proposal generation step entirely, YOLO is fast by design, running in real time at 45 FPS and a fast version, i.e. Fast YOLO , running at 155 FPS. Since YOLO sees the entire image when making predictions, it implicitly encodes contextual information about object classes and is less likely to predict false positives on background. YOLO makes more localization errors resulting from the coarse division of bounding box location, scale and aspect ratio. YOLO may fail to localize some objects, especially small ones, possibly because the grid division is quite coarse, and because by construction each grid cell can only contain one object. It is unclear to what extent YOLO can translate to good performance on datasets with significantly more objects, such as the ILSVRC detection challenge.

2. **SSD (Single Shot Detector):** In order to preserve real-time speed without sacrificing too much detection accuracy, Liu et al. proposed SSD, which is faster than YOLO and has accuracy competitive with state-of-the-art region-based detectors, including Faster RCNN. SSD effectively combines ideas from RPN in Faster RCNN , YOLO and multiscale CONV features to achieve fast detection speed while still retaining high detection quality. Like YOLO, SSD predicts a fixed number of bounding boxes and scores for the presence of object class instances in these boxes, followed by an NMS step to produce the final detection. The CNN network in SSD is fully convolutional, whose early layers are based on a standard architecture, such as VGG (truncated before any classification layers), which is referred as the base network. Then several auxiliary CONV layers, progressively decreasing in size, are added to the end of the base network. The information in the last layer with low resolution may be too coarse spatially to allow precise localization. SSD uses shallower layers with higher resolution for detecting small objects. For objects of different sizes, SSD performs detection over multiple scales by operating on multiple CONV feature maps, each of which predicts category scores and box offsets for bounding boxes of appropriate sizes. For a 300 $\times$ 300 input, SSD achieves 74.3% mAP on the VOC2007 test at 59 FPS on a Nvidia Titan X.

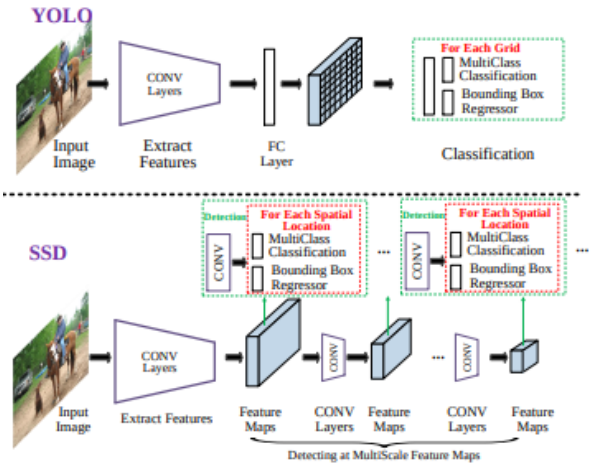

Figure 26: 8 High level diagrams of the leading unified frameworks for generic object detection.

2.4 Face Recognition

A facial recognition system is used to identify a person's face. It can work with both images or videos.Liveness detection within a video face recognition system prevents the network from identifying a face in an image. It can be learned by supervised deep learning using a dataset for live human and in-live human and sequence learning.The face recognition problem states that given a database of K people and an input image, the network needs output the ID of the image if it's of any of the K persons and in case its not the network should predict not recognized.

2.4.1 One Shot Learning

One of the challenges that deep learning based facial recognition systems have to solve is the one shot learning problem which means that the recognition system should be able to recognize a person, learning from one image. We very well know by now that deep learning doesn't work well with a small number of data. So to make these recognition systems work, the problem is tackled not as a classification problem but instead the network is trained to learn a similarity function between images.

2.4.2 Similarity Function

Let us denote the similarity function as variable d. So d(img1, img2) has to be small if two images are similar and large in case they are far apart. To tell if two images are different or same we use a threshold τ. If d(img1,img2) is lower than τ, we say images are similar else its different.To summarize the network should output

$$output = \begin{cases} Similar, & \text{if } d(img1, img2) \leq \tau. \\ Different, & \text{otherwise.} \end{cases} \tag{1}$$

2.4.3 Siamese Network

The similarity function is implemented using a type of neural network called Siamease Network which accepts multiple inputs to the two or more networks with the same architecture and parameters. The goal of learning is that the parameters of Neural network define an encoding $f(x^{(i)})$ and the network has to learn parameters such that

1. If $x^{(i)}$ and $x^{(j)}$ are the same person then $||f(x^{(i)}) - f(x^{(i)})||^2$ is small.
2. If $x^{(i)}$ and $x^{(j)}$ are the different then $||f(x^{(i)}) - f(x^{(i)})||^2$ is large.

2.4.4 Triplet Loss

Triplet Loss is one of the loss functions we can use to solve the similarity distance in a Siamese network. Our learning objective in the triplet loss function is to get the distance between an **Anchor** image and a **positive** or a **negative** image where positive means same person, while negative means different person. More formally we want,

$$||f(A) - f(P)||^2 \leq ||f(A) - f(N)||^2 \implies ||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 \leq 0$$

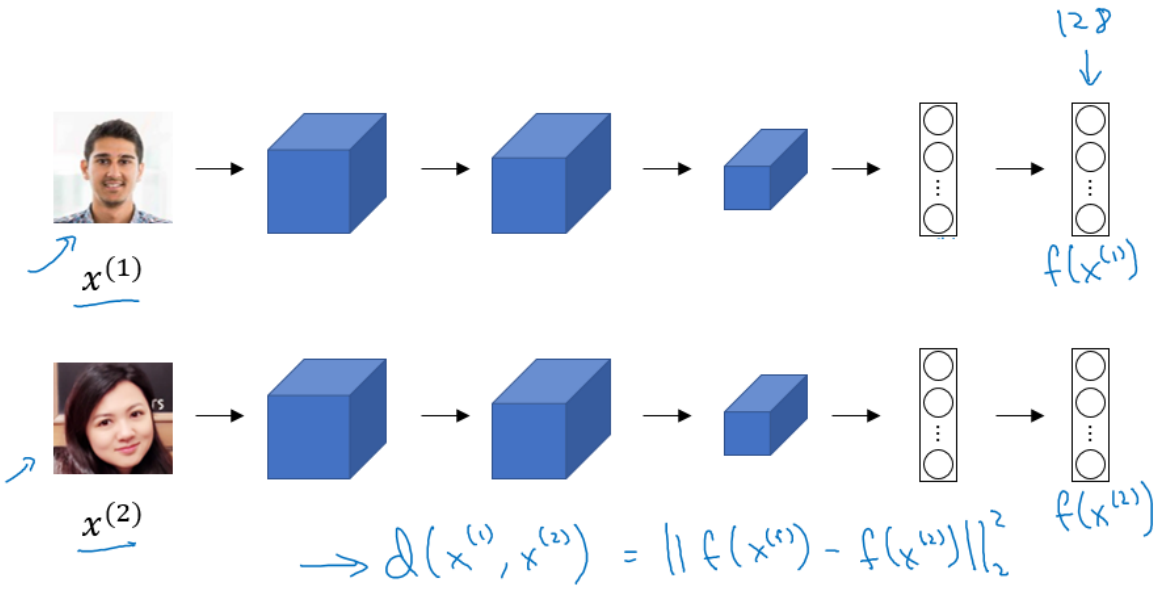

Figure 27: Siamese Network

The issue with the above equation is that if the neural network outputs everything as 0, then the equation is easily satisfied, so to fix that we introduce a constant α

$$||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 \leq -\alpha$$
$$\implies ||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha \leq 0$$

So the final loss function can be states as

$$L = max(||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha, 0)$$

One important thing to note is that if during training A,P and N are chosen randomly then the constraint is easily satisfied so the network doesn't train properly so we should chose triplets such that it is hard for the network to learn them.

2.5 CNNs for NLP

Location Invariance and local Compositionality made intuitive sense for images, but not so much for NLP. One probably does care a lot where in the sentence a word appears. Pixels close to each other are likely to be semantically related (part of the same object), but the same isn't always true for words. In many languages, parts of phrases could be separated by several other words. The compositional aspect isn't obvious either. Clearly, words compose in some ways, like an adjective modifying a noun, but how exactly this works what higher level representations actually "mean" isn't as obvious as in the Computer Vision case.

Given all this, it seems like CNNs wouldn't be a good fit for NLP tasks. Recurrent Neural Networks make more intuitive sense. They resemble how we process language (or at least how we think we process language): Reading sequentially from left to right. Fortunately, this doesn't mean

that CNNs don't work. All models are wrong, but some are useful. It turns out that CNNs applied to NLP problems perform quite well. The simple Bag of Words model is an obvious oversimplification with incorrect assumptions, but has nonetheless been the standard approach for years and lead to pretty good results.

A big argument for CNNs is that they are fast. Very fast. Convolutions are a central part of computer graphics and implemented on a hardware level on GPUs. Compared to something like n-grams, CNNs are also efficient in terms of representation. With a large vocabulary, computing anything more than 3-grams can quickly become expensive. Even Google doesn't provide anything beyond 5-grams. Convolutional Filters learn good representations automatically, without needing to represent the whole vocabulary. It's completely reasonable to have filters of size larger than 5. One can think that many of the learned filters in the first layer are capturing features quite similar (but not limited) to n-grams, but represent them in a more compact way.

Let's try to understand CNN's for NLP using a case study. The task we will be tackling is sentiment analysis, the input is a sentence which might be a movie review or a tweet and the output is a class label which tells if the review is positive or negative. The model we train will have three components to it, namely:

1. Embedding layer.
2. Multi-channel CNN layer.
3. Pooling layer.
4. Output layer.

Here is an overview of the architecutre:

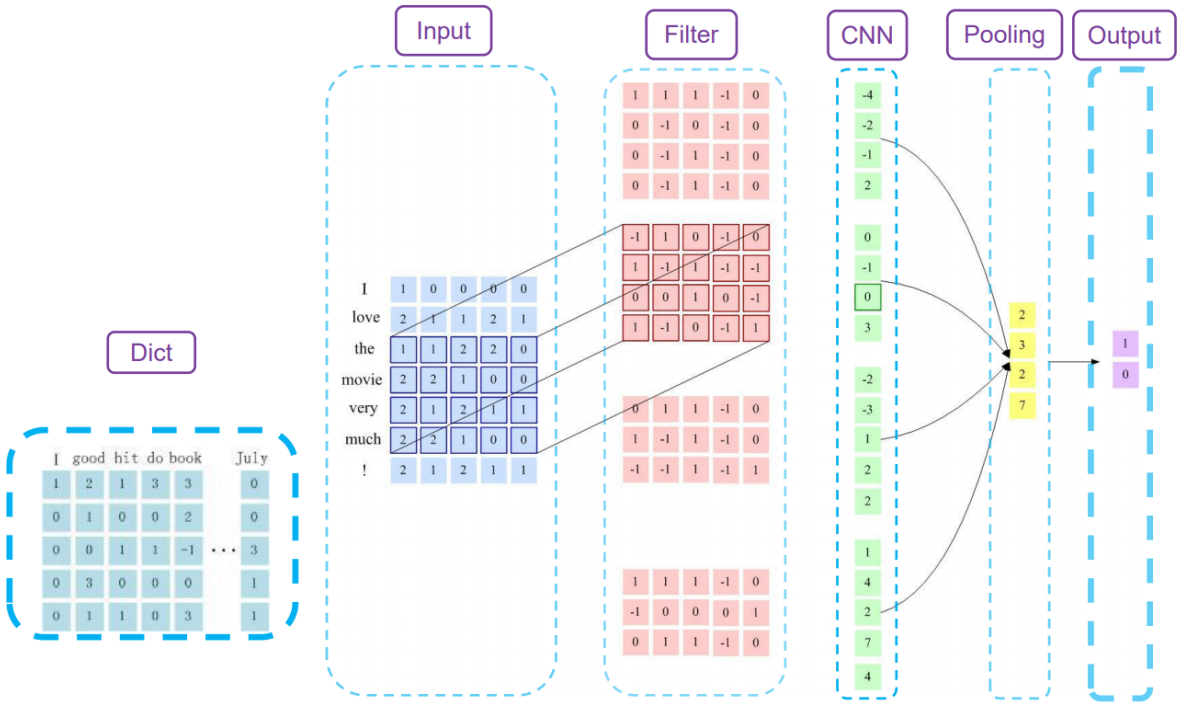

Figure 28: CNN for Text

2.5.1 Embedding Layer

The embedding layer is basically a lookup table or some fine tuned pre trained NLP model which can convert text to meaningful numbers. Common choices include gloVe, word2vec, ElMO, BERT, etc.(you will learn about these in detail in the NLP module, for now consider these as a black box which can take in text and spit out meaningful numbers).

2.5.2 Convolutional Layer

Using a stride of 1, the filters are applied to the input, convolution operation is performed and the output of the operation is sent to the narrow green array.

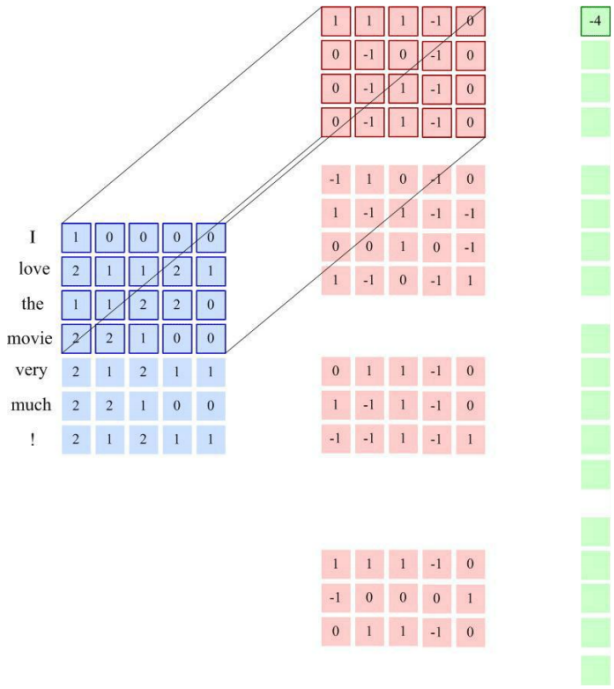

Figure 29: Convolutional Layer

2.5.3 Max Pooling

Now max pooling is done on each segment of the green array and the numbers generated are then sent to a feedforward network which has a softmax at the final layer and outputs a probability for each class, 0 being negative and 1 being positive.

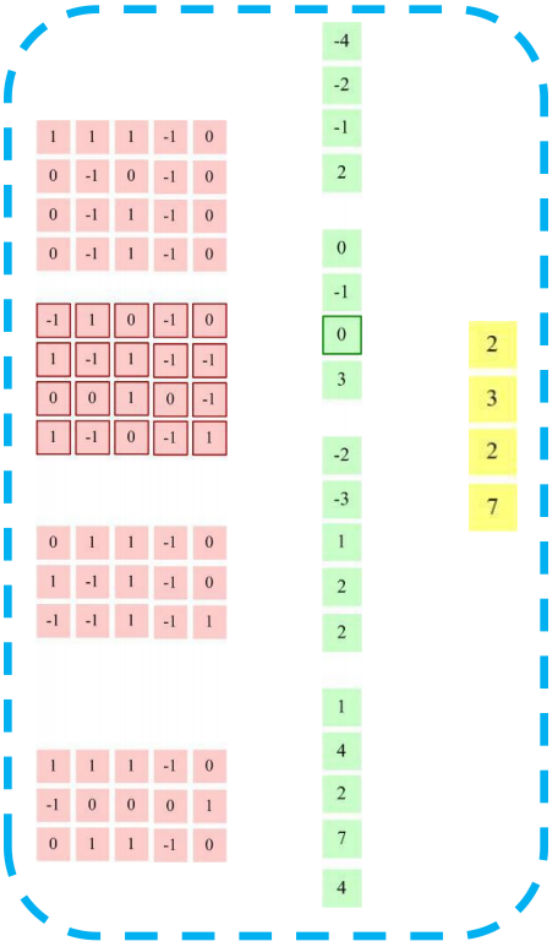

Figure 30: Pooling Layer

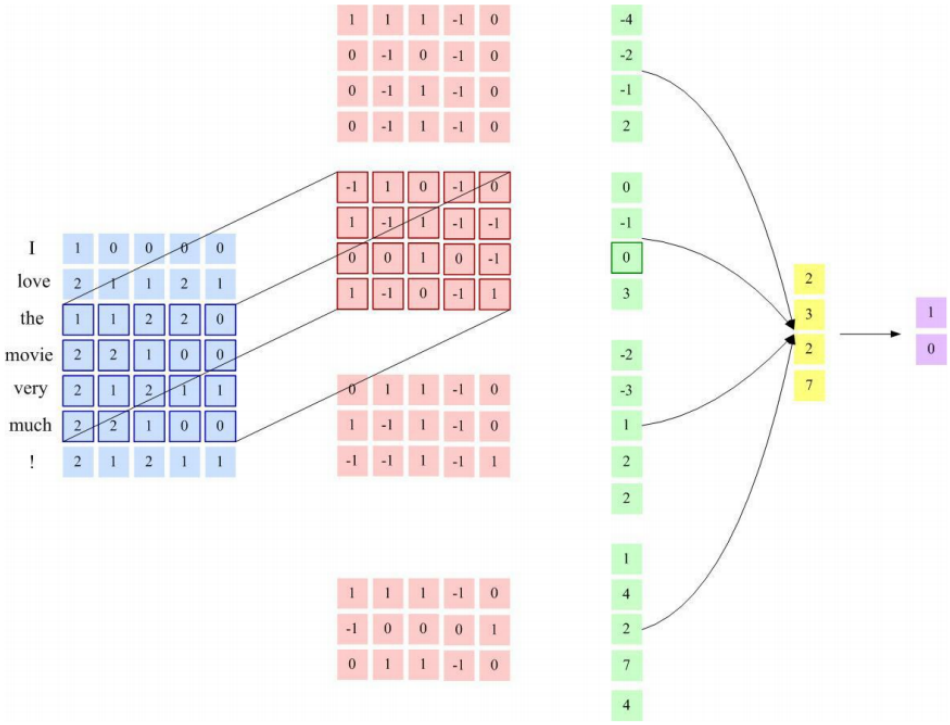

Figure 31: Architecture

References

[1] An overview of resnet and its variants. https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035.

[2] A simple guide to the versions of the inception network. https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202.

[3] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2016.

[6] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey, 2020.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[9] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

- http://cs231n.github.io/convolutional-networks/
- https://arxiv.org/pdf/1809.02165v1.pdf
- https://www.coursera.org/learn/convolutional-neural-networks/home/week/4
- http://www.phontron.com/class/nn4nlp2020/assets/slides/nn4nlp-03-cnn.pdf
- http://d2l.ai/chapter_convolutional-neural-networks/index.html