Ex7 - Deep Learning Questions - Computer Vision

Question 1
Explain each of these terms in a sentence or two:

1. Cross entropy
2. Residual connections
3. Adam optimizer
4. Cyclic learning rate
5. Dropout
6. Bottleneck layer
7. 1x1 convolution
8. DenseNet

Answer 1

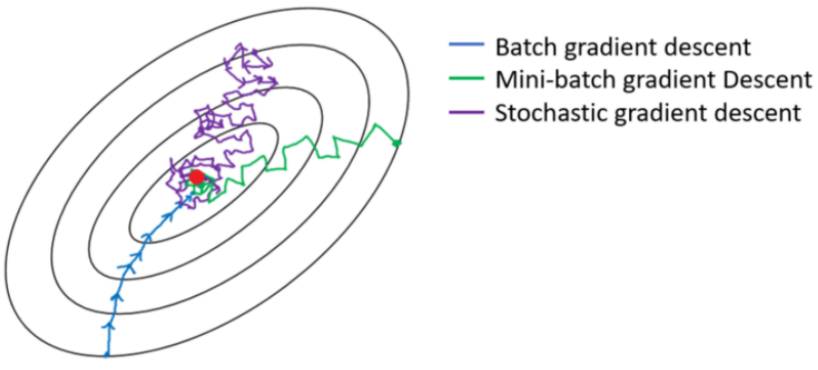
1. Cross entropy
    a. Cross entropy is a measure from the field of information theory, building upon entropy and generally calculating the difference between probability distributions.
    b. In ML it is often used as a loss function (a.k.a log loss) and sometimes for optimization, in the field of supervised classification problems (binary classes / multiclass problems).
2. Residual Connections
    a. Residual connections, a.k.a skip-connections are a way to provide another path for data to reach latter parts of the neural network by skipping some layers in the forward feed.
    b. Motivation: The training process of a neural network with residual connections is empirically shown to converge much more easily. Also helps to resolve the shattered gradients problem in deep networks.
3. Adam Optimizer
    a. Adam (adaptive moment estimation) - is an optimization algorithm for parameters in neural networks, it is an extension to stochastic gradient descent optimizer, and uses both the benefits of AdaGrad and RMSProp.
    b. The algorithm sets an adaptive learning rate for each parameter and keeps the exponentially decaying average of the past gradients when calculating the next step.
4. Cyclic learning rate
    a. Method for setting a dynamic learning rate (like schedulers), cyclic learning rate allows our learning rate to oscillate back and forth between the lower and upper bounds we pre-define.
    b. Motivation - The network may become stuck in either saddle points or local minima, and the low learning rate due to scheduler in later epochs may not be sufficient to break out. In addition, it helps to overcome poor initial choice in learning rate.
5. Dropout
    a. A regularization method which randomly zeroes some of the inputs / outputs of a NN layer with probability p.
    b. Usually used in FC layers at the deeper and final layers of the network. Rarely used for conv layers, not used in RNNs and in general it helps to avoid overfitting.
6. Bottleneck layer
    a. Generally, a bottleneck layer is a layer that has fewer nodes compared to the previous layers. It is a layer that is often used for dimensionality reduction.
    b. An example of it can be seen in Google's Inception network, bottleneck layers are added to reduce the number of channels in the network (by using 1x1 convolutions).
7. 1x1 Convolution
    a. 1X1 Convolution simply means the filter is of size 1X1, it is used to reduce the number of channels while introducing non-linearity, and it enables building deeper networks for that reason.
    b. Widely used in modern CNN architectures, like GoogleNet, ResNet and SqueezeNet
8. DenseNet
    a. A CNN architecture used for image classification that was published in 2017 and uses dense connection.
    b. In dense connections, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers. Each layer is receiving a “collective knowledge” from all preceding layers.
    c. Since each layer receives feature maps from all preceding layers, the network can be thinner and compact (less channels), and it was shown to be more accurate (compared to resnet for example).

Question 2
Explain the pros and cons of using small and large batch sizes.

Answer 2

- Large Batch size
  - Pros
    - As the batch is larger we get a more accurate estimation of the gradient in the “correct” direction (see reference image below)
  - Cons
    - Requires more memory
    - Longer training time (in certain cases - see explanation in the small batch size pros)
- Small batch size
  - Pros
    - Requires less memory (especially important if you are not able to fit large batches of the dataset in your machine's memory)
    - Faster training convergence times with mini-batches: m samples mini batch requires O(m) computation and memory but reduces the uncertainty in the gradient only by O(sqrt(m))
  - Cons
    - Less accurate the estimate of the gradient compared to full dataset batch (see reference image below)

Batch size convergence comparison reference:



Question 3

How many 3x3 filters are needed to replace a 7x7 kernel? Compare the number of parameters in each option.

Answer 3

Three 3x3 filters achieve the same overall effective receptive field in the original input as a single 7x7 filter and therefore can replace a single 7x7 filter.

Let’s compare the number of parameters (assuming we have 3 channels for input):

Formula for calculation:
$$N_{parameters} = ((F_{width} \times F_{height} \times N_{in\ channels}) + 1) \times N_{kernels}$$

One 7x7 filter:
$$N_{parameters} = ((F_{width} \times F_{height} \times N_{in\ channels}) + 1) \times N_{kernels} = ((7 \times 7 \times 3) + 1) \times 1 = 148$$

Three 3x3 filter:
$$N_{parameters} = ((F_{width} \times F_{height} \times N_{in\ channels}) + 1) \times N_{kernels} = ((3 \times 3 \times 3) + 1) \times 3 = 84$$

Some of the benefits of replacing 7x7 filter with three 3x3 are:

- The features that would be extracted will be highly local. This helps in capturing smaller, fine-grained features in the image.
- Using kernels sequentially (i.e increasing number of layers) allows the network to learn more complex features.
- We reduce the number of parameters so the network can be deeper and train faster

Question 4

You are training a neural network for classifying images on a custom dataset and it doesn't seem to learn anything. Describe your approach to solving the issue.

Answer 4

1. Firstly, I would try to review my entire data processing code
    a. I will review the image processing, will print some examples, will also verify that the labels were loaded correctly
    b. I'll verify that the data loaders for my samples work as expected and that the labels per image are stitched correctly
    c. Will inspect places where we use the data loaders and verify they are used correctly
2. Verify the network architecture code
    a. Printing the network summary
    b. Verify that all the layers I've built were connected correctly and as expected when we defined the NN class
    c. Verify stacked layers, activations, batch norms, strides, losses
3. Verify the training code and add prints as much as possible to detect problems in the training process (even within batches and not just for entire epochs)
    a. Look for NaNs (inputs / predictions)
    b. 0/0
    c. Sqrt of negative numbers
4. If we used a pre-trained network in the process
    a. I will try to verify, did I scale the images as the network expects (0,255 vs. 0,1 vs. -0.5,0.5)
5. If we built the network from scratch:
    a. Try to change the initial weights and retrain
    b. Loss
        i. Try to increase / decrease loss and retrain
        ii. Verify the loss we calculated is what expected (logits vs. probability)
    c. In general I will print the losses and examples for predictions and inputs, even per batch, to verify we get correct numbers and I don't get odd values / NaNs
6. If all of these did not work, I would restart the kernel, take a simple pre-trained network and try to apply transfer learning. I will try to debug each step in the flow and try to see where the problem is while attempting to get to a point where I see that we learned something.

Question 5
Mention the problems imbalanced datasets can cause to Deep Learning problems, and suggest a few ways to avoid them.

Answer 5

First of all, I would mention that I would try to focus on the problem statement, it all depends on the problem we are trying to solve and what that is important to us, for example if we try to predict if we need to buy a house (investment-wise), then we probably don't want to make mistakes of buying bad houses while we care less when our model predicts not to buy a house we should have.

Some ways to handle imbalance datasets:

- Evaluation metric -
  - In imbalanced datasets we need to carefully choose our evaluation metric, for example, accuracy will be a bad measurement, while precision / recall / f1 can work better.
- Class weights
  - we can define, based on the number of samples per class what is the weight we want, this is available in the loss function in PyTorch or in the training API in keras for example
- Over / under sampling
  - For example, in the above houses problem, we can decide to over sample the "buy house" samples (which means we randomly take such samples more than 1 time as input for training)
  - Similar approach is under sampling samples that we want to decrease weights for / that has many of the same label
- Advanced over / under sampling
  - If we have a low number of samples from a certain class and we want to increase it's weight, we can create augmentations of samples in the same class instead of just over sampling the same sample for example
  - Instead of under sampling randomly, if we know the data and familiar with the phenomenon in it and problems we have in specific areas (for example we are overfitted to specific class of subclass in the model) we can under sample this class only, we can also use clustering to identify such problems and potentially under / over sample from problematic clusters.
- Naive additional solution
  - If possible, and it is necessary to solve our problem, we can increase the dataset size and create a balanced dataset by collecting more samples and labelling them :)

Question 6
Given two networks: SRCNN (https://arxiv.org/pdf/1501.00092.pdf) and Unet (https://arxiv.org/pdf/1505.04597.pdf).
What is the receptive field of a pixel in each network?
You should consider one pixel (let's say the center pixel of the output) and go back to see what neighbors at the input image influence this pixel.

Answer 6

SRCNN
Going backward per pixel in the output we go through:

- 5x5 Convolution: which increases the receptive field by "padding" 2 pixels in each direction
- 9x9 Convolution: which increases the receptive field by "padding" additional 4 pixels in each direction

Meaning that the receptive field per pixel in the output of the network is 13x13 pixels - which are 169 pixels in the original image.

Unet

When talking about Unet, we have 2 sections in the process - The encoding and downsampling part, and the decoding and up sampling (up-convs) part.
We can see that the effective receptive field for a finite size input image is the entire input image.
This is due the exponential increase of the receptive field in each max-pool operation done to reduce the layers dimension by 2-fold.

Question 7
Given a standard CNN consists of convolutional layers and then fully-connected layers.
Explain what layer in CNN can reach a lot of parameters? How can we avoid it?

Answer 7

Usually the transfer from the CNN layers to the fully-connected layers will contain the largest number of parameters.
The reason is that the first FC layer input is composed from the multi-dimensional output of the convolutional layers which can get pretty big, as we need to flatten the output of the convolutional layer we have a high number of inputs for the FC layer.

We can avoid it by:

- Using pooling layers - To reduce the number of parameters after the convolutional layers, the pooling layers summarizes the features extracted in the convolutional layers so we can keep most of what the network learned while reducing the number of params greatly.
- Reducing the number of outputs in the first FC layer.

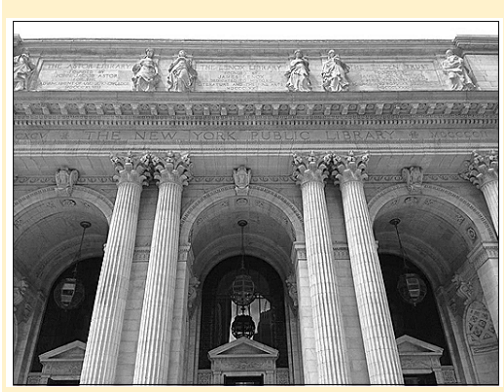
Question 8
What is the result of convolving an image X with a filter h = [-1 -1 -1; 0 0 0; 1 1 1].

Answer 8

It will be very similar to a bottom sobel filter, which is [-1 -2 -1; 0 0 0; 1 2 1]

The results will be inverted colors and colored lines will reflect edge detection of vertical lines where the bottom is emphasized.

For example (left - orig, right - the above filter):




Question 9
(a) What if all the weights are initialized with the same value?
(b) What happens if we set all the biases to be zero (ignoring the biases)?

Answer 9
(a) Same init value is usually bad, it means that all the neurons will have the same value and all the neurons will have the same gradient. We need to introduce asymmetry in the network. The same init value approach makes the model no better than a linear model.

(b) What happens if we set all the biases to be zero (ignoring the biases)?
If we initialize the weights randomly, setting the cias to 0 is a common practice, as weights already break the asymmetry. For ReLU, to reduce the risk of dying neurons, some initialize bias to be a small positive number (e.g 0.01).

Question 10
Write down two NN models (it can also be convolutional networks) that have (more or less) the same number of parameters, but with different power of computational. Explain your answer.

Answer 10

1st Model

Architecture:
- Convolution layer (input=28x28, kernel=5x5, kernels=32, padding=0, stride=1)
- Max pooling (input=24x24, pool=2x2, padding=1, stride=2)
- Output: (32, 12, 12)

Params:
- Convolution layer - parameters: 832, activation size: 25,088, output: (32, 24, 24)
- Max pooling - parameters: 0, activation size: 18,432, output: (32, 12, 12)

Number of parameters - 832 + 0 = 832
Number of activations - 25,088 + 18,432 = 43,520

2nd Model
Architecture:
- Convolution layer 1 (input=28x28, kernel=3x3, kernels=32, padding=0, stride=1)
- Convolution layer 2 (input=26x26, kernel=3x3, kernels=32, padding=0, stride=1)
- Max pooling (input=24x24, pool=2x2, padding=1, stride=2)
- Output: (32, 12, 12)

Params:
- Convolution layer 1 - parameters: 320, activation size: 25088, output: (32, 26, 26)
- Convolution layer 2 - parameters: 320, activation size: 21632, output: (32, 24, 24)
- Max pooling - parameters: 0, activation size: 18432, output: (32, 12, 12)

Number of parameters - 320 + 320 = 640
Number of activations - 25,088 + 21,632 + 18,432 = 65,152

The two neural networks have a similar number of parameters (832 vs. 640), but the computational power is different.
The reason is that in the first model we use a large filter instead of 2 smaller ones, in this case we have more parameters (weights) to optimize in our network but less layers overall.
The first model is less complex and with less layers so there are less computations and the power of computations is lower.

Open Questions
The two questions below are open questions. We would like to see the way you develop algorithms using deep learning. Your answers should be written (not code). You can use pseudo code if it is more convenient for you.

Question 11
You moved into a new apartment and you would like to make your front door smarter. Write an algorithm that alerts every time there is someone outside your door who is not a part of your family (you decide who is considered to be family).

Answer 11
Since we did not define it, let's say that we want to be alerted when there is a non-family member outside our door for at least 10 seconds.

Technically, in order to get the video signal that I'll discuss below and use it for classification I would use a camera with a motion sensor that can send the video seen in the camera to a server that I can access and apply my algorithm on, the camera will send videos with length of 10 seconds each time.

In order to solve the problem, I will split it into two separate problems.

<u>The first problem - Face detection</u>
I will implement the face detection by using a pretrained Multi-Task Cascaded Convolutional Neural Network (MTCNN) which provides state of the art results for face detection.
The only main concern here I can think of is that MTCNN can be slow for video with high FPS (frames per second), I can address this issue in multiple approaches:

- Either find another state-of-the-art pre-trained model that works faster
- Based on my camera FPS I can decide to use striding in the prediction (predict the bounding box only each X frames, so for example if striding is 3 the frame 0 bounding box will also be used for frames 1,2 - assuming that faces do not move too much between frames when we have high FPS)
- I can decide to limit it even furtherly: if we have files of 10 seconds I can decide to detect faces only X times per second (so we will sample frames and not use all of them).

In addition, I would also apply a small mechanism to avoid FPs that can occur only due to movement which does not involve someone coming to my door - based on the MTCNN output, I will decide if there is a face to classify.
So for example if we got 0 / low number of bounding boxes in X frames that the network inferred I would not continue to the recognition part as the clip probably initiated due to a movement not related to a person detected by the motion sensor and not because of a real person in front of my door.

<u>For the second problem - Face recognition</u>
After installing the local camera at the door and applying the solution for the first problem (face detection) I will try to build a dataset and collect a high number of detected faces from the camera that will represent our family members in a balanced manner, I will create X classes for X family members and an additional class for Non family members - so we will have a dataset built of X+1 classes (I did not use this as a binary classification problem in order to have better understanding of FPs and error analysis - for example, maybe I'm not good at detecting only one specific family member).

After building the dataset, I would apply transfer learning (I'll try to evaluate multiple models such as VGG-16 / Resnet / Inception / etc.) or any newer state-of-the-art models that can work for my problem (based on the performance that I'd like to get - how fast do I want to be alerted).

After training my model, I would use the trained model as a predictor on the output of the MTCNN first network (which will have Y detected faces frames per 10 second video from my camera), based on the Y outputs I get I will decide based on majority for example whether the Y frames included a family member or not.

Some problems that may appear in the second problem solution and how to handle them:

- Performance problems: Here I would take similar approach based on the first problem solution - I can limit the numbers of frames we predict on out of the 10 seconds video

- Not enough examples of non-family members: As my labels collection was based on the camera in my door, I assume that I won't have too many non-family members there, potentially this is the real "distribution" of faces that the camera sees and it can be fine but I can consider adding a lot of non family members images collected from the web and extract the faces using my MTCNN network (and use these faces for the non-family member class), this can also create problems as images in the web won't look the same as my usual camera images that might have specific lighting and pixels but it might be worth a try.

So finally I will describe the overall flow of the algorithm:

1. My camera with a motion sensor sends 10 second video to my server
2. In the server I run MTCNN on the video clip and extracts X frames with faces -
    a. If the predictor did not find at least X frames with faces I will not continue to the next steps
3. After extracting the X faces and resizing them to the correct shape my face recognition network expects, I will predict whether or not each of the X faces frames is a family member or not.
4. Per the X predictions I will take the majority (of is / isn't a family member), if the majority says: not a family member - I will generate an alert with the video and will send it to my device
    a. Just an idea to add here - I can also add a feedback loop to the algorithm, where I look on the video after the alert and say whether the algorithm was correct or not and save the video as additional labels that I can use for training when improving my model.

Question 12

These days, most of the parking lots have a system that recognizes the license plate number of an entering car. This helps to automatically open the gate in case that the driver paid before arriving at the exit gate. For this purpose, two cameras are needed - one at the entrance and one at the exit. Both capture the license plate number and translate it into a series of numbers.

Write an algorithm that for a given image containing a car with a license plate number, recognizes all the numbers in this plate. Pay attention that the angle and the location of the license plates can vary between images.

For simplicity: all the license plates have the same size (in the real world, not in the image), have 7 digits and have the same font. The plate is yellow with black digits. Some pictures are attached.

Moreover, no trucks or motorcycles can enter the parking lot.

If you have more assumptions, please write them down.

Answer 12

Same here, I would approach this problem with 2-3 separate models per problem that will run sequentially.

<u>First step - Plate localization</u>

To handle this first problem, given an image of a car, I would have used a pretrained network and transfer learning to predict 2 things:

1. Bounding box of the license plate
2. Landmarks of the 4 corners of the license plates

If possible, I would prefer to create a dataset based on Israeli license plate real examples but labelling can be very costly here, so I would start with existing datasets in the web and if we will see performance problems we will handle them later (we can use datasets in the web and changing the image into grayscale to avoid the problems of different background colors in different countries for example).

Second step - Skew correction

The reason for the 2 outputs above is that after the localization we will use the 4 anchors and the partial image of the license plate (the image inside the predicted bounding box) to straighten the license plate (due to the different shooting angles that can impact our license plate numbers recognition).
This can be done with classical approaches from vision, after we have the bounding box and the 4 anchors of the corners of the image, we can use perspective transform from four corner points which is covered by a method in the opencv python module to do it.

After this step, we will have a straightened version of the license plate from the original image.

Third step - Character recognition (OCR)

Same here, I would use a pretrained model, probably will try multiple CRNN models or even simple built in models for OCR tasks (there are some in keras for example), I would run all of them and will choose the one with the best results.
I would consider applying grayscale coloring to the input license plate patch and evaluate results again.

Other possible approach - using SSD - Single Shot MultiBox Detector (localization + classification done in a single pass of the network).