MA30279 Machine Learning 2 Assessed Coursework 1 (2023-24)
CN: 24075

Introduction: Provide a mathematical context for the problem. In particular:

(a) Specify in mathematical terms the desired criteria for this problem (e.g., how do you express in mathematical terms that "the resulting image C should look almost visually indistinguishable from B "?). [3]

Define a message tensor $A \in \mathbb{R}^{\tilde{\eta}_H \times \tilde{\eta}_W \times \tilde{\eta}_C}$ and a cover tensor $B \in \mathbb{R}^{\eta_H \times \eta_W \times \eta_C}$, with

$$\tilde{\eta}_H \leq \eta_H \quad , \quad \tilde{\eta}_W \leq \eta_W \quad \text{and} \quad \tilde{\eta}_C \leq \eta_C$$

Define the concatenating function $f(A, B)$ which produces a resulting tensor $C \in \mathbb{R}^{\eta_H \times \eta_W \times \eta_C}$. The tensor C represents the concealment of the tensor A completely contained in the tensor B. In order for the tensors C and B to look visually indistinguishable, it is required to train the appropriate function $f(A, B)$ to minimise the absolute difference between tensor B and C over the respective dimensions (i.e. minimise the absolute difference between each pixel of images B and C over each RGB channel). This results in the minimisation of the following Mean Error function (ME)

$$ME(B, C) = \frac{1}{\eta_H \eta_W \eta_C} \sum_{k=1}^{\eta_C} \sum_{j=1}^{\eta_W} \sum_{i=1}^{\eta_H} |f(A, B)_{ijk} - B_{ijk}| = \frac{1}{\eta_H \eta_W \eta_C} \sum_{k=1}^{\eta_C} \sum_{j=1}^{\eta_W} \sum_{i=1}^{\eta_H} |C_{ijk} - B_{ijk}|$$

Do you think deep learning is suitable in this context? Explain why or why not. [2]

Deep Learning is certainly suitable in this context. Given the large verity of possible images B and A, it would be difficult for a human to decide what features of B would be important to extract and select when attempting to hide image A effectively. Deep learning has critical advantages here; it is able to learn which features are important based on an increasingly large available data set. Also, since the features are not needed to be precisely designed, the features are learnt instead through optimisation. This will help make the network more robust at producing a good image C.

Unlike a few decades ago, there are many large image data sets available, possible examples include ImageNet and CIFAR. Hence, it would be possible to create an appropriate large data set for training and validation for this scenario. This makes deep learning more suitable because deep learning tends to perform better than more traditional methods when a large amount of data is available. This is even without potential scalability issues which could be present in non-DL techniques. There has been a lot of progress on scalabilty in deep learning over the past decade e.g. ResNet

Data handling: Explain how you can prepare and handle data for this problem. In particular:

Explain what are your input datapoints and your output data. Specify the dimensions and if there should be any specific limitations to the size of the images involved. [5]

In the mathematical formulation of the problem discussed above, the image A was designed to be either completely contained inside the image B, or be the same dimensions as the image B. There was also a restriction placed on the number of available channels for the image A with respect to B. For simplicity, it has been chosen to limit the size of A to the same dimensions as B with the 2D dimensions being a power of 2 (see 1). It is also assumed that both images A and B have only RBG channels ($\eta_C = 3$).

$$A, B \in \mathbb{R}^{\eta_H \times \eta_W \times \eta_C}, \quad \text{with} \quad \eta_W = \eta_H = 2^n \quad \text{for } n \in \mathbb{N}$$

Proceed to define a concatenated tensor for the input $\tilde{C} \in \mathbb{R}^{\eta_H \times \eta_W \times \eta_C^2}$ using the same notation as before. This tensor $\tilde{C}$ is created by 'placing tensor B on top of tensor A'. The output image, $C \in \mathbb{R}^{\eta_H \times \eta_W \times \eta_C}$, dimensions remain the same as B

Explain how you organise the datapoints in suitable datasets for training, validation and testing. [5]

Suppose that for $i = 1, ..., N$, we have N many images $D_i \in \mathbb{R}^{\eta_H \times \eta_W \times 3}$, with $D_j \neq D_i$ when $j \neq i$ and $D_i \in D$. So, the D dataset is a collection of N many unique images D_i all of the same dimensions. Then, since the ordering of pairs of D_i in D does not matter (i.e. If A is hidden and B is cover, this is a different pair from B hidden and A cover. But, A hidden, A cover not allowed), the number of pairings for $D_i \in D$ is N *choose 2*. Therefore, we then obtain a larger dataset $\tilde{D} = \{\{D_0, D_N\}, \ldots, \{D_i, D_j\}, \ldots, \{D_N, D_0\}\}, i \neq j$ that represents a collection of N *choose 2* many pairings for cover and hidden images. We can then perform a concatenation operation on all the pairs in the data set $\tilde{D}$, where we place D_i on top of D_j to form a concatenated image $\tilde{C}_{ij}$. The following dataset X is then formed of all the $\tilde{C}_{ij}$ images and cover images D_i.

$$X = \{\{\tilde{C}_{0N}, D_0\}, \ldots, \{\tilde{C}_{ij}, D_i\}, \ldots \{\tilde{C}_{N0}, D_N\}\} \qquad |X| = \frac{N(N-1)}{2}$$

This dataset X can then be separated into randomly selected training, validation and testing data sets. This is done by first computing a random permutation of X. Then, split X to create $\tilde{X}_{train}$ to hold $80 - 90\%$ of the data in X, with X_{test} holding the rest. Then, further reduce $\tilde{X}_{train}$ to X_{train} and $X_{validation}$, where $X_{validation}$ holds $10 - 20\%$ of the data in $\tilde{X}_{train}$ with X_{train} holding the rest. Hence, we have now organised data points into three datasets X_{train}, $X_{validation}$ and X_{test}.

Model: Devise a suitable model for the concealing operation. In particular:

Describe a suitable deep learning architecture (choosing between a feed-forward neural network and a convolution neural network), explaining why you chose it and why it is suitable for this application. [5]

When choosing a suitable good neural network architectures, it is a good idea to look at famous architectures which are good at performing similar tasks. CNNs have revolutionised image and signal processing tasks. U-Nets are specialised CNNs for image segmentation, they have a contracting and expansion U shaped symmetrical structure. This structure is particularly appealing for this problem. This is because the contracting path will learn the features of A and B which will allow it to hide the details of A inside the features of B. Then, on the expansion path, the lost fine grained spatial information is added to the high level spatial information via the skip connections; this detailing is very important for this problem since if some of the pixels of C were oddly coloured, it would be clear the image was a fake.

CNN architectures also have many attractive properties. Parameter sharing and sparse connectivity leads to a massive reduction in trainable parameters. Hierarchical feature extraction and max pooling will not only build translation invariance into the model, but will force the network to only keep the most important information. This is important for our problem because it will help the network better recognise different patterns in B and A, and it will reduce the processing and memory requirements of the model. Translation invariance is critical, since even if the images A and B are perhaps rotated, the same important features and details for hiding should be detected.

Specify width and depth of the network, activation function(s), and all the other necessary details to practically implement your network (e.g., if you chose a fully connected NN you should specify width and depth of the NN, or if you chose a CNN you should specify the filter size, stride and padding). Explain what considerations informed your strategy while designing your network. [5]

The U-Net architecture presented in the lecture notes was developed for biomedical imaging. These images often have a constant background, so loss of information at the boarders was not important. Information at the boarders is important for this application. Also, unlike the U-Net application in the notes, we need to preserve the 2D dimensions of the image and have RGB channels for the output image C. Hence, adaptations were made when proposing the model (1).

The padded convolution layers help the network learn low level and high level features, preserve the spatial information at the edges of the images, and keep the 2D dimensions the same. Padding is important for this application because the image C will be very distinguishable from B if the network cannot learn how to hide the pixels at

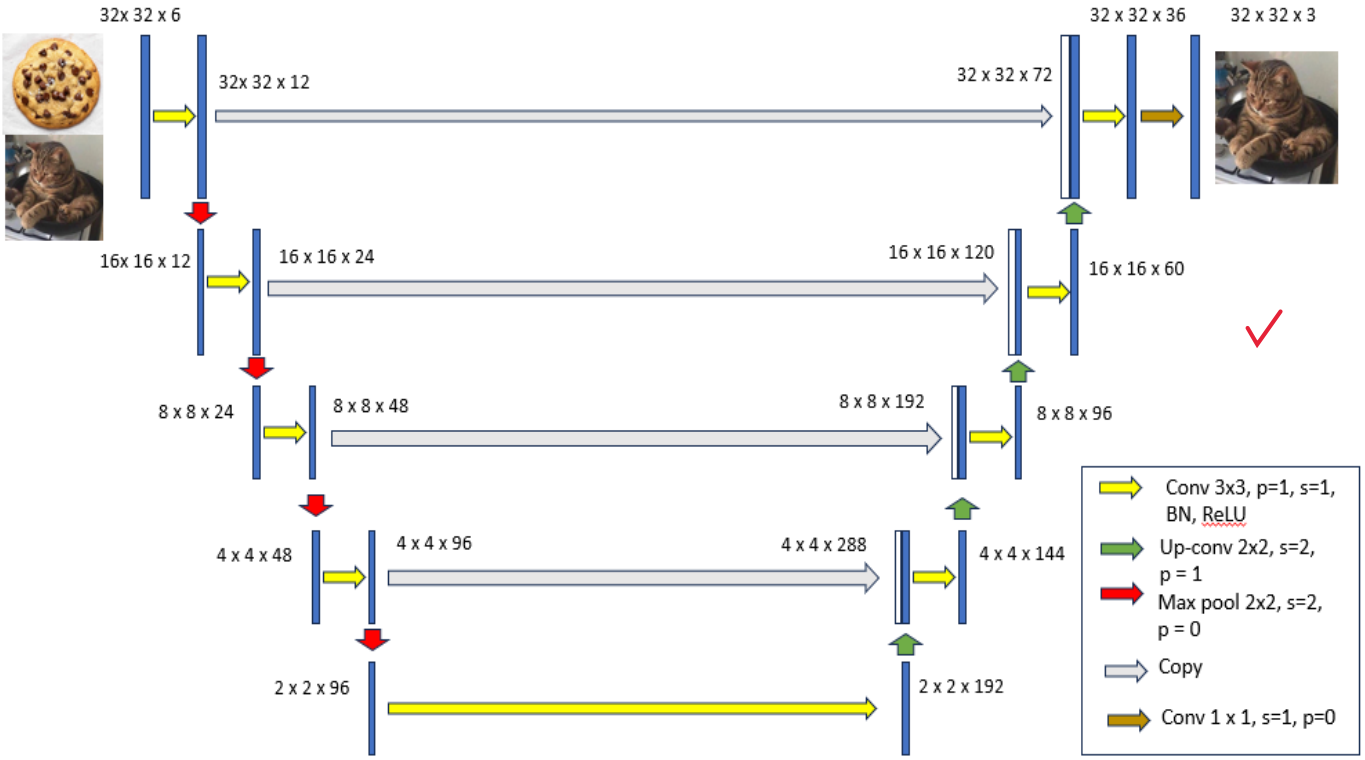

Figure 1: Proposed Architecture based on U-Net. $32 \times 32 \times 6$ example, but this is easy to extend to $2^n \times 2^n \times 6$, $n \in \mathbb{N}$

the boarder well. Nested convolution layers with max pooling aid hierarchical feature extraction and computational speed whilst reducing the learnable parameter space.

Adaptations where also made in the expansive path. Transpose convolutions aid slow recovery of the dimensions and allow the network to localise/unpack the important features from the contracting path. Skip connections, without 'crop', allows the network to recuberate lost detailing, including the detailing at the boarders. The final special channel-wise convolution was adapted to produce a C of the correct dimensions.

It is critical for the company to have an architecture that trains quickly and is stable. The activation function ReLU was used to reduce the vanishing gradient problem and increase rate of convergence. Skip connections also speed up optimisation by adding stability to the optimisation of gradients; they also mitigate the problem of vanishing and exploding gradients. Batch normalisation in between layers will not only significantly speed up training and ensure more stability, but will also enhance generalisation by reducing overfitting. This is important because we want the model to be able to be good at hiding lots of different styles of images taking advantage of different cover image features.

Provide a Python code snippet for the PyTorch class implementing the forward method of the neural network you devised. [8] Note: *torch.rand((batch_size, 6, 32, 32))*

```
class ProposedUNet_32x32(nn.Module):
    def __init__(self):
        super(ProposedUNet_32x32, self).__init__()
        self.downsample_1 = nn.Sequential(
            nn.Conv2d(6, 12, kernel_size=3, padding=1),
            nn.BatchNorm2d(12),
            nn.ReLU(inplace=True))
        self.downsample_2 = nn.Sequential(
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(12, 24, kernel_size=3, padding=1),
            nn.BatchNorm2d(24),
            nn.ReLU(inplace=True))
```

```
        self.downsample_3 = nn.Sequential(
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(24, 48, kernel_size=3, padding=1),
            nn.BatchNorm2d(48),
            nn.ReLU(inplace=True))
        self.downsample_4 = nn.Sequential(
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(48, 96, kernel_size=3, padding=1),
            nn.BatchNorm2d(96),
            nn.ReLU(inplace=True))
        self.downsample_5 = nn.Sequential(
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(96, 192, kernel_size=3, padding=1),
            nn.BatchNorm2d(192),
            nn.ReLU(inplace=True))
        self.upsample_1 = nn.Sequential(
            nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True))
        self.upsample_2 = nn.Sequential(
            nn.Conv2d(96+192, 144, kernel_size=3, padding=1),
            nn.BatchNorm2d(144),
            nn.ReLU(inplace=True),
            nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True))
        self.upsample_3 = nn.Sequential(
            nn.Conv2d(144+48, 96, kernel_size=3, padding=1),
            nn.BatchNorm2d(96),
            nn.ReLU(inplace=True),
            nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True))
        self.upsample_4 = nn.Sequential(
            nn.Conv2d(96+24, 60, kernel_size=3, padding=1),
            nn.BatchNorm2d(60),
            nn.ReLU(inplace=True),
            nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True))
        self.upsample_5 = nn.Sequential(
            nn.Conv2d(60+12, 36, kernel_size=3, padding=1),
            nn.BatchNorm2d(36),
            nn.ReLU(inplace=True),
            nn.Conv2d(36, 3, kernel_size=1, padding=0))
    def forward(self, x):
        downsample1 = self.downsample_1(x)
        downsample2 = self.downsample_2(downsample1)
        downsample3 = self.downsample_3(downsample2)
        downsample4 = self.downsample_4(downsample3)
        downsample5 = self.downsample_5(downsample4)
        upsample1 = self.upsample_1(downsample5)
        upsample2 = self.upsample_2(torch.cat([downsample4, upsample1], dim=1))
        upsample3 = self.upsample_3(torch.cat([downsample3, upsample2], dim=1))
        upsample4 = self.upsample_4(torch.cat([downsample2, upsample3], dim=1))
        upsample5 = self.upsample_5(torch.cat([downsample1, upsample4], dim=1))
        return upsample5
```

Training: Devise a suitable training procedure. In particular:

Specify the loss (and cost) function and the learnable parameters, explaining why you chose this loss (for example, why do you expect to work well for this problem?) and which favourable mathematical properties it has. [5]

Define the loss function function as

$$\mathcal{L}(B,C) = \frac{1}{\eta_H \eta_W \eta_C} \sum_{k=1}^{\eta_C} \sum_{j=1}^{\eta_W} \sum_{i=1}^{\eta_H} (f(A,B)_{ijk} - B_{ijk})^2 = \frac{1}{\eta_H \eta_W \eta_C} \sum_{k=1}^{\eta_C} \sum_{j=1}^{\eta_W} \sum_{i=1}^{\eta_H} (C_{ijk} - B_{ijk})^2$$

The cost function measures how well we are doing over the entire training set. Suppose we have m many i.i.d training examples, the cost function is then

$$\mathcal{J}(B,C) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(B_i, C_i)$$

The loss function has appealing properties. It does a good job at penalising pixels of C that are far away from the value of the corresponding pixel in B. Therefore, if C and B are very visually distinguishable, the loss function will be large, whereas if B and C are visually indistinguishable, then the loss value will be small. Because of the squaring, the value of the cost function will be dominated by pixels who's values are very different. This means learning will be accelerated. Also, minimising this loss function will lead to minimising the variance and bias. This will help reduce under fitting and over fitting in the model, which intern will help the model generalise better as long as the model complexity is right.

Specify which optimisation algorithm you will use for the training phase, choosing among the algorithms covered during the course, and explain why it is amenable to the loss function you devised. Discuss parameter choices related to the algorithm of your choice (e.g., how do you select the learning rate and how do you initialise the learnable parameters). [5]

We need to use an algorithm that is first order for back propagation and computational efficiency. Adam will be used because it has many superior properties compared to other first order optimisation methods in the notes. Unlike steepest/stochastic decent like methods, Adam adapts the learning rate based on the curvature of the objective function. It also uses momentum, which helps the algorithm avoid local minimum traps, and accelerate convergence. This is particularly important in this application since the objective function devised above is most likely very non-convex. For Adam, select $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\delta = 10^{-8}$. Tune the learning rate by initially setting $\alpha = 0.001$, and then increasing until the loss starts increasing rapidly (or code breaks). Ideally, we would like a higher learning rate so convergence is faster. Initialise the weights using small positive numbers form a uniform distribution. This will help speed up learning initially because we are using the ReLU activation function. We could also divide each column of the weights by the positive square root of the dimension. This could make learning more stable initially.

c) In addressing the above points, consider the various challenges training poses and discuss if any of the possible solutions we covered in the course (e.g., regularisation, early stopping, data augmentation) would be sensible and/or necessary within the deep learning strategy you are devising. [5]

Generalisation is certainly a challenge and data augmentation would be sensible to help with this. It is important that the company is able to choose any suitably defined images A and B, outside of the training dataset, and produce a good C image. The authors of the U-net architecture, which inspired the architecture (1) above, commented that data augmentation lead to very good performance with few labelled examples required. Computing transformations to the hidden image A, such as flipping or rotations, or adding noise should not affect the networks ability to produce an image C visually indistinguishable from a cover image B. This will be necessary to incorporate since deep learning models in general perform far better with very large datasets.

There are also training challenges with hyper-parameter selection, since they not only partly define the architecture of the algorithm, but also impact model performance and training time. For our implementation, the hyper-parameters we are particularly concerned about are the learning rate α, the number of mini-batches M, and the number of epochs n_{epoch}. One very efficient method to select the optimal hyper-parameters is by incorporating early stopping into the training algorithm. It allows us to measure trade off between training times and accuracy (particularly for different learning rates); reduce overfitting by computing prediction error on the validation set (improves generalisation error); and aids analysis on the effect of a wide range of different combinations of parameters. We may want to try a few different batch sizes M. This is because smaller batch sizes tend to speed up computation, but often add more instability, whereas larger batch sizes increase accuracy, but cost far more. Changing n_{epoch} is important since we aim to balance training times with squeezing performance and generalisation improvements.

The minimisation of non-convex objective functions is a challenge of particular importance to the deep learning community over the last decade. It is difficult to know whether you have reached a true global minimum, or whether the local minimum you have reached reduces the objective function sufficiently. Strategies to tackle this have been implemented. Adam mimics second order methods by adapting the learning rates mimicking the curvature of the objective function; it also uses momentum to escape local minimum traps. The model defined (1) is sufficiently deep that even if Adam gets trapped in a local minimum, the value of the objective function at the global minimum will likely not be much smaller.

(d) Provide a Python code snippet implementing the loop over epochs for training your network

using PyTorch. This should include the call to the cost function, back-propagation, and optimiser operations. There is no need to include code for printing or plotting results. [7]

```
model = ProposedUNet_32x32()
loss = nn.MSELoss()
optimiser = torch.optim.Adam(model.parameters(),lr=0.001, betas=(0.9, 0.999), eps=1e-08)
n_epochs = 100
top_val_loss = 0
for n in range(n_epochs):
    for tildeC, D in train_loader:
        model_pred = model.forward(tildeC)
        loss_value = loss(model_pred, D)
        loss_value.backward()
        optimiser.step()
        optimiser.zero_grad()
    tot_val_loss = 0.
    with torch.no_grad():
        for tildeC, D in val_loader:
            model_pred = model.forward(tildeC)
            loss_value = loss(model_pred, D)
            tot_val_loss = loss_value.item()*tildeC.shape[0] + tot_val_loss
    if tot_val_loss > top_val_loss:
        torch.save(model,'top_UNet_model')
        top_val_loss = tot_val_loss
        count = 0
    else:
        count += 1
    if count == 5:
        break
```

5. Assessment of results: Explain how you will evaluate the correct functioning of your strategy. In particular:

(a) Explain when you can consider the testing phase satisfactory. [2]

We will have tuned the hyper-parameters α, n_{epochs} and M to compute a model with the best performance on the training data $\tilde{X}_{train}$ i.e. a high training accuracy on X_{train} (not underfitted), which also has a high accuracy on the validation set $X_{validation}$ (not overfitted). The model should perform well on the test set X_{test} i.e. high accuracy/low loss. We will have also completed some sense check tests on a small subset of images to verify that the built C is indeed visually indistinguishable from B for images outside and inside $\tilde{X}_{train}$. This is to make sure that we are learning properly, and that the minimum loss we can produce using the optimised parameters, is actually a good enough trained model to produce a good C to the human eye.

(b) Discuss which quality metrics you would use in view of the learning strategy you devised. [3]

There are three quality metrics we could use to measure the accuracy of the learning stratagy devised:

- The Mean Error function defined in Q1(a) can be used to measure the accuracy between a cover image B and a learned cover image C. If the Mean Error was small, this would imply that the image pixel values are approximately the same.
- Measurement for structure differences: It would also be a good idea to measure any differences in structure between the images C and B. This is because we might be able to see some distortion between images B and C if the structure is not that similar i.e. for the images to be visually indistinguishable, the contrast, brightness and colour of the pixels must be similar for each image, B and C ,to the human eye.
- Relative error: Following ideas from the *Machine Learning 1* coursework, when measuring the difference between a image and a reconstructed image, the relative error was computed $||C - B||_F / ||B||_F$ as a metric to quantify the accuracy of C. A relative error close to zero shows high accuracy and therefore confidence in our reconstructed cover image C. This might be a more fair way of comparing model accuracy if larger images where used as inputs (can be done by adapting model specification appropriately).