Assignment 2

- **(v1.3)** Updated 3.2 to have simplified FCNN architecture for calculations.

- **(v1.2)** Further clarification to question 3.2: updated image of FCNN architecture to specify output channels in hyphens and added sentence to say the number after the hyphen specifies the number of output channels or units of a layer in the image. Removed the artefact from older question stating "Assume zero padding is used in convolutional layers such that the output dimension is equal to the input dimension", as output and input dimensions are specified by images and are not equal.

- **(v1.1)** Updated question 3.2 to REMOVE that all convolutional layerrs only have one output channel. Please refer to the numbers in the images to see the output channels in each convolutional layer.

Version: 1.1
Version Release Date: 2023-02-08

Deadline: Friday, Feb. 24th, at 11:59pm.

Submission: You must submit two files through MarkUs[1]: (1) a PDF file containing your writeup, titled `a2-writeup.pdf`, and (2) your code file `a2-code.ipynb`. There will be sections in the notebook for you to write your responses. Your writeup must be typed. Make sure that the relevant outputs (e.g. `print_gradients()` outputs, plots, etc.) are included and clearly visible.
See the syllabus on the course website[2] for detailed policies. You may ask questions about the assignment on Piazza[3]. *Note that 10% of the assignment mark (worth 2 pts) may be removed for lack of neatness.*

You may notice that some questions are worth 0 pt, which means we will not mark them in this Assignment. Feel free to skip them if you are busy. However, you are expected to see some of them in the midterm. So, we won't release the solution for those questions.

The teaching assistants for this assignment are Mica Consens, Amir Peimani and Yun-Chun Chen. Send your email with subject "*[CSC413] A2 ...*" to `csc413-2023-01-tas@cs.toronto.edu` or post on Piazza with the tag `a2`.

---

[1] `https://markus.teach.cs.toronto.edu/2023-01`
[2] `https://uoft-csc413.github.io/2023/assets/misc/syllabus.pdf`
[3] `https://piazza.com/class/lcp8mp3f9dl7lp`

Important Instructions

Read the following before attempting the assignment.

Overview and Expectations

You will be completing this assignment with the aid of large language models (LLMs) such as ChatGPT, text-davinci-003, or code-davinci-002. The goal is to help you (i) develop a solid understanding of the course materials, and (ii) gain some experience in using LLMs for problem solving. Think of this as analogous to (i) understanding the rules of addition and multiplication, and (ii) learning how to use a calculator. Note that LLMs may not be a reliable "calculator" (yet) — as you will see, GPT-like models can generate incorrect and contradicting answers. It is therefore important that you have a good grasp of the lecture materials, so that you can evaluate the correctness of the model output, and also prompt the model toward the correct solution.

Prompt engineering. In this assignment, we ask that you try to (i) solve the problems yourself, and (ii) use LLMs to solve a selected subset of them. You will "guide" the LLMs toward desired outcomes by typing *text prompts* into the models. There are a number of different ways to prompt an LLM, including direct copy-pasting LaTeX strings of a written question, copying function docstrings, or interactively editing the previously generated results. Prompting offers a natural and intuitive interface for humans to interact with and use LLMs. However, LLM-generated solutions depend significantly on the quality of the prompt used to steer the model, and most effective prompts come from a deep understanding of the task. You can decide how much time you want to spend as a university student vs. a prompt engineer, but we'd say it's probably not a good idea to use more than 25% of your time on prompting LLMs. See Best Practices below for the basics of prompt engineering.

What are LLMs good for? We have divided the assignment problems into the following categories, based on our judgment of how difficult it is to obtain the correct answer using LLMs.

- [**Type 1**] LLMs can produce almost correct answers from rather straightforward prompts, e.g., minor modification of the problem statement.

- [**Type 2**] LLMs can produce partially correct and useful answers, but you may have to use a more sophisticated prompt (e.g., break down the problem into smaller pieces, then ask a sequence of questions), and also generate multiple times and pick the most reasonable output.

- [**Type 3**] LLMs usually do not give the correct answer unless you try hard. This may include problems with involved mathematical reasoning or numerical computation (many GPT models do not have a built-in calculator).

- [**Type 4**] LLMs are not suitable for the problem (e.g., graph/figure-related questions).

Program trace and show your work. For questions in [**Type 1**] [**Type 2**] **, you must include the** *program trace* **(i.e., screenshots of your interaction with the model) in your submission**; the model output does not need to be the correct answer, but you should be able to *verify* the solution or find the mistake. Whereas for questions in [Type 3] [Type 4] , you do not have to include the program traces, but we encourage you to experiment with LLMs and get creative in the prompting; for problems labeled [EC] in these two categories, extra credit will be given to prompts that lead to the correct solution.

Grading.   We will be grading the assignments as follows.

- **Written Part**

  - For questions in [Type 1] [Type 2] , we require you to submit (*i*) **the program trace**, (*ii*) **critique of the model output** (is it correct? which specific part is wrong?), (*iii*) **your own solution**. You will not lose points if you cannot prompt the LLM to generate the correct answer, as long as you can identify the mistake. However, you will receive 0 point on the question if you do not include the program trace, even if you solve the problem by yourself; the goal of this is to help you familiarize with the use of LLMs.
    If you are confident that the model output is correct, you can directly use it as your own solution. Make sure you cite the model properly, that is, **include the model name, version (date), and url if applicable**.
  - For questions in [Type 3] [Type 4] , you will be graded based on the correctness of your own solution, and you do not need to include screenshots of the model output. Creative prompts that lead to correct model output will be rewarded.

- **Programming Part**

  - For writing questions labeled [Type 1] [Type 2] please submit (*i*) the program trace, (*ii*) critique of the model output, (*iii*) your own solution. The grading scheme is the same as the previous written part.
  - For coding questions labeled [Type 1] [Type 2] , submit (*i*) the program trace, (*ii*) your own solution. You will be graded based on the correctness of your own solution, that is, you will receive full marks if your code can execute and produce the desired outcome. From our experience, the most efficient way is to start with the LLM-generated code, which often gives the correct solution under minor modifications. Again, **make sure you cite the model properly**.

– For questions in [Type 3] [Type 4] , you will be graded based on the correctness of your own solution, and you do not need to include screenshots of the model output.

Written Assignment

What you have to submit for this part

See the top of this handout for submission directions. Here are the requirements.

- The zero point questions (in black below) will not be graded, but you are more than welcome to include your answers for these as well in the submission.

- For (nonzero-point) questions labeled [Type 1] [Type 2] you need to submit (*i*) **the LLM program trace**, (*ii*) **critique of the model output**, (*iii*) **your own solution**. You will receive 0 point on the question if you do not include the program trace, even if you solve the problem by yourself. Your own solution can be a copy-paste of the LLM output (if you verify that it is correct), but make sure you **cite the model properly**.

- For (nonzero-point) questions in [Type 3] [Type 4] you only need to submit your own written solution, but we encourage you to experiment with LLMs on some of them.

- If you attempt the extra credit problems [EC] using LLMs, include your program trace in the beginning of the writeup document.

For reference, here is everything you need to hand in for the first half of the PDF report `a2-writeup.pdf`.

- **Problem 1:** 1.1.1[Type 2] , 1.2.1[Type 1] , 1.2.2[Type 1]

- **Problem 2:** 2.1.1[Type 4] , 2.1.2[Type 1] , 2.1.3[Type 1] , 2.2.1[Type 2] , 2.2.2[Type 3] , 2.2.3[Type 2] , 2.3.1[Type 3] , 2.3.2[Type 3] , 2.3.3[Type 1]

- **Problem 3:** 3.1[Type 4] , 3.2[Type 3] , 3.3 [Type 1]

Useful prompts

You could start by naively copy-pasting the question and the context as the prompt, and try to improve the generated answers by trial and error. Raw LaTeX dumps are made available for the written questions to facilitate this process.

- https://uoft-csc413.github.io/2023/assets/assignments/a2_raw_latex_dump.tex

- https://uoft-csc413.github.io/2023/assets/assignments/a2_macros.tex

1 Optimization

This week, we will continue investigating the properties of optimization algorithms, focusing on stochastic gradient descent and adaptive gradient descent methods. For a refresher on optimization, refer to: `https://uoft-csc413.github.io/2023/assets/slides/lec03.pdf`.

We will continue using the linear regression model established in Homework 1. Given n pairs of input data with d features and scalar labels $(\mathbf{x}_i, t_i) \in \mathbb{R}^d \times \mathbb{R}$, we want to find a linear model $f(\mathbf{x}) = \hat{\mathbf{w}}^T \mathbf{x}$ with $\hat{\mathbf{w}} \in \mathbb{R}^d$ such that the squared error on training data is minimized. Given a data matrix $X \in \mathbb{R}^{n \times d}$ and corresponding labels $\mathbf{t} \in \mathbb{R}^n$, the objective function is defined as:

$$\mathcal{L} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2 \tag{1}$$

1.1 Mini-Batch Stochastic Gradient Descent (SGD)

Mini-batch SGD performs optimization by taking the average gradient over a mini-batch, denoted $\mathcal{B} \in \mathbb{R}^{b \times d}$, where $1 < b \ll n$. Each training example in the mini-batch, denoted $\mathbf{x}_j \in \mathcal{B}$, is randomly sampled without replacement from the data matrix X. Assume that X is full rank. Where $\mathcal{L}$ denotes the loss on $\mathbf{x}_j$, the update for a single step of mini-batch SGD at time t with scalar learning rate η is:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta}{b} \sum_{\mathbf{x}_j \in \mathcal{B}} \nabla_{\mathbf{w}_t} \mathcal{L}(\mathbf{x}_j, \mathbf{w}_t) \tag{2}$$

Mini-batch SGD iterates by randomly drawing mini-batches and updating model weights using the above equation until convergence is reached.

1.1.1 Minimum Norm Solution [2pt] [Type 2]

Recall Question 3.3 from Homework 1. For an overparameterized linear model, gradient descent starting from zero initialization finds the unique minimum norm solution $\mathbf{w}^*$ such that $X\mathbf{w}^* = \mathbf{t}$. Let $\mathbf{w}_0 = \mathbf{0}$, $d > n$. Assume mini-batch SGD also converges to a solution $\hat{\mathbf{w}}$ such that $X\hat{\mathbf{w}} = \mathbf{t}$. Show that mini-batch SGD solution is identical to the minimum norm solution $\mathbf{w}^*$ obtained by gradient descent, i.e., $\hat{\mathbf{w}} = \mathbf{w}^*$.

Hint: Be more specific as to what other solutions? Or is $\mathbf{x}_j$ or $\mathcal{B}$ contained in span of X? Do the update steps of mini-batch SGD ever leave the span of X?

1.2 Adaptive Methods

We now consider the behavior of adaptive gradient descent methods. In particular, we will investigate the RMSProp method. Let w_i denote the i-th parameter. A scalar learning rate η is used.

At time t for parameter i, the update step for RMSProp is shown by:

$$w_{i,t+1} = w_{i,t} - \frac{\eta}{\sqrt{v_{i,t}} + \epsilon} \nabla_{w_{i,t}} \mathcal{L}(w_{i,t}) \tag{3}$$

$$v_{i,t} = \beta(v_{i,t-1}) + (1 - \beta)(\nabla_{w_{i,t}} \mathcal{L}(w_{i,t}))^2 \tag{4}$$

We begin the iteration at $t = 0$, and set $v_{i,-1} = 0$. The term ϵ is a fixed small scalar used for numerical stability. The momentum parameter β is typically set such that $\beta \geq 0.9$ Intuitively, RMSProp adapts a separate learning rate in each dimension to efficiently move through badly formed curvatures (see lecture slides/notes).

1.2.1   Minimum Norm Solution [1pt] [Type 1]

Consider the overparameterized linear model $(d > n)$ for the loss function defined in Section 1. Assume the RMSProp optimizer converges to a solution. Provide a proof or counterexample for whether RMSProp always obtains the minimum norm solution.

Hint: Compute a simple 2D case. Let $\mathbf{x}_1 = [2, 1]$, $w_0 = [0, 0]$, $t = [2]$.

1.2.2   [0pt] [Type 1]

Consider the result from the previous section. Does this result hold true for other adaptive methods (Adagrad, Adam) in general? Why might making learning rates independent per dimension be desirable?

2   Gradient-based Hyper-parameter Optimization

In this problem, we will implement a simple toy example of *gradient-based hyper-parameter optimization*, introduced in Lecture 3.

Often in practice, hyper-parameters are chosen by trial-and-error based on a model evaluation criterion. Instead, *g*radient-based hyper-parameter optimization computes gradient of the evaluation criterion w.r.t. the hyper-parameters and uses this gradient to directly optimize for the best set of hyper-parameters. For this problem, we will optimize for the learning rate of gradient descent in a regularized linear regression problem.

Specifically, given n pairs of input data with d features and scalar label $(\mathbf{x}_i, t_i) \in \mathbb{R}^d \times \mathbb{R}$, we wish to find a linear model $f(\mathbf{x}) = \hat{\mathbf{w}}^\top \mathbf{x}$ with $\hat{\mathbf{w}} \in \mathbb{R}^d$ and a L2 penalty, $\lambda \|\hat{\mathbf{w}}_2^2\|$, that minimizes the squared error of prediction on the training samples. λ is a hyperparameter that modulates the impact of the L2 regularization on the loss function. Using the concise notation for the data matrix $X \in \mathbb{R}^{n \times d}$ and the corresponding label vector $\mathbf{t} \in \mathbb{R}^n$, the squared error loss can be written as:

$$\tilde{\mathcal{L}} = \frac{1}{n} \|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2 + \lambda \|\hat{\mathbf{w}}\|_2^2.$$

Starting with an initial weight parameters $\mathbf{w}_0$, gradient descent (GD) updates $\mathbf{w}_0$ with a learning rate η for t number of iterations. Let's denote the weights after t iterations of GD as $\mathbf{w}_t$, the loss as $\mathcal{L}_t$, and its gradient as $\nabla_{\mathbf{w}_t}$. The goal is the find the optimal learning rate by following the gradient of $\mathcal{L}_t$ w.r.t. the learning rate η.

2.1   Computation Graph

2.1.1   [0.5pt] [Type 4]

Consider a case of 2 GD iterations. Draw the computation graph to obtain the final loss $\tilde{\mathcal{L}}_2$ in terms of $\mathbf{w}_0, \nabla_{\mathbf{w}_0}\tilde{\mathcal{L}}_0, \tilde{\mathcal{L}}_0, \mathbf{w}_1, \tilde{\mathcal{L}}_1, \nabla_{\mathbf{w}_1}\tilde{\mathcal{L}}_1, \mathbf{w}_2, \tilde{\lambda}$ and η.

2.1.2   [0.5pt] [Type 1]

Then, consider a case of t iterations of GD. What is the memory complexity for the forward-propagation in terms of t? What is the memory complexity for using the standard back-propagation to compute the gradient w.r.t. the learning rate, $\nabla_\eta \tilde{\mathcal{L}}_t$ in terms of t?
Hint: Express your answer in the form of $\mathcal{O}$ in terms of t.

2.1.3   [0pt] [Type 1]

Explain one potential problem for applying gradient-based hyper-parameter optimization in more realistic examples where models often take many iterations to converge.

2.2   Optimal Learning Rates

In this section, we will take a closer look at the gradient w.r.t. the learning rate. To simplify the computation for this section, consider an unregularized loss function of the form $\mathcal{L} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2$. Let's start with the case with only one GD iteration, where GD updates the model weights from $\mathbf{w}_0$ to $\mathbf{w}_1$.

2.2.1   [1pt] [Type 2]

Write down the expression of $\mathbf{w}_1$ in terms of $\mathbf{w}_0$, η, $\mathbf{t}$ and X. Then use the expression to derive the loss $\mathcal{L}_1$ in terms of η.
Hint: If the expression gets too messy, introduce a constant vector $\mathbf{a} = X\mathbf{w}_0 - \mathbf{t}$

2.2.2   [0pt] [Type 3]

Determine if $\mathcal{L}_1$ is convex w.r.t. the learning rate η.
Hint: A function is *convex* if its second order derivative is positive

2.2.3   [1pt] [Type 2]

Write down the derivative of $\mathcal{L}_1$ w.r.t. η and use it to find the optimal learning rate η^* that minimizes the loss after one GD iteration. Show your work.

2.3   Weight decay and L2 regularization

Although well studied in statistics, L2 regularization is usually replaced with explicit weight decay in modern neural network architectures:

$$\mathbf{w}_{i+1} = (1 - \lambda)\mathbf{w}_i - \eta \nabla \mathcal{L}_i(X) \tag{5}$$

In this question you will compare regularized regression of the form $\tilde{\mathcal{L}} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2 + \tilde{\lambda}\|\hat{\mathbf{w}}\|_2^2$ with unregularized loss, $\mathcal{L} = \frac{1}{n}\|X\hat{\mathbf{w}} - \mathbf{t}\|_2^2$, accompanied by weight decay (equation 5).

2.3.1   [0.5pt] [Type 3]

Write down two expressions for $\mathbf{w}_1$ in terms of $\mathbf{w}_0$, η, $\mathbf{t}$, λ, $\tilde{\lambda}$, and X. The first one using $\tilde{\mathcal{L}}$, the second with $\mathcal{L}$ and weight decay.

2.3.2   [0.5pt] [Type 3]

How can you express $\tilde{\lambda}$ (corresponding to L2 loss) so that it is equivalent to λ (corresponding to weight decay)?
Hint: Think about how you can express $\tilde{\lambda}$ in terms of λ and another hyperparameter.

2.3.3   [0pt] [Type 1]

Adaptive gradient update methods like RMSprop (equation 4) modulate the learning rate for each weight individually. Can you describe how L2 regularization is different from weight decay when adaptive gradient methods are used? In practice it has been shown that for adaptive gradients methods weight decay is more successful than l2 regularization.

3   Convolutional Neural Networks

The last set of questions aims to build basic familiarity with convolutional neural networks (CNNs).

3.1 Convolutional Filters [0.5pt] [Type 4]

Given the input matrix $\mathbf{I}$ and filter $\mathbf{J}$ shown below, compute $\mathbf{I} * \mathbf{J}$, the output of the convolution operation (as defined in lecture 4). Assume zero padding is used such that the input and output are of the same dimension. What feature does this convolutional filter detect?

$$\mathbf{I} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad \mathbf{J} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \mathbf{I} * \mathbf{J} = \begin{bmatrix} ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? \end{bmatrix}$$

3.2 Size of Conv Nets [1pt] [Type 3]

CNNs provides several advantages over fully connected neural networks (FCNNs) when applied to image data. In particular, FCNNs do not scale well to high dimensional image data, which you will demonstrate below. Consider the following CNN architecture on the left:

CNN architecture   FCNN architecture
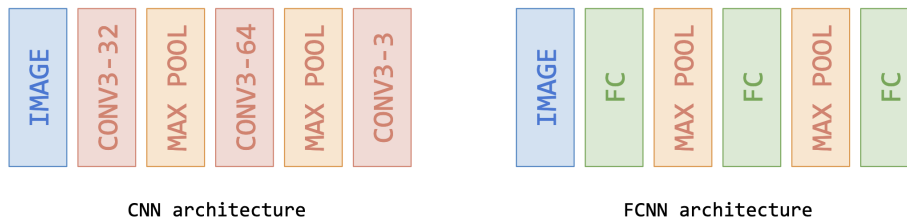
The input image has dimension 32×32 and is RGB (three channel). For ease of computation, all convolutional layers use 3×3 kernels and the output channels of each layer are described in the images. This means that the number after the hyphen specifies the number of output channels or units of a layer (e.g. Conv3-64 layer has 64 output channels). Assume zero padding is used in convolutional layers. Each max pooling layer has a filter size of 2×2 and a stride of 2. Furthermore, ignore all bias terms.

We consider an alternative architecture, shown on the right, which replaces convolutional layers with fully connected (FC) layers. Assume the fully connected layers do not change the output shape of inputs. For both the CNN architecture and the FCNN architecture, compute the total number of neurons in the network, and the total number of trainable parameters. You should report four numbers in total. Finally, name one disadvantage of having more trainable parameters.

9

3.3   Receptive Fields [0.5pt] [Type 1]

The receptive field of a neuron in a CNN is the area of the image input that can affect the neuron (i.e. the area a neuron can 'see'). For example, a neuron in a 3×3 convolutional layer is computed from an input area of 3×3 of the input, so it's receptive field is 3×3. However, as we go deeper into the CNN, the receptive field increases. One helpful resource to visualize receptive fields can be found at: `https://distill.pub/2019/computing-receptive-fields/`.

List 3 things that can affect the size of the receptive field of a neuron and briefly explain your answers.

Programming Assignment

What you have to submit for this part

See the top of this handout for submission directions. Here are the requirements.

- The zero point questions (in black below) will not be graded, but you are more than welcome to include your answers for these as well in the submission.

- For (nonzero-point) writing questions labeled [Type 1] [Type 2] you need to submit (*i*) **the program trace**, (*ii*) **critique of the model output**, (*iii*) **your own solution**. You will receive 0 point on the question if you do not include the program trace, even if you solve the problem by yourself. **Make sure you cite the model properly**.

- For (nonzero-point) coding questions labeled [Type 1] [Type 2] you need to submit (*i*) **the LLM program trace** (in the writeup PDF), (*ii*) **your own solution**.

- For (nonzero-point) questions in [Type 3] [Type 4] you only need to submit your own solution, but we encourage you to experiment with LLMs on some of them.

- If you attempt the extra credit problems [EC] using LLMs, include your program trace in the beginning of the writeup document.

For reference, here is everything you need to hand in:

- This is the second half of your PDF report `a2-writeup.pdf`. Please include the solutions to the following problems. You may choose to export `a2-code.ipynb` as a PDF and attach it to the first half of `a2-writeup.pdf`. Do not forget to append the following program traces/screenshots to the end of `a2-writeup.pdf`:

  - **Problem 4**:
    * 4.1 [0.5pt] [Type 1]
      · program trace
      · critique of the model output
      · code for model `PoolUpsampleNet` (screenshot or text)
    * 4.2 [0.5pt] [Type 4]
      · visualizations and your commentary
    * 4.3 [1.0pt] [Type 3]
      · your answer (6 values as function of **NIC**, **NF**, **NC**)
  - **Problem 5**:
    * 5.1 [0.5pt] [Type 1]

· program trace

· critique of the model output

· code for model `ConvTransposeNet` (screenshot or text)

* 5.2 [0.5pt] [Type 4]

· your answer, 1 plot figure (training/validation curves)

* 5.3 [0.5pt] [Type 3]

· your answer

* 5.4 [0.5pt] [Type 3]

· your answer

– **Problem 6**:

* 6.1 [0.5pt] [Type 1]

· program trace

· critique of the model output

· code for model `UNet` (screenshot or text)

* 6.2 [0.5pt] [Type 4]

· your answer, 1 plot figure (training/validation curves)

* 6.3 [1.0pt] [Type 3]

· your answer

• Your code file `a2-code.ipynb`

Introduction

This assignment will focus on the applications of convolutional neural networks in various image processing tasks. The starter code is provided as a Python Notebook on Colab (`https://colab.research.google.com/github/uoft-csc413/2023/blob/master/assets/assignments/a2-code.ipynb`). First, we will train a convolutional neural network for a task known as image colourization. Given a greyscale image, we will predict the colour at each pixel. This is a difficult problem for many reasons, one of which being that it is ill-posed: for a single greyscale image, there can be multiple, equally valid colourings. In the second half of the assignment, we will perform fine-tuning on a pre-trained object detection model.

Image Colourization as Classification

In this section, we will perform image colourization using three convolutional neural networks (Figure 1). Given a grayscale image, we wish to predict the color of each pixel. We have provided

a subset of 24 output colours, selected using k-means clustering[4]. The colourization task will be framed as a pixel-wise classification problem, where we will label each pixel with one of the 24 colours. For simplicity, we measure distance in RGB space. This is not ideal but reduces the software dependencies for this assignment.

We will use the CIFAR-10 data set, which consists of images of size 32x32 pixels. For most of the questions we will use a subset of the dataset. The data loading script is included with the notebooks, and should download automatically the first time it is loaded.

Helper code for Section 4 is provided in `a2-code.ipynb`, which will define the main training loop as well as utilities for data manipulation. Run the helper code to setup for this question and answer the following questions.

4 Pooling and Upsampling

4.1 [0.5pt] [Type 1]

Complete the model `PoolUpsampleNet`, following the diagram in Figure 1a. Use the PyTorch layers `nn.Conv2d`, `nn.ReLU`, `nn.BatchNorm2d`[5], `nn.Upsample`[6], and `nn.MaxPool2d`. Your CNN should be configurable by parameters `kernel`, `num_in_channels`, `num_filters`, and `num_colours`. In the diagram, `num_in_channels`, `num_filters` and `num_colours` are denoted **NIC**, **NF** and **NC** respectively. Use the following parameterizations (if not specified, assume default parameters):

- `nn.Conv2d`: The number of input filters should match the second dimension of the *input* tensor (e.g. the first `nn.Conv2d` layer has **NIC** input filters). The number of output filters should match the second dimension of the *output* tensor (e.g. the first `nn.Conv2d` layer has **NF** output filters). Set kernel size to parameter `kernel`. Set padding to the `padding` variable included in the starter code.

- `nn.BatchNorm2d`: The number of features should match the second dimension of the output tensor (e.g. the first `nn.BatchNorm2d` layer has **NF** features).

- `nn.Upsample`: Use `scaling_factor = 2`.

- `nn.MaxPool2d`: Use `kernel_size = 2`.

Note: grouping layers according to the diagram (those not separated by white space) using the `nn.Sequential` containers will aid implementation of the `forward` method.

---

[4]https://en.wikipedia.org/wiki/K-means_clustering
[5]https://gauthamkumaran.com/batchnormalization/amp/
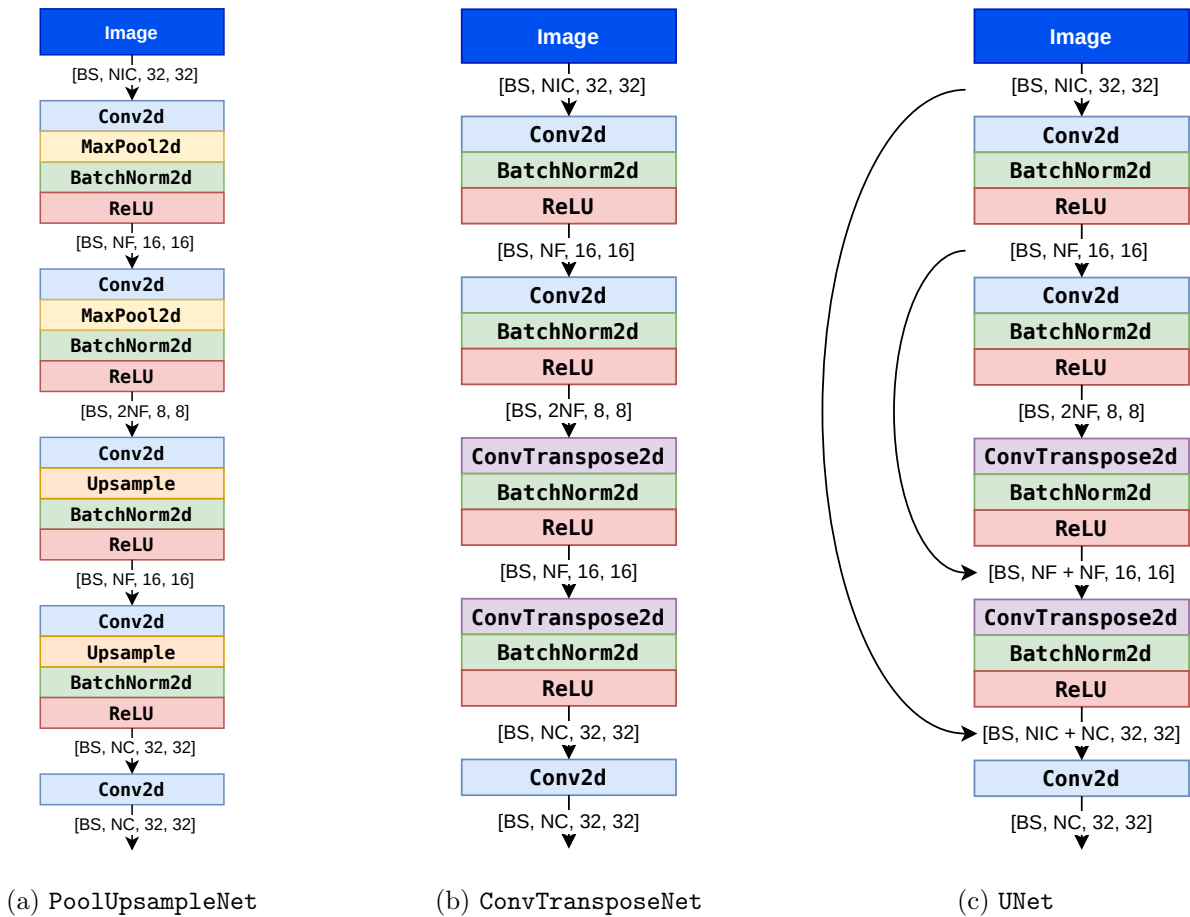[6]https://machinethink.net/blog/coreml-upsampling/

Figure 1: Three network architectures that we will be using for image colourization. Numbers inside square brackets denote the shape of the tensor produced by each layer: **BS**: batch size, **NIC**: num_in_channels, **NF**: num_filters, **NC**: num_colours.

4.2   [0.5pt] [Type 4]

Run main training loop of `PoolUpsampleNet`. This will train the CNN for a few epochs using the cross-entropy objective. It will generate some images showing the trained result at the end. Do these results look good to you? Why or why not?

4.3   [1.0pt] [Type 3]

Compute the number of weights, outputs, and connections in the model, as a function of **NIC**, **NF** and **NC**. Compute these values when each input dimension (width/height) is doubled. Report all 6 values.

   Note:

1. Please ignore biases when answering the questions.

2. Please ignore `nn.BatchNorm2d` when answering the number of weights, outputs and connections, but we still accept answers that do.

   Hint:

1. `nn.Upsample` does not have parameters (this will help you answer the number of weights).

2. Think about when the input width and height are both doubled, how will the dimension of feature maps in each layer change? If you know this, you will know how dimension scaling will affect the number of outputs and connections.

5   Strided and Transposed Dilated Convolutions [2 pts]

For this part, instead of using `nn.MaxPool2d` layers to reduce the dimensionality of the tensors, we will increase the step size of the preceding `nn.Conv2d` layers, and instead of using `nn.Upsample` layers to increase the dimensionality of the tensors, we will use *transposed* convolutions. Transposed convolutions aim to apply the same operations as convolutions but in the opposite direction. For example, while increasing the stride from 1 to 2 in a convolution forces the filters to skip over every other position as they slide across the input tensor, increasing the stride from 1 to 2 in a transposed convolution adds "empty" space around each element of the input tensor, as if reversing the skipping over every other position done by the convolution. We will be using a `dilation rate of 1` for the transposed convolution. Excellent visualizations of convolutions and transposed convolutions have been developed by Dumoulin and Visin [2018] and can be found on their GitHub page[7].

---

[7]`https://github.com/vdumoulin/conv_arithmetic`

5.1   [0.5pt] [Type 1]

Complete the model `ConvTransposeNet`, following the diagram in Figure 1b. Use the PyTorch layers `nn.Conv2d`, `nn.ReLU`, `nn.BatchNorm2d` and `nn.ConvTranspose2d`. As before, your CNN should be configurable by parameters `kernel`, `dilation`, `num_in_channels`, `num_filters`, and `num_colours`. Use the following parameterizations (if not specified, assume default parameters):

- `nn.Conv2d`: The number of input and output filters, and the kernel size, should be set in the same way as Section 4. For the first two `nn.Conv2d` layers, set `stride` to 2 and set `padding` to 1.

- `nn.BatchNorm2d`: The number of features should be specified in the same way as for Section 4.

- `nn.ConvTranspose2d`: The number of input filters should match the second dimension of the *input* tensor. The number of output filters should match the second dimension of the *output* tensor. Set `kernel_size` to parameter `kernel`. Set `stride` to 2, set `dilation` to 1, and set both `padding` and `output_padding` to 1.

5.2   [0.5pt] [Type 4]

Train the model for at least 25 epochs using a batch size of 100 and a kernel size of 3. Plot the training curve, and include this plot in your write-up.

5.3   [0.5pt] [Type 3]

How do the results compare to Section 4? Does the `ConvTransposeNet` model result in lower validation loss than the `PoolUpsampleNet`? Why may this be the case?

5.4   [0.5pt] [Type 3]

How would the `padding` parameter passed to the first two `nn.Conv2d` layers, and the `padding` and `output_padding` parameters passed to the `nn.ConvTranspose2d` layers, need to be modified if we were to use a kernel size of 4 or 5 (assuming we want to maintain the shapes of all tensors shown in Figure 1b)? Note: PyTorch documentation for `nn.Conv2d`[8] and `nn.ConvTranspose2d`[9] includes equations that can be used to calculate the shape of the output tensors given the parameters.

---

[8]https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html
[9]https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html

5.5   [0pt] [Type 4]

Re-train a few more `ConvTransposeNet` models using different batch sizes (e.g., 32, 64, 128, 256, 512) with a fixed number of epochs. Describe the effect of batch sizes on the training/validation loss, and the final image output quality. You do *not* need to attach the final output images.

6   Skip Connections

A skip connection in a neural network is a connection which skips one or more layer and connects to a later layer. We will introduce skip connections to the model we implemented in Section 5.

6.1   [0.5pt] [Type 1]

Add a skip connection from the first layer to the last, second layer to the second last, etc. That is, the final convolution should have both the output of the previous layer and the initial greyscale input as input (see Figure 1c). This type of skip-connection is introduced by Ronneberger et al. [2015], and is called a "UNet". Following the `ConvTransposeNet` class that you have completed, complete the `__init__` and `forward` methods of the `UNet` class in Section 6 of the notebook. Hint: You will need to use the function `torch.cat`.

6.2   [0.5pt] [Type 4]

Train the model for at least 25 epochs using a batch size of 100 and a kernel size of 3. Plot the training curve, and include this plot in your write-up.

6.3   [1.0pt] [Type 3]

How does the result compare to the previous model? Did skip connections improve the validation loss and accuracy? Did the skip connections improve the output qualitatively? How? Give at least two reasons why skip connections might improve the performance of our CNN models.

References

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.