Deep Learning and Neural Networks
Assignment 2

Due: 16/10/2022, 11:55pm

Do not leave this to the last minute! Each coding section will take a while to run and you need to allocate time to analyse results.

General Comments.

- Please submit your report along with the code (Jupyter notebook is preferred) as a single zip file. The minimum your submission has will be: report (as PDF), code (as Jupyter notebook), and converted code (as .py file).
- Include your name and student number in the filename for both the zip file and PDF. Do not send doc/docx files.
- Include your name and email address on the report and use a single column format when you prepare your report.
- Please ensure all of your results are included in your report. Marking is based on what is in the report. This includes plots, tables, code screen-shots etc.
- Make sure you answer questions in full and support any discussions with relevant additional information/experiments if necessary.

Late submission. Late submission of the assignment will incur a penalty of 10% for each day late. That is with one day delay, the maximum mark you can get from the assignment is 90 out of 100, so if you score 99, we will (sadly) give you 90. Assignments submitted with more than a week delay will not be assessed. Please apply for special consideration for late submission as soon as possible (*e.g.*, documented serious illness).

Note from ECE4179/5179/6179 Team. The nature of assignments in ECE4179/5179/6179 is different from many other courses. Here, we may not have a single solution to a problem. Be creative in your work and feel free to explore beyond questions.

Good Luck

Question:	1	2	3	4	5	6	Total
Points:	10	5	15	30	15	15	90
Score:							

Note: The DAE task has bonus marks that will be applied to your continuous assessment (2% of unit total)

Calculation Exercises

For the calculation exercises, ensure that you document any calculations/working-out. You can use a word editor to format your answers properly. You will receive a 0 if you do not provide calculations/working-out.

Multilayer Perceptron (MLP)

1. Consider the MLP network in Figure 1. Table 1 shows the value of each of the parameters of the network. The activation functions for all neurons are shown in Table 2. The neurons in the input layer, *i.e.* neuron 1 and neuron 2, do not use any activation function and simply pass in the input to the network. All the neurons in the hidden layer use a ReLU activation function, *i.e.*, $a_3(x) = a_4(x) = a_5(x) = \text{ReLU}(x) = \max(0, x)$. The neuron in the output layer uses a Sigmoid activation function, *i.e.*, $a_6(x) = \sigma(x) = 1/\big(1 + \exp(-x)\big)$. Answer the following questions.

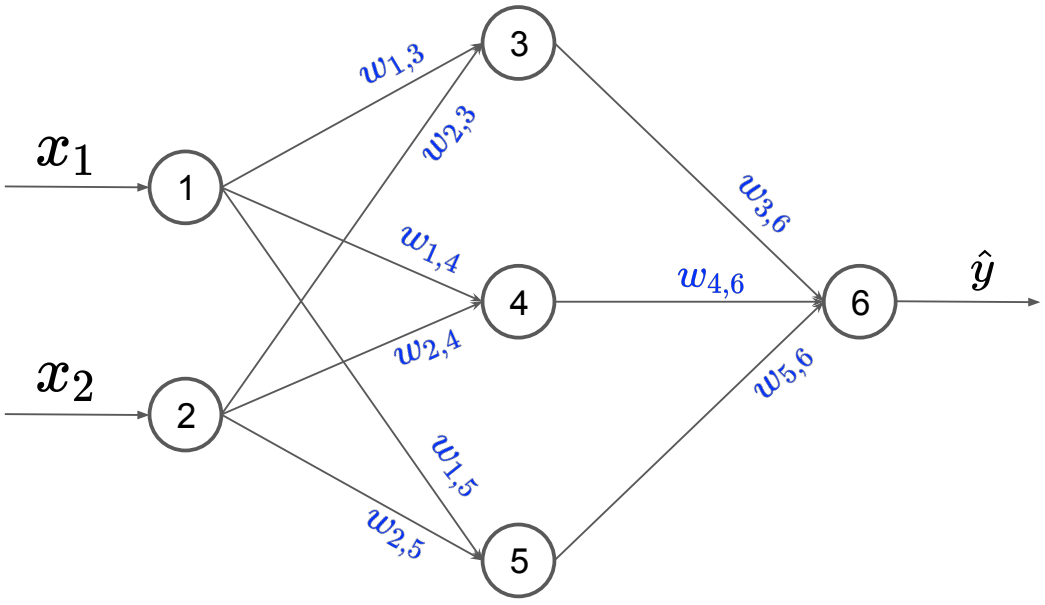

Figure 1: An MLP with one hidden layer (Question 1).

Parameter	Value
$w_{1,3}$	0.1
$w_{1,4}$	0.3
$w_{1,5}$	-0.2
$w_{2,3}$	0.6
$w_{2,4}$	0.4
$w_{2,5}$	0.0
$w_{3,6}$	-0.5
$w_{4,6}$	-0.3
$w_{5,6}$	0.8

Table 1: Parameter values of the MLP (Figure 1).

Neuron	Activation function
a_1	None
a_2	None
a_3	ReLU
a_4	ReLU
a_5	ReLU
a_6	Sigmoid

Table 2: Activation functions of the MLP (Figure 1).

1.1. [2 points] Compute the output of the network for $\boldsymbol{x} = \big(x_1, x_2\big)^\top = \big(1, 2\big)^\top$

1.2. [1 point] Assume the label of $\boldsymbol{x} = \big(x_1, x_2\big)^\top = \big(1, 2\big)^\top$ is $y = 0$. If we use the Binary Cross Entropy (BCE) loss to train our MLP, what will be the value of the loss for $\big(\boldsymbol{x}, y\big)$?

1.3. [1 point] Now assume the label of $\boldsymbol{x} = \big(x_1, x_2\big)^\top = \big(1, 2\big)^\top$ is $y = 1$. For BCE loss, what will be the value of the loss for $\big(\boldsymbol{x}, y\big)$? Do you expect the loss to be bigger or smaller compared to the previous part? Why? Explain your answer and your observation.

1.4. [3 points] Assume the learning rate of the SGD is $\text{lr} = 0.01$. For a training sample $\boldsymbol{x} = \big(x_1, x_2\big)^\top = \big(1, 2\big)^\top$ and $y = 1$, obtain the updated value of $w_{3,6}$.

1.5. [3 points] Using the assumptions from the previous part (*i.e.*, the learning rate of the SGD is $\text{lr} = 0.01$, the training sample is $\boldsymbol{x} = \big(x_1, x_2\big)^\top = \big(1, 2\big)^\top$ and $y = 1$), obtain the updated value of $w_{2,5}$.

Activation Function

2. [5 points] Consider the following activation function:

$$z = \begin{cases} \exp(x) - \exp(-x), & -1 \leq x \leq 1 \ , \\ \exp(1) - \exp(-1), & x > 1 \ , \\ \exp(-1) - \exp(1), & x < -1 \ . \end{cases} \tag{1}$$

We stack 1,000 of z, to form $z_{1000} = \underbrace{z \circ z \circ \cdots \circ z}_{1000 \text{ times}}$, where $\circ$ denotes function composition. What would be the response of z_{1000} to $x \in \mathbb{R}$? You need to discuss the behaviour of $z_{1000}(x)$ for $x \in \mathbb{R}$. We plot the activation function in Equation (1) along with the 45-degree line in Figure 2 for your convenience.

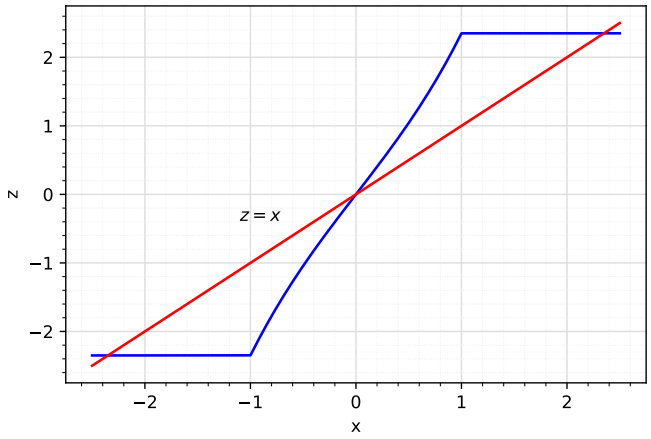

Figure 2: Activation function for Question 2.

Coding Exercises

It is recommended that you code up these exercises with the provided skeleton notebook. The results you obtain from the notebook need to be placed into a written report (ie. word doc), and discussions need to be added within this report. Remember to submit the PDF version of your report (and also your python notebook and .py file).

Denoising autoencoder (DAE) with MNIST

Whenever you measure a signal, noise creeps in. Denoising (aka noise reduction) is the process of removing noise from a signal and is a profound and open engineering problem. In this exercise, you will learn and implement a Denoising AutoEncoder (DAE) to remove additive Gaussian noise from images.

Autoencoders. An autoencoder is a neural model that is trained to attempt to copy its input to its output. Internally, it has a hidden layer $\boldsymbol{z}$ that describes a code used to represent the input. To be precise, an autoencoder realizes two functions/networks, a function f_{enc} to transform the input $\boldsymbol{x} \in \mathbb{R}^n$ to a new representation $\boldsymbol{z} \in \mathbb{R}^m$, followed by a function f_{dec}, which converts the new representation $\boldsymbol{z}$ back into the original representation. We call these two functions the encoder and the decoder (see Figure 3 for an illustration).

You may ask yourself if an autoencoder succeeds in simply learning to set $f_{\text{dec}}\big(f_{\text{enc}}(\boldsymbol{x})\big) = \boldsymbol{x}$ everywhere, then what is it useful for? The short answer to that is, the new representation $\boldsymbol{z}$ learned by the autoencoder captures useful information about the structure of the data in a compact form that is friendly to ML algorithms. But that is not all. In deep learning, we can borrow the idea of autoencoding and design many useful solutions, denoising being one.

DAE. In a DAE, instead of showing $\boldsymbol{x}$ to the model, we show a noisy input as $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \epsilon$. The noise ϵ is task specific. If you are working with MRI images, this noise should model the noise of an MRI machine. If you are working on speech signals, the noise could be the babble noise recorded in a cafe. What you will ask your DAE to do is now to generate clean samples, *i.e.* $\hat{\boldsymbol{x}} = \boldsymbol{x}$ (see Figure 3 for an illustration). So in essence, knowing about the problem and associated noise will enable you to simulate it. This will give you a very task-specific solution, which is in many cases desirable.

3. [15 points] In this section, you will implement a DAE to reduce the amount of noise within the dataset. Table 3 shows the hyperparameters and network architecture that you will be using.

Parameter	Value	Description
λ	1e-3 (0.001)	Learning rate
Weight decay	1e-5 (0.00001)	Weight decay in optimizer
# Encoder layer(s)	1	The number of layers in the encoder
# Neurons	{128}	Number of neurons in the encoder
# Decoder layer(s)	{*???*}	The number of layers in the decoder
# Neurons	{*???*}	Number of neurons in the decoder
Activation function	ReLU	*Use* ***"F" module (Functional)***
Loss function	*???*	For the entire DAE
Optimiser	Adam	For the entire DAE

Table 3: Hyperparameters for the DAE model. Note that some of the hyperparameters have been replaced with "*???*"

The tasks for this section are the following:

- Task D1: Create the dataloader for MNIST. Visualise a few samples.

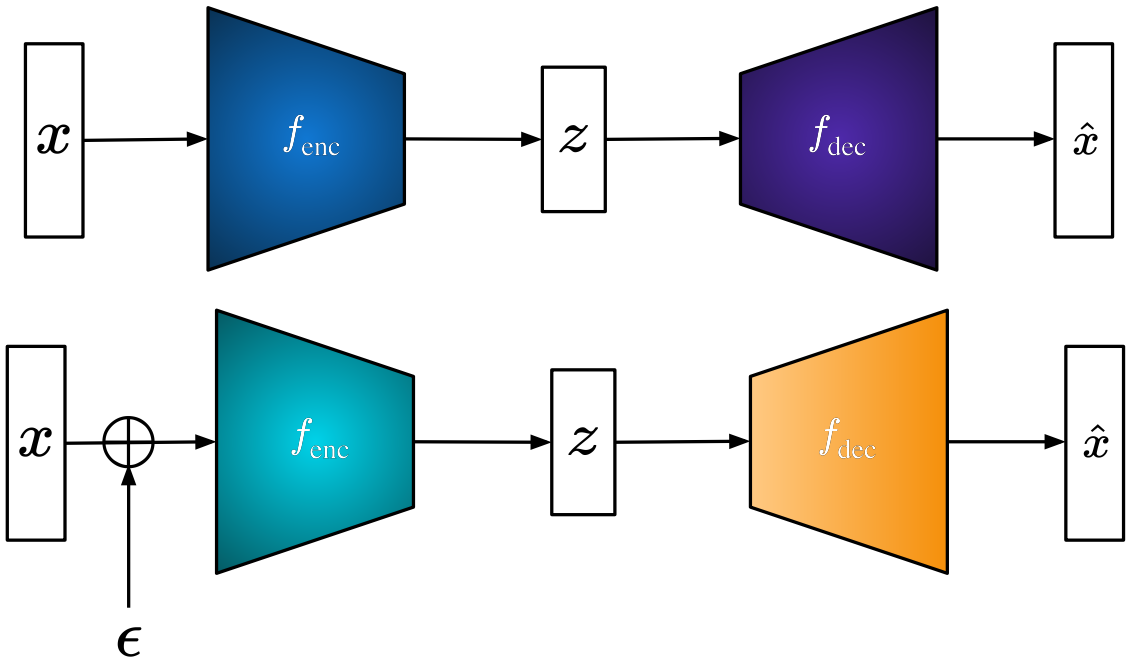

Figure 3: **Top.** An autoencoder is comprised of two subnetworks, an encoder and a decoder. In its vanilla form, the goal is to extract useful strcuture and information about the input. **Bottom.** In a DAE, you feed the network with a noisy input and train the model to produce clean outputs.

- Task D2: Design your network architecture for the DAE.
- Task D3: Train and evaluate the DAE results.
- Task D4: Visualise predictions along with the clean data.
- Task D5: Discuss both the DAE training results and the visualised predictions of the outputs.

Note: To create the noisy data, you will inject random noise into the data samples during the *training_step* within your network. This is shown in the skeleton code. You will also need to reshape the output predictions to visualize the output image.

Bonus marks [2% of unit total]

For the DAE section, you can earn an additional 2% (out of 50%) towards your continuous assessment. Your continuous assessment mark will not exceed the total maximum mark (it does not add to your overall marks).

To earn the bonus marks, you must complete at least two of the given ideas to some satisfactory degree. The ideas/improvements must be related to auto encoders **AND** the MNIST dataset. If you choose to implement your own improvement, it must be something worthwhile (not just simply changing λ for example). The results must be shown, and if you end up any external references, please cite it here and in your report.

Here are a few ideas to choose from:

- Improve the design of the network (hence improve the performance)
- Handle various noise types (speckle noise, Gaussian noise, inpainting etc.)
- Instead of downsampling to a smaller latent space, can you increase it while reducing noise?
- Any other improvements related to this task

MLP vs. CNN Skip this if you are studying ECE5179/6179. Go to *Why my CNN says this is a dog?*

4. [30 points] This section requires you to compare MLPs against CNNs on an image classification problem. We will be using the SVHN (Street View House Number) dataset. This section assumes you have the fundamental principles in applying the modelling process. You will need to create your dataloaders and two models (one MLP, one CNN). You will also be comparing their performance in several ways. The tasks are split as follows:

   - Task C1 Download the SVHN Train and Test datasets.
   - Task C2 Visualise a few training samples.
   - Task C3 Define an MLP and a CNN model.
   - Task C4 Train and evaluate both models.
   - Task C5 Visualise the results for both using validation accuracies.
   - Task C6 Discuss the performance and number of parameters used for each model.

MLP - SVHN

The hyperparameters for the MLP model can be found in Table 4. Train your MLP model and compare it against the CNN version.

Parameter	Value	Description
λ	1e-1 (0.1)	Learning rate
# Hidden layers	3	The layers you will be coding
# Neurons	{256, 64, 10}	Number of neurons in each layer
Activation function	ReLU	*Use **"F" module (Functional)***
Loss function	*???*	Loss function for a classification problem
Optimiser	SGD	Optimiser used for training

Table 4: Hyperparameters for the MLP model. Note that some of the hyperparameters have been replaced with "*???*"

CNN - SVHN

The entire CNN architecture can be found in the coloured box below. The convolutional blocks are similar to the previous lab, and we will only be using 3 convolutional blocks in this section.

image ($\boldsymbol{x} \in \mathbb{R}^{B\times C\times H\times W}$) $\rightarrow$ **conv-blk1** $\rightarrow$ **conv-blk2** $\rightarrow$ AvgPool $\rightarrow$ **fc** $=$ $\hat{\boldsymbol{y}}$

End-to-end from input image to predicted class for the CNN. Use (3,3) for AvgPool.

The box below shows the structure of one convolutional block. Tables 5 and 6 highlight the details of each block/layer.

Conv1 $\rightarrow$ **ReLU** $\rightarrow$ **Conv2** $\rightarrow$ **ReLU**

The convolutional layers and activation functions within one convolutional block (*conv-blk*).

Note: For task C6, ensure that you add in additional discussion as specified in the skeleton code.

Add in discussions that compare the two models for this dataset. Include additional analysis as specified within the lab document.

Name	Type	in	out	kernel size	padding	stride
Conv1	Conv2D	c_i	c_o	3×3	0	1
Conv2	Conv2D	c_o	c_o	3×3	1	1

Table 5: Details of the convolutional block

Name	in	out
conv-blk1	3	64
conv-blk2	64	128
fc1	1152	10

Table 6: Block structure

Why my CNN says this is a dog?

Imagine you have trained a CNN to distinguish dogs from cats. You measure the accuracy of your model on some validation data, and it is 95%. Is it enough to deploy your model? There are many stories where just checking training/validation accuracies led to disasters (see this infamous story). So what else can you do to ensure that your trained model does what it is supposed to do? You may want to use explanation techniques to understand how your model works (to a certain degree).

One way of explaining the predictions of a neural network is called "Occlusion Sensitivity". Basically, one hides a portion of the input (*e.g.*, a rectangular patch of an image) and measures the softmax score for the partially occluded input. If the hidden portion contains essential information about the input, the softmax score should drop significantly (see Figures 4 and 5 for an illustration). By sliding the patch-mask over the input, one can identify and visualize which parts of the image are more important for classification. For more details, check Section 4.2 of the paper "Visualizing and Understanding Convolutional Networks" by Zeiler and Fergus.

5. [15 points] We would like to apply the occlusion sensitivity on a trained ResNet model. Write a function to slide a patch of size $p \times p$ over an image with a stride s (each move is s elements away). Zero out the elements inside the patch and measure the confidence of the ResNet (*i.e.*, the softmax score) for its top prediction. Create a heatmap using the softmax score and plot it next to the image. Use the provided images along with images of your choice. Does ResNet behave the way you expect it to?

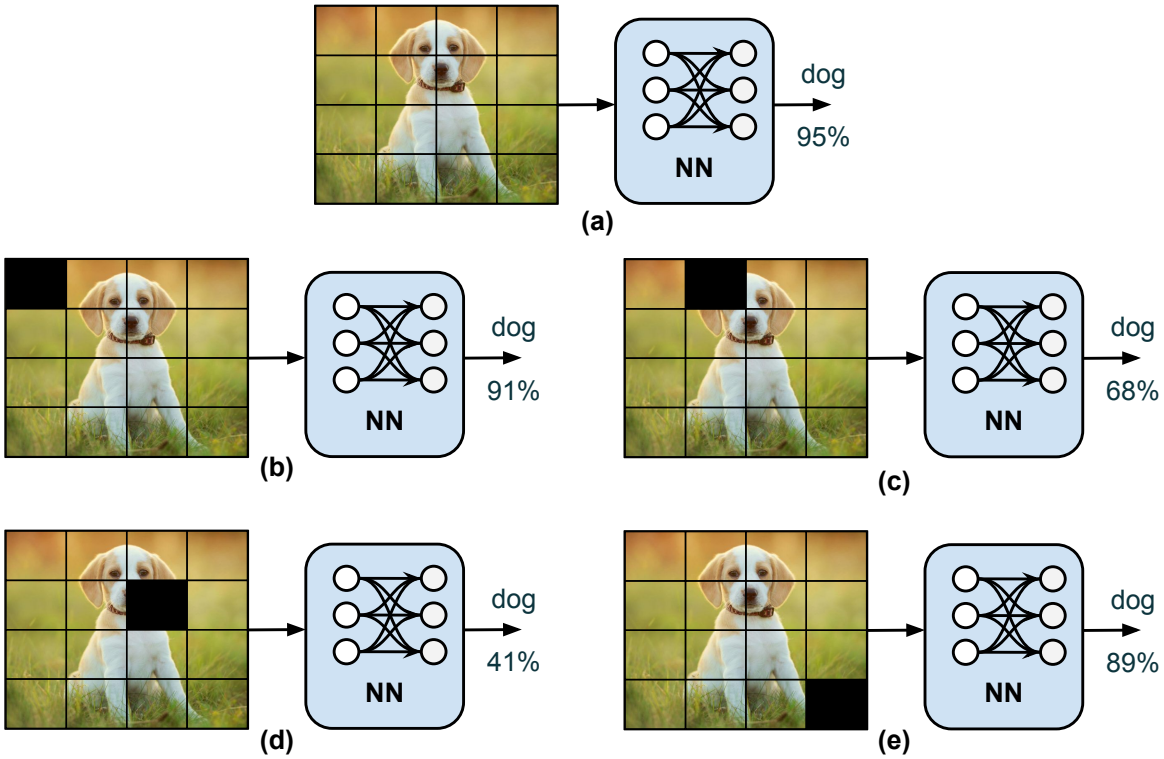

Figure 4: An example of explaining a CNN's output via occlusion sensitivity. If your CNN classifies dogs, you expect by applying the occlusion sensitivity, softmax scores of the class dog drop when the occlusion mask covers parts of the dog's face. This shows that your CNN mainly focuses on these areas and is not just picking up clues from the background to recognize dogs.

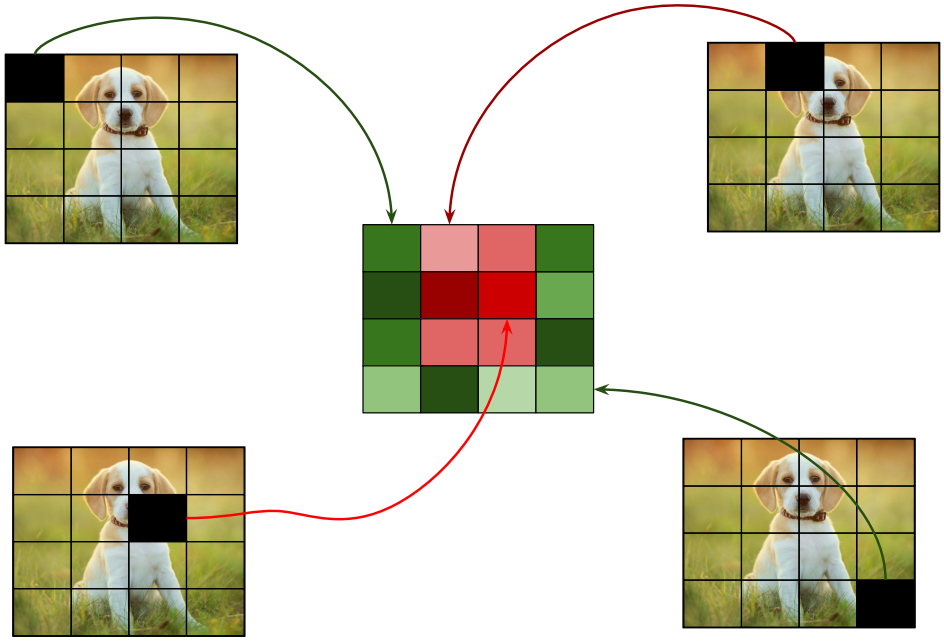

Figure 5: The confidence of a CNN (*i.e.*, softmax score) over images with predefined occlusions (here, black patches at certain locations) will provide us with a tool to deduce what is important for the model in terms of its predictions.

Face Landmarks Detection (FLD) with Transfer Learning

6. [15 points] In this final coding task, you will be applying your knowledge to a new application. You will be using a method called transfer learning (see Figure 6 which utilises a pre-trained network (in our case a ResNet18) for a new task.

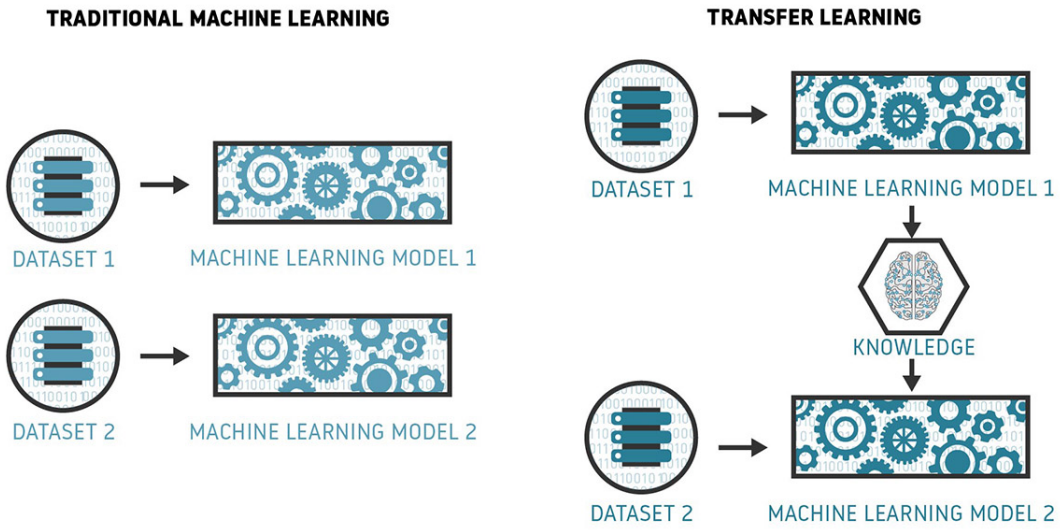

Figure 6: Original ResNet18 Architecture

Pre-trained networks are usually trained on large datasets, so the learnt features are quite general. What does this mean? Features are usually derived through the convolutional layers, and as you stack more convolutional layers, the features that are learnt by the network become more complex and more useful – but also require more data for training.

The ResNet18 model (Figure 7) we will be using has already been trained on millions of RGB images. What we can do with this is to utilise all the convolutional blocks of this architecture (and their learnt parameters), and only replace the fully connected layer with the fully connected layer we want.

Now the question is, *why do we do this?* Remember, the whole purpose of the convolutional layers is to extract useful features. The useful features are inputs to the final fully connected layer (before softmax). The fully connected layer then weighs up each of the extracted feature in order to make a classification prediction. This means we do not have to train a model from scratch to learn these features! We only need to learn how to combine them in the appropriate way to solve our problem at hand.

So for this section, you will be loading a pre-trained ResNet18 model, and you will replace the final fully connected layer with a modified fully connected layer. See Table 7 for the number of neurons required.

The dataset contains photos of people's faces and have associated "landmarks" that identify a part of a person's face. Each person has 68 landmarks (corresponding to an (x,y) coordinate on the image). The dataset itself is quite small, so we will only use 10% of the training data as validation. The tasks for this section are divided as follows:

- Task FLD1 Create a dataloader for the face landmarks dataset. Visualize a few samples of the data along with the landmarks.
- Task FLD2 Load a pre-trained ResNet18 network. Adapt the code to include transfer learning.
- Task FLD3 Train and test the model.
- Task FLD4 Visualise the predicted landmarks for some of the images.

Although we do not do it here, the face landmarks could be used for purposes such as head pose estimation, identifying gaze direction, detecting facial gestures, and swapping faces etc. This is why it would be useful to have a model that can detect face landmarks ! Ensure that you have added in required discussions.

Note the following hints:

- Ensure that the weights of the ResNet18 model are *frozen*[1]. You can do this by training your model for one epoch, and compare the original weight of the model with your new model (just look at the first conv1 layer). You can also verify that the 'requires_grad' attributes of the parameters are set to False, which is essentially what tells your PyTorch optimiser that these parameters shouldn't be updated.
- Ensure you pick the appropriate loss. Remember, we are dealing with coordinates here, so it is not a classification problem!

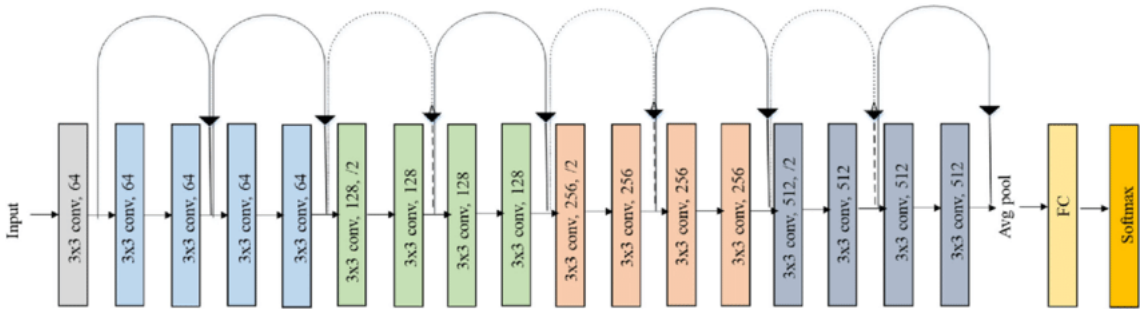

Figure 7: Original ResNet18 Architecture

Parameter	Value	Description
λ	5e-4 (0.0005)	Learning rate
FC layer	{Input, Output} = {512, 136}	Number of neurons in each layer
Loss function	???	Remember, you are dealing with coordinates!
Optimizer	SGD	Optimizer for the entire model

Table 7: Hyperparameters for the pre-trained ResNet18. Note that some of the hyperparameters have been replaced with "???".

Hopefully we have given you a taste of what deep learning is like over the last few months! Although it may have felt like a long and arduous journey, we hope that the content we have delivered and the insights you have gained will be helpful in the coming years.

[1] See this discussion thread for freezing model weights