Deep Learning and Neural Networks Assignment

Due: 14/10/2024, 4:30PM

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 Notebooks as a single zip file.
- Include your name and student number in the filename for both the zip file and PDF. Do not send doc/docx files. These will NOT be assessed.
- 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 working out for the calculation exercises are visible in your report.
- Please ensure all of the code outputs are visible in the notebook upon submitting, with all the discussion questions answered.

Academic integrity and the Use of Generative AI

Academic integrity

Every assignment submission will be screened for any collusion and/or plagiarism. Breaches of academic integrity will be investigated thoroughly and may result in a zero for the assessment and interviews with the plagiarism officers at Monash University.

Late submission policy and grading guidelines

Late submissions

Late assignment submission 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% of the total assignment mark, so if you score 95%, we will (sadly) give you 90%. Assignment 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).

The learning outcomes for this assessments are:

- Applying forward and backpropagation in a deep learning setting
- Furthering your understanding of linear regression models
- Adapting MLPs to denoise data
- Implementing modular programming for repeated convolutional blocks
- Deploying transfer learning for landmark detection
- Utilise pre-trained audio-to-vector models for speech recognition

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	7	Total
Points:	10	5	20	20	20	15	10	100
Score:								

Calculation Exercises - General Comments

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.

Calculation Exercise 1: 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.*, $a_6(x) = \sigma(x) = 1/\big(1 + \exp(-x)\big)$. For cross entropy loss, **ensure you use $\log_e$**. Answer the following questions.

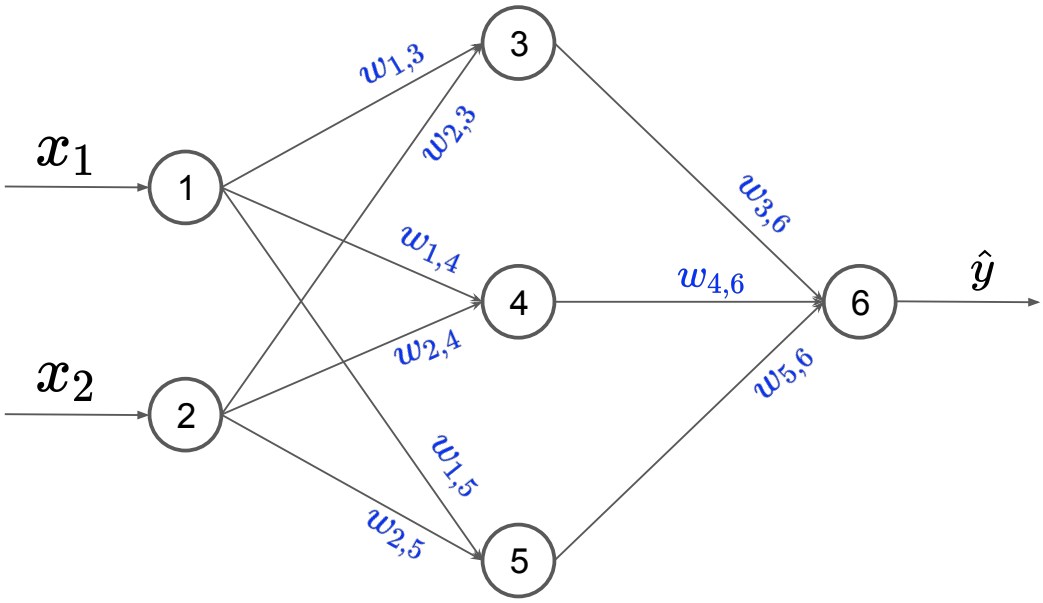

Figure 1: An MLP with one hidden layer (Question 1).

Parameter	Value
$w_{1,3}$	0.4
$w_{1,4}$	-0.2
$w_{1,5}$	-0.3
$w_{2,3}$	0.0
$w_{2,4}$	0.7
$w_{2,5}$	0.1
$w_{3,6}$	-0.2
$w_{4,6}$	0.5
$w_{5,6}$	-0.6

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 $\text{lr} = 0.1$. For a training sample $\boldsymbol{x} = \big(x_1, x_2\big)^\top = \big(1, 2\big)^\top$ and $y = 0$, 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 $\text{lr} = 0.1$, the training sample is $\boldsymbol{x} = \big(x_1, x_2\big)^\top = \big(1, 2\big)^\top$ and $y = 0$), obtain the updated value of $w_{2,5}$.

Calculation Exercise 2: 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.

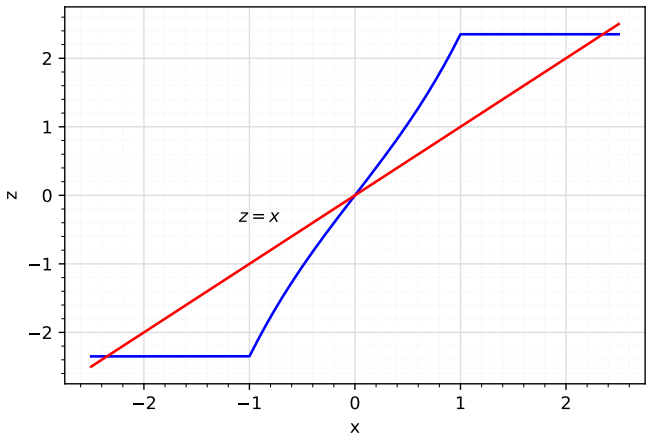

Figure 2: Activation function for Question 2.

Coding Exercises - General Comments

You will code up these exercises with the provided skeleton notebooks. The results and discussions you obtain from the notebook can stay within the notebook without going into a separate PDF. Each task will have its own discussion questions.

Linear and Logistic Regression

3. [20 points] You are probably thinking, these models again! Okay you are right, but there will be additional ideas presented here.

   In the first subtask, you will explore the effects of outliers in your dataset and how that can affect your model performance. You will apply a weighted linear regression model so that the resulting model is more robust.

   In the second subtask, you will be exploring ways to model non-linearities in the dataset with a linear regression model by transforming your inputs to a linearised space and performing classification in the transformed space with a technique called decay learning rate.

Task 1.1. Weighted linear regression

In this coding exercise, we study the iterative weighted linear regression model. In some problems, every data point might not have the same importance. Having weights associated to samples will provide us with a principal way to model such problems. Consider a data set in which each data point (x_i, y_i), $x_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}$ is associated with a weighting factor $0 < \alpha_i \leq 1$. Define the loss of a linear model with parameters $w \in \mathbb{R}^n$ as the weighted sum-of-squares error:

$$\mathcal{L}_{\text{wSE}}(\boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} \alpha_i (w^\top x_i - y_i)^2 \tag{2}$$

It can be shown that the optimal weights can be obtained as:

$$w^* = (X^\top A X)^{-1} X^\top A Y \tag{3}$$

Here, X is a matrix of size $m \times n$ where every row is one input sample (i.e., row i in X is x_i). Similarly, Y is an m dimensional vector storing y_i and A is a diagonal matrix of size mm with $A[i, i] = \alpha_i$.

The purpose of this task is to train an iterative linear regression model that will weigh the samples as the model is being trained. You need to develop the mechanism to weigh each sample as the model is being iteratively trained. Here, the diagonal matrix A is updated by using the following equation:

$$\hat{\alpha_i} = exp(-\sigma ||y_i^{noisy} - \hat{y_i}||^2) \tag{4}$$

where σ is a hyperparemeter you will tune.

You are supposed to implement a weighted linear regression model to explain the data. The data for this question has two sets, a training set $\left(\boldsymbol{X}_{\text{trn}}, \boldsymbol{Y}_{\text{trn}}^{\text{noisy}}\right)$ and a validation set $\left(\boldsymbol{X}_{\text{val}}, \boldsymbol{Y}_{\text{val}}\right)$. You are supposed to use only the training set $\left(\boldsymbol{X}_{\text{trn}}, \boldsymbol{Y}_{\text{trn}}^{\text{noisy}}\right)$ to train your model. Note that the training set is noisy (*i.e.*, $\boldsymbol{Y}_{\text{trn}}^{\text{noisy}}$ is noisy). You will use the validation set to evaluate your model and as you might have noticed, the validation set is noise free.

Since in this case, the weight of each sample is not provided, you need to develop a mechanism to approximate how noisy a sample is. One way of doing this is to first fit a linear regression model to your data and then check and see which samples your model cannot explain well (*i.e.*, the prediction is not good). For such samples, you need to devise a way to assign a reduced sample weight. Once you attain the weights according to the error of your predictions, you can fit a weighted linear regression model and repeat this process a few times till you obtain a model that can explain the validation data well.

Implement the aforementioned strategy and report your error on the validation set. For your convenience, a sketch of a possible solution is provided in Algorithm 1.

Hints.

- You will craft new non-linear features.
- The error of the validation set is

$$\text{validation-error} = \frac{1}{m}\sum_{i=1} \left\| y_i^{\text{val}} - \hat{y}_i^{\text{val}} \right\|^2 . \tag{5}$$

  Here, $\hat{y}_i^{\text{val}} = \boldsymbol{w}\boldsymbol{x}_i^{\text{val}}$ is the prediction of your model for a validation sample and m is the number of samples within your validation set.
- You have to tune σ yourself.
- You have to choose a stopping condition.

Algorithm 1: Iterative Weighted Linear Regression

Data: The training set $\boldsymbol{X}_{\text{trn}} \in \mathbb{R}^{m\times n}$, $\boldsymbol{Y}_{\text{trn}}^{\text{noisy}} \in \mathbb{R}^m$
Data: The validation set $\boldsymbol{X}_{\text{val}} \in \mathbb{R}^{p\times n}$, $\boldsymbol{Y}_{\text{val}} \in \mathbb{R}^p$
Result: The parameters of the optimal model $\boldsymbol{w}^* \in \mathbb{R}^n$
$\boldsymbol{A} \leftarrow 1$;
for *iter in max-iter* **do**
  $\boldsymbol{w} \leftarrow$ Fit a weighted linear regression model using $\boldsymbol{X}_{\text{trn}}, \boldsymbol{Y}_{\text{trn}}^{\text{noisy}}, \boldsymbol{A}$; `/* Use ?? and note that` $\boldsymbol{A}$ `is a diagonal matrix keeping the sample weights` a_i `*/`
  $\hat{y}_i = \boldsymbol{w}^\top \boldsymbol{x}_i^{\text{trn}}$; `/* Evaluate your model on all training samples */`
  $\hat{a}_i = \exp\left(-\sigma \|y_i^{\text{noisy}} - \hat{y}_i\|^2\right)$; `/* Obtain the weight of the training samples */`
  Obtain the error of your model on the validation set according to Equation (5);
end
$\boldsymbol{w}^* \leftarrow$ the model with the minimum validation error

Task 1.2. Using non-linear features and learning rate decay for better classification

You have been given a set of samples $x = (x_1, x_2)^\top \in \mathbb{R}^2$. In this task, you will apply the knowledge you learned from previous labs to train a logistic model via Gradient Descent and try to improve the performance by implementing a decaying learning rate.

Decaying learning rate can be simply implemented as:

$$lr_{new} = (1 - \alpha) * lr_{old}, \tag{6}$$

where $0 < \alpha < 1$ is a hyperparameter. In this task, we use $\alpha = 0.1\%$ for the learning rate decay.

You will follow the standard procedure of first loading in your data and visualising it. From there, you will apply nonlinear transformations to your data via GD. You can then visualise your decision boundary and write a decaying learning rate functino to further improve your model.

You can reuse functions **sigmoid**, **compute_loss_and_grad** and **predict** from your previous labs for this task. Plot the decision boundary for above on test data. Compute the accuracy of the resulting model on the test data (X_test, y_test).

Task 2. Denoising autoencoder (DAE) with Face Data

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 random 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.

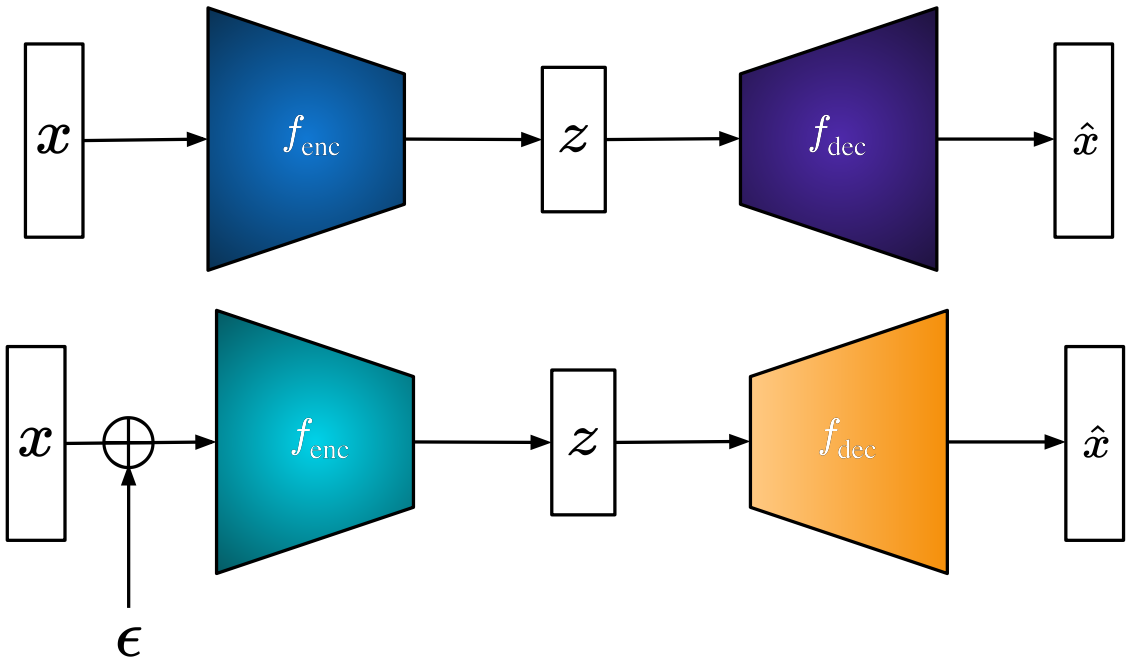

Figure 3: **Top.** An autoencoder is comprised of two subnetworks, an encoder and a decoder. In its vanilla form, the goal is to extract useful structure and information about the input. **Bottom.** In a DAE, you feed the network with a noisy input and train the model to produce clean outputs.

4. [20 points] In this task, 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	Leaky ReLU	Only at the encoder
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 "*???*"

You will need to Compute Peak Signal-to-Noise Ratio (PSNR) for test dataset. The PSNR is a commonly used metric to measure the quality of a reconstructed or denoised signal or image. The PSNR is calculated as the ratio of the peak signal power to the noise power, typically measured in decibels (dB). PSNR can be defined as

$$\text{PSNR} = 10 \cdot \log_{10} \left\{ \frac{\text{MAX}^2}{\text{MSE}} \right\} \tag{7}$$

Where:

- PSNR is the Peak Signal-to-Noise Ratio in dB.
- MAX is the maximum possible pixel value (e.g., 255 for an 8-bit image).
- MSE is the Mean Squared Error between the original and the reconstructed (or denoised) image. It's calculated as the average of squared pixel-wise differences between the two images.

You will be asked to show the PSNR values for:

1. Between the noisy image and original image
2. Between the reconstructed image and original image

We provide you a handy class *torchmetrics.image.PeakSignalNoiseRatio* for PSNR calculation.

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.

Task 3. GTSRB

5. [20 points] This task is to apply CNN to classify German traffic signs using the GTSRB dataset (German Traffic Sign Recognition Benchmark). This task assumes you have the fundamental principles in applying the modelling process. You will need to create your dataloaders and achieve the deliverables listed below.

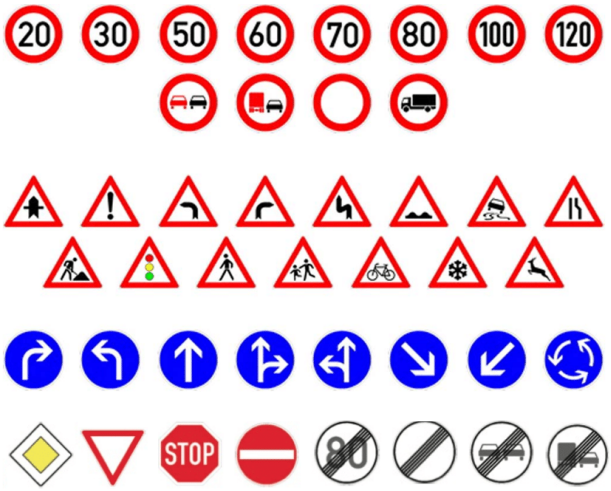

Figure 4: GTSRB dataset with some of the classes

The final deliverables for this task are:

1. Accuracy of more than 85% on the test dataset using CNN.[1]
2. Update the callback function within the pytorch-lightning module to visualize predictions
3. Convergence of train and validation loss/accuracy curves.
4. Two additional analysis listed in the notebook

The only constraint with the model, you need to be able to program modular convolutional blocks which you can layer easily. An example of a convolutional block can be seen below:

Input $\rightarrow$ **Conv1** $\rightarrow$ **Activation** $\rightarrow$ **Conv2** $\rightarrow$ **Activation** $\rightarrow$ **Output**

Example of a convolutional block

Of course, you can add other layers in between the activations and convolutional blocks such as drop out, batchnorm2d, pooling etc. Laying out the convolutional block allows modular stacking of the blocks to increase the number convolutional layers easily:

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

Example of layering the convolutional blocks

You will also need to layer at least two convolutional blocks for this task to obtain full marks for the model architecture. Your model must also have shown convergence on the validation set.

[1] You can use techniques learnt in lectures and workshops such as skip connections, 1x1 convolutions etc.

Task 4. Face Landmarks Detection (FLD) with Transfer Learning

6. [15 points] In this task, you will be applying your knowledge to a new application. You will be using a method called transfer learning see Figure 5 which utilises a pre-trained network (in our case a ResNet50 model) for a new task. The application of transfer learning will be to plot facial landmarks on RGB images.

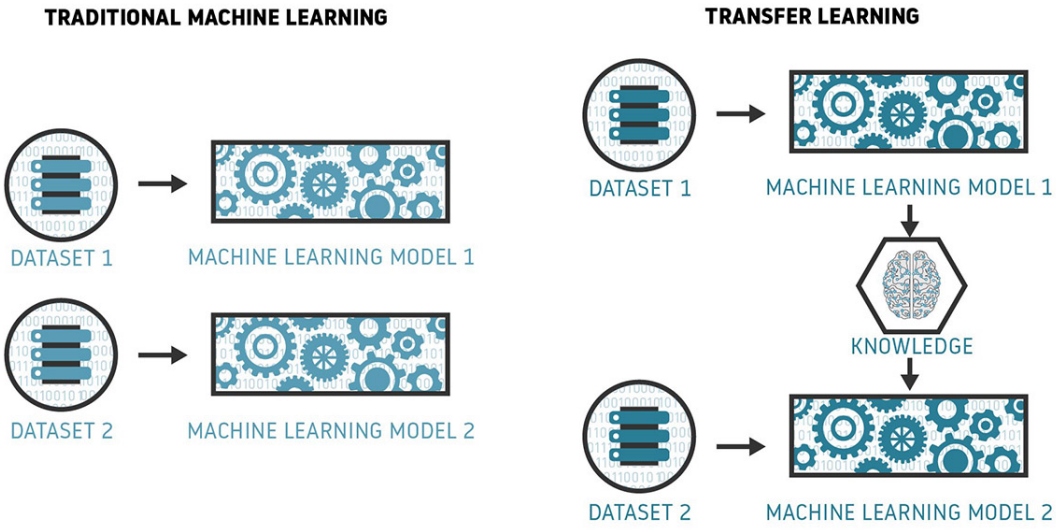

Figure 5: Traditional vs. transfer learning

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 ResNet50 model (Figure 6) 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 learned parameters), and only replace the fully connected layer with the fully connected layer we want.

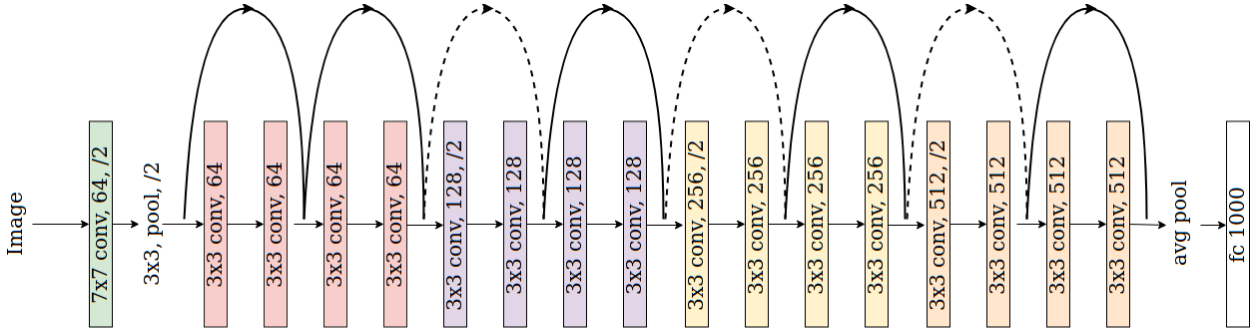

Figure 6: Resnet Structure

The face landmark dataset contains photos of people's faces and have associated "landmarks"[2] 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.

Note the following hints:

- Ensure loading the pre-trained weights of the model. [3]
- Ensure you pick the appropriate loss. Remember, we are dealing with coordinates here, so it is not a classification problem!

The final deliverables for this task are:

1. Successfully load a pre-trained model.
2. Apply the pre-trained model to the face landmarks dataset
3. Obtain a MSE loss of less than 650 on the validation set in less than 100 epochs. [4]

[2] 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 !

[3] You can confirm this by training your model for one epoch, and comparing the original weight of the model with your new model (just look at the first layer).

[4] You can tune the hyper parameters such as learning rate and have additional fully connected layers. You may also choose to unfreeze layer by layer within your model

Task 5. Speech Command Recognition

7. [10 points] In this final task of your assignment, you will be loading a speech recognition model from the following options:

   - Word2Vec2
   - Whisper
   - HuBERT

   And using it to extract feature representations from audio files. The goal is to train a model that can recognize ten different voice commands, including "Yes", "No", "Down", and more.

   To accomplish this task, you will follow these steps:

   - **Dataset:** You will work with the Speech Commands dataset, which consists of audio recordings of various command words. The dataset is divided into three folders: Train, Validation, and Test. The *Train* folder contains audio files for training the models, the *Validation* folder contains audio files for hyperparameter tuning, and the *Test* folder contains audio files for final predictions.
   - **Preprocessing:** You will preprocess the audio data by extracting meaningful features using your model of choice. You can choose a pre-trained model that has been trained on labeled speech data and can effectively capture spoken language patterns and features effectively. The pre-processing model normally comes with the pre-trained model that we have listed above.
   - **Model Training:** You will train a simple MLP (Multi-Layer Perceptron) model to predict the labels of the voice commands. The MLP model will take the extracted features as input and learn to classify the voice commands into ten different classes.
   - **Evaluation:** You will evaluate the performance of your trained model using accuracy as the evaluation metric. Accuracy measures the percentage of correctly predicted labels.
   - **Testing:** Finally, you will make predictions on the test dataset using your trained model and submit your results to Kaggle (instruction is provided in the notebook).
   - **Marking:**
     - 5/10 points are allocated to the notebook quality.
     - 5/10 points are allocated to the Kaggle Competition results. You need to achieve at least 85% accuracy on the test set. The competition marks will be linearly awarded from 0 to 5 points according to accuracy from 85% to highest accuracy in this competition.

   Kaggle Competition: The link to the competition is provided here: https://www.kaggle.com/competitions/ecse-4179-5179-6179-2024-s-2-assignment-1

   - Your student email has been registered for the competition.
   - You should create a Kaggle account with your student email address and then attempt the competition.
   - You have access to the dataset from **Data** tab in the Kaggle page.

   By completing this task, you will gain hands-on experience in speech recognition using the pre-trained speect recognition models and learn how to train a model to recognize voice commands. This skill has wide-ranging applications, including voice-controlled systems for smart home devices, navigation systems, and automotive applications.

Final words

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.