

---

# COMP47650 Deep Learning Project

---

Version 2.0

## 1 Overview

This project will examine your ability to apply deep learning architectures to and address challenges relating to complex, real world problems. This project is intended for groups of three students. We will leave it to you to form these groups ~~but group details should be entered into the allocated Google Form<sup>1</sup>, no later than 17:00., January 31st, 2020. If you don't have a group or you are missing a person, please still fill this form out.~~

## 2 Specification

The project is split into three sections:

### 2.1 Code

Projects should be completed in **either** Tensorflow ( $\geq 2.0$ ) **or** Pytorch ( $\geq 1.0$ ). Part of this project is to familiarise yourself with these frameworks, so please make sure to start early in case you run into any problems! Code should be clear, commented and most importantly, experiments must be **reproducible**<sup>2</sup>. If you use code from elsewhere (for preprocessing, pre-trained models etc.), make sure to cite the resource used by providing an appropriate link within the code/ as a footnote in the report. You are free to use pretrained models to aid performance but you should explain the motivation behind using a particular model.

### 2.2 Report

The report should be in a conference paper-style format, using the NeurIPS LaTeX template (compile as "preprint") and style guide (available at <https://nips.cc/Conferences/2019/PaperInformation/StyleFile>). Reports should be 4–6 pages in length (excluding references) and should contain (but are not limited to) the following sections:

- Abstract
- Introduction
- Related Work
- Experimental Setup
- Results
- Conclusion & Future Work
- References

A description of these sections is included in the "Resources" section at the end of this document.

---

<sup>1</sup>Link to Google Form <https://forms.gle/FcVdcZ2JVX5YS8P28>

<sup>2</sup>Here's a quick guide to reproducible results: <https://machinelearningmastery.com/reproducible-machine-learning-results-by-default/>

### 2.3 Peer/Self-assessment Report

You will be asked to assess the contribution of your group members along with your own contribution to the project. Ideally, the work should be split equally between each team member. This will allow for evaluation of the quality of each team member's contribution. The self-assessment form is available on Moodle and should be submitted along with the assignment. Describe your contribution in the self-assessment section in **no more than 1 page** (please do not change font or formatting in order to fit more text). You should also rate your fellow group members' contribution to the project in the table provided.

## 3 Dataset

A dataset from one of the three following categories will be randomly assigned to your group. Each dataset is sourced from Kaggle.com and presents various challenges that your team should address in your project.

### (A) Image

This dataset is comprised of chest X-Ray images to be classified as either 'normal' or 'pneumonia'.

**Link:** <https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia/tasks>

**Data format:** jpeg

**Task:** Binary classification

**Test data:** Use provided test data.

**Evaluation Measure:** AUC

**Challenges:** Imbalanced data, dataset not very large.

### (B) Text

This is a Twitter sentiment classification task where each class represents a polarity of sentiment (0=negative, 0=neutral, 4=positive).

**Link:** <https://www.kaggle.com/kazanova/sentiment140>

**Data format:** csv

**Task:** multi-class classification

**Test data:** Shuffle data with random seed=0, take last 20% of data for testing.

**Evaluation Measure:** Accuracy

**Challenges:** Data is sequential and requires a language model.

### (C) Audio

This is an audio tagging task where clips of audio are classified with one of 41 classes.

**Link:** <https://www.kaggle.com/c/freesound-audio-tagging/data>

**Data format:** wav

**Task:** multi-class classification

**Test data:** Use provided test data.

**Evaluation Measure:** F1-score

**Challenges:** Using audio data (raw data? pre-processed?), imbalanced dataset.

## 4 Experiments

### 4.1 Useful considerations before starting

- What pre-processing is necessary to perform before modelling data?
- What is the structure of the data? Is it sequential?
- Is the data balanced?
- If so, what techniques can be used to address this imbalance?
- What deep learning architectures are most appropriate for this data?
- Can the data be changed to fit a particular architecture?
- Are there existing pre-trained models that could be fine-tuned?
- Can the model be parallelised (make better use of GPUs for quicker training)?
- How many weights are there in the model? How much memory does this consume?

### 4.2 Evaluating model performance

A validation set should be created either using a subset of the training set or use the one provided to evaluate your model's ability to generalise and to perform hyper-parameter tuning **Never use the test set in this process**. You should create a plot showing the training and validation loss on the y-axis and the iteration of training on the x-axis to identify the point where generalisation is best. The model that performs best of the validation set should then be used for testing. The evaluation measure allocated for each dataset should be used and the motivation for using it explained.

It should be noted that the training procedure will likely be repeated during grading and any substantial differences (small differences are expected due to random initialisation) between the reported training loss and that obtained upon repeating the experiment will bring the results - and therefore overall project - into question.

## 5 Submission Details

### 5.1 What to submit

Please submit in the form of a Zip file the following:

- All related scripts/ jupyter notebooks
- Model weights for each model trained. <sup>3</sup>
- A "requirements.txt"
- A short README file explaining how to run scripts.
- Your peer/self-evaluation form.
- Your final paper in PDF format.

**Do not upload dataset**, we have this! Any feature extraction or re-writing of data into other formats should all be reproducible within the code. The submission deadline is: 17:00., 24th of April, 2020.

### 5.2 Plagiarism

All submissions will be checked for plagiarism. Please review the School's plagiarism policy here. Any suspected plagiarism will be dealt with accordingly.

---

<sup>3</sup>Use either the standard Pytorch format or H5 format. More details on tensorflow to H5 here: [https://www.tensorflow.org/tutorials/keras/save\\_and\\_load](https://www.tensorflow.org/tutorials/keras/save_and_load)

## Resources

### Section Descriptions

<b>Abstract</b>	General, short overview of your paper (draw in the readers attention).
<b>Introduction</b>	Introduce the problem in more detail, the motivation for your work, a brief description of your approach and findings along with a description of how the rest of the paper will proceed.
<b>Related Work</b>	A short summary of past approaches to this problem, the state-of-the-art (if available). You should compare and contrast your own work to these existing works.
<b>Experimental Setup</b>	A detailed description of the dataset used, preprocessing applied (if any), proposed algorithm, baseline approach, hyper-parameter tuning done etc.
<b>Results</b>	Explain evaluation technique and motivation behind using it (i.e. Accuracy, RMSE, F1-score etc.) Compare your approach to baselines (i.e. a simpler model) or state of the art.
<b>Conclusion &amp; Future Work</b>	An overview of your main findings. You should also propose how this model might be improved in future (i.e. what would you do if you had more time or could do the project again?).
<b>References</b>	Reference any papers that you used in your related work or as a reference for your approach.

### Pytorch

**Download** instructions: <https://pytorch.org/>

**Beginner tutorials:** <https://pytorch.org/tutorials/>

**Forum:** <https://discuss.pytorch.org/>

### Tensorflow

**Download instructions:** <https://www.tensorflow.org/install>

**Beginner tutorials:** <https://www.tensorflow.org/tutorials>

**Forum:** <https://www.tensorflow.org/community/forums>

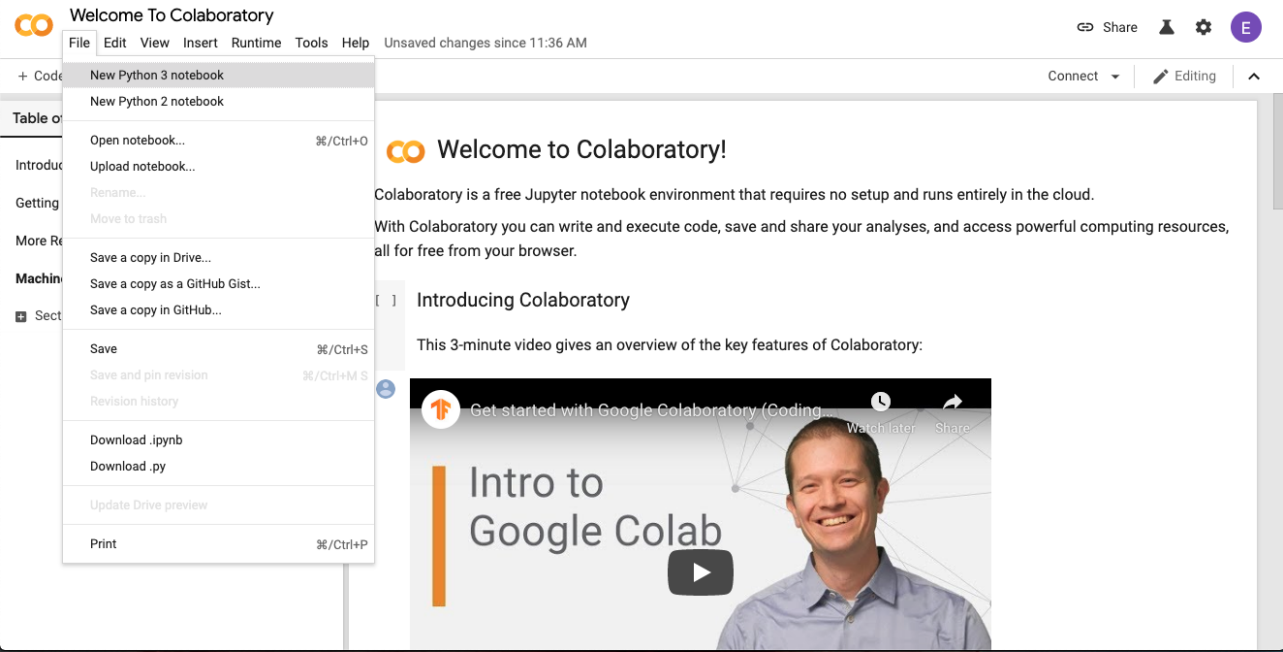
### LaTeX

**Online LaTeX compiler:** <https://www.overleaf.com/>

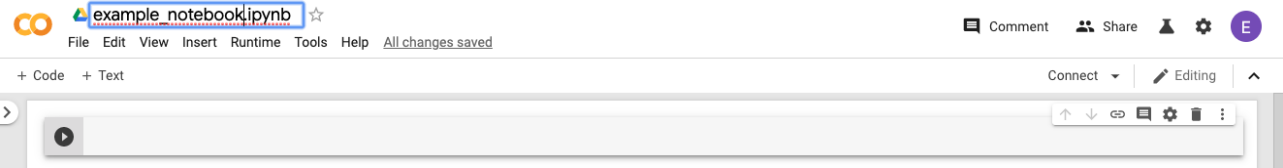
**LaTeX tutorials:** <https://www.overleaf.com/learn/latex/Tutorials>

# Google Colab

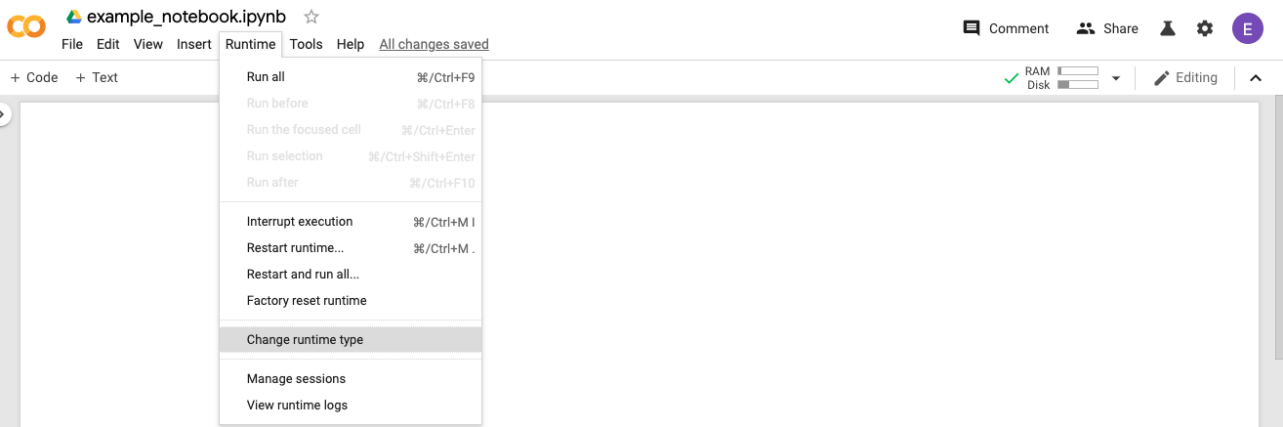
## Create new notebook



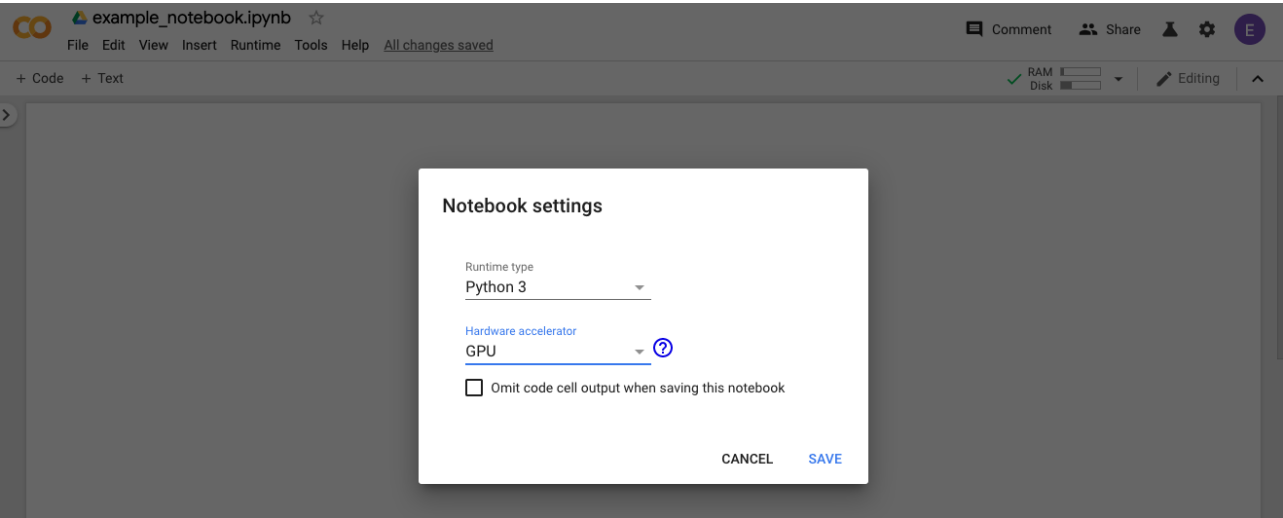
## Rename notebook



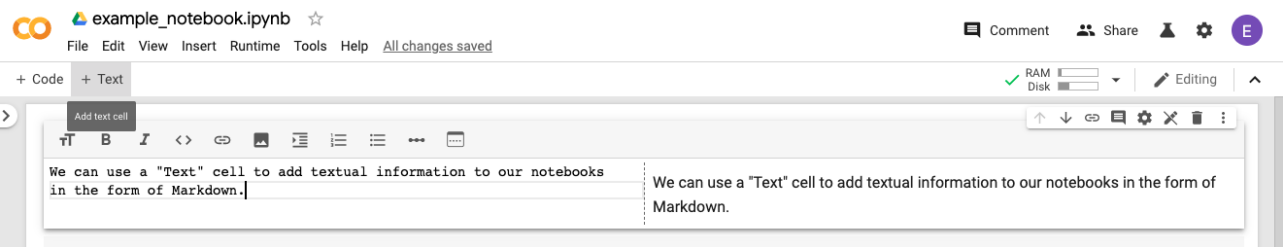
In order to switch between CPU, GPU and TPU, change runtime.



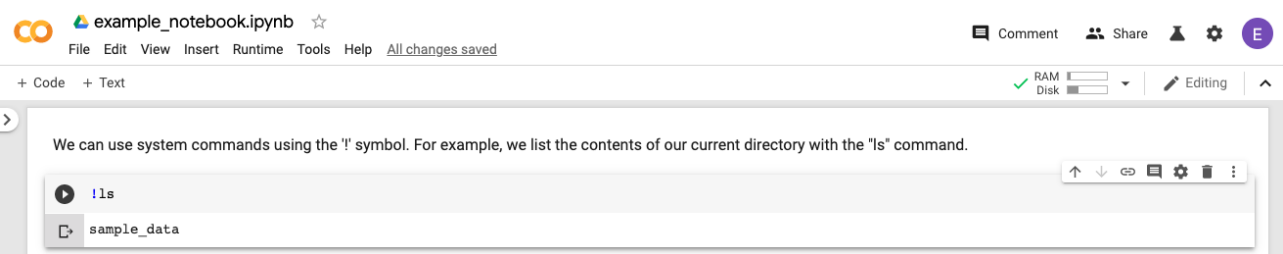
Select either **None**, **GPU** or **TPU**.



Write details about code or additional information using the text cells.



You can use system commands (i.e. linux commands) using the '!' symbol in code cells.



## Installing Pytorch and using it with a GPU.

The screenshot shows a Jupyter Notebook titled "example\_notebook.ipynb". The interface includes a top bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help" menus, along with a status bar indicating "All changes saved". On the right, there are icons for "Comment", "Share", "Settings", and a user profile. Below the top bar, there are tabs for "+ Code" and "+ Text". The notebook content consists of two code cells. The first cell contains the command `! pip3 install torch torchvision`. The output shows that the requirements for `torch` and `torchvision` are already satisfied. The second cell contains the command `import torch`. The output shows that `torch` is working with the GPU. The third cell contains the command `print("Is cuda available?", torch.cuda.is_available())`, `print("Number of devices: ", torch.cuda.device_count())`, and `print("Device name (i.e. type):", torch.cuda.get_device_name())`. The output shows that CUDA is available, there is 1 device, and the device name is "Tesla P100-PCIE-16GB".

```
[1] ! pip3 install torch torchvision
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.6/dist-packages (1.3.1)
Requirement already satisfied: torchvision in /usr/local/lib/python3.6/dist-packages (0.4.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from torch) (1.17.5)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.6/dist-packages (from torchvision) (6.2.2)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from torchvision) (1.12.0)
```

Check that torch is working

```
[2] import torch
```

Check that torch is working with the GPU we have available.

```
[4] print("Is cuda available?", torch.cuda.is_available())
print("Number of devices: ", torch.cuda.device_count())
print("Device name (i.e. type):", torch.cuda.get_device_name())
```

```
Is cuda available? True
Number of devices: 1
Device name (i.e. type): Tesla P100-PCIE-16GB
```

## Installing Tensorflow and using it with a GPU.

The screenshot shows a Jupyter Notebook titled "Check that tensorflow is working with the GPU we have available". The notebook content consists of a single code cell. The code imports `tensorflow` as `tf`, imports `eager` from `tensorflow.python`, imports `device_lib` from `tensorflow.python.client`, and lists the local devices. It then filters the devices to find the GPU. The output shows that TensorFlow is working with the GPU. The output also shows the device name, device type, memory limit, and locality information for the GPU.

```
import tensorflow as tf
from tensorflow.python import eager
from tensorflow.python.client import device_lib
devices = list(device_lib.list_local_devices())

gpus = [device for device in devices if "GPU" in str(device)]

print("Is cuda available?", tf.test.is_built_with_cuda())
print("Number of devices: ", eager.context.num_gpus())
print("Device name (i.e. type):", gpus)
```

```
2.1.0
Is cuda available? True
Number of devices: 1
Device name (i.e. type): [name: "/device:XLA_GPU:0"
device_type: "XLA_GPU"
memory_limit: 17179869184
locality {
}
incarnation: 9983085339974938302
physical_device_desc: "device: XLA_GPU device"
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 14912199066
locality {
  bus_id: 1
  links {
  }
}
incarnation: 534488733954009849
physical_device_desc: "device: 0, name: Tesla T4, pci bus id: 0000:00:04:0, compute capability: 7.5"
]
```

## Uploading data from your local disk

The image illustrates the process of uploading a local dataset to Google Colab. It is divided into three horizontal sections showing the state of a Colab notebook at different stages.

**Top Section:** A file explorer window is open, showing a list of files in a directory named 'digit-recognizer'. The files are:

Name	Date Modified	Size	Kind
train.csv	11 Dec 2019 at 20:01	76.8 MB	CSV Document
sample_submission.csv	11 Dec 2019 at 20:01	241 KB	CSV Document
test.csv	11 Dec 2019 at 20:01	51.1 MB	CSV Document

The 'train.csv' file is selected. Below the file list are buttons for 'Options', 'Cancel', and 'Open'.

**Middle Section:** The Colab notebook interface shows a code cell with the following code:

```
[11] from google.colab import files  
  
uploaded = files.upload()
```

Below the code cell, a file selection dialog is open, showing 'Choose Files', 'No file chosen', and 'Cancel upload' buttons.

**Bottom Section:** The Colab notebook interface shows the same code cell, but the upload process is complete. The file selection dialog now shows 'train.csv' as the selected file. Below the dialog, a progress bar and file details are displayed:

- train.csv(text/csv) - 76775041 bytes, last modified: 11/12/2019 - 2% done
- Saving train.csv to train (1).csv

You can also use the kaggle API to download data: <https://www.kaggle.com/docs/api>