Homework 2

18739 Spring 2020

Due: Tuesday, March 03, 2020 10:20am PST / 1:20pm EST

Academic Integrity Policy

This is an individual homework. Discussions are encouraged but you should write down your own code and answers. No collaboration on code development is allowed. We refer to CMU's Policy on Cheating and Plagiarism (`https://www.cmu.edu/policies/`). Any code included in your submission will be examined under Similarity Check automatically.

Late Day Policy

Your solutions should be uploaded to Gradescope (`https://www.gradescope.com/`) by the deadline. You have 8 late days in total to spend during the whole course but no more than 3 days can be applied to any one homework.

Written Exercises

If a homework contains exercises with written answers you will need to include a `.pdf` file containing your solutions in the zip mentioned below. Please indicate your Andrew ID and name on the first page. We will not accept hard-copies and do not hand-write your answers. Latex (Overleaf: `https://www.overleaf.com`) and Markdown are two recommended languages to generate the clean layout but you are free to use other software. You will need to submit the written part separately from you code submission to HomeworkX-PDF on Gradescope.

Coding Exercises

Submit the coding portions of homeworks to Gradescope according to the following procedure:

1. Compress your `.py` and other requested files into a zip file called `submission.zip` (No other name is allowed otherwise Gradescope will fail to grade your submission). Do NOT put all files into a folder first and compress the folder. **Create the zip file directly on the top of all required files**. Each homework will specify the `.py` and other files expected to be included in the zip.

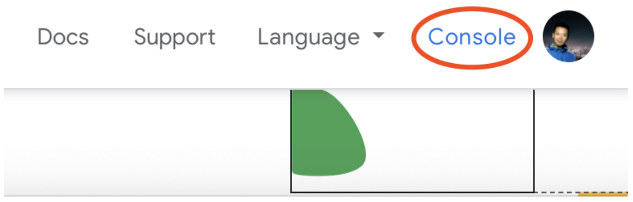
2. Submit `submission.zip` to Gradescope. Your code implementation will be auto-graded by Gradescope and you have an infinite number of submissions before the deadline but only the last submission will count towards the grading. Allow the server to take some time (around 2 min) for execution, which means that do not submit until the last moment when everyone is competing for resources.

Contents

Cloud Server Setup

Step One: Open an GCP account

Go to the website of Google Cloud Platform (GCP: `https://cloud.google.com`) and log in with one of your **non-andrew** Google account. For SV students, we suggest you to use your sv email account. For Pittsburgh students, you can use one of your personal accounts. Using your `@andrew.cmu.edu` account may cause organization error. After login, you may click `Console` button on the right top corner.



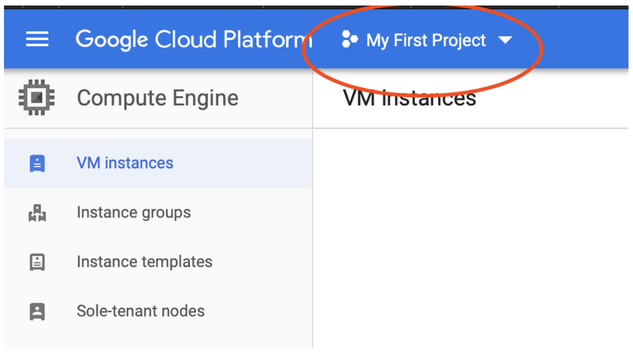
Step Two: Redeem Your Credit

Visit https://console.cloud.google.com/education to redeem the credit you receive from this course into your GCP account. You will receive an email containing the code to redeem your account. Please do not share that with other people.

Step Three: Start a New Project

This is the most important step to prevent you from going bankrupt. Once you have redeemed your credit, you need to select the correct billing account before you start your virtual machine, otherwise you will be using your free $300 credits from Google or even your own credit card. **Notice, this $300 credits are not for this course. You do not need to use this for completing the homework.** We will give you the credits only for this course by emails.

Start a new project first:

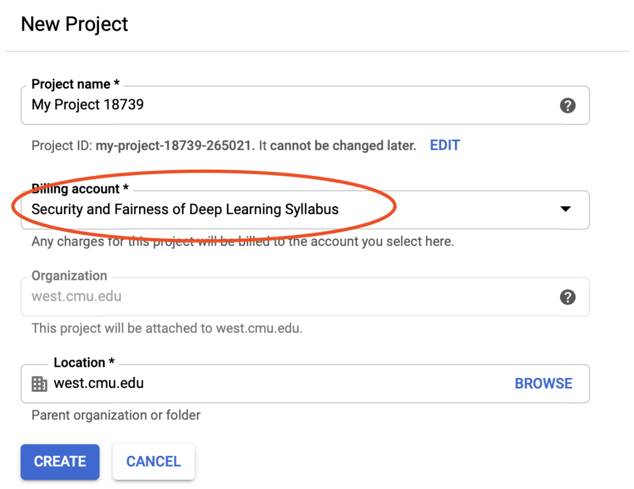


Your can choose your project name.

Billing Account

Ignore this part if you do not see a dropdown asking you to select the Billing account, which means you only have one billing option – the $ 50 credit we give you. If you still have doubt about this, come to the Office Hour.

But if you do see an option to choose different Billing account, make sure you change the is to **Security and Fairness of Deep Learning Syllabus**.


New Project

Project name *
My Project 18739

Project ID: **my-project-18739-265021**. It cannot be changed later. EDIT

Billing account *
Security and Fairness of Deep Learning Syllabus
Any charges for this project will be billed to the account you select here.

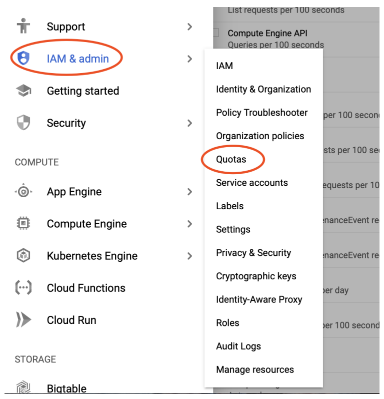
Organization
west.cmu.edu
This project will be attached to west.cmu.edu.

Location *
west.cmu.edu BROWSE
Parent organization or folder

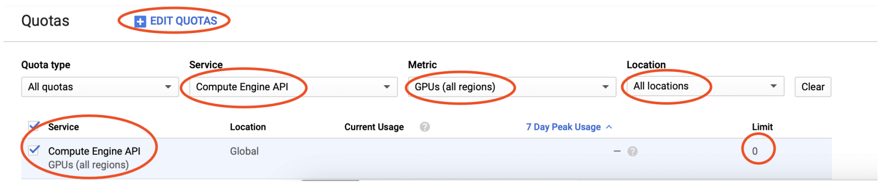
CREATE CANCEL

Step Four: Request Increase of GPU Quota.

Navigate to IAM & admin → Quotas



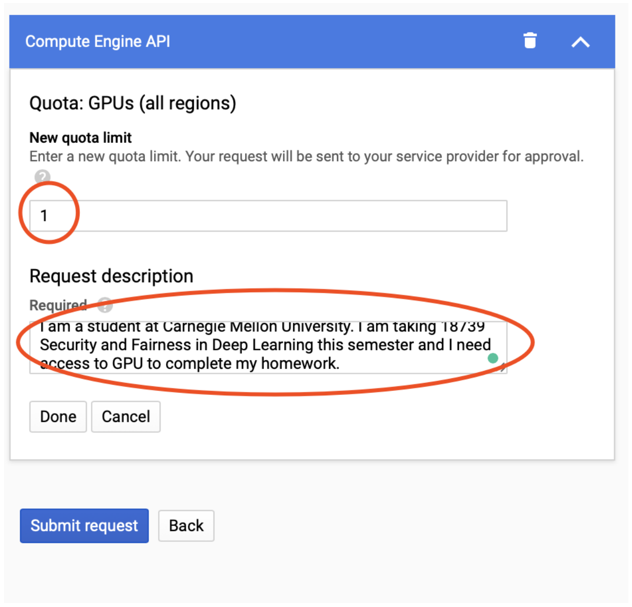
Change the filter options shown as below (if GCP asks you to upgrade the account again, click upgrade and refresh the webpage). After you check the box of `Compute Engine API`, you should see your current limit of GPU is 0. Click **EDIT QUOTAS** at the top to start requesting increase of your limit.


Quotas EDIT QUOTAS

Quota type: All quotas
Service: Compute Engine API
Metric: GPUs (all regions)
Location: All locations
Clear

Service	Location	Current Usage	7 Day Peak Usage	Limit
✓ Compute Engine API GPUs (all regions)	Global		–	0

Enter your contact information and you will see the following page.



Compute Engine API

Quota: GPUs (all regions)

New quota limit

Enter a new quota limit. Your request will be sent to your service provider for approval.

1

Request description

Required

I am a student at Carnegie Mellon University. I am taking 18739 Security and Fairness in Deep Learning this semester and I need access to GPU to complete my homework.

Please enter exactly the same content as above and click **Submit request**. Allow 1 - 5 days for Google to process your request and you will receive an increase of your GPU limit. If you accidentally redeemed your credit into your andrew account, please follow the note on Piazza (`https://piazza.com/class/k579gjgsqgq3bg?cid=40`)

1 Introduction

In this homework, you will be implementing new techniques introduced in lectures on Neural Networks, and learn to train Convolutional Neural networks. You will understand the architecture of Convolutional Neural Networks and gain experience with training these models on benchmark datasets. There are 3 parts in this homework. In the first part, you will be revisiting your Keras implementation from last assignment and implementing convolutional structures to improve the accuracy by a great margin. In the second part, you will be implementing three specific structures/layers common in CNNs using `numpy`. In the third part, you will be training a CNN Model on the CIFAR–10 dataset.

2 Methods and Test Cases

This homework comes with test cases that you can run on your local computer (or GCP) without relying on gradescope's autograder. Your grade, however, will depend on a separate set of testcases on autograder which largely mimic those available to you but will vary in inputs provided to the various methods tested. In Part II, you will have to train a model and upload the results of the provided test cases along with your submission (further instructions in Part II). Finally, Part III features a hidden test dataset over which your solution will be evaluated on gradescope and requires you to include your predictions on that dataset along with your submission (further details in Part III).

Each method you will be implementing in this homework needs to start with a specific call:

```
hw2_utils.exercise(
    andrew_username="pmardzie", # <<< set your andrew username here
    seed=42
)
```

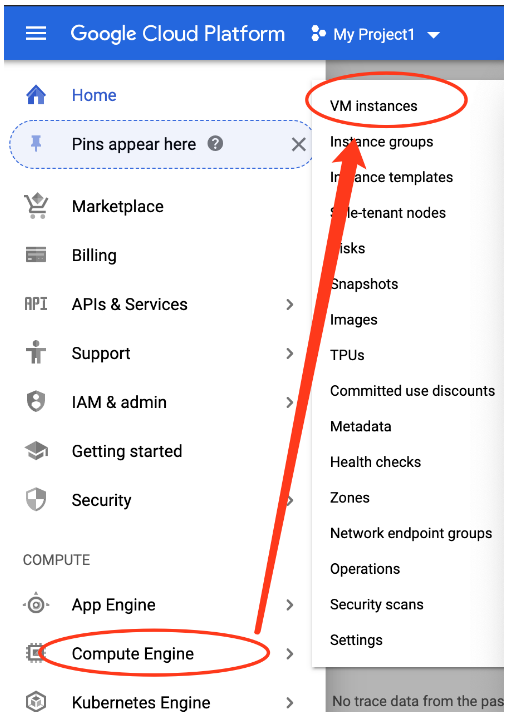
You need to set your andrew username as part of this call. This method also sets the random state for numpy and tensorflow which should result in consistent results for every run of your solution. You can adjust the random seed as you see fit but the test case outcomes you include in your submission must match exactly those produced under the combination of username and seed in the code you submit.

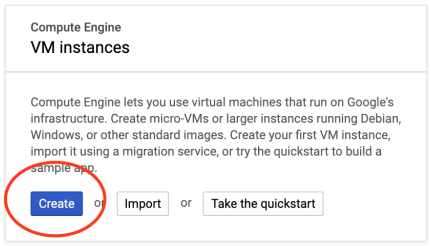
3 Setup

In PART I and II and 7.1 & 7.2 of PART III, you can run your program with your computer locally. However, it is recommended that you use GPU for 7.3 of PART III. Please reuse the conda environment we distributed in the Homework 1 for running conda on your local machine. For remote development on GCP, the package installation varies. Please find the details below.

3.1 Start your Virtual Machine on GCP

Step I Start a Virtual Machine



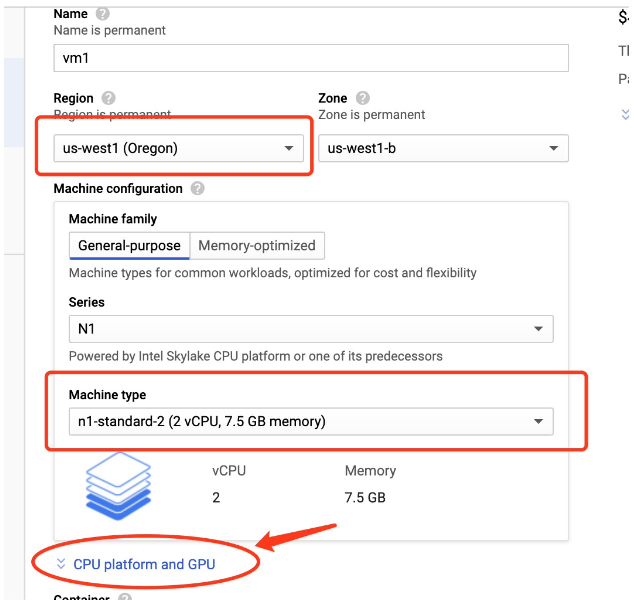

Compute Engine

VM instances

Compute Engine lets you use virtual machines that run on Google's infrastructure. Create micro-VMs or larger instances running Debian, Windows, or other standard images. Create your first VM instance, import it using a migration service, or try the quickstart to build a sample app.

Step II Configure Your VM

First let's configure your CPU. Normally 7.5G is big enough for loading the dataset. However, if you find it is not big enough, you can select an instance with more memory. The selection of region usually affects GPU availability; if you find there is no GPU in the region you selected, try changing the region. Usually Oregon and North Virginia have more GPU resources.



Name
Name is permanent
vm1

Region
Region is permanent
us-west1 (Oregon)

Zone
Zone is permanent
us-west1-b

Machine configuration

Machine family
General-purpose | Memory-optimized
Machine types for common workloads, optimized for cost and flexibility

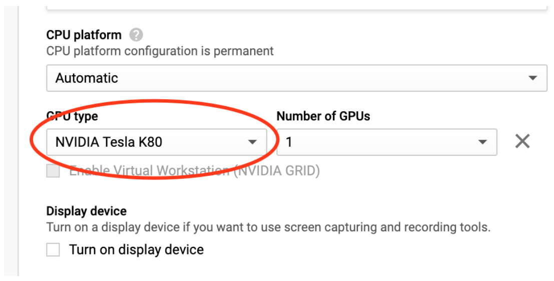
Series
N1
Powered by Intel Skylake CPU platform or one of its predecessors

Machine type
n1-standard-2 (2 vCPU, 7.5 GB memory)

vCPU	Memory
2	7.5 GB

CPU platform and GPU

After your configure your CPU, open the GPU drop-down menu and select a GPU. Note that more powerful GPU, like V100 or P100, may result in faster computation but will cost more credits. Monitor your credits and decide if you have to use a more powerful GPU. We recommend you start with Tesla K80.



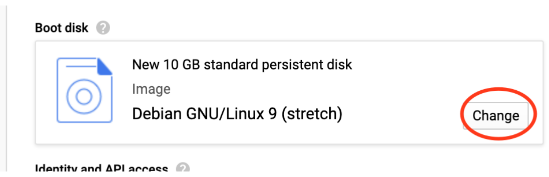
CPU platform
CPU platform configuration is permanent
Automatic

GPU type: NVIDIA Tesla K80
Number of GPUs: 1

Enable Virtual Workstation (NVIDIA GRID)

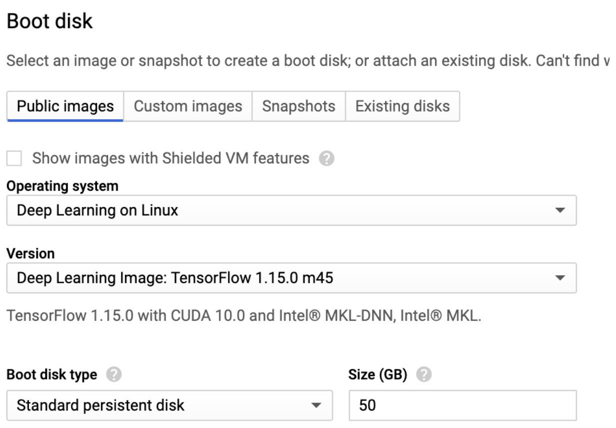
Display device
Turn on a display device if you want to use screen capturing and recording tools.
Turn on display device

Proceed to the OS selection to change from the default OS



Boot disk
New 10 GB standard persistent disk
Image
Debian GNU/Linux 9 (stretch)
Change

to the following OS:


Boot disk

Select an image or snapshot to create a boot disk; or attach an existing disk. Can't find v

☐ Show images with Shielded VM features

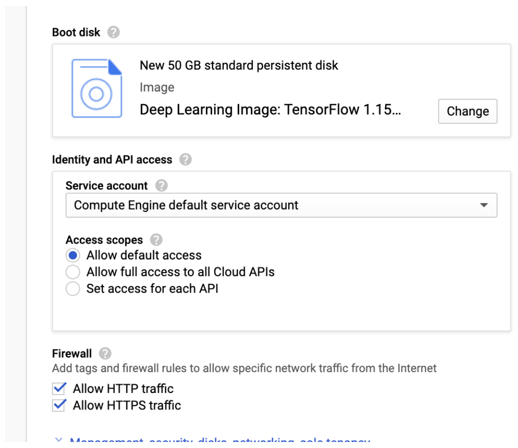
Operating system: Deep Learning on Linux

Version: Deep Learning Image: TensorFlow 1.15.0 m45

TensorFlow 1.15.0 with CUDA 10.0 and Intel® MKL-DNN, Intel® MKL.

Boot disk type: Standard persistent disk

Size (GB): 50


Boot disk

New 50 GB standard persistent disk

Image

Deep Learning Image: TensorFlow 1.15...

Identity and API access

Service account: Compute Engine default service account

Access scopes

[✓ Allow default access]

Allow full access to all Cloud APIs

Set access for each API

Firewall

Add tags and firewall rules to allow specific network traffic from the Internet

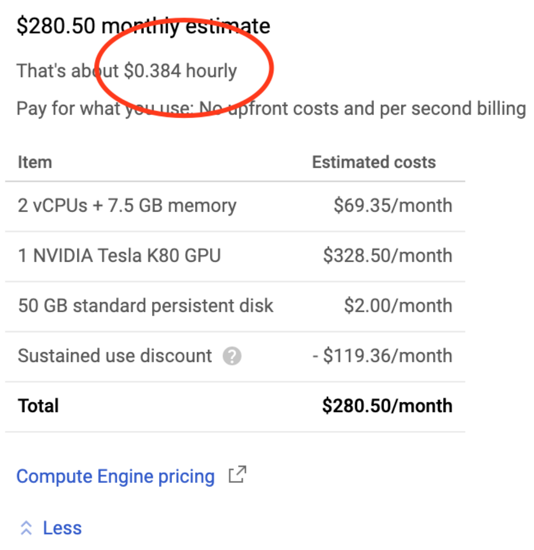
✓ Allow HTTP traffic

✓ Allow HTTPS traffic

In the OS above, tensorflow-gpu, CUDA, cudnn and other acceleration tools are already installed. You do not need to reinstall tensorflow again.

Step III Budget Control

Okay, everything is settled down. Before you enjoy training, notice how much you are going to spend with your credits.


$280.50 monthly estimate

That's about $0.384 hourly

Pay for what you use: No upfront costs and per second billing

Item	Estimated costs
2 vCPUs + 7.5 GB memory	$69.35/month
1 NVIDIA Tesla K80 GPU	$328.50/month
50 GB standard persistent disk	$2.00/month
Sustained use discount	- $119.36/month
Total	**$280.50/month**

Compute Engine pricing

Less

Always stop your VM if you are not using it!!!

Always stop your VM if you are not using it!!!

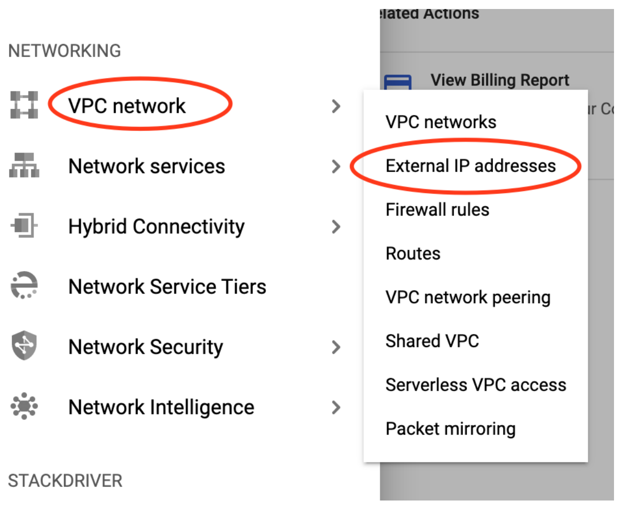
Always stop your VM if you are not using it!!!

If you keep the machine running for 24h, you will lose about $9. Doing this for a week will use up all your credits! Double-check your VM before you go for a vacation.

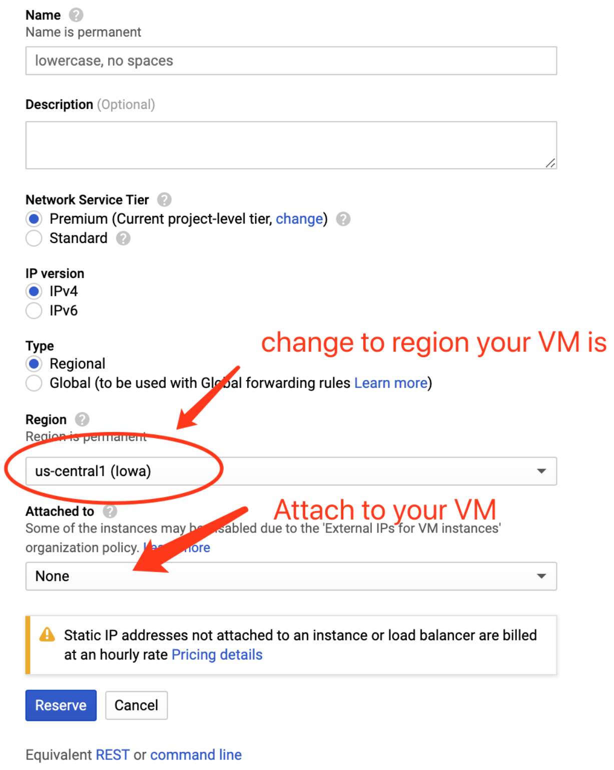
(Optional) Step IV Setup SSH

If you want SSH into your VM without using Google's web-based SSH, you need to assign a static external IP to your VM.

Navigate to External IP addresses



Create a new IP and attach it to your VM.



Name
Name is permanent
lowercase, no spaces

Description (Optional)

Network Service Tier
[✓ Premium (Current project-level tier, change)]
Standard

IP version
[✓ IPv4]
IPv6

Type
[✓ Regional]
Global (to be used with Global forwarding rules Learn more)

Region
Region is permanent
us-central1 (Iowa)

Attached to
Some of the instances may be disabled due to the 'External IPs for VM instances' organization policy. Learn more
None

Static IP addresses not attached to an instance or load balancer are billed at an hourly rate Pricing details

4 Background: tf.keras

In Homework1, you have built up Tensorflow models using the low-level APIs, like `tf.add` and `tf.matmul`. Tt can become inefficient to build complex models using these basic building blocks. In this homework we are going to build models using Tensorflow high-level APIs, `tf.keras`, which is the Tensorflow's implementation of Keras. It contains more functionality than the original Keras and allows the use of the tensorflow API alongside models constructed in the higher-level `tf.keras` API. The documentation can be found here https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/keras.

We will build CNNs with the Functional API, which gives us more control and access to each layers compared to Sequential API. We elaborate on the basic elements and layers of a CNN. You will have to consult tf.keras documentation for other needed layers in this homework.

Input Layer

On the top of the model, we will need an input layer, which functions similarly to a placeholder tensor. Use the following code to create an Input layer.

```
input = tf.keras.Input(shape=(...))
```

Where the shape refers to the single instance input size (not the size of a batch of inputs).

Dense Layer

Use the following code to build a dense layer and its post-activation output tensor.

```
layer = tf.keras.layers.Dense(out_dim, activation='relu')

output = layer(input)
```

Where out_dim is the output dimension, input is the input tensor, output is the output tensor. You do not need to specify the input dimensions. The activation function can be selected with the keyword arguement as shown.

Convolutional Layer

Use the following code to build a convolutional layer for 2D input and its post-activation output tensor.

```
output = tf.keras.layers.Conv2D(F, (H, W), S, activation='relu')(input)
```

Where F is the number of filters, (H, W) is the shape of each filter, S is stride size.

5 PART I. Convolution Upgrade for MNIST *[15 points]*

In this Part, you will be implementing a 2-layer Convolutional Neural Network (CNN) for MNIST classification. Complete your solution in `hw2_mnist.py` which contains separate model building and model training methods for each of the two models in this exercise. The tests in `hw2_test_mnist.py` should run on your own computer. As part of your submission, you need to include `evaluation_mnist.txt`, the log of the public tests. You can generate this file by running the below or using the provided `Makefile` target.

```
python hw2_test_mnist.py > evaluation_mnist.txt
```

5.1 Coding Exercise: CNN Model *[3 points]*

First, you need to reshape the input data into 28×28 using `numpy.reshape()` (since these are black and white images, there is no color channel). Your layers should have the following structures:

- Layer 1: A convolutional layer (Documentation): 32 filters, size 5×5, stride of 1, Relu activation.
- Layer 2: A maxpooling layer (Documentation): size/stride of 2.
- Layer 3: A Dropout layer(Documentation): randomly churns 20% of the neurons.
- Layer 4: Fully connected layer: ReLu activation, 128 neurons.
- Output Layer: Fully connected (10 output neurons), softmax activation.

The first 2 layers are usually grouped together to perform "one round" of convolutional operation for images. In this model, we only have one such round. Typically the state–of–art models of image classification have much more such convolutional operations to deal with larger and more complicated images.

5.2 Coding Exercise: Build Your Own Model *[5 points]*

In this section, you are free to design your own model. Remember, you can add or get rid of features/structures from your second model. Since this is a simple dataset, a lot of the benefits of some regularization techniques are not obvious, however, they will be really useful in training on a larger and more complicated dataset such as CIFAR–10 in PART III. Please try out different opimitizers and non-linear activation functions and make sure your new model can achieve **Faster Convergence** or **Higher Test Accuracy**. This is tested by the last test case of `hw2_test_mnist.py`.

5.3 Written Report/Exercises *[7 points]*

Report the following results for both models.

- What is your best validation accuracy and the corresponding test accuracy? What is the batch size and the number of training epochs, respectively? *[1 points]*
- Please give a summary of your model architecture, including the hyper-parameters you chose for each layer. You can use `print(model.summary())` to print a summary representation of your model. The verbatim mode in latex is useful in including raw text in your report. *[1 points]*

Answer the following questions:

- If an image I with dimension (width, height, depth) $W \times H \times D$ is passed through a convolutional layer with N filters of size F, stride S and zero padding Z, what is the dimension of the filtered output for I? Write your answer in terms of the variables mentioned. *[1 points]*
- Consider an input of size $M \times M \times C$ and a neural network with one Convolution layer and two Dense layers. Each Convolution layer has N filters of size F, stride S, and zero padding Z. Aassume we do not have bias terms in the Convolution layer. The dimensions of two dense layers are K_1 and K_2 respectively. All layers are connected as follows: [Conv, Dense1, Dense2]. Write down the number of trainable parameters in terms of the variables mentioned. *[2 points]*
- Describe the difference of the training phase and test phase of Dropout layer. *[2 points]*

6 PART II. Numpy implementations *[12 points]*

In this part, you will be implementing some basic structures such as CNN or DFN by only using `numpy`. You will be completing 5 functions in `hw2_numpy.py`. Each of these functions comes with test code to test your implementations. Note that in this homework, we are only asking you implement the forward passes of these operations, while in reality you need to have a backward pass function for each forward function. You are allowed to use loops for these exercises.

After you have completed all missing pieces in the starer code, you can check your implementation locally by running the public tests in `hw2_test_numpy.py`. As part of your submission, you need to include `evaluation_numpy.txt`, the log of the public tests. You can generate this file by running the below or using the provided `Makefile` target.

```
python hw2_test_numpy.py > evaluation_numpy.txt
```

6.1 Coding Exercise: Convolution Implementation *[5 points]*

Complete the method `convolution` to implement the forward pass of a convolution layer.

Complete the method `grayscale_filter` which modifies the second channel of the provided filter weights to implement Luma coding: $0.299R + 0.587G + 0.114B$, where R, G, B are the Red, Green and Blue input channels.

6.2 Coding Exercise: ReLU Implementation *[2 points]*

Complete method `relu` to implement the forward pass for a relu layer.

6.3 Coding Exercise: MaxPooling Implementation *[3 points]*

Complete method `maxpool` to implement the forward pass for a maxpooling layer.

6.4 Coding Exercise: Dropout Implementation *[2 points]*

Complete the method `dropout` to implement the forward pass for a dropout layer, which randomly churns neurons in a network. Since a dropout layer behaves differently during training and testing, make sure to implement the operation for both modes.

7 PART III. CIFAR–10 Training in Tensorflow *[20 points]*

In this part, you will train a CNN on a benchmark dataset. You should run 7.1 and 7.2 on you laptop and 7.3 on the google cloud. First retrieve the dataset: run `sh get_dataset.sh`. This will also download and extract the subsets for Section 7.3 .

7.1 Coding Exercise: Build a specific model *[2 points]*

In this section, you need to construct a specific model. You do not have to use the GPU yet. Using the starter code in `hw2_cifar.py` construct a model with the following specifications:

- Input Layer that takes input of shape (32, 32, 3) (given in the starter code).
- 7x7 Convolutional Layer with 32 filters and stride of 1 with ReLU activation.
- Spatial Batch Normalization layer (specify mode with `is_training`).
- 2x2 Max Pooling layer with a stride of 2
- Flatten layer that reshapes the activations from the last layer to (N, C). N is the batch size and C is the number of features.
- Dense layer with 1024 output units with ReLU activation.
- Dense layer from 1024 input units to 10 outputs.

- Softmax Layer (a good habit is to sperate the Softmax Activition from the final Dense layer as a individual layer. We will explain the motivation in the next homework).

Your model should have 8 layers excluding the input. Complete this model in `hw2_cifar.py`. After you have completed the script, you can check your implementation locally by running the public tests in `hw2_test_cifar.py`. As part of your submission, you need to include `evaluation_cifar.txt`, the log of the public tests. You can generate this file by running the below or using the provided `Makefile` target.

```
python hw2_test_cifar.py > evaluation_cifar.txt
```

7.2 Coding Exercise: Optimizer *[2 points]*

While `tf.keras` does provide Keras-like interface like `model.compile()` and `model.fit()` for training, you must use `tf.Session()` in this homework. In subsequent assignments we will need to have more access to the model where we need to compute intermediate values which can not be provided by idiomatic `tf.keras`. You will need to set up the optimizer by hand. Follow the steps below to set up an optimizer for running Back Propagation.

1. Get the output `probit` of the model you build in Section 7.1
2. Compute the mean loss `mean_loss` using your prediction `probit` and a placeholder tensor `Y` of the ground truth.
3. Set up an **RMSprop optimizer** (using a 1e-3 learning rate) with `tf.train.RMSPropOptimizer`. See the TensorFlow documentation (`https://www.tensorflow.org/versions/r1.15/api_docs/python/tf/train/RMSPropOptimizer`) for more information.
4. Minimize your loss by running `train_step = optimizer.minimize(mean_loss)`. The symbolic tensor `train_step` is what your want to pass into `tf.Session().run()` to update all trainable parameters, just like `update_W_op` in HW1.

7.3 Coding Exercise: Your Own Model *[16 points]*

In this section, you are going to implement your own model and train one CIFAR-10-739 dataset (739 means the dataset is under some processing we perform and you can not download a CIFAR-10 from the Internet to pass this section.). We will distribute a training dataset and a validation dataset with features and labels. We will also distribute a test dataset without labels. You will submit your predictions to Gradescope in the form of a CSV fileand your accuracy on the test dataset will earn you corresponding credits.

When you generate your predictions CSV, you can use the `save_csv` method of `hw2_cifar_my_model.py`. When you build your model, **you are not allowed to use transfer learning (do not use ImageNet Pretrained weights) or pre-built model from Tensorflow/Keras**. If you want to implement a well-known architecture, like VGG, ResNet or DenseNet, you have to build by hand, layer by layer. In this part, we want you to focus on model selection and training, **so you are free to use either Keras or Tensorflow**. We do not provide starter code in the section but you will be asked to include all your code for the submission.

Dataset

The `get_datasets.sh` command you ran earlier should have downloaded and unzipped the another subset of CIFRA. You should see the dataset in `.npy` files. You should train your model on `train_X.npy` with its labels `train_y.npy`. You can test your model on `public_test_X.npy` with its labels `public_test_y.npy`. **You submission will be your prediction labels of `private_test_X.npy`.** You can assume the private test set are drawn from the similar distribution of the public test set, which means your test accuracy for the public test set should be close to what you will get from the private test set. You should not train your model with the public test set, which may cause overfitting. Generate your predictions and submit the results as `submission.csv`.

We also provide another small dataset for you to develop your model and test locally, which should save your a lot of time in loading the actually dataset when doing functionality verification and debuging. You should first develop your model locally. After you have fixed all your bugs, then you can move to the bigger dataset and train your model on GCP. The smaller datasets should have extracted into `val_X.npy` and `val_y.npy`.

Things to try

- Filter size: In model 2 we used 7x7; this makes pretty pictures but smaller filters may be more efficient
- Number of filters: In model 2 we used 32 filters. Do more or fewer do better?
- Pooling vs Strided Convolution: Do you use max pooling or just stride convolutions?
- Batch normalization: Try adding spatial batch normalization after convolution layers and vanilla batch normalization after affine layers. Do your networks train faster?
- Network architecture: The network above has two layers of trainable parameters. Can you do better with a deep network? Good architectures to try include:
  - (conv-relu-pool)$\times$N $\rightarrow$ (dense)$\times$M $\rightarrow$ (softmax or SVM)
  - (conv-relu-conv-relu-pool)$\times$N $\rightarrow$ (dense)$\times$M $\rightarrow$ (softmax or SVM)
  - (batchnorm-relu-conv)$\times$N $\rightarrow$ (dense)$\times$M $\rightarrow$ (softmax or SVM)
- Regularization: Add l2 weight regularization, or perhaps use Dropout.

Training Tips

For each network architecture that you try, you should tune the learning rate and regularization strength. When doing this there are a couple important things to keep in mind:

- If the parameters are working well, you should see improvement within a few hundred iterations
- Remember the coarse-to-fine approach for hyperparameter tuning: start by testing a large range of hyperparameters for just a few training iterations to find the combinations of parameters that are working at all.
- Once you have found some sets of parameters that seem to work, search more finely around these parameters. You may need to train for more epochs.
- You can use a validation set for hyperparameter search or functional verification. You can split a validation dataset from the training set by yourself.
- Don't stick to the book! Feel free to try something novel.

7.3.1 Written Report: Report your work/findings *[2 points]*

This is a written section, which means you need to include this into your PDF submission to Gradescope. Please indicate the following things if applicable:

- Citation of any sources (paper, github repo, website) that help you develop your homework
- Your model's architecture. Describe your model: e.g. what kind of architecture you are using, what is the optimizer, how many structures you have tried and how do they perform?
- Any preprocessing that you have performed to the data and how that helps to improve your accuracy.

7.3.2 Grading

For your own model you will receive 3 credits if your test accuracy is $\geq 70\%$. If your accuracy is $\geq 75\%$, you will receive 5 credits. After your accuracy reaches $\geq 80\%$, you will receive 8 credits and if it reaches $\geq 83\%$ it will earn you 10 credits. 2 credits account for your report and citation of any possible resources that helps you in the homework development. The rest of 4 credits account for the implementation. If you fail to pass any accuracy test, we will give you at most 4 credits based on your implementation.

Have fun and happy training!

Submission

1. Submit

   ```
   hw2_mnist.py
   hw2_numpy.py
   hw2_cifar.py
   submission.csv
   evaluation_mnist.txt
   evaluation_numpy.txt
   evaluation_cifar.txt
   ```

   and you python scripts of your own model for training CIFAR. Please include all scripts you use for training, prepossessing data, testing. Create a single zip file as specified by the submission procedures at the beginning of this document.

2. Typeset your answer to written questions into `hw2_written.pdf` and follow the submission procedures at the beginning of this document.