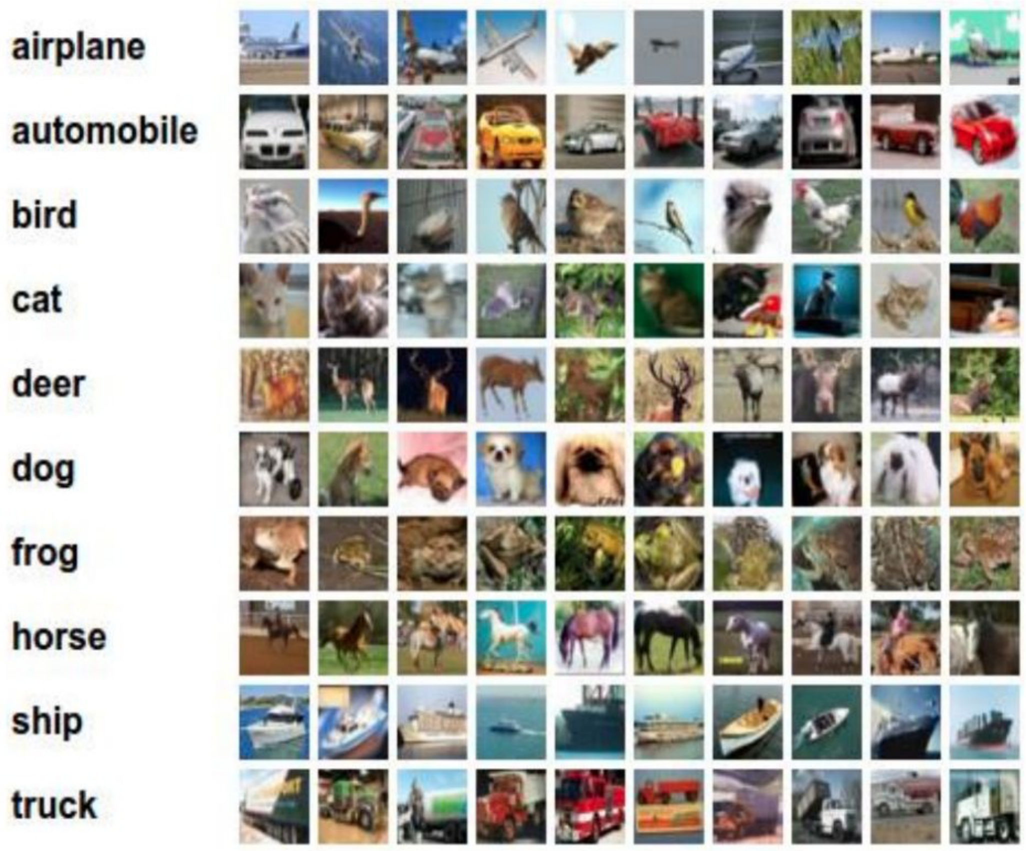
# CMPUT 328 Fall 2020
# Assignment 3 Worth 10% of the total weight

## Image Classification on CIFAR-10 Dataset using CNN

Your task in this assignment is to do classification on CIFAR-10 dataset using PyTorch CIFAR-10 dataset:

Contains 32x32x3 RGB images. There are 50000 images in the train set and 10000 images in the test set. Each image in CIFAR-10 dataset belongs to one of the ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.



For more information, see https://www.cs.toronto.edu/~kriz/cifar.html

## A. Tasks:

In this assignment, you are going to implement the Convolutional Neural Network to do classification on CIFAR-10 dataset.

The network architecture in this assignment is not fixed. You will design the network architecture yourself.

**However, there are some requirements that your network architecture must satisfy:**

a) Your network must have at least 5 layers (a layer means convolution or fully connected layer) into an activation function like Relu, Softmax, Tanh or sigmoid..., (input layer, max pooling or strided convolution layer used to downsample activation map doesn't count) The number of layers in your network will be very close to the number of activation functions)

b) Must have at least 1 convolution layer
c) Must have at least 1 fully connected layer at the end
d) Must have at least 1 max pooling or strided convolution layer (i.e. Convolution with stride > 1)
e) Must use batch normalization **(will be explained in section B)**
f) Must use dropout **(will be explained in section B)**
g) Must use skip connection **(will be explained in section B)**

**If your network architecture doesn't satisfy any of the above requirements, you will lose marks. For every requirement that is not satisfied, you lose 10% of your mark.**

**In addition to the above requirement, to do well in this assignment, you may have to:**

- Use He/Xavier initialization **(will be explained in section B)**
- Regularization
- Try different activation functions (relu, tanh, sigmoid, elu...)
- Try different optimizer (For example: Vanilla SGD, Adam, RMSProp, AdaDelta...)
- Try different loss function (For example: cross entropy, square error)
- Annealing the learning rate **(will be explained in section B)**

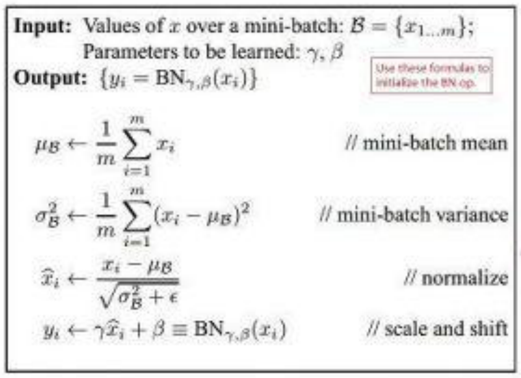
## B. Explanation for the terms mentioned above:

1. **Batch Normalization:** A tradition technique that is used frequently to reduce the effect of internal covariate shift in Deep Neural Network by normalizing the input into each activation function to have a zero mean and unit variance. Batch Normalization do this by computing the mean and variance of each batch during training time, then subtract the mean from the input and divide the result by variance. A linear transformation is applied after to allow the network to restore the original signal to the activation function. At test time, Batch Normalization use the mean and variance of the whole population set to normalize the input into activation function instead of using batch statistics. The mean and variance of the whole population are approximated by a moving mean and variance during training that is updated during training time.

   More information can be found at: https://arxiv.org/abs/1502.03167(the original paper), https://gab41.lab41.org/batch-normalization-what-the-hey-d480039a9e3b,

   https://wiki.tum.de/display/lfdv/Batch+Normalization

**Full Batch Normalization algorithm:**

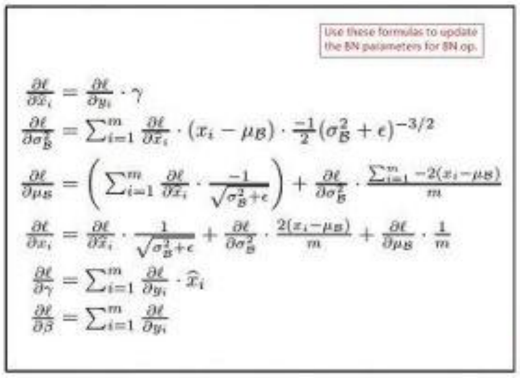


```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
       Parameters to be learned: $\gamma, \beta$
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

Use these formulas to initialize the BN op.

$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                      // mini-batch mean
$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$     // mini-batch variance
$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$                  // normalize
$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$          // scale and shift
```

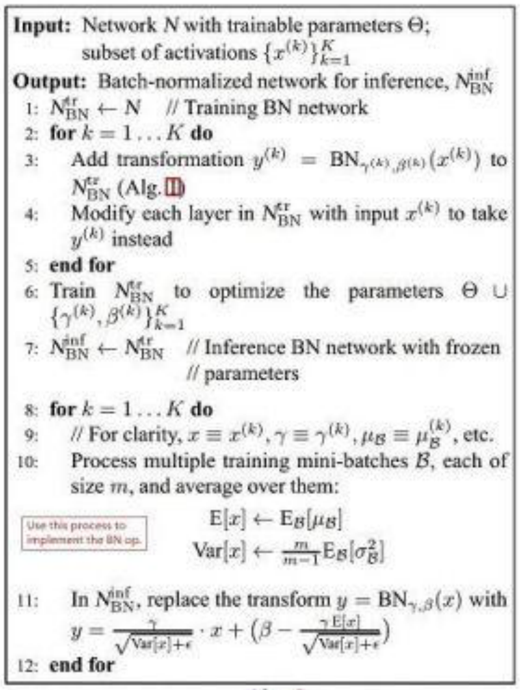
Alg. 1



Use these formulas to update the BN parameters for BN op.

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$
$$\frac{\partial \ell}{\partial \sigma_\mathcal{B}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_\mathcal{B}) \cdot \frac{-1}{2}(\sigma_\mathcal{B}^2 + \epsilon)^{-3/2}$$
$$\frac{\partial \ell}{\partial \mu_\mathcal{B}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}\right) + \frac{\partial \ell}{\partial \sigma_\mathcal{B}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_\mathcal{B})}{m}$$
$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_\mathcal{B}^2} \cdot \frac{2(x_i - \mu_\mathcal{B})}{m} + \frac{\partial \ell}{\partial \mu_\mathcal{B}} \cdot \frac{1}{m}$$
$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$
$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Formula set 1



```
Input: Network $N$ with trainable parameters $\Theta$;
       subset of activations $\{x^{(k)}\}_{k=1}^{K}$
Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$
 1: $N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$    // Training BN network
 2: for $k = 1 \ldots K$ do
 3:     Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to
        $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
 4:     Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take
        $y^{(k)}$ instead
 5: end for
 6: Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup$
    $\{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
 7: $N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$   // Inference BN network with frozen
                     // parameters
 8: for $k = 1 \ldots K$ do
 9:     // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_\mathcal{B} \equiv \mu_\mathcal{B}^{(k)}$, etc.
10:     Process multiple training mini-batches $\mathcal{B}$, each of
        size $m$, and average over them:
Use this process to implement the BN op.
            $\mathrm{E}[x] \leftarrow \mathrm{E}_\mathcal{B}[\mu_\mathcal{B}]$
            $\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_\mathcal{B}[\sigma_\mathcal{B}^2]$
11:     In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with
        $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
12: end for
```

Alg. 2

<u>**Notes:**</u>

- Batch Normalization should always be applied after a matrix multiplication and before an activation function.
- Batch Normalization are usually not applied for the first layer after the input layer as the statistics of the input layer can be computed easily from the dataset to do a simple normalization instead of batch normalization.
- Batch Normalization should NEVER be applied to the output layer of a network. Doing so will force the output of a network to follow a certain distribution which is usually not true.

**Batch Normalization in PyTorch:** There is a high-level API function that can be used to implement Batch Normalization in PyTorch (A simple Google search would yield plenty results).

---

2. **Dropout:** Dropout is a regularization technique used to prevent overfitting in training Deep Neural Networks. Dropout drops random unit (along with their connections) from the neural network during training. This prevents units from co-adapting too much. Dropout has somewhat fallen out of favor since Batch Normalization has similar regularization effect. However, it almost never hurt if Dropout is used correctly in a neural network. Dropout are usually used at the layer just before the output layer of a neural network. Dropout probability is usually 0.5 or 0.2 (correspond to dropout_kept_prob of 0.5 or 0.8)

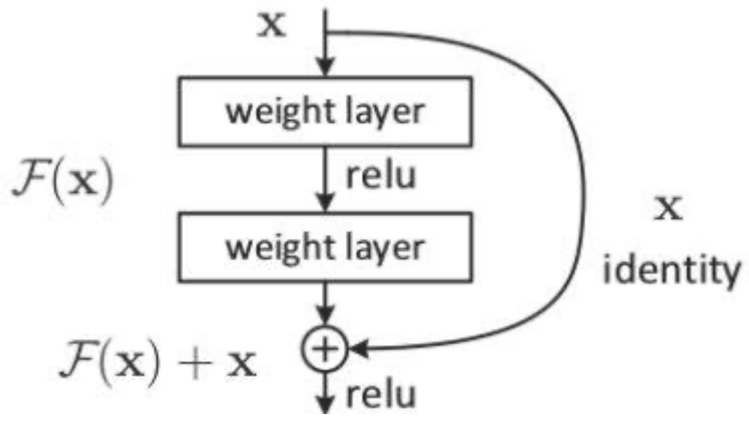
Note that dropout also have different training and test time behaviors. In practice, during training phase, dropout kept probability is set to a value that is less than 1.0 but during testing that value must be set to 1.0

For more information: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf (original paper)

---

3. **Skip connections:** In Deep Learning, skip connections are simply connection from an earlier layer in the network to a later layer. This allows better gradients for very deep networks during training and helps improve performance. You can find a clear example of them in ResNet, in each residual block.

   For more information: https://arxiv.org/abs/1512.03385,
   https://stats.stackexchange.com/questions/56950/neural-network-with-skip-layer-conn ections
   **An example of a skip connection:**


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

In PyTorch, a simple implementation of skip connection is just adding the input into an activation function by the output of an earlier layer.

---

**4. Use He/Xavier initialization:**
See https://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization

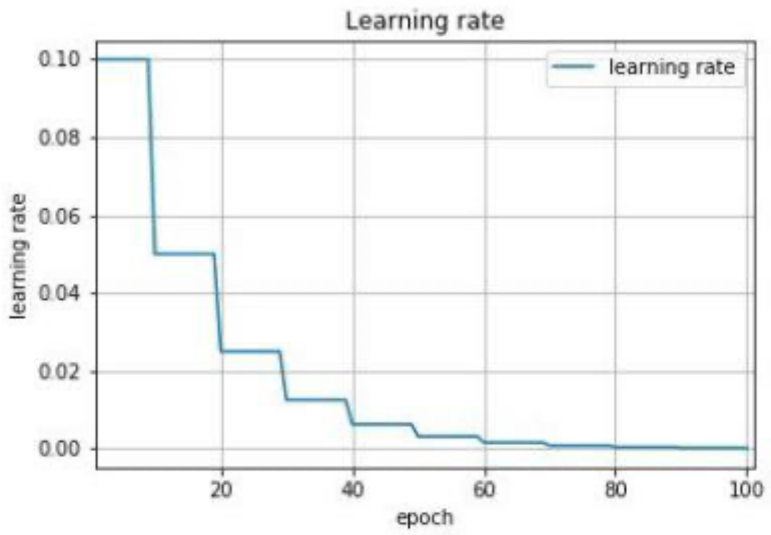
You don't need to implement it manually, should be straightforward to do in PyTorch.

---

**5. Learning Rate Annealing:**

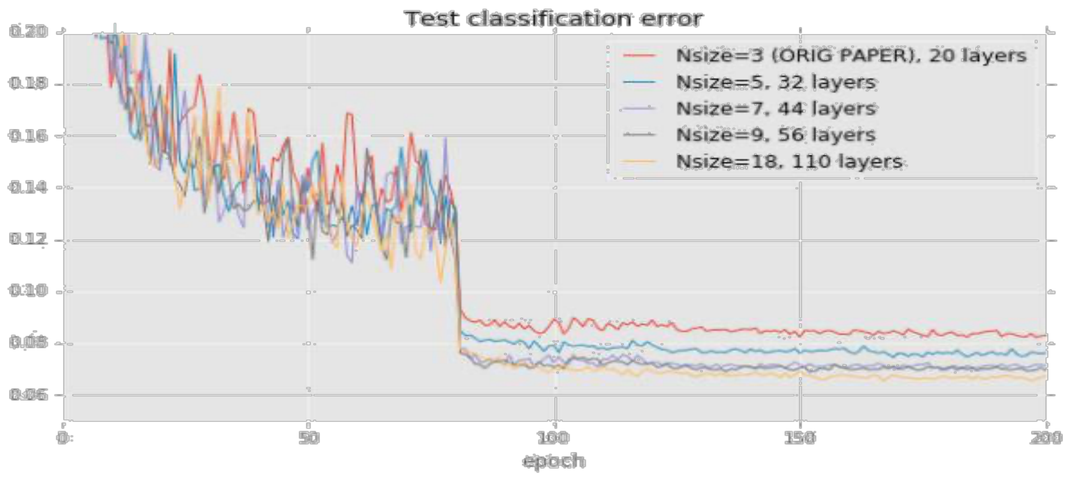
During training deep network, it's a good idea to drop your learning rate overtime. Most optimizer assume the learning rate decreasing overtime as a necessary condition for convergence. Decreasing the learning rate overtime allow your network to search on a finer scale of the parameters space and allow improvement of performance.

How much to drop the learning rate and when to drop is important. Drop the learning rate too soon or too much and your network will be very slow to train. A common scheme is to halve the learning rate after a few epochs similar to a staircase function. Another scheme is progressively reduce the learning rate after every iteration. **An example of a staircase learning rate:**



**An example on effect of learning rate annealing (the part where test error drop dramatically is the part where the learning rate is dropped significantly)**

More information: http://cs231n.github.io/neural-networks-3/#anneal



## C. What to do:

You are given some code. You are advised to build upon what you implemented in Assignment 3. There are 4 main parts in the code you need to complete:

1. load_dataset: You should load cifar-10 dataset, create the data loaders, split the dataset into training, validation and testing. Take the last 5k of the training data to be validation data (not a random split)

2. Network/model: you should define the network class in this part + any utilities related to weight/bias initialization. The architecture of the network you design inside this function must satisfy the all requirements mentioned in section A. For each requirement that is not satisfied, you lose 10% of your mark on this assignment.

3. Train: you should have the training and validation in this part. However, in this assignment, you should and probably have to add code to save your best model during training in order to resume your training from the best checkpoint and avoid training from scratch (whenever training stops) as your model will train for long. Best model is chosen based on the accuracy on the validation data. We also want this to be saved so that training and testing can run separately.

4. Test: you should test the model in this part and return predictions. You function should load the trained saved model, as the training function will not be called during marking time.

## D. SUBMISSION:

You need to submit 2 files:

1. Download the notebook as **.py file (not .ipynb)** and submit it.
2. Submit the saved model **(model_YOURIDNUMBER.pt)** **(IMPORTANT!)**

**DUE DATE:** The due date is Friday, **October 16th by 11:59 pm**. The assignment is to be submitted online on eclass. For late submissions' rules please, check the course information on eclass

## E. MARKING:

A third of the marks will be for documentation and remaining two-thirds will be for the code:
Code marks will depend on correctness of the implementation and following metrics:

**1. Accuracy:** Your algorithm will get the marks based on its accuracy on Cifar-10 dataset:

- <= 0.65: 0
- >= 0.80: Full
- 0.65 <= accuracy < 0.80: Scaled

**2. Run timw:** Your whole program (testing part) should not run for more than **100s (With Google colab CPU), using batch size of 1 for the test data-loader.**

**COLLABORATION POLICY:** This must be your own work. Do not share or look at the code of other people (whether they are inside or outside the class). Do not search for or copy the code from the Internet.

You can talk to others that are in the class about solution ideas (but not so detailed that you are verbally sharing or hearing about or seeing the code). You must cite whom you talked to in the comments of your programs.