

I SRN declare that I have completed this assignment completely and entirely on my own, without any consultation with others.  I understand that any breach of the UAB Academic Honor Code may result in severe penalties.

Srushti R Nayak

SP2024 CS Deep Learning

Hw4: CNN

**CNN for Image classification**

1. **Background and Method Introduction:**

**CNN are better than simple 2-layer neural network in terms of that CNNs can capture spatial hierarchies and patterns in image data with the use of convolutional layers. CNNs are used commonly in image recognition.**

**CNN typically consists of multiple layers. It has different type of layers including convolutional layers, pooling layers, fully connected layers.**

* **Convolution layers consists of set of learnable filters that are convolved with the input data to produce feature maps.**
* **Pooling layers are used to reduce dimensions. It helps model to prevent from overfitting. It basically reduces spatial dimension while keeping important information.**

**In CNN, images are given as input to the neural network and CNN gradually extracts features from input data. The final layer usually consists of one or more fully connected layers which produces the output predictions.**

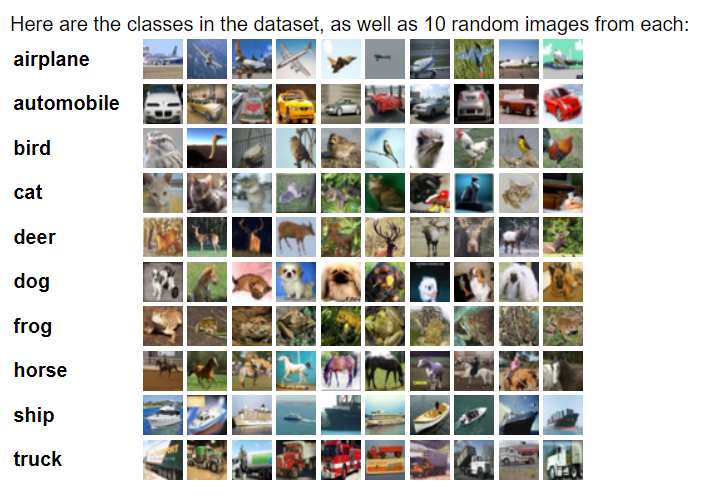
**Similar to other neural networks, loss is computed during training and using backpropagation, gradients are computed. According to computed gradients, weights and bias are adjusted. Activation functions are used to find complex non-linear patterns. To prevent overfitting, techniques such as regularizations, dropout are used.**

1. **Dataset and Tasks Description**:

**2.1. CIFAR10 dataset:**

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.



**Fig1: Image classes in CIFAR10 database**

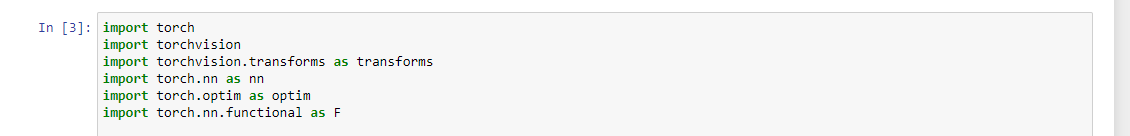
**Image from:** [**https://www.cs.toronto.edu/~kriz/cifar.html**](https://www.cs.toronto.edu/~kriz/cifar.html)

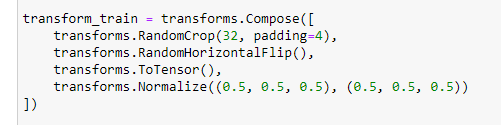
**2.2. Tasks Description:**

**This report details how to use python’s libraries to build CNN on the CIFAR-10 dataset. It also describes different optimizers and effect of each optimizer on CIFAR10. Additionally, it examines the impact of incorporating L2 regularization. Furthermore, the report provides insights into enhancing the training process of the neural network model through effective hyperparameter tuning.**

1. **Algorithms Used**:

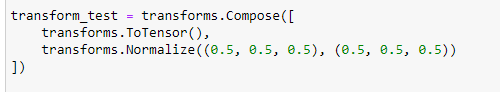
**3.0 Loading dataset and preparing data:**

* Importing required libraries:  
  
* Performing transforms on input data:  
   - for train

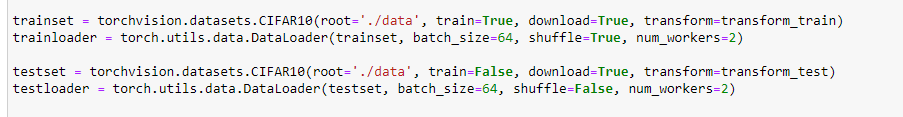


* Data augmentation to include variations in training data
  + Input train images are cropped randomly to the size of 32x32 pixels with padding of 4 pixels on each side.
* Randomly flipping input images with 0.5 probability to increase diversity
* Converting input images into pytorch tensors for further processing.
* Normalizing tensor images with mean and standard deviation of 0.5 for every channel.

- for test



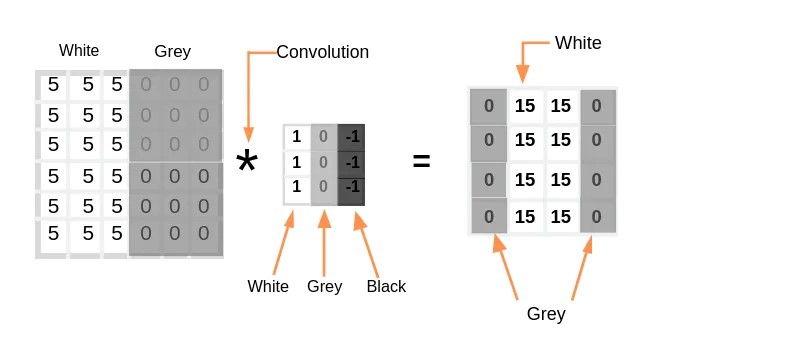
* Test data does not need augmentation and other transformations like training data.
* Test data are only converted into tensors and normalized.



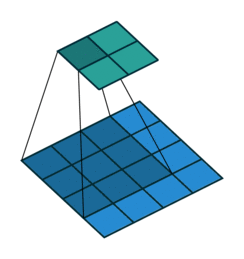
The DataLoader helps in training by organizing data into batches. Here, 64 samples are in each batch is set and shuffling is set to true, extra workers(multiple subprocesses introducing parallelization) are used for faster loading.

**3.1 Layers**

* **Convolution layer:**



**[fig1 Convolution operation:** [**https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529**](https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529)**]**



**[fig2: Visualization of convolution:** [**https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529**](https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529)**]**

Convolution involves applying various filters on incoming data to extract features. As shown in fig1 and fig2. As shown in fig1, filter is applied on grayscale image to detect edge.

* **Pooling layer**

Pooling layer is used to reduce dimensionality. It downsamples the data and prevent overfitting. It keeps important part to move forward eliminating redundant part.

Max pooling:

In max pooling, in part of input data let’s say of size 2\*2, the highest value is taken to move forward.

Ex. [1 0

4,2]

From this part of data, only 4 will move further in process.

* **Fully connected layer**

In fully connected layer, each neuron is connected with the neuron in next layer just like traditional Artificial neural network. It is different than convolution which operates on local input region. Fully connected layer carries global information.

**3.2 Activation functions:**

Various activation functions are available to use

|  |  |
| --- | --- |
| Activation function | formula |
| Sigmoid | f(x) = |
| Tanh | f(x) = |
| ReLU | f(x) = 0, x<0  x, x>=0 |
| Leaky ReLU | f(x) = max(0.1\*x,x) |
| Parametric ReLU | f(x) = max(ax,x) |
| ELU | f(x) = x, x>=0  , x<0 |

In this report ReLU is used with all the optimizers.

**3.3 Optimizers:**

**3.3.1: SGD**

SGD stands for stochastic gradient descent. It is optimizing technique to minimize loss function in which parameters (such as weights and bias) are updated using a single sample or a small of sample at a time. It is different than the traditional gradient descent where entire dataset is used to update parameters. SGD makes training fast and memory efficient.

Although, SGD is good optimizer, it has some drawbacks where it fails:

* If the loss function has local minima or saddle point
* Zero gradient
* Gradient comes from minibatches so it can be noisy

To solve problems with SGD, we use other optimizers such as SGD+Momentum, Adagrad, RMSProp, Adam.

**3.3.2: SGD with Momentum**

Momentum helps accelerate SGD using result of previous iteration. It adds the fraction of the update vector from previous iteration to the current update. So momentum controls the influence of the previous update on the current update. It prevents SGD from sticking into local minima and fastens the convergence process.

**3.3.3: ADAgrad**

ADAgrad stands for Adaptive Gradient Algorithm.

In process of convergence some parameters are reaching minima faster and some are slower. So, process will become quick if use different learning rate for different parameters. ADAgrad follows similar concept. It adapts the learning rate for each parameter based on the historical gradients of that parameter observed for that parameter during training. ADAgrad is more useful when there is sparse data with large number of features.

The main idea behind AdaGrad is to give a larger learning rate to parameters that have been updated infrequently and a smaller learning rate to parameters that have been updated frequently.

It maintains sum of the squared gradients of each parameter.

ADAGrad sometimes makes learning rate too small and this can lead to slow convergence.

**3.3.4: RMSProp**

RMSProp stands for Root Mean Square Propagation. It overcomes the limitation of ADAgrad. It introduces the concept of decay rate. RMSProp maintains an exponentially decaying average of the squared gradients for each parameter. RMSProp also adapts learning rate for each parameter. But due to decay rate it prevents learning rates form decreasing too rapidly and allows faster convergence and improved performance.

**3.3.5: Adam**

Adam stands for Adaptive Moment Estimation. It has benefits of Momentum and RMSProp. It incorporates adaptive learning rate and momentum.

Adam maintains exponentially decaying moving averages of past gradients (first moment) and squared gradients (second moment) for each parameter. It uses two decay rates b1 and b2 for updating moving averages. It uses bias correction as initially both first and second moment are kept as 0 and so they are bias towards 0.

**3.4 Regularization:**

Regularization is used to prevent model from overfitting the training data. It adds regularization penalty and stop any parameter from dominating. There are mainly 3 types of regularizations.

R1or lasso regularization

R2 or Ridge regularization

Elastic net regularization

In this assignment, only L2 regularization is used.

**3.5 Batch Normalization:**

Batch normalization is added between layers to improve training stability and faster convergence. In batcg normalization, activations in each layer are normalized in mini batches.

For each feature (or neuron) in a layer, batch normalization normalizes its activations by subtracting the mean and dividing by the standard deviation of the activations across the mini-batch.

Batch normalization is usually added after fully connected or convolutional layers and before nonlinearity.

**[ppt 6: lec slide 56,** **loffe and Szegedy, 2015].**

1. **Results**

Rest of the hyper parameters are kept same.

|  |  |  |
| --- | --- | --- |
| **Optimizer** | **Regularization** | **Accuracy** |
| SGD | No | **26%** |
| SGD + Momentum | No | **56%** |
| ADAgrad | No | **51%** |
| RMSProp | No | **65%** |
| Adam | No | **78%** |
| RMSProp | Yes | **65%** |
| Adam | Yes | **76%** |

* It can be seen that regularization is not good choice. Adding regularization has no difference for RMSProp and for Adam accuracy is decreased after adding regularization.
* Among all the optimizers, Adam achieves highest accuracy.
* SGD can be improved by adding momentum. More than 100%(from 26% to 56%. 100% increase for 26 = 52) increase in accuracy can be in seen after adding momentum.
* SGD+Momentum achieves better accuracy than ADAgrad.

1. **Methods of improvement**
2. Batch normalization
3. Hyper parameter tuning

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Optimizer** | **Batch normalization** | **Epochs** | **Learning rate** | **Accuracy** |
| RMSProp | Yes | 10 | 0.001 | 75% |
| Adam | Yes | 10 | 0.001 | 78% |
| RMSProp | Yes | 15 | 0.005 | 77% |
| Adam | No | 15 | 0.005 | 61% |
| RMSProp | Yes | 15 | 0.001 | 77% |
| Adam | No | 15 | 0.001 | 80% |

* Adam optimizer gives better accuracy than RMSProp
* Adding batch normalization improves the accuracy of RMSProp to great extent.
* Adam performs really well even without adding batch normalization.
* Increased learning rate can be reason for reduced accuracy.
* Increasing epochs also improves algorithm’s performance.

1. **Conclusion:**

**Following conclusions can be drawn in terms of CIFAR10 datset.**

* Adam optimizer outperforms all other optimizers across various configurations.
* Momentum can significantly improve accuracy of SGD.
* Adding regularization does not contribute in increasing accuracy.
* Adding batch normalization improves accuracy.
* Increasing learning rate too much might result in fail to find minima and overshooting can happen.
* Increasing epochs improves accuracy.
* Optimal configuration: Adam optimizer with 0.001 learning rate and 15 epochs.

1. **References:**

[**https://www.cs.toronto.edu/~kriz/cifar.html**](https://www.cs.toronto.edu/~kriz/cifar.html)

[**https://pytorch.org/**](https://pytorch.org/)

[**https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529**](https://towardsdatascience.com/covolutional-neural-network-cb0883dd6529)

**Lecture slides**

**loffe and Szegedy, 2015**