National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering

Faculty Member: Dr. Latif Anjum **Date: 18/12/2024**

Course/Section: CV **Semester: 7th**

CS-477 Computer Vision

Lab 12: Implementation of a Simple Convolutional Neural Network

Lab 9: Implementation of a Simple Convolutional Neural Network

		PLO4 - CLO4	PLO 5 -CLO5	PLO 8 -CLO6	PLO 9 -CLO7
Name	**Reg. No**	**Investigation** **5 Marks**	**Modern Tool Usage** **5 Marks**	**Ethics** **5 Marks**	**Individual and Team Work** **5 Marks**
Muhammad Anser Sohaib	**367628**				
Muhammad Saad	**372985**				
Zuha Fatima	**385647**				

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Objectives

Understand PyTorch's Tensor library and neural networks at a high level.
Introduction to CNN
Training a CNN classifier

Lab Report Instructions

All questions should be answered precisely to get maximum credit. Lab report must ensure following items:

- ✓ Lab objectives
- ✓ Python codes
- ✓ Results (graphs/tables) duly commented and discussed
- ✓ Conclusion

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Introduction to CNN

Convolutional Neural Network (CNN): A CNN is a type of deep neural network designed to recognize and process visual data with a grid-like structure, such as images. CNNs are particularly effective in image recognition, object detection, and other computer vision tasks.

Grid-like Topology: Images can be thought of as a grid of pixels, where each pixel represents the smallest unit of information. The arrangement of pixels creates a

grid-like structure, and CNNs leverage this spatial organization for more effective feature extraction.

Digital Image: In the context of CNNs, digital images are represented as a grid of pixels. Each pixel's position in the grid corresponds to a specific location in the image, and the pixel value represents the color and intensity at that point. The combination of pixel values across the grid forms the complete visual representation of the image.

Binary Representation: While you mentioned a binary representation, it's important to note that digital images typically use a range of values to represent colors. In the RGB (Red, Green, Blue) color space, for example, each pixel is often represented by three values corresponding to the intensity of each color channel. The values are usually integers ranging from 0 to 255, or they could be normalized to the range [0, 1].

Pixel Values: Pixel values indicate the brightness and color information of the image. In grayscale images, each pixel has a single value representing intensity. In color images, each pixel has multiple values corresponding to the intensities of different color channels.

CNNs use convolutional layers to automatically and adaptively learn spatial hierarchies of features from input images. These layers contain filters that are convolved with the input image to extract features such as edges, textures, and more complex patterns. Pooling layers are often used to reduce the spatial dimensions of the data, and fully connected layers integrate the learned features for final classification or regression tasks.

In summary, CNNs are a powerful class of neural networks designed for processing grid-like data, making them especially effective for tasks involving images and spatial relationships within those images.

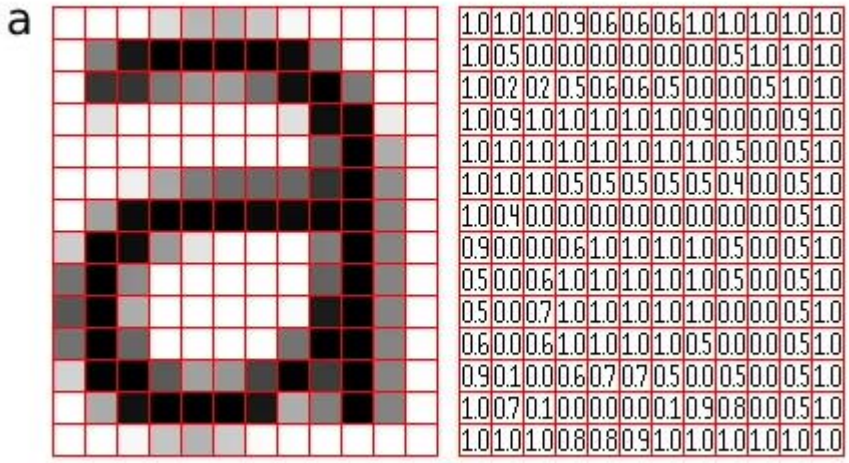


Figure 1: Source: https://pippin.gimp.org/image_processing/images/sample_grid_a_square.png

Convolutional Neural Network (CNN) typically consists of three fundamental layers: a convolutional layer, a pooling layer, and a fully connected layer.

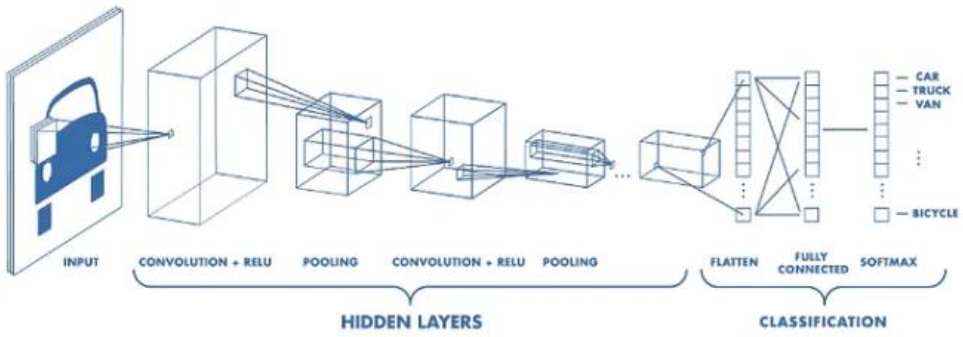


Figure 2: Source https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

Convolution Layer

At the core of the CNN architecture lies the convolutional layer, bearing the primary computational load of the network. This layer executes a dot product operation between two matrices – one matrix comprises learnable parameters known as a kernel, and the other matrix represents a confined section of the receptive field. While the kernel is spatially smaller than the image, it possesses greater depth. For instance, in an image with three (RGB) channels, the kernel's height and width are spatially small, yet its depth extends across all three channels.

Illustration of Convolution Operation

In the forward pass, the kernel traverses the height and width of the image, generating the image representation of the receptive region. This process yields a two-dimensional representation known as an activation map, showcasing the kernel's response at each spatial position within the image. The extent of movement of the kernel is termed as the "stride."

For an input of size W x W x D, with Dout being the number of kernels, a spatial size of F, a stride of S, and a specified amount of padding P, the size of the output volume is determined by the following formula:

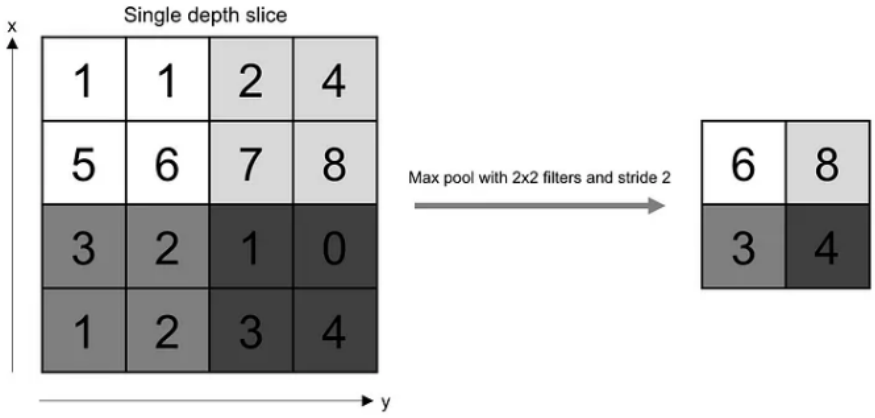
$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Formula for Convolution Layer

Pooling Layer

Following the convolutional layer, the pooling layer plays a crucial role in the Convolutional Neural Network (CNN) architecture. Its function involves replacing specific locations in the network's output by computing a summary statistic of the neighboring outputs. This strategic substitution aids in diminishing the spatial size of the representation, leading to a reduction in computational demands and the number of weights. Notably, the pooling operation is applied independently to each slice of the representation.

Various pooling functions exist, each offering distinct ways of summarizing information within a neighborhood. Options include the average of the rectangular neighborhood, the L2 norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel. However, among these, max pooling stands out as the most widely employed method. Max pooling entails selecting the maximum output value from the neighborhood, providing a robust approach to retaining essential features while reducing the dimensionality of the representation.



If we have an activation map of size W x W x D, a pooling kernel of spatial size F, and stride S, then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F}{S} + 1$$

Formula for Padding Layer

This will yield an output volume of size Wout x Wout x D.In all cases, pooling provides some translation invariance which means that an object would be recognizable regardless of where it appears on the frame.

Fully Connected Layer

Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect. The FC layer helps to map the representation between the input and the output.

Non-Linearity Layers

Given that convolution is inherently a linear operation and images exhibit significant non-linearity, non-linearity layers are frequently inserted immediately after the convolutional layer to impart non-linearity to the activation map.

Several types of non-linear operations are commonly employed, with notable examples including:

Sigmoid:

The sigmoid non-linearity is expressed mathematically as $\sigma(\kappa) = 1/(1+e^{-}\kappa)$. It takes a real-valued number and compresses it into a range between 0 and 1. However, a drawback of the sigmoid function is its tendency to generate gradients close to zero, particularly at its tails. This property can hinder the backpropagation process, potentially causing the gradient to become too small and impede effective weight updates. Additionally, when the input data is consistently positive, the sigmoid may produce outputs that are either entirely positive or entirely negative, leading to a zig-zag dynamic in gradient updates for weights.

Tanh:

Tanh transforms a real-valued number to the range [-1, 1]. Similar to sigmoid, tanh activations can saturate, but unlike sigmoid, its output is zero-centered.

ReLU (Rectified Linear Unit):

ReLU has gained widespread popularity in recent years. It computes the function $f(\kappa) = \max(0, \kappa)$, effectively thresholding the activation at zero. Compared to sigmoid and tanh, ReLU offers faster convergence, accelerating the learning process by a factor of six. Despite its advantages, ReLU does have a potential drawback during training. If a large gradient flows through it, the neuron may undergo an update in such a way that further updates become unlikely. This issue can be mitigated by carefully selecting an appropriate learning rate

Watch following video to understand how CNN works

https://www.youtube.com/watch?v=HGwBXDKFk9I

Task 1: Convolution on Images _____

Download a color image for this task. Write a function in python that takes as input arguments: an image, a square filter (3x3, 5x5 etc.), padding size and number of strides. The output of the function must be an image undergoing the convolution operation on the input image.Implement convolution and showcase the result by trying different filters, padding values and number of strides. Provide the code and at least 4 screenshots for the final outputs.

TASK 1 EXPLANATION STARTS HERE

Convolution is a fundamental operation in image processing and computer vision. It involves applying a filter (kernel) to an image to extract features like edges, textures, or patterns. The operation slides the filter over the input image, computes weighted sums of pixel values, and produces an output image.

Key Parameters:

1. **Image**: The input image to which the convolution will be applied.
2. **Filter**: A square matrix (e.g., 3x3, 5x5) containing weights used for convolution.
3. **Padding**: Adds extra pixels around the edges of the input image to control the size of the output. Common types are:
    - Zero Padding: Fills the border with zeros.
    - Same Padding: Ensures the output size matches the input size.

4. **Strides**: The number of pixels the filter moves at each step. Larger strides reduce the size of the output image.

Steps:

1. **Prepare the image**: Load the image in color (RGB).
2. **Apply padding**: If specified, add padding to the image.
3. **Convolve the filter**: Perform element-wise multiplication of the filter with the image region it covers and sum the results. Repeat this operation while sliding the filter across the image using the specified strides.
4. **Return the output image**: Store and display the resulting image after the convolution.

TASK 1 EXPLANATION ENDS HERE

TASK 1 CODES START HERE

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

def apply_convolution(image, kernel, padding=0, stride=1):

    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # 
Convert to grayscale

    kernel_height, kernel_width = kernel.shape
    padded_image = np.pad(image, [(padding, padding), (padding, 
padding)], mode='constant', constant_values=0)

    output_height = (padded_image.shape[0] - kernel_height) // 
stride + 1
    output_width = (padded_image.shape[1] - kernel_width) // 
stride + 1

    output = np.zeros((output_height, output_width), 
dtype=np.float32)

    for y in range(0, output_height):
        for x in range(0, output_width):
```

```
            region = padded_image[y * stride:y * stride +
kernel_height, x * stride:x * stride + kernel_width]
            output[y, x] = np.sum(region * kernel)

    return np.clip(output, 0, 255).astype(np.uint8)

# Example filters
filter1 = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]])  # Edge
detection (Sobel)
filter2 = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])  #
Sharpening
filter3 = np.ones((3, 3)) / 9                               #
Smoothing (average)

# Read image
image = cv2.imread("house_image.jpeg")

# Convolution examples
output1 = apply_convolution(image, filter1, padding=1, stride=1)
output2 = apply_convolution(image, filter2, padding=1, stride=1)
output3 = apply_convolution(image, filter3, padding=1, stride=1)
output4 = apply_convolution(image, filter3, padding=2, stride=2)

# Plot results
fig, axs = plt.subplots(1, 5, figsize=(20, 10))
axs[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
axs[0].set_title("Original Image")
axs[0].axis("off")

axs[1].imshow(output1, cmap="gray")
axs[1].set_title("Edge Detection")
axs[1].axis("off")

axs[2].imshow(output2, cmap="gray")
axs[2].set_title("Sharpening")
axs[2].axis("off")

axs[3].imshow(output3, cmap="gray")
axs[3].set_title("Smoothing")
axs[3].axis("off")

axs[4].imshow(output4, cmap="gray")
axs[4].set_title("Smoothing (Stride=2)")
axs[4].axis("off")

plt.show()
```

TASK 1 CODES END HERE

TASK 1 SCREENSHOTS START HERE

TASK 1 SCREENSHOTS END HERE

Task 2: Simple CNN _____

Build a simple convolutional neural network in PyTorch and train it to recognize handwritten digits using the MNIST dataset (Training a *classifier* on the MNIST dataset can be regarded as the *hello world* of image recognition).

TASK 2 EXPLANATION STARTS HERE

This task involves creating a CNN using PyTorch to classify handwritten digits from the MNIST dataset. The process includes designing the network architecture, loading the dataset, training the model to optimize its parameters, and evaluating its performance. This serves as a foundational exercise in image recognition.

TASK 2 EXPLANATION ENDS HERE

TASK 2 CODES START HERE

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader


class SimpleCNN(nn.Module):
    def _init_(self):
        super(SimpleCNN, self)._init_()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1,
padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1,
padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 14 * 14, 128)
```

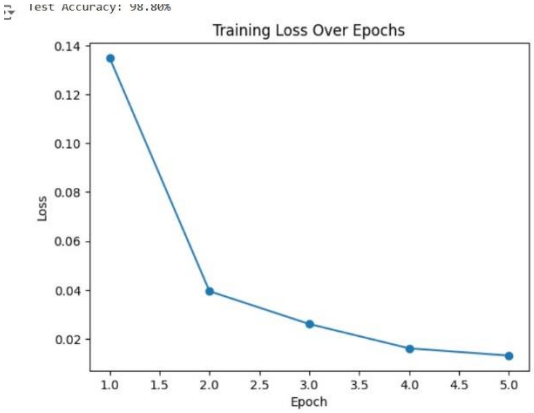
```
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = self.pool(torch.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x


device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')


transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = datasets.MNIST(root='./data', train=True,
download=True, transform=transform)
test_dataset = datasets.MNIST(root='./data', train=False,
download=True, transform=transform)

train_loader = DataLoader(train_dataset, batch_size=64,
shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64,
shuffle=False)

model = SimpleCNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epochs = 5
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)


        outputs = model(images)
        loss = criterion(outputs, labels)
```

```
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f"Epoch [{epoch+1}/{num_epochs}], Loss:
{running_loss/len(train_loader):.4f}")


model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Test Accuracy: {100 * correct / total:.2f}%")
```

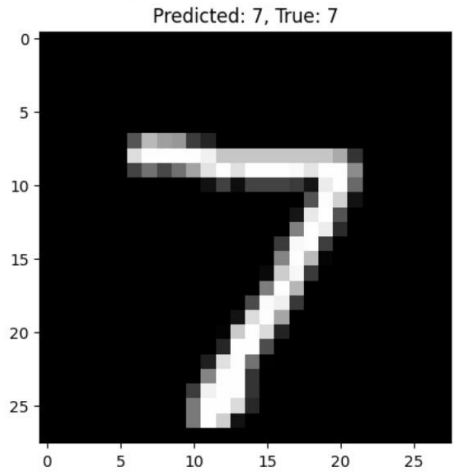
TASK 2 CODES END HERE

TASK 2 SCREENSHOTS START HERE

```
Epoch [1/5], Loss: 0.1354
Epoch [2/5], Loss: 0.0388
Epoch [3/5], Loss: 0.0250
Epoch [4/5], Loss: 0.0171
Epoch [5/5], Loss: 0.0125
Test Accuracy: 98.73%
```



First 10 Predictions:



TASK 2 SCREENSHOTS END HERE

Task 3: CNN _____

Build a simple convolutional neural network in PyTorch and train it to recognize following fashion object using the fashion MNIST dataset which contains 10 classes (T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shift, Sneaker, Bag, Ankle boot).

TASK 3 EXPLANATION STARTS HERE

This task involves building a CNN in PyTorch to classify images from the Fashion MNIST dataset, which contains 10 fashion item categories. The process includes designing the network, loading and preprocessing the dataset, training the model to identify patterns in the images, and evaluating its classification accuracy.

TASK 3 EXPLANATION ENDS HERE

TASK 3 CODES START HERE

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader, Subset
import matplotlib.pyplot as plt

# Check for GPU
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
```

```
print(f"Using device: {device}")

# Load Fashion MNIST dataset
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Subset for 20,000 images
train_dataset = datasets.FashionMNIST(root='./data', train=True,
transform=transform, download=True)
train_subset = Subset(train_dataset, range(20000))
test_dataset = datasets.FashionMNIST(root='./data', train=False,
transform=transform, download=True)

train_loader = DataLoader(train_subset, batch_size=64,
shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64,
shuffle=False)

# Define the CNN model
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1,
padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1,
padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(64 * 14 * 14, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(x.size(0), -1)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Instantiate model, move to device
model = CNN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

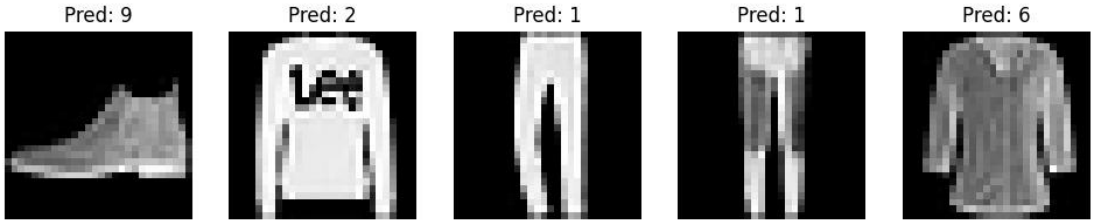
```
# Training loop
def train_model(model, train_loader, criterion, optimizer,
epochs=5):
    model.train()
    for epoch in range(epochs):
        running_loss = 0.0
        for images, labels in train_loader:
            images, labels = images.to(device),
labels.to(device)  # Move to GPU
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
        print(f"Epoch {epoch+1}, Loss:
{running_loss/len(train_loader):.4f}")

# Testing loop
def test_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device),
labels.to(device)  # Move to GPU
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    print(f"Test Accuracy: {100 * correct / total:.2f}%")

# Train and evaluate the model
train_model(model, train_loader, criterion, optimizer, epochs=5)
test_model(model, test_loader)

# Visualize some test results
dataiter = iter(test_loader)
images, labels = next(dataiter)
images, labels = images.to(device), labels.to(device)
outputs = model(images)
_, predictions = torch.max(outputs, 1)

# Plot the images and predictions
```

```
fig, axs = plt.subplots(1, 5, figsize=(12, 3))
for i in range(5):
    axs[i].imshow(images[i].cpu().squeeze(), cmap='gray')   # Move
to CPU for display
    axs[i].set_title(f"Pred: {predictions[i].item()}")
    axs[i].axis('off')
plt.show()
```

TASK 3 CODES END HERE

TASK 3 SCREENSHOTS START HERE

```
Using device: cuda

Epoch 1, Loss: 0.5292

Epoch 2, Loss: 0.3094

Epoch 3, Loss: 0.2448

Epoch 4, Loss: 0.1919

Epoch 5, Loss: 0.1520

Test Accuracy: 88.88%
```



TASK 3 SCREENSHOTS END HERE

Helpful links

https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939
https://www.youtube.com/watch?v=HGwBXDKFk9I
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
https://colab.research.google.com/github/pytorch/tutorials/blob/gh-pages/_downloads/4e865243430a47a00d551ca0579a6f6c/cifar10_tutorial.ipynb#scrollTo=PP9km88QkiZp