"Don't expect to be easy to work, because if you're easy to do, your heart will be flattered and arrogant."
_ Buddha

APT: PDE_Machine Learning

Presenter: Dr. Ha Viet Uyen Synh.

Warm-up: Face Recognition

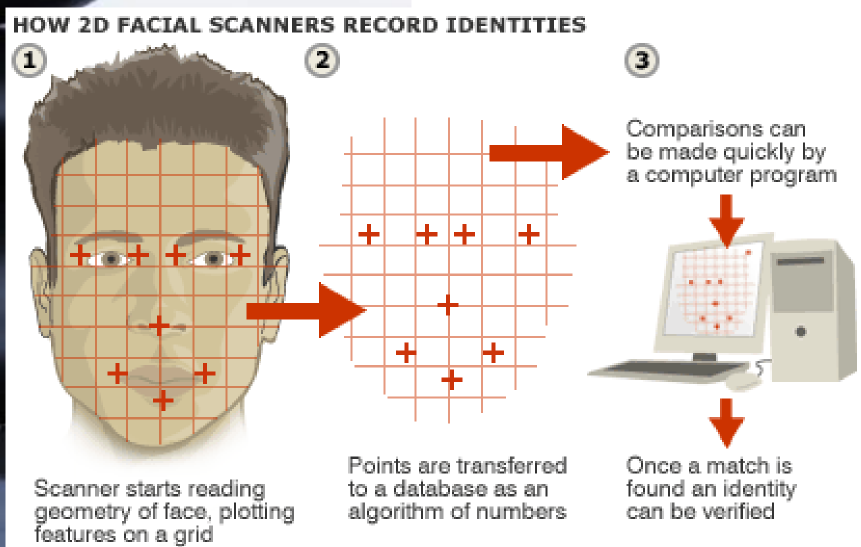
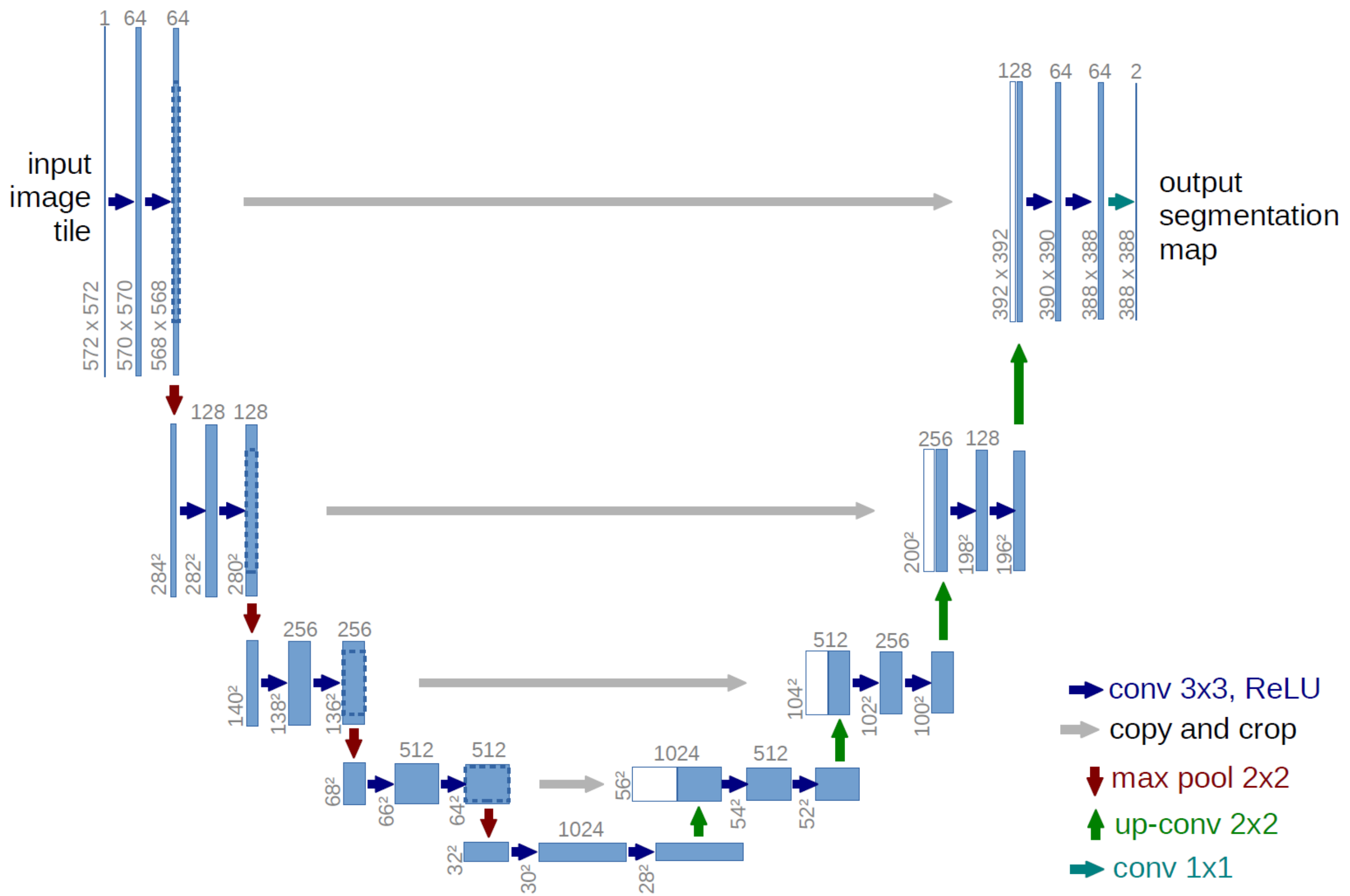


Image Segmentation

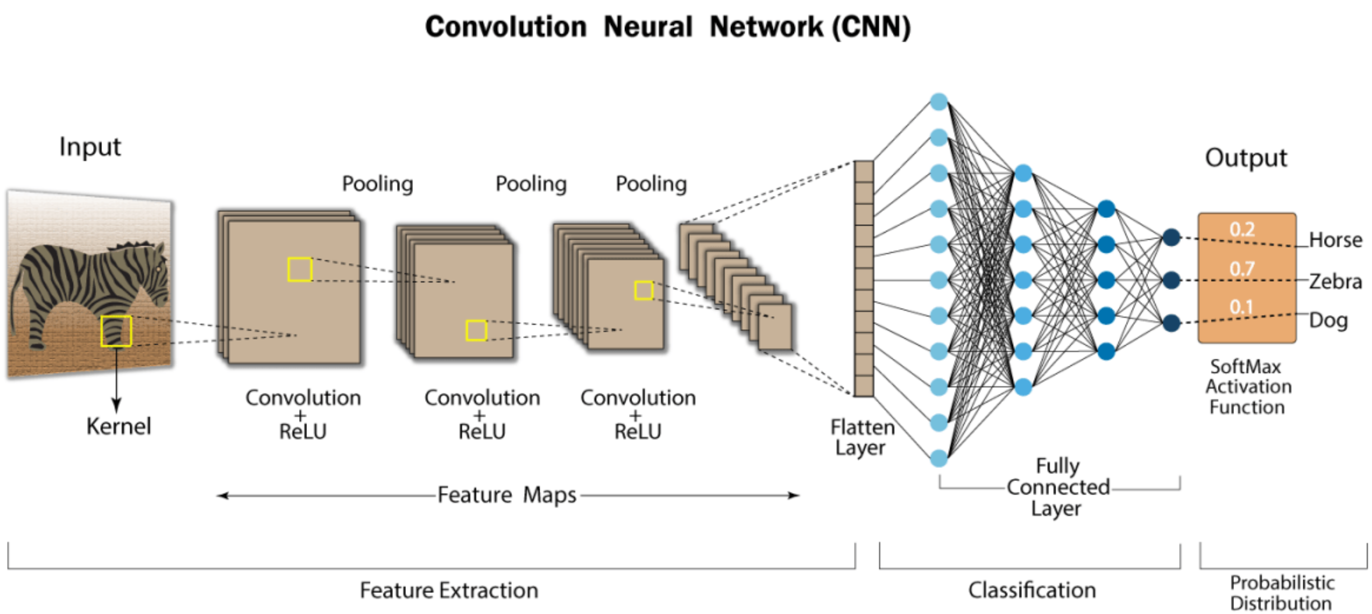
Warm-up: U-Net


input image tile
output segmentation map
1
64
64
572 x 572
570 x 570
568 x 568
128
128
284^2
282^2
280^2
256
256
140^2
138^2
136^2
512
512
68^2
66^2
64^2
1024
32^2
30^2
28^2
1024
512
56^2
54^2
52^2
512
256
104^2
102^2
100^2
256
128
200^2
198^2
196^2
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Part A

CONVOLUTIONAL NEURAL NETWORK

1. Convolutional Neural Network _ CNN



In CNN, each input image will pass through a sequence of convolution layers along with pooling, fully connected layers, filters (Also known as kernels). After that, we will apply the Soft-max function to classify an object with probabilistic values 0 and 1.

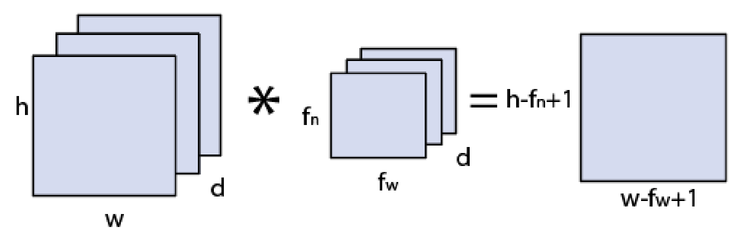
Convolution Layer

Convolution layer is the first layer to extract features from an input image. By learning image features using a small square of input data, the convolutional layer preserves the relationship between pixels. It is a mathematical operation which takes two inputs such as image matrix and a kernel or filter.

The dimension of the image matrix is **h×w×d.**

The dimension of the filter is **f_h×f_w×d.**

The dimension of the output is **$(h-\lfloor f_h/2 \rfloor+1)\times(w-\lfloor f_w/2 \rfloor+1)\times 1$.**



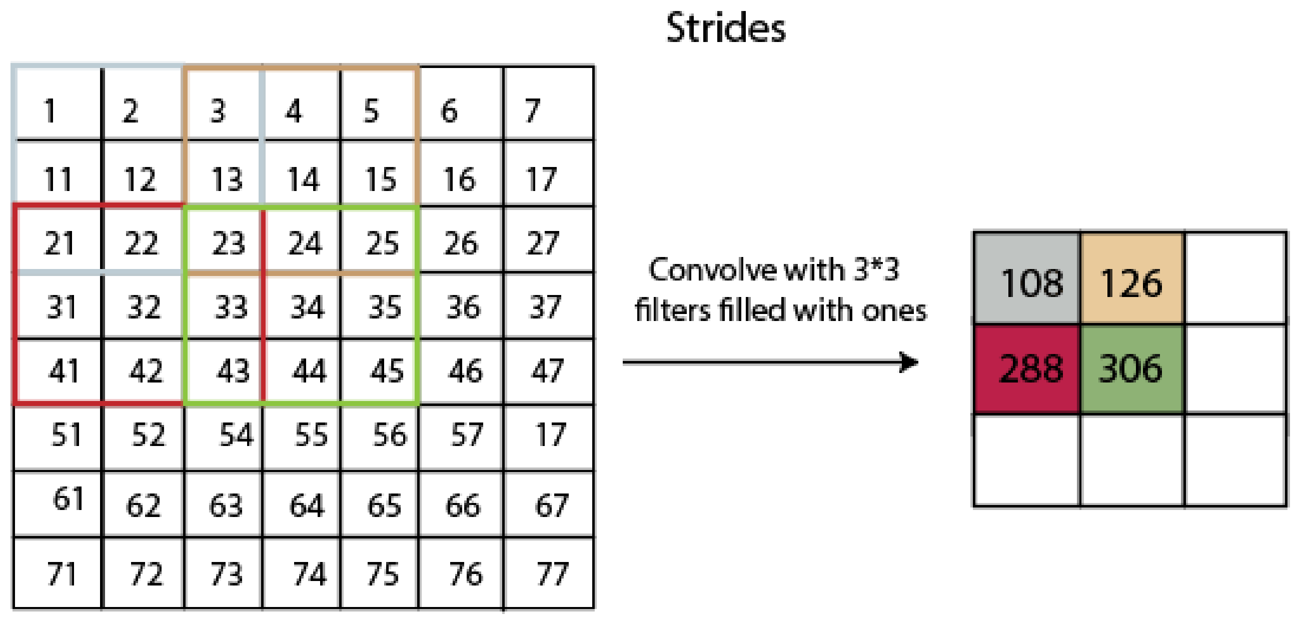
Ex:

Image matrix multiplies kernl or filter matrix

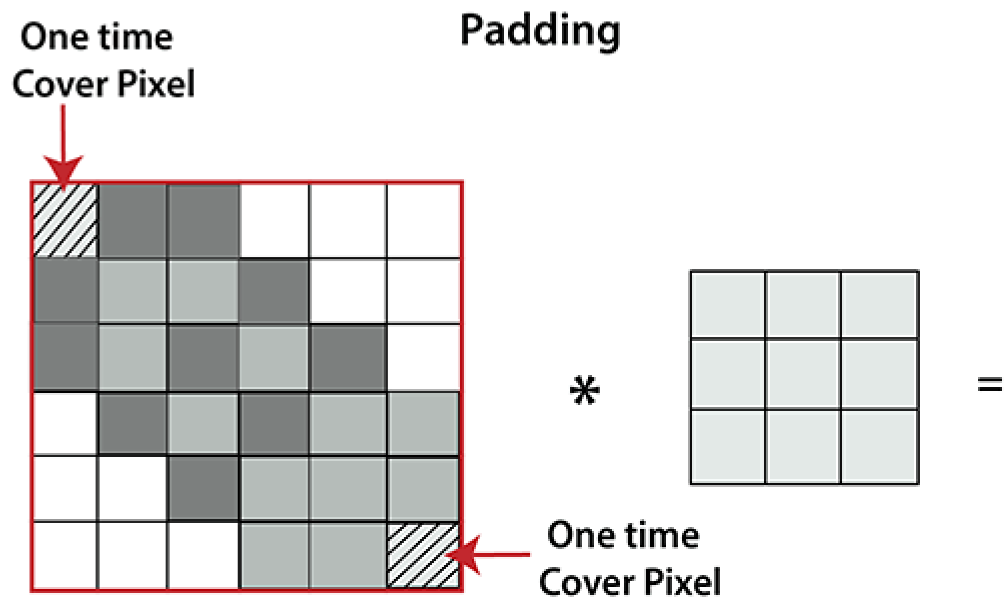
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 3 & 4 \\ 2 & 4 & 3 \\ 2 & 3 & 4 \end{bmatrix}$$

Convolved Feature

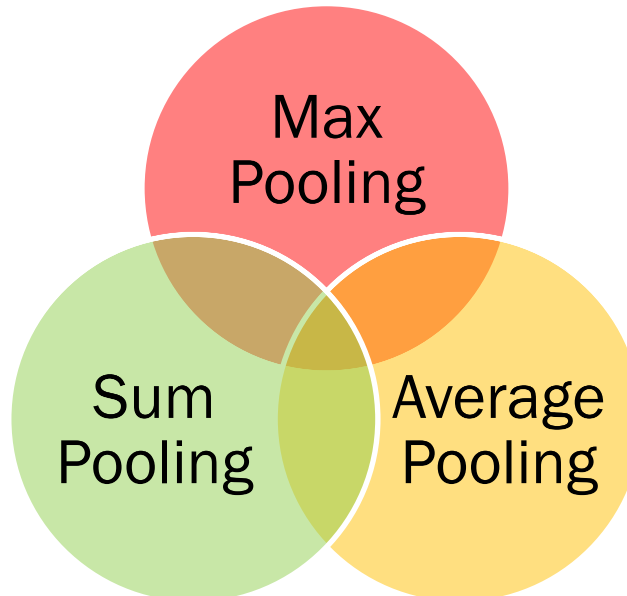
Strides (recall)



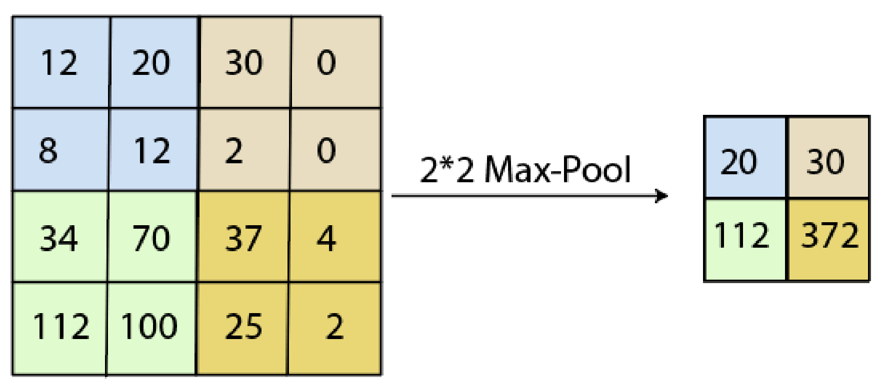
Padding (recall)



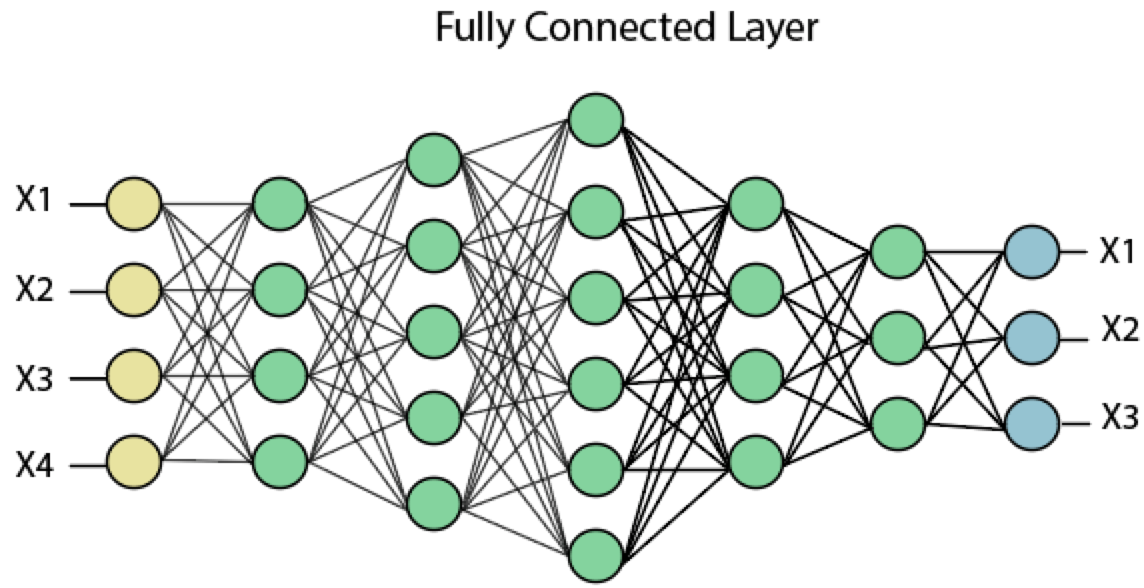
Pooling Layer

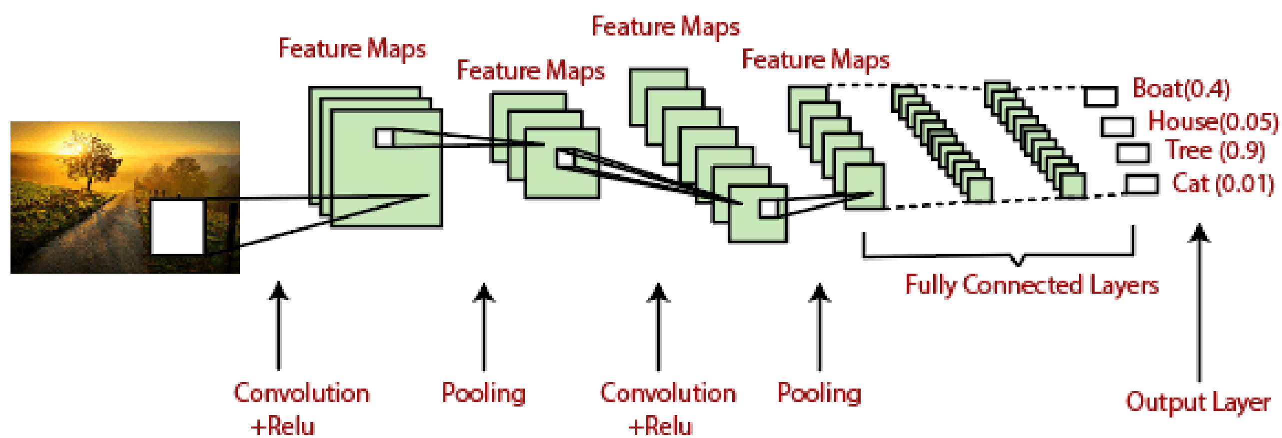


Max Pooling



Fully Connected Layer





2. Image Transforms

Step 1:

```
import torch
import matplotlib.pyplot as plt
import numpy as np
from torchvision import datasets,transforms
```

Step 2:

```
transform1=transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5,),(0.5,))])
training_dataset=datasets.MNIST(root='./data',train=True,download=True,transform=transform1)
training_loader=torch.utils.data.DataLoader(dataset=training_dataset,batch_size=100,shuffle=True)
```

Note: *Our MNIST images are 28*28 grayscale images which would imply that each image is a two dimensional number by array 28 pixels wide and 28 pixels long and each pixel intensity ranging from 0 to 255.*

Step 3:

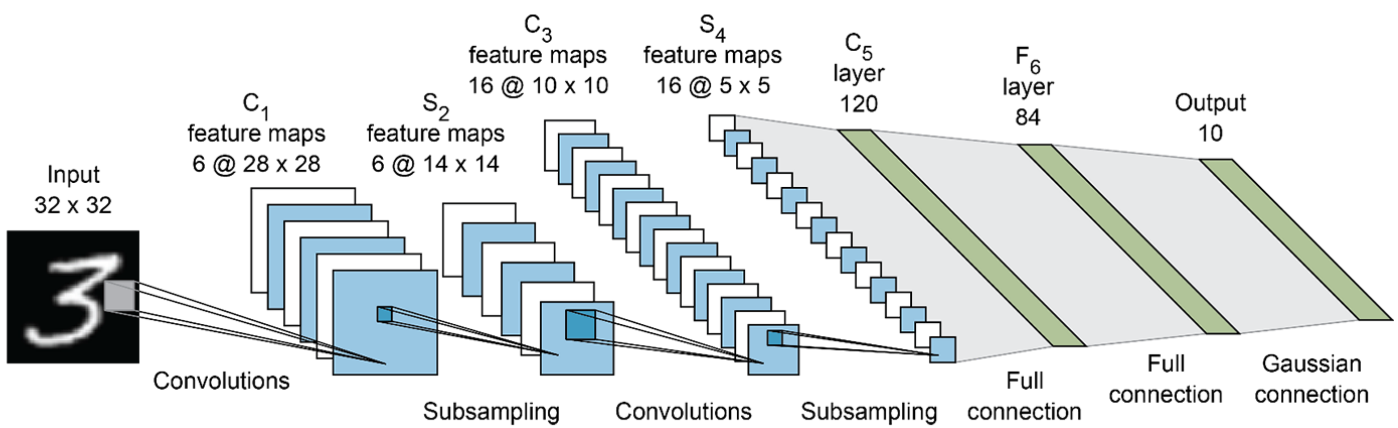
```
def im_convert(tensor):
    image=tensor.clone().detach().numpy()
    image=image.transpose(1,2,0)
    print(image.shape)
    image=image*(np.array((0.5,0.5,0.5))+np.array((0.5,0.5,0.5)))
    image=image.clip(0,1)
    return image
```

Step 4:

```
dataiter=iter(training_loader)
images,labels=dataiter.next()
fig=plt.figure(figsize=(25,4))
for idx in np.arange(20):
    ax=fig.add_subplot(2,10,idx+1)
    plt.imshow(im_convert(images[idx]))
    ax.set_title([labels[idx].item()])
plt.show()
```

3. CNN Implementation

```
class LeNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1=nn.Conv2d(1,20,5,1)
        self.conv2=nn.Conv2d(20,50,5,1)
        self.fully1=nn.Linear(4*4*50,500)
        self.fully2=nn.Linear(500,10)
    def forward(self,x):
        x=func.relu(self.conv1(x))
        x=func.max_pool2d(x,2,2)
        x=func.relu(self.conv2(x))
        x=func.max_pool2d(x,2,2)
        x=x.view(-1,4*4*50)
        x=func.relu(self.fully1(x))
        x=self.fully2(x)
        return x
```



4. Training of CNN

Step 1:

```
device=torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

transform1=transforms.Compose([transforms.Resize((28,28)),transforms.ToTensor(),transforms.Normalize((0.5,),(0.5,))])

training_dataset=datasets.MNIST(root='./data',train=True,download=True,transform=transform1)
validation_dataset=datasets.MNIST(root='./data',train=False,download=True,transform=transform1)

training_loader=torch.utils.data.DataLoader(dataset=training_dataset,batch_size=100,shuffle=True)
validation_loader=torch.utils.data.DataLoader(dataset=validation_dataset,batch_size=100,shuffle=False)
```

Step 2:

```
model=LeNet().to(device)
criteron=nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=0.00001)
```

Step 3:

```
epochs=12
loss_history=[]
correct_history=[]
val_loss_history=[]
val_correct_history=[]
for e in range(epochs):
    loss=0.0
    correct=0.0
    val_loss=0.0
    val_correct=0.0
    for input,labels in training_loader:
        input=input.to(device)     #device
        labels=labels.to(device)   #device
        outputs=model(input)
        loss1=criteron(outputs,labels)
        optimizer.zero_grad()
        loss1.backward()
        optimizer.step()
        _,preds=torch.max(outputs,1)
        loss+=loss1.item()
        correct+=torch.sum(preds==labels.data)
```

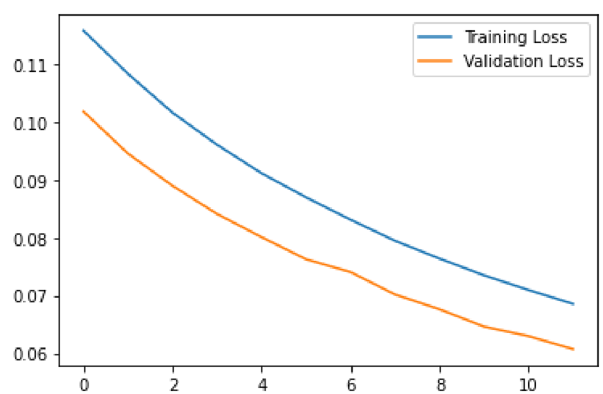
```
    else:
        with torch.no_grad():
            for val_input,val_labels in validation_loader:
                val_input=val_input.to(device)
                val_labels=val_labels.to(device)
                val_outputs=model(val_input)
                val_loss1=criteron(val_outputs,val_labels)
                _,val_preds=torch.max(val_outputs,1)
                val_loss+=val_loss1.item()
                val_correct+=torch.sum(val_preds==val_labels.data)
        epoch_loss=loss/len(training_loader)
        epoch_acc=correct.float()/len(training_loader)
        loss_history.append(epoch_loss)
        correct_history.append(epoch_acc)
        val_epoch_loss=val_loss/len(validation_loader)
        val_epoch_acc=val_correct.float()/len(validation_loader)
        val_loss_history.append(val_epoch_loss)
        val_correct_history.append(val_epoch_acc)
        print('training_loss:{:.4f},{:.4f}'.format(epoch_loss,epoch_acc.item()))
        print('validation_loss:{:.4f},{:.4f}'.format(val_epoch_loss,val_epoch_acc.item()))
```

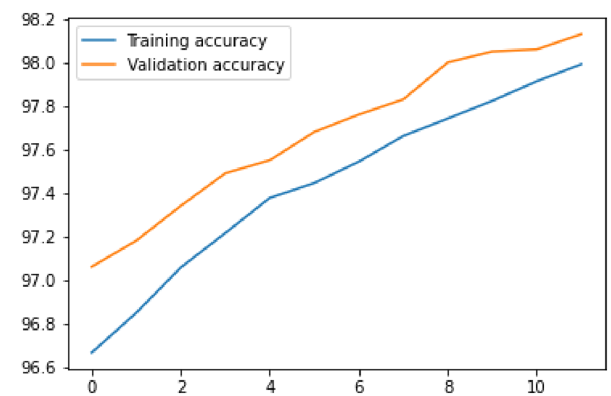
Result:

```
training_loss:1.8134,63.1500
validation_loss:1.0848,79.7000
training_loss:0.7461,83.5250
validation_loss:0.5079,86.8700
training_loss:0.4459,88.2317
validation_loss:0.3603,90.1300
training_loss:0.3410,90.6333
validation_loss:0.2901,91.9600
training_loss:0.2827,92.0683
validation_loss:0.2442,93.1200
training_loss:0.2431,93.0583
validation_loss:0.2127,93.8600
training_loss:0.2138,93.8633
validation_loss:0.1863,94.5200
training_loss:0.1906,94.5000
validation_loss:0.1690,94.9800
training_loss:0.1719,95.0100
validation_loss:0.1509,95.5300
training_loss:0.1562,95.4267
validation_loss:0.1376,95.9000
training_loss:0.1431,95.8233
validation_loss:0.1273,96.2200
training_loss:0.1323,96.1267
validation_loss:0.1164,96.4900
```

Step 4:

```
plt.plot(loss_history,label='Training Loss')
plt.plot(val_loss_history,label='Validation Loss')
plt.legend()
plt.show()
plt.plot(correct_history,label='Training accuracy')
plt.plot(val_correct_history,label='Validation accuracy')

plt.legend()
plt.show()
```





5. Validation of CNN

Step 1:

```python
import torch
import matplotlib.pyplot as plt
import numpy as np
import torch.nn.functional as func
import PIL.ImageOps
from torch import nn
from torchvision import datasets,transforms
```

Step 2:

```python
device=torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
transform1=transforms.Compose([transforms.Resize((28,28)),transforms.ToTensor(),transforms.Normalize((0.5,),(0.5,))])
training_dataset=datasets.MNIST(root='./data',train=True,download=True,transform=transform1)
validation_dataset=datasets.MNIST(root='./data',train=False,download=True,transform=transform1)
training_loader=torch.utils.data.DataLoader(dataset=training_dataset,batch_size=100,shuffle=True)
validation_loader=torch.utils.data.DataLoader(dataset=validation_dataset,batch_size=100,shuffle=False)
```

Step 3:

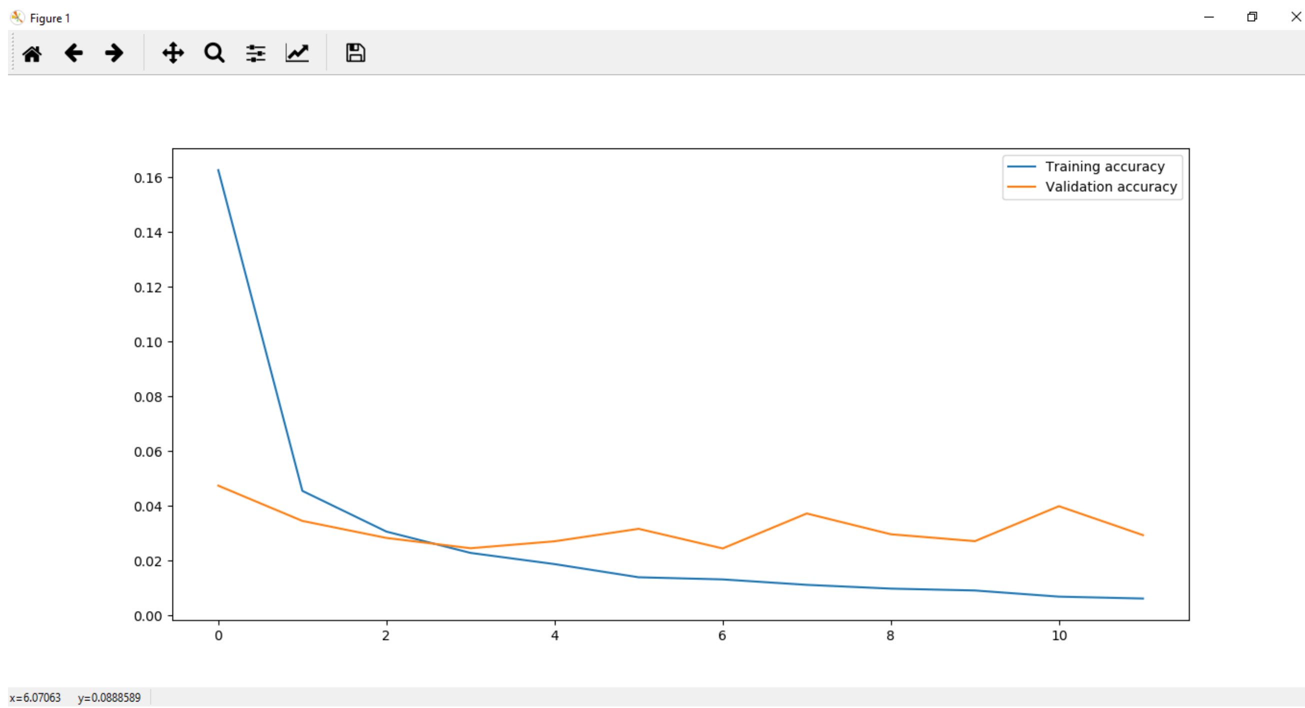
```
class LeNet(nn.Module):
        def __init__(self):
            super().__init__()
            self.conv1=nn.Conv2d(1,20,5,1)
            self.conv2=nn.Conv2d(20,50,5,1)
            self.fully1=nn.Linear(4*4*50,500)
            self.dropout1=nn.Dropout(0.5)#<======
            self.fully2=nn.Linear(500,10)
        def forward(self,x):
            x=func.relu(self.conv1(x))
            x=func.max_pool2d(x,2,2)
            x=func.relu(self.conv2(x))
            x=func.max_pool2d(x,2,2)
            x=x.view(-1,4*4*50)
            x=func.relu(self.fully1(x))
            x=self.dropout1(x)   #<========
            x=self.fully2(x)     #<========
            return x
```

Step 4:

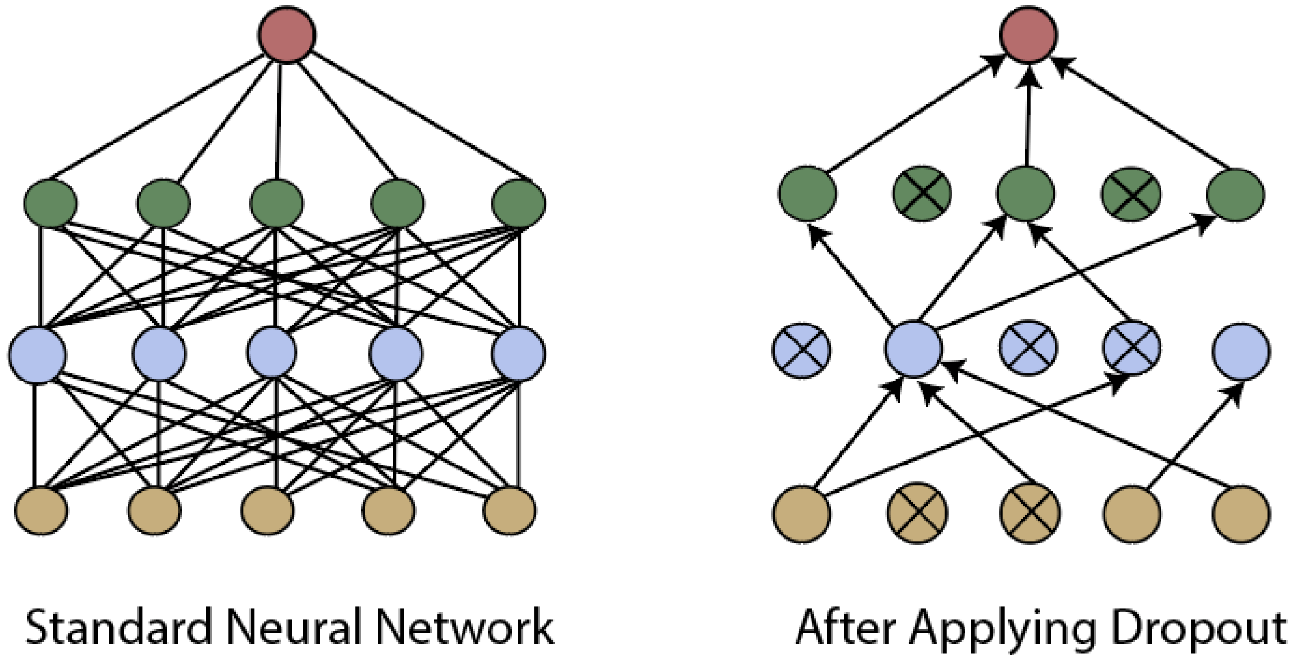
```
model=LeNet().to(device)
criteron=nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=0.00001)
```

Dropout Layer

Dropout Layer is a technique to reduce overfitting occurs



Dropout Layer



Dropout layers will be placed between the convolutional layers and between the fully connected layers. The dropout layer is used in between layers which have a high number of parameters because these high parameter layers are more likely to overfit and memorize the training data.

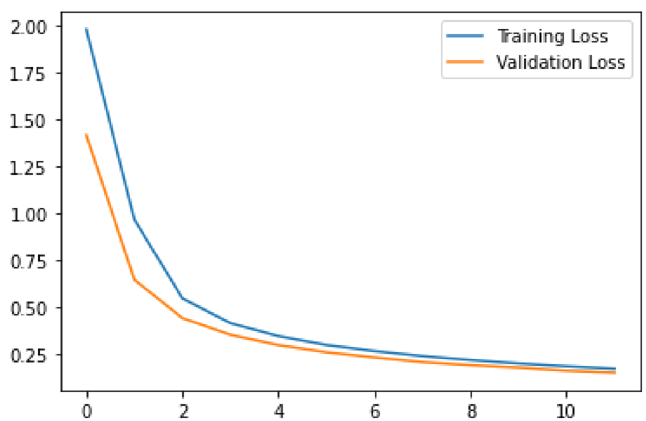
Step 5:

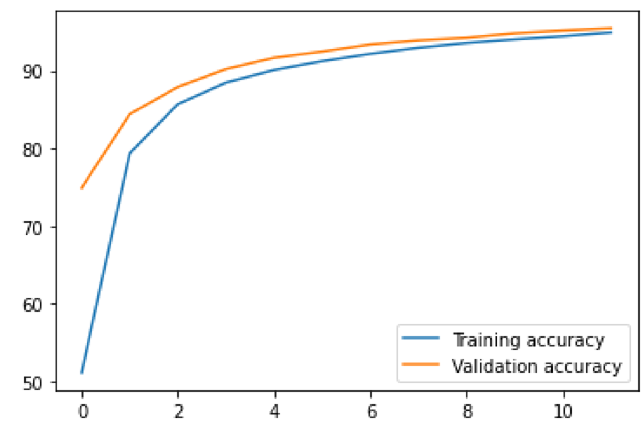
```
epochs=12
loss_history=[]
correct_history=[]
val_loss_history=[]
val_correct_history=[]
for e in range(epochs):
    loss=0.0
    correct=0.0
    val_loss=0.0
    val_correct=0.0
    for input,labels in training_loader:
        input=input.to(device)     #<====
        labels=labels.to(device)   #<====
        outputs=model(input)
        loss1=criteron(outputs,labels)
        optimizer.zero_grad()
        loss1.backward()
        optimizer.step()
        _,preds=torch.max(outputs,1)
        loss+=loss1.item()
        correct+=torch.sum(preds==labels.data)
```

```
    else:
        with torch.no_grad():
            for val_input,val_labels in validation_loader:

                val_input=val_input.to(device)
                val_labels=val_labels.to(device)
                val_outputs=model(val_input)
                val_loss1=criteron(val_outputs,val_labels)

                _,val_preds=torch.max(val_outputs,1)
                val_loss+=val_loss1.item()
                val_correct+=torch.sum(val_preds==val_label
s.data)
        epoch_loss=loss/len(training_loader)
        epoch_acc=correct.float()/len(training_loader)
        loss_history.append(epoch_loss)
        correct_history.append(epoch_acc)
        val_epoch_loss=val_loss/len(validation_loader)
        val_epoch_acc=val_correct.float()/len(validation_lo
ader)
        val_loss_history.append(val_epoch_loss)
        val_correct_history.append(val_epoch_acc)
        print('training_loss:{:.4f},{:.4f}'.format(epoch_lo
ss,epoch_acc.item()))
        print('validation_loss:{:.4f},{:.4f}'.format(val_ep
och_loss,val_epoch_acc.item()))
```

Step 6:

```
plt.plot(loss_history,label='Training Loss')
plt.plot(val_loss_history,label='Validation Loss')
plt.legend()
plt.show()
plt.plot(correct_history,label='Training accuracy')
plt.plot(val_correct_history,label='Validation accuracy')
plt.legend()
plt.show()
```





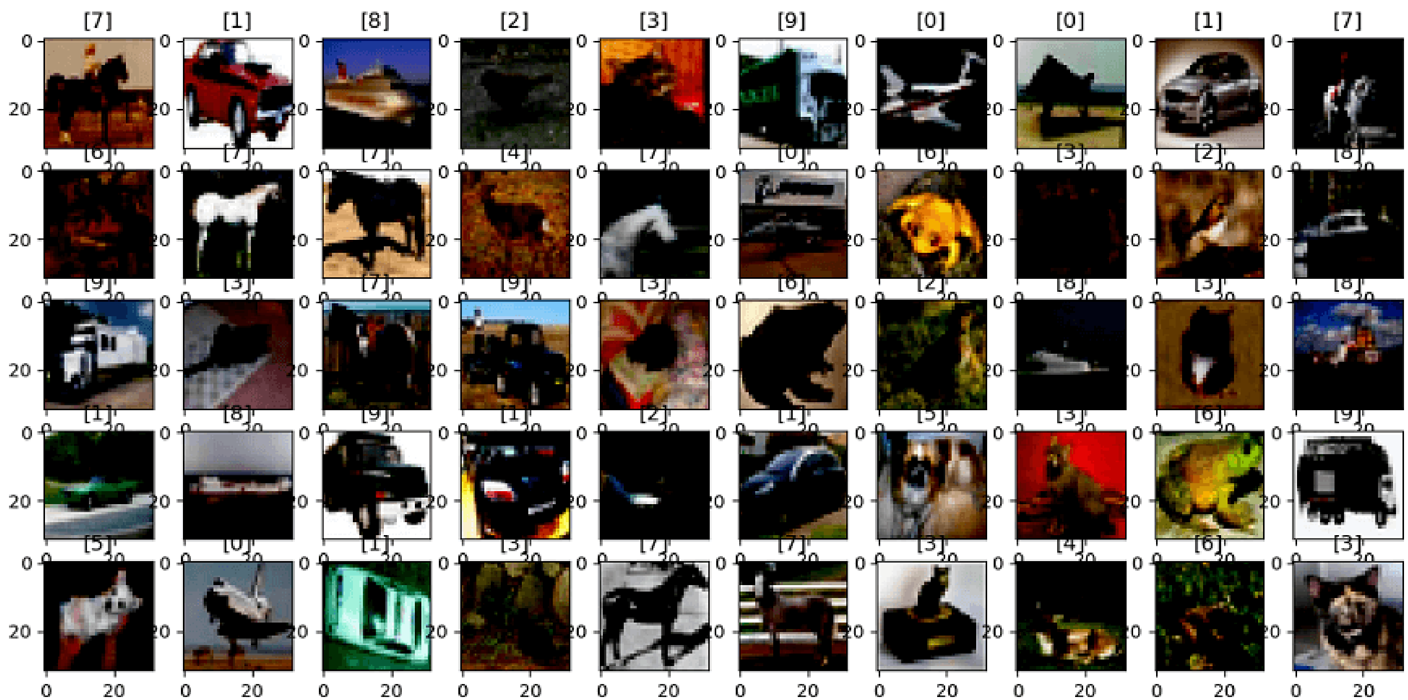
Part B

IMAGE CLASSIFICATION

CIFAR-10 Dataset

The **CIFAR 10(Canadian Institute for Advanced Research)** will be harder to classify and will come with new barriers which we will need to overcome. It is a collection of the image which is commonly used to train machine learning and computer vision algorithms.

The CIFAR-10 dataset consists of 60000 thirty by thirty color images in 10 classes means 6000 images per class. This dataset is divided into one test batch and five training batches. Every batch contains 10000 images. In the test batch, there are 1000 images which are randomly selected from each class.



CIFAR-100 Dataset

S. No	Superclass	Classes
1.	aquatic mammals	beaver, dolphin, otter, seal, whale
2.	flowers	orchids, poppies, roses, sunflowers, tulips
3.	fish	aquarium fish, flatfish, ray, shark, trout
4.	food containers	bottles, bowls, cans, cups, plates
5.	household electrical devices	clock, computer keyboard, lamp, telephone, television
6.	fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
7.	household furniture	bed, chair, couch, table, wardrobe
8.	large carnivores	bear, leopard, lion, tiger, wolf
9.	insects bee, beetle, butterfly, caterpillar, cockroach	
10.	large man-made outdoor things	bridge, castle, house, road, skyscraper
11.	large natural outdoor scenes	cloud, forest, mountain, plain, sea
12.	medium-sized mammals	fox, porcupine, possum, raccoon, skunk
13.	large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
14.	non-insect invertebrates	crab, lobster, snail, spider, worm
15.	reptiles	crocodile, dinosaur, lizard, snake, turtle
16.	people	baby, boy, girl, man, woman
17.	trees	maple, oak, palm, pine, willow
18.	small mammals	hamster, mouse, rabbit, shrew, squirrel
19.	vehicles 1	bicycle, bus, motorcycle, pickup truck, train
20.	vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

1. LeNet Model for CIFAR-10 Dataset

Step 1:

```
import torch
import matplotlib.pyplot as plt
import numpy as np
import torch.nn.functional as func
import PIL.ImageOps
from torch import nn
from torchvision import datasets,transforms
import requests
from PIL import Image
```

Step 2:

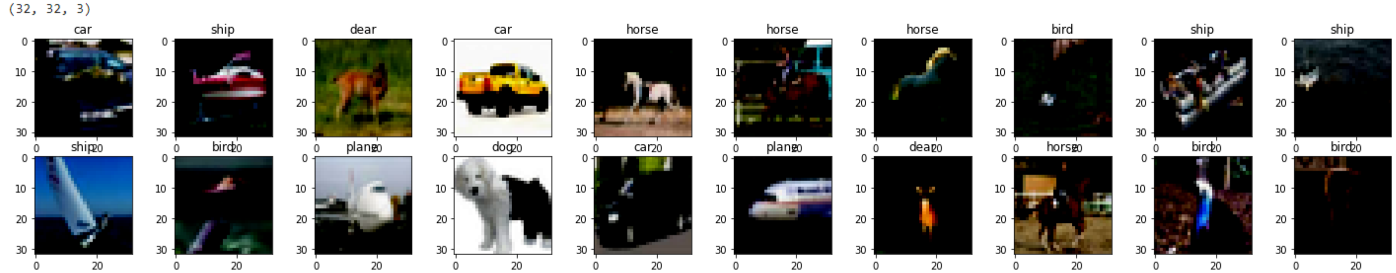
```
device=torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
transform1=transforms.Compose([transforms.Resize((32,32)),transforms.ToTensor(),transforms.Normalize((0.5,),(0.5,))])

training_dataset=datasets.CIFAR10(root='./data',train=True,download=True,transform=transform1)
validation_dataset=datasets.CIFAR10(root='./data',train=False,download=True,transform=transform1)

training_loader=torch.utils.data.DataLoader(dataset=training_dataset,batch_size=100,shuffle=True)
validation_loader=torch.utils.data.DataLoader(dataset=validation_dataset,batch_size=100,shuffle=False)
```

Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
170499072/? [00:03<00:00, 52671694.25it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified

Step 3:

```
def im_convert(tensor):
    image=tensor.cpu().clone().detach().numpy()
    image=image.transpose(1,2,0)
    print(image.shape)
    image=image*(np.array((0.5,0.5,0.5))+np.array((0.5,0.5,0.5)))
    image=image.clip(0,1)
    return image

classes=('plane','car','bird','cat','dear','dog','frog','horse','ship','truck')
dataiter=iter(training_loader)
images,labels=dataiter.next()
fig=plt.figure(figsize=(25,4))
for idx in np.arange(20):
    ax=fig.add_subplot(2,10,idx+1)
    plt.imshow(im_convert(images[idx]))
    ax.set_title(classes[labels[idx].item()])
```

```
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
```



Step 4:

```
class LeNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1=nn.Conv2d(3,20,5,1)
        self.conv2=nn.Conv2d(20,50,5,1)
        self.fully1=nn.Linear(5*5*50,500)
        self.dropout1=nn.Dropout(0.5)
        self.fully2=nn.Linear(500,10)
    def forward(self,x):
        x=func.relu(self.conv1(x))
        x=func.max_pool2d(x,2,2)
        x=func.relu(self.conv2(x))
        x=func.max_pool2d(x,2,2)
        x=x.view(-1,5*5*50)
        x=func.relu(self.fully1(x))
        x=self.dropout1(x)
        x=self.fully2(x)
        return x

model=LeNet().to(device)
criteron=nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=0.00001)
```

Step 5:

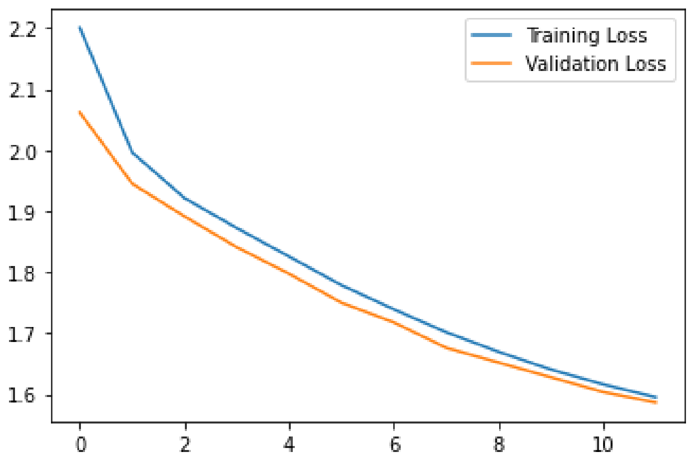
```
epochs=12
loss_history=[]
correct_history=[]
val_loss_history=[]
val_correct_history=[]
for e in range(epochs):
    loss=0.0
    correct=0.0
    val_loss=0.0
    val_correct=0.0
    for input,labels in training_loader:
        input=input.to(device)
        labels=labels.to(device)
        outputs=model(input)
        loss1=criteron(outputs,labels)
        optimizer.zero_grad()
        loss1.backward()
        optimizer.step()
        _,preds=torch.max(outputs,1)
        loss+=loss1.item()
        correct+=torch.sum(preds==labels.data)
```

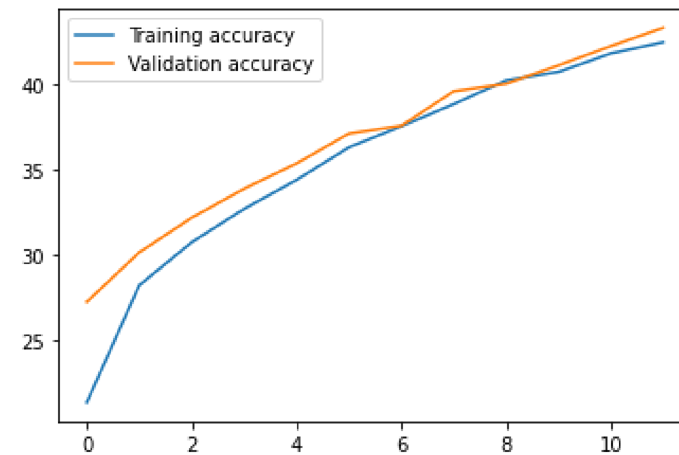
```
else:
        with torch.no_grad():
            for val_input,val_labels in validation_loader:
                val_input=val_input.to(device)
                val_labels=val_labels.to(device)
                val_outputs=model(val_input)
                val_loss1=criteron(val_outputs,val_labels)
                _,val_preds=torch.max(val_outputs,1)
                val_loss+=val_loss1.item()
                val_correct+=torch.sum(val_preds==val_labels.data)
        epoch_loss=loss/len(training_loader)
        epoch_acc=correct.float()/len(training_loader)
        loss_history.append(epoch_loss)
        correct_history.append(epoch_acc)
        val_epoch_loss=val_loss/len(validation_loader)
        val_epoch_acc=val_correct.float()/len(validation_loader)
        val_loss_history.append(val_epoch_loss)
        val_correct_history.append(val_epoch_acc)
        print('training_loss:{:.4f},{:.4f}'.format(epoch_loss,epoch_acc.item()))
        print('validation_loss:{:.4f},{:.4f}'.format(val_epoch_loss,val_epoch_acc.item()))
```

```
training_loss:2.2006,21.3820
validation_loss:2.0616,27.2700
training_loss:1.9963,28.2420
validation_loss:1.9451,30.1700
training_loss:1.9216,30.7620
validation_loss:1.8916,32.2000
training_loss:1.8724,32.7180
validation_loss:1.8410,33.9000
training_loss:1.8256,34.4120
validation_loss:1.7974,35.3800
training_loss:1.7788,36.3240
validation_loss:1.7500,37.1300
training_loss:1.7393,37.5580
validation_loss:1.7179,37.5900
training_loss:1.7018,38.8580
validation_loss:1.6762,39.6100
training_loss:1.6696,40.2460
validation_loss:1.6521,40.0500
training_loss:1.6407,40.7400
validation_loss:1.6280,41.1300
training_loss:1.6163,41.8280
validation_loss:1.6038,42.2600
training_loss:1.5954,42.4840
validation_loss:1.5867,43.3300
```

Step 6:

```
plt.plot(loss_history,label='Training Loss')
plt.plot(val_loss_history,label='Validation Loss')
plt.legend()
plt.show()
plt.plot(correct_history,label='Training accuracy')
plt.plot(val_correct_history,label='Validation accuracy')

plt.legend()

plt.show()
```

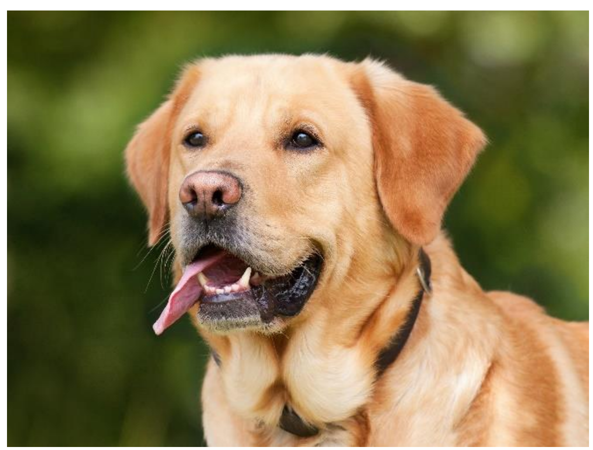


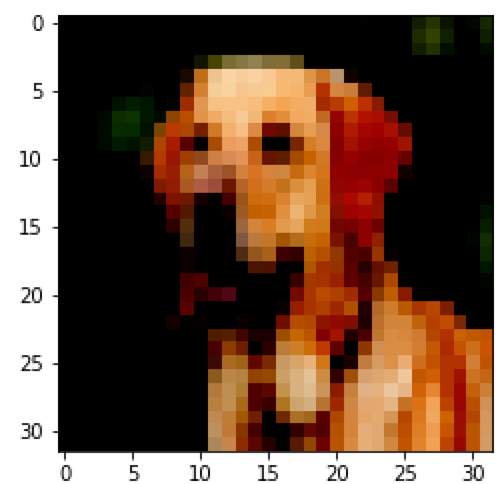


Step 7:

```
url='https://3c1703fe8d.site.internapcdn.net/newman/gfx/news/hires/2018/2-dog.jpg'
response=requests.get(url,stream=True)
img=Image.open(response.raw)
img=transform1(img)
image1=img.to(device).unsqueeze(0)
output=model(image1)
_,pred=torch.max(output,1)
print(classes[pred.item()])
```

```
dog
```

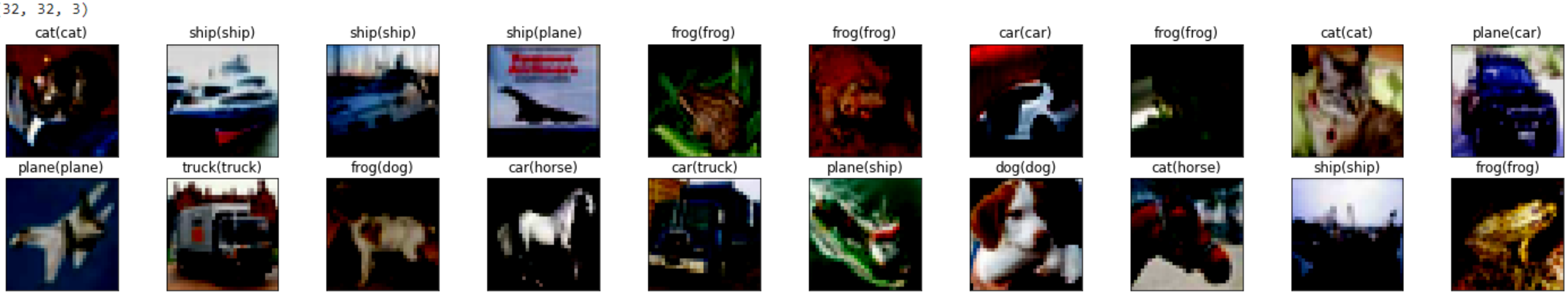




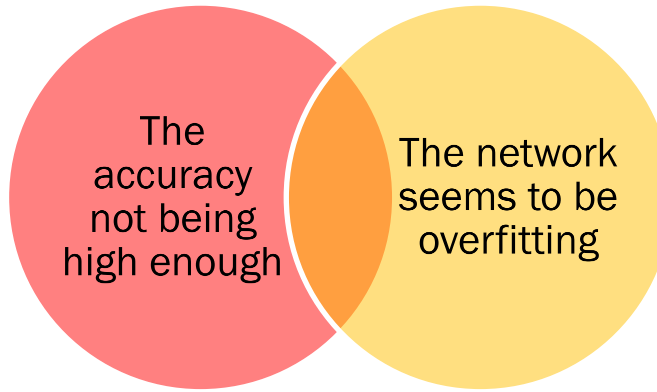
Step 8:

```
dataiter=iter(validation_loader)
images,labels=dataiter.next()
images_=images.to(device)
labels=labels.to(device)
output=model(images_)
_,preds=torch.max(output,1)
fig=plt.figure(figsize=(25,4))
for idx in np.arange(20):
    ax=fig.add_subplot(2,10,idx+1,xticks=[],yticks=[])
    plt.imshow(im_convert(images[idx]))
    ax.set_title("{}({})".format(str(classes[preds[idx].item()]),str(classes[labels[idx].item()]),color=("green" if preds[idx]==labels[idx] else "red")))
plt.show()
```

```
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
(32, 32, 3)
```

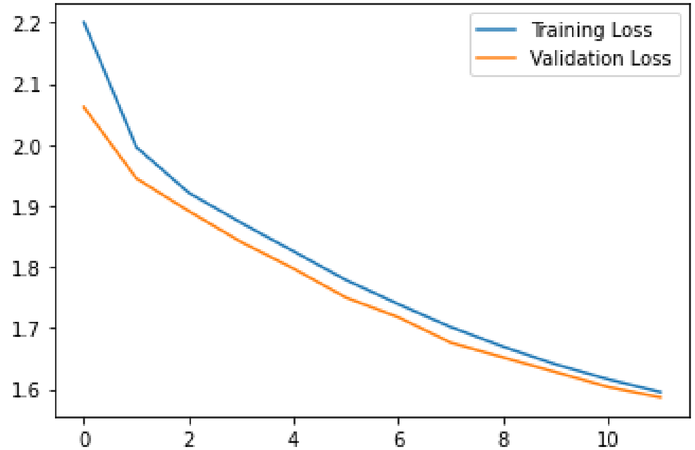


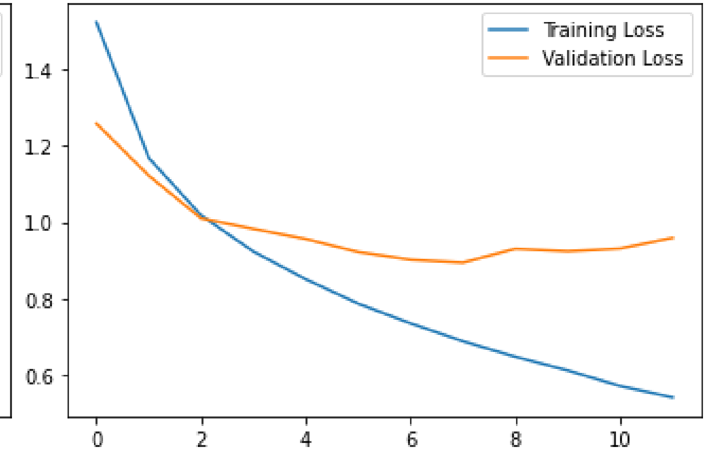
3. Hyperparameter Tuning Technique

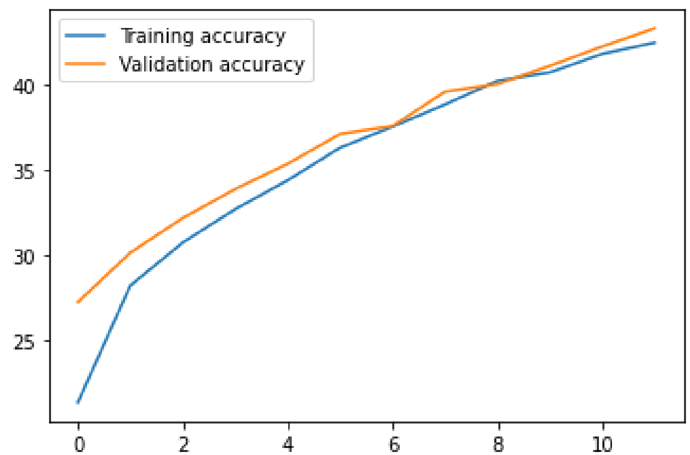


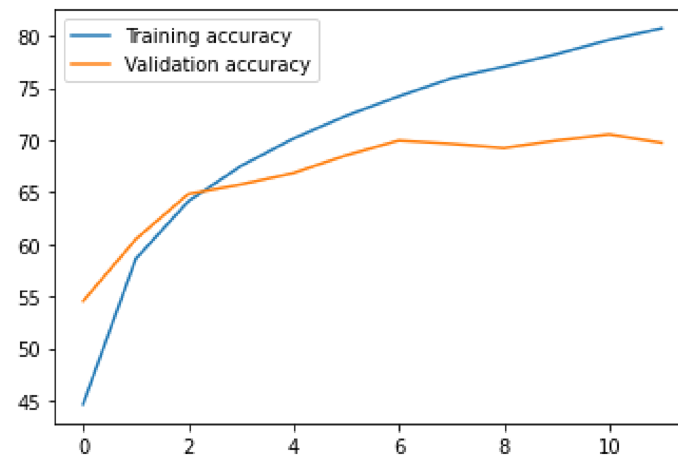
The fine-tuning of the model is important and can improve the model performance significantly

1. The first modification will be focused on the learning rate.
2. The second modification is very effective. We will simply add more convolution of layers.
3. A larger kernel implies more parameters. We will use a smaller kernel size to remove overfitting.







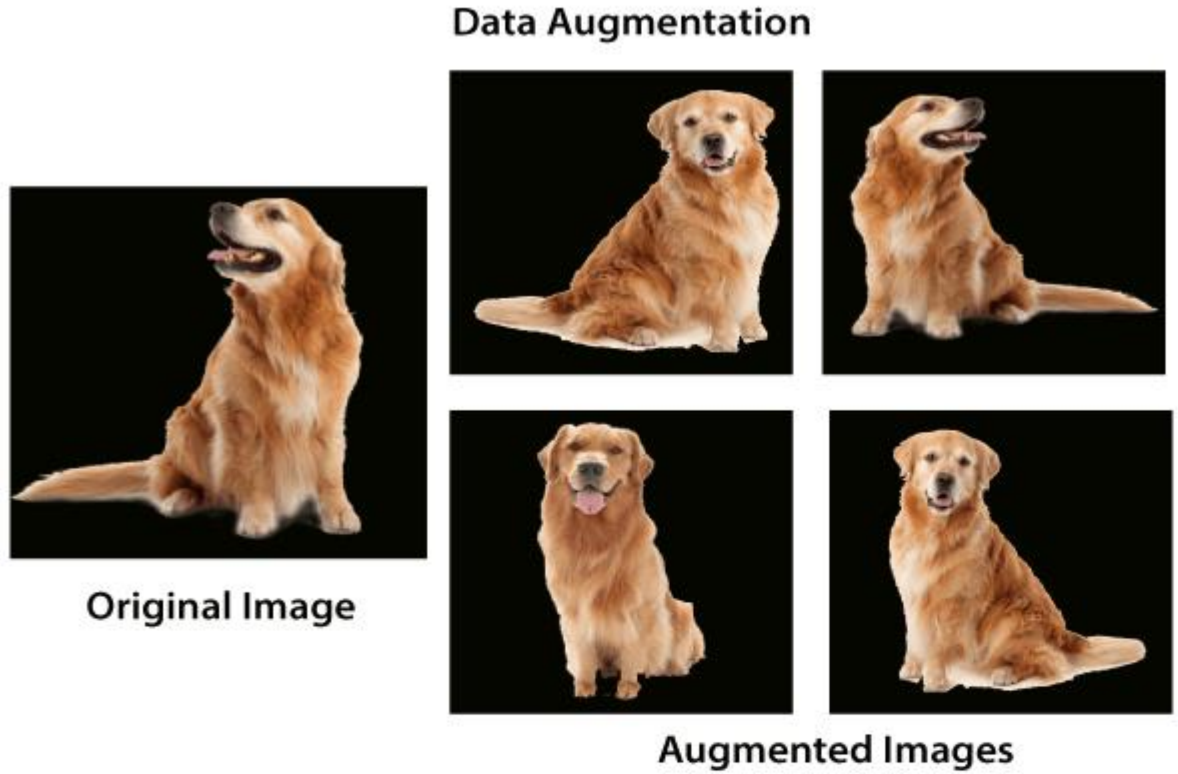


Old model
Lr =0.00001

New model
Lr =0.001

```
class LeNet(nn.Module):  #Modify
        def __init__(self):
            super().__init__()
            #self.conv1=nn.Conv2d(3,20,5,1)
            #self.conv2=nn.Conv2d(20,50,5,1)
            self.conv1=nn.Conv2d(3,16,5,1)
            self.conv2=nn.Conv2d(16,32,5,1)
            self.conv3=nn.Conv2d(32,64,5,1)
            #self.fully1=nn.Linear(5*5*50,500)
            self.fully1=nn.Linear(5*5*64,500)
            self.dropout1=nn.Dropout(0.5)
            self.fully2=nn.Linear(500,10)
        def forward(self,x):
            x=func.relu(self.conv1(x))
            x=func.max_pool2d(x,2,2)
            x=func.relu(self.conv2(x))
            x=func.max_pool2d(x,2,2)
            #x=x.view(-1,5*5*50)
            x=x.view(-1,5*5*64)
            x=func.relu(self.fully1(x))
            x=self.dropout1(x)
            x=self.fully2(x)
            return x

model=LeNet().to(device)
criteron=nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=0.001)
```

4. Data Augmentation Process



The data augmentation technique is useful because it allows our model to look at each image in our dataset from a variety of different perspective. After applying the transformation, the newly created images are known as augmented images because they essentially allow us to augment our dataset by adding new data to it.

Data Augmentation Implementation

Step 1:

```
import torch
import matplotlib.pyplot as plt
import numpy as np
import torch.nn.functional as func
import PIL.ImageOps
from torch import nn
from torchvision import datasets,transforms
import requests
from PIL import Image
```

Step 2:

```
device=torch.device("cuda:0" if torch.cuda.is_availab
le() else "cpu")
transform_train=transforms.Compose(
[transforms.Resize((32,32)),
 transforms.RandomHorizontalFlip(),
 transforms.RandomRotation(10),
 transforms.RandomAffine(0,shear=10,scale=(0.8,1.2)),
 transforms.ColorJitter(brightness=0.2,contrast=0.2,s
aturation=0.2),
 transforms.ToTensor(),
 transforms.Normalize((0.5,),(0.5,))])

transform1=transforms.Compose([transforms.Resize((32,
32)),transforms.ToTensor(),transforms.Normalize((0.5,
),(0.5,))])

training_dataset=datasets.CIFAR10(root='./data',train
=True,download=True,transform=transform_train)
validation_dataset=datasets.CIFAR10(root='./data',tra
in=False,download=True,transform=transform1)

training_loader=torch.utils.data.DataLoader(dataset=
training_dataset,batch_size=100,shuffle=True)
validation_loader=torch.utils.data.DataLoader(dataset
= validation_dataset,batch_size=100,shuffle=False)
```

Step 3:

```
def im_convert(tensor):
    image=tensor.cpu().clone().detach().numpy()
    image=image.transpose(1,2,0)
    print(image.shape)
    image=image*(np.array((0.5,0.5,0.5))+np.array((0.5,0.5,0.5)))
    image=image.clip(0,1)
    return image

classes=('plane','car','bird','cat','dear','dog','frog','horse','ship','truck’)

dataiter=iter(training_loader)
images,labels=dataiter.next()
fig=plt.figure(figsize=(25,4))
for idx in np.arange(20):
    ax=fig.add_subplot(2,10,idx+1)
    plt.imshow(im_convert(images[idx]))
    ax.set_title(classes[labels[idx].item()])
```

Step 4:

```
class LeNet(nn.Module):
        def __init__(self):
            super().__init__()
            self.conv1=nn.Conv2d(3,16,3,1, padding=1)
            self.conv2=nn.Conv2d(16,32,3,1, padding=1)
            self.conv3=nn.Conv2d(32,64,3,1, padding=1)
            self.fully1=nn.Linear(4*4*64,500)
            self.dropout1=nn.Dropout(0.5)
            self.fully2=nn.Linear(500,10)

        def forward(self,x):
            x=func.relu(self.conv1(x))
            x=func.max_pool2d(x,2,2)
            x=func.relu(self.conv2(x))
            x=func.max_pool2d(x,2,2)
            x=func.relu(self.conv3(x))
            x=func.max_pool2d(x,2,2)
            x=x.view(-1,4*4*64)
            x=func.relu(self.fully1(x))
            x=self.dropout1(x)
            x=self.fully2(x)
            return x

model=LeNet().to(device)
criteron=nn.CrossEntropyLoss()
optimizer=torch.optim.Adam(model.parameters(),lr=0.001)
```

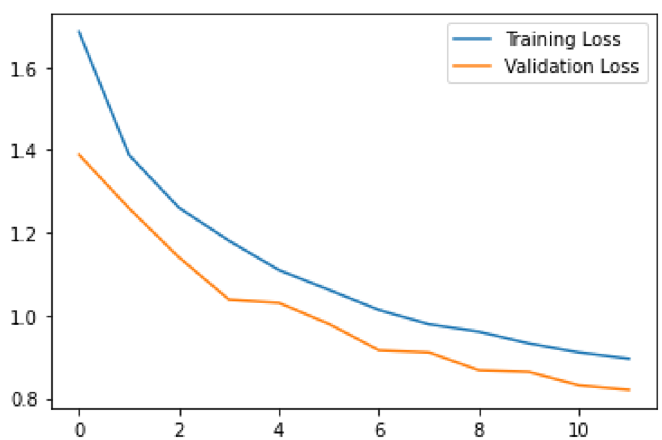
Step 5:

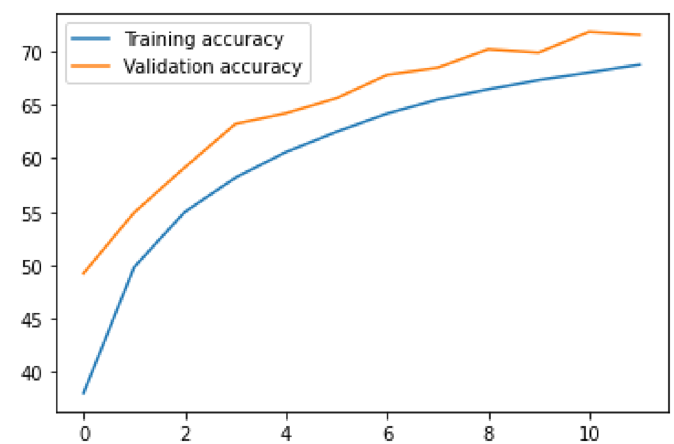
```
epochs=12
loss_history=[]
correct_history=[]
val_loss_history=[]
val_correct_history=[]
for e in range(epochs):
    loss=0.0
    correct=0.0
    val_loss=0.0
    val_correct=0.0
    for input,labels in training_loader:
        input=input.to(device)
        labels=labels.to(device)
        outputs=model(input)
        loss1=criteron(outputs,labels)
        optimizer.zero_grad()
        loss1.backward()
        optimizer.step()
        _,preds=torch.max(outputs,1)
        loss+=loss1.item()
        correct+=torch.sum(preds==labels.data)
```

```
    else:
        with torch.no_grad():
            for val_input,val_labels in validation_loader:

                val_input=val_input.to(device)
                val_labels=val_labels.to(device)
                val_outputs=model(val_input)
                val_loss1=criteron(val_outputs,val_labels)

                _,val_preds=torch.max(val_outputs,1)
                val_loss+=val_loss1.item()
                val_correct+=torch.sum(val_preds==val_label
s.data)
        epoch_loss=loss/len(training_loader)
        epoch_acc=correct.float()/len(training_loader)
        loss_history.append(epoch_loss)
        correct_history.append(epoch_acc)
        val_epoch_loss=val_loss/len(validation_loader)
        val_epoch_acc=val_correct.float()/len(validation_lo
ader)
        val_loss_history.append(val_epoch_loss)
        val_correct_history.append(val_epoch_acc)
        print('training_loss:{:.4f},{:.4f}'.format(epoch_lo
ss,epoch_acc.item()))
        print('validation_loss:{:.4f},{:.4f}'.format(val_ep
och_loss,val_epoch_acc.item()))
```

Step 6:

```
plt.plot(loss_history,label='Training Loss')
plt.plot(val_loss_history,label='Validation Loss')
plt.legend()
plt.show()
plt.plot(correct_history,label='Training accuracy')
plt.plot(val_correct_history,label='Validation accuracy')

plt.legend()
plt.show()
```





Step 7:

```
url='https://akm-img-a-
in.tosshub.com/indiatoday/images/story/201810/white_stork
.jpeg?B2LINO47jclcIb3QCW.Bj9nto934Lox4'
response=requests.get(url,stream=True)
img=Image.open(response.raw)
img=transform1(img)
image1=img.to(device).unsqueeze(0)
output=model(image1)
_,pred=torch.max(output,1)
print(classes[pred.item()])
```

Step 8:

```
dataiter=iter(validation_loader)
images,labels=dataiter.next()
images_=images.to(device)
labels=labels.to(device)
output=model(images_)
_,preds=torch.max(output,1)
fig=plt.figure(figsize=(25,4))
for idx in np.arange(20):
      ax=fig.add_subplot(2,10,idx+1,xticks=[],yticks=[])
      plt.imshow(im_convert(images[idx]))

ax.set_title("{}({})".format(str(classes[preds[idx].item()]),str(classes[labels[idx].item()]),color=("green" if classes[preds[idx]]==classes[labels[idx]] else "red")))
plt.show()
```

Any Questions?