AMSI VACATIONRESEARCH SCHOLARSHIPS 2022–23

SET YOUR SIGHTS ON RESEARCH THIS SUMMER

# Image Compression using Convolutional Networks

Sarah Kawaguchi

Supervised by Laurence Park and Vishal Patel

Western Sydney University

Vacation Research Scholarships are funded jointly by the Department of Education

AMSI AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE

## Contents

# 1 Prelude

## 1.1 Acknowledgement

I would like to thank Park Laurence for his guidance throughout this project and for his advice in matters both regarding the project and my career. I would also like to thank Vishal Patel for all his thoughtful insight and his support.
Finally, I would like to thank my family and friends for their love and support during this time.

## 1.2 Abstract

Digital images generally contain significant amounts of redundancy. Image compression is used to reduce this redundancy. The main advantages of image compression include reduction in storage and transmission time.
This paper discuses Convolutional Neural Networks, a technique used for image compression, and its parameters and how they affect the size and quality of the compressed images.

## 1.3 Statement of Authorship

The code for model2 was developed by Takya Mamoto from Osaka University. All the other code and work was developed by Sarah Kawaguchi.

## 2 Data Set

For this project, I focused on one dataset only, the CIFAR-10 dataset. This data set consists of 60000 32x21 color images in 10 classes, with 6000 images per class. The classes include: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. All these classes are completely mutually exclusive, meaning there is no overlap between any of them.

## 3 Introduction

Image compression is a type of data compression which is used to save the cost of storing and transferring images.
This paper discusses compressing an image using a convolutional neural network autoencoder, which involves images going into an encoder which compresses them and turns their pixels into a series of numbers which is called the latent representation. These numbers are then put into a decoder to return the compressed version of the images.

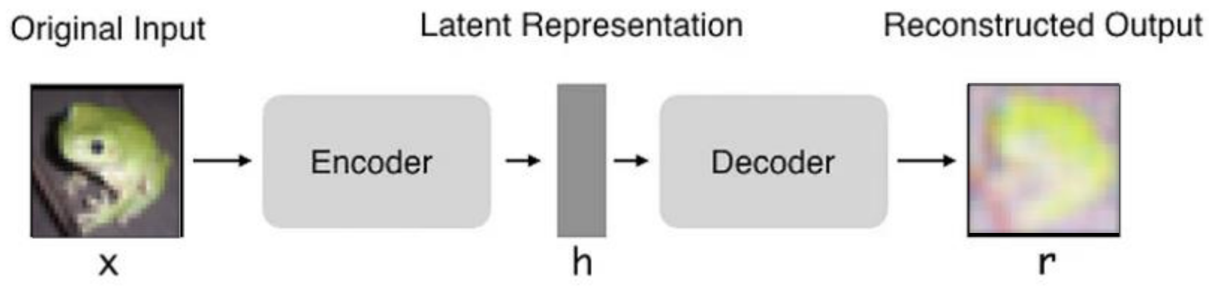


*Figure 1 Architecture of an Autoencoder [1]*

In this paper, we look at various convolutional neural network models and their parameters, and how changing certain parameters such as model complexity and epochs can affect the quality of the compressed images.

# 4 Convolutional Neural Networks

A neural network is an algorithm created to simulate biological neural networks. It is comprised of an input layer, one or more hidden layers and an output layer, where each layer is made up of interconnected neurons.

Convolutuonal neural networks (CNN) are a type of neural networks which uses convolutional layers. CNN is most commonly used for image recognition and tasks that involve the processing of pixel data such as image classification. This is because the convolutional layers reduce the high dimensionality of images without losing information.

Convolutional neural netowrks consist of three main types of layers:

- Convolutional layer
- Pooling layer
- Fully connected layer

# 5 Autoencoders

An autoencoder is a structure used to compress and encode data and then reconstruct the data from the reduced encoded representation to a representation that is as close to the original input as possible.

An autoencoder consists of three main parts:

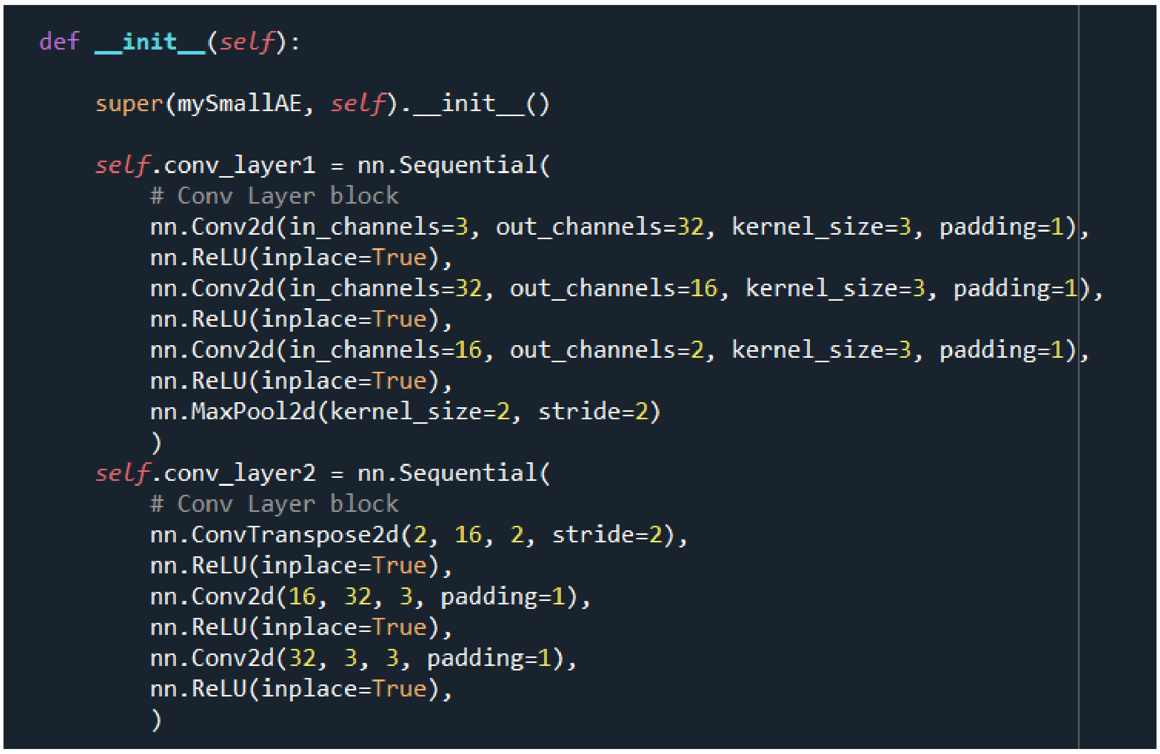
- Encoder: takes the images as input and compresses them into an encoded representation.
- Bottleneck: contains the compressed representation of the images.
- Decoder: reconstructs the encoded representation to as close to the original as possible.

Another big part of an encoder is reconstruction loss, which is the measure of how well the autoencoder is performig and how much information was lost during the process.

# 6 CNN Autoencoder

## 6.1 Model1

This is the first convolutional neural network autoencoder model that was used to compress the images in CIFAR-10. It contains three convolutional layers and one pooling layer.



```
def __init__(self):

    super(mySmallAE, self).__init__()

    self.conv_layer1 = nn.Sequential(
        # Conv Layer block
        nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(in_channels=32, out_channels=16, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(in_channels=16, out_channels=2, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.MaxPool2d(kernel_size=2, stride=2)
        )
    self.conv_layer2 = nn.Sequential(
        # Conv Layer block
        nn.ConvTranspose2d(2, 16, 2, stride=2),
        nn.ReLU(inplace=True),
        nn.Conv2d(16, 32, 3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(32, 3, 3, padding=1),
        nn.ReLU(inplace=True),
        )
```

*Figure 2 Convolutional Neural Network Autoencoder model1*

## 6.2 Epochs

An epoch means training the neural network with all the training data for one cycle [2]. In a single epoch, all of data is used exactly once. Increasing the number of epochs can boost precision, but if the number of epochs is too high it can lead to overfitting the model.

I first ran model1 for 10 epochs and these were the results I acheievd.

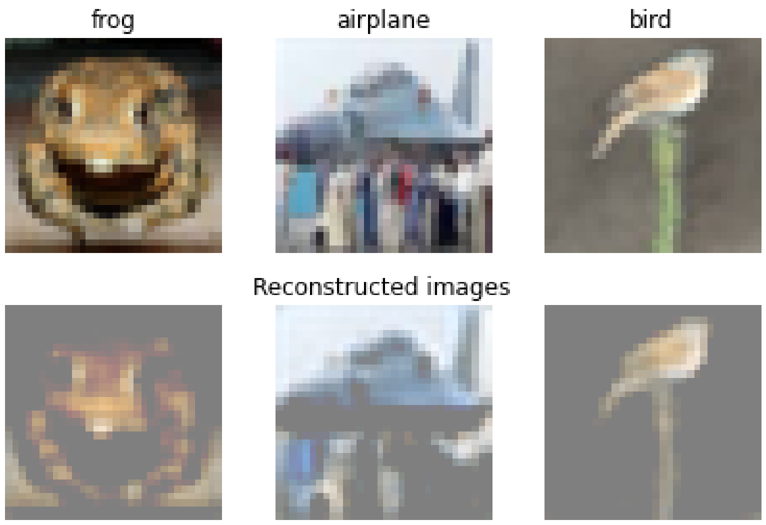


*Figure 3 Three random images from CIFAR-10 before and after compression using model1*

From figure 3 we can see that while the quality of the images after compression is not great, the quality before compression is not good either. Some quality was lost after compression but the objects retained their shapes even after compression.

I ran this model again for 30 epochs and these were the results.

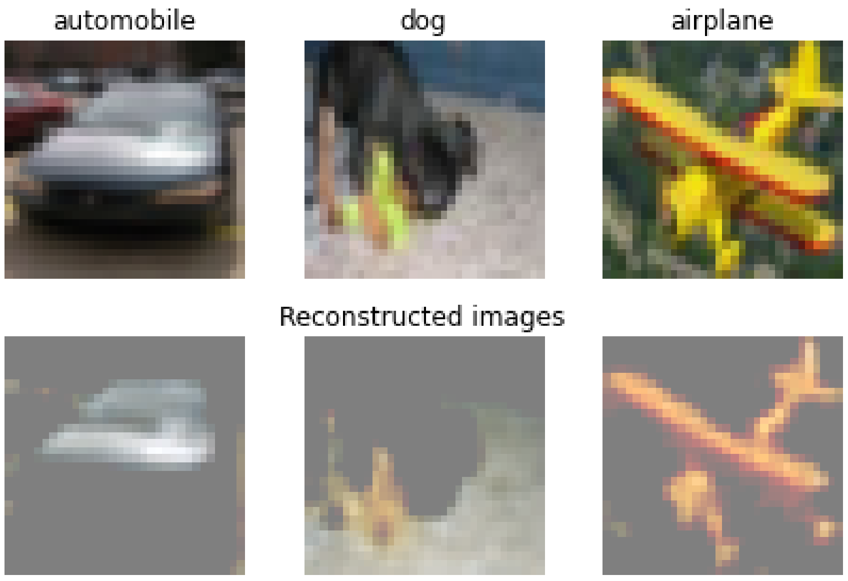


*Figure 4 Three random images from CIFAR-10 before and after compression using model1*

Figure 4 shows the training and validation loss after 10 epochs. There doesn't seem to be a notable difference in quality between figure 3 and figure 5, but the difference can be noted in the training and validation loss.

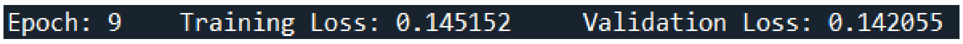
## 6.3 Reconstruction Loss

'Loss' is a concept used to assess the performance of a model. In this case, we have two different types which are validation loss and training loss.

Training loss is used to assess how a model fits the training data. Note that the training data is a portion of the dataset used to initially train the model [2].

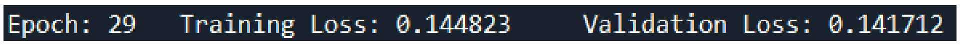
Validation loss is used to assess the performance of a model on the validation set. The validation set is a portion of the dataset set used to validate the performance of the model [2].

Having various validation and training losses often helps with assessing the model accuracy and performance. In a good model, the training loss and validation loss should both decrease as the number of epochs increases and should eventually stabilize at a specific point.



```
Epoch: 9     Training Loss: 0.145152      Validation Loss: 0.142055
```

*Figure 5 Training and validation loss after 10 epochs for model1*



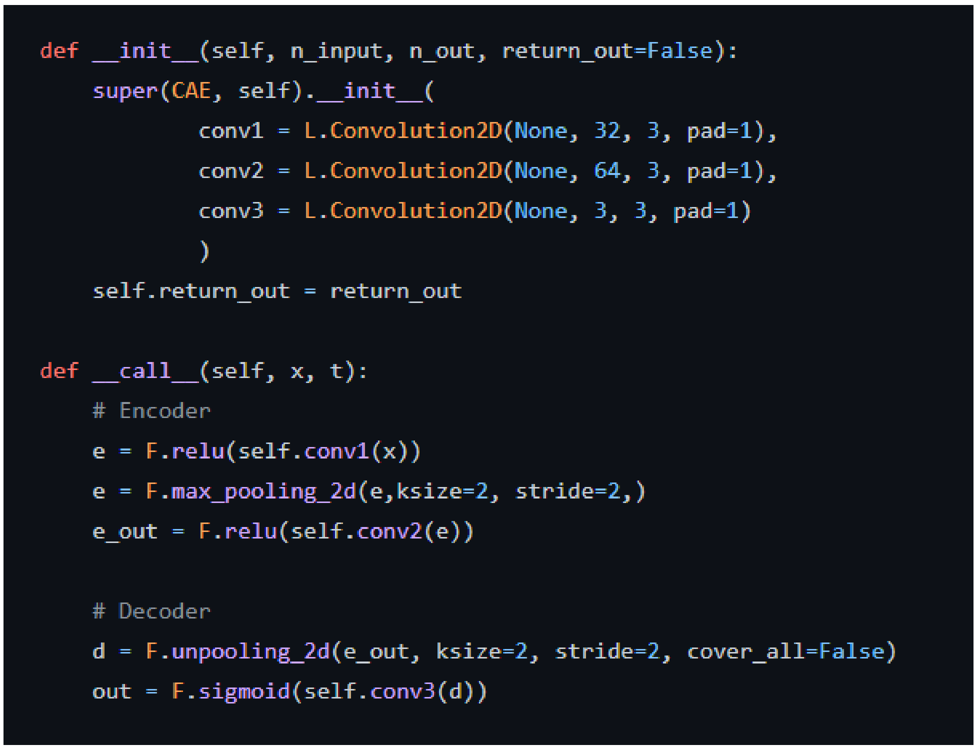
```
Epoch: 29    Training Loss: 0.144823      Validation Loss: 0.141712
```

*Figure 6 Training and validation loss after 30 epochs for model1*

Figures 5 and 6 show that the training loss has decreased from 0.145 to 0.144 when the number of epochs has been increased by 20. The validation loss has also decreased  by 0.001 over the course of 20 epochs. This indicates that this model is neither overfitted nor underfitted.

## 6.4 Model2

This is another model that was created to compress the images in CIFAR-10. Model2 contains two convolutional layers and one pooling layer.



```
def __init__(self, n_input, n_out, return_out=False):
    super(CAE, self).__init__(
            conv1 = L.Convolution2D(None, 32, 3, pad=1),
            conv2 = L.Convolution2D(None, 64, 3, pad=1),
            conv3 = L.Convolution2D(None, 3, 3, pad=1)
            )
    self.return_out = return_out

def __call__(self, x, t):
    # Encoder
    e = F.relu(self.conv1(x))
    e = F.max_pooling_2d(e,ksize=2, stride=2,)
    e_out = F.relu(self.conv2(e))

    # Decoder
    d = F.unpooling_2d(e_out, ksize=2, stride=2, cover_all=False)
    out = F.sigmoid(self.conv3(d))
```

*Figure 7 Convolutional Neural Network Autoencoder model2*

Figure 7 shows all the different layers in model2

This model ran for 20 epochs and the images before and after compression are shown for every class in the CIFAR-10 dataset.

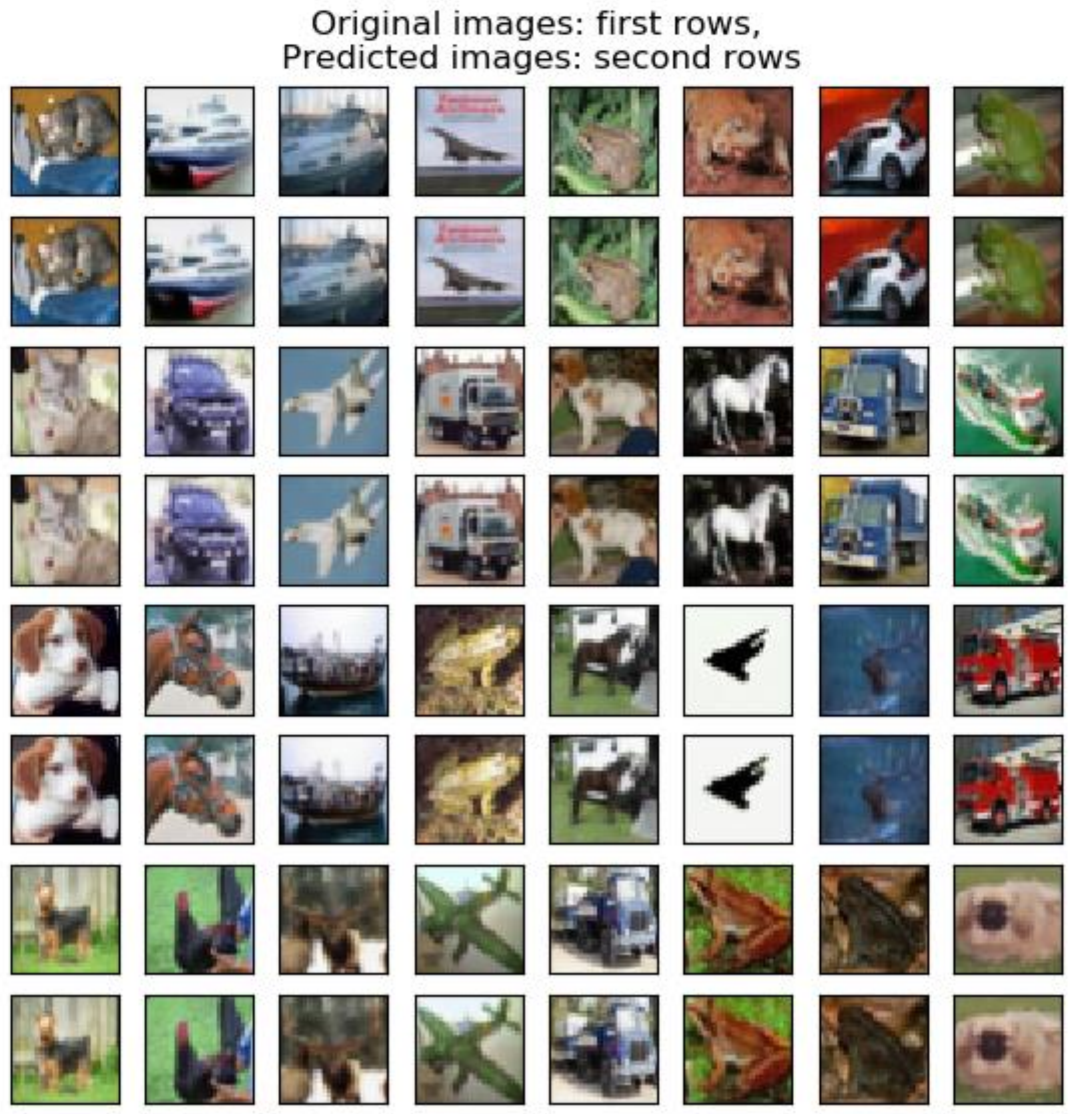


*Figure 8 Images before and after being compressed using model2. The first row shows the before images and the second row shows the after images.*

From figure 8, we can observe that the quality after compression is great in comparison to before compression. The images look very similar both before and after.

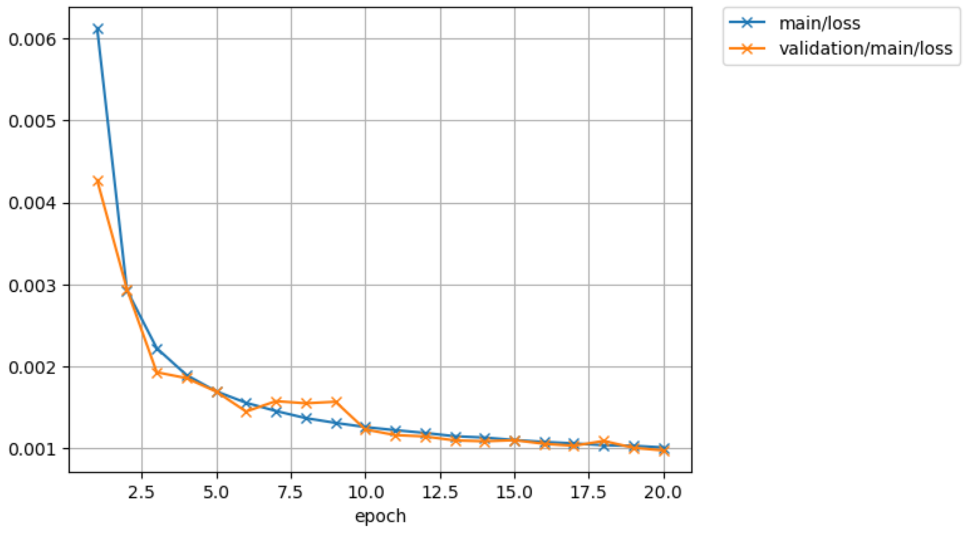


*Figure 9 Reconstruction loss plot over 20 epochs*

In figure 9, it is apparent that both the training and validation losses are reducing as the number of epochs increases and both the validation and training loss seem to be close to 0.0012 after only 10 epochs. This indicates that this model is performing well.

By comparing model1 and model2, we notice that model2 is less complex than model1 as it has less layers. But model2 is performing better than model1 in terms of quality of reocnstructed images. Model2 has a validation loss of approximately 0.0012 while the validation loss for model1 after the same number of epochs is 0.1421 which is significanty larger. All of this points towards model2 being better than model1 for this specific dataset in terms of compressing images.

# 7 Conclusion

In this paper, we focused on creating a convolutional neural network autoencoder to compress images from the CIFAR-10 dataset. We also worked on studying the different parameters that affect a convolutional neural network autoencoder model. The results of this work were fascinating as we discovered that complexity does not always equal better performance and every convolutional neural network autoencoder has to be adjusted for the dataset that is going to be used, otherwise the model would not be as accurate and performance would drop.

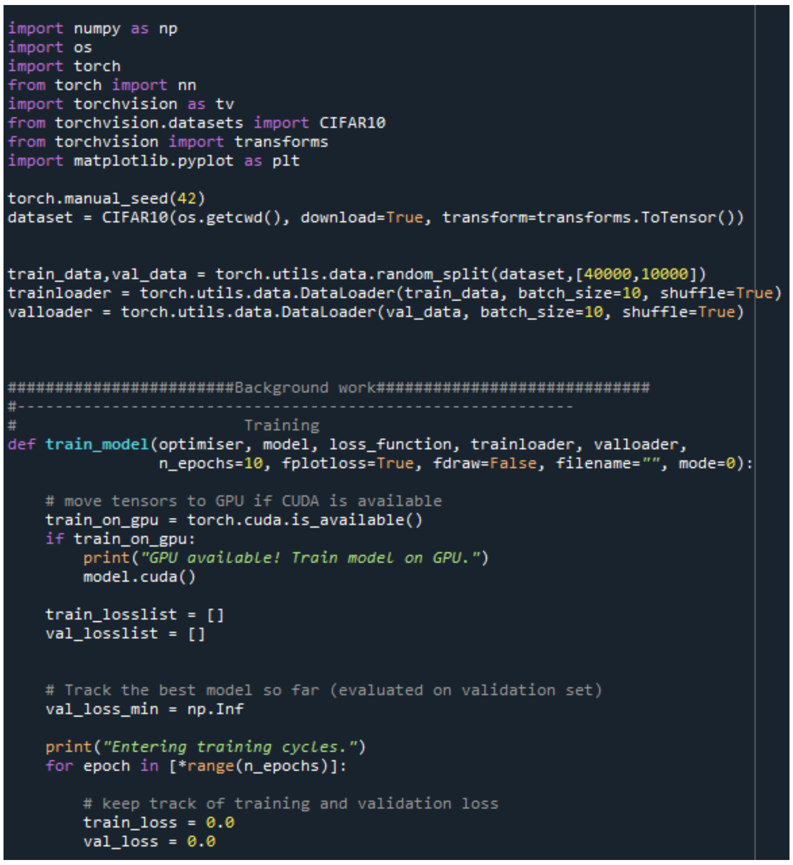
# 8 References

1. Hubens, N. (2018) *Deep inside: Autoencoders*, *Medium*. Towards Data Science. Available at: https://towardsdatascience.com/deep-inside-autoencoders. (Accessed: February 20, 2023).
2. baeldung (2022) *Epoch in neural networks*, *Baeldung on Computer Science*. Available at: https://www.baeldung.com/cs/epoch-neural-networks (Accessed: February 20, 2023).

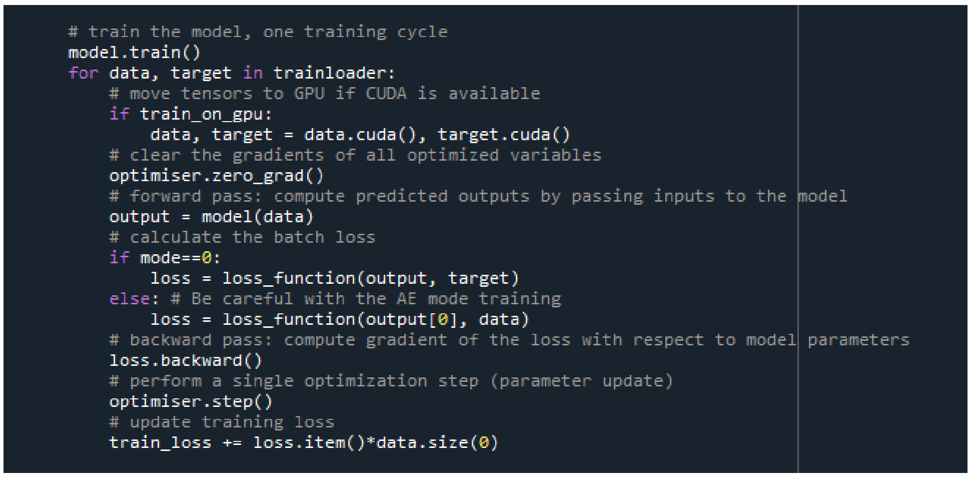
# 9 Appendix

We have the source code for both models in the appendix.

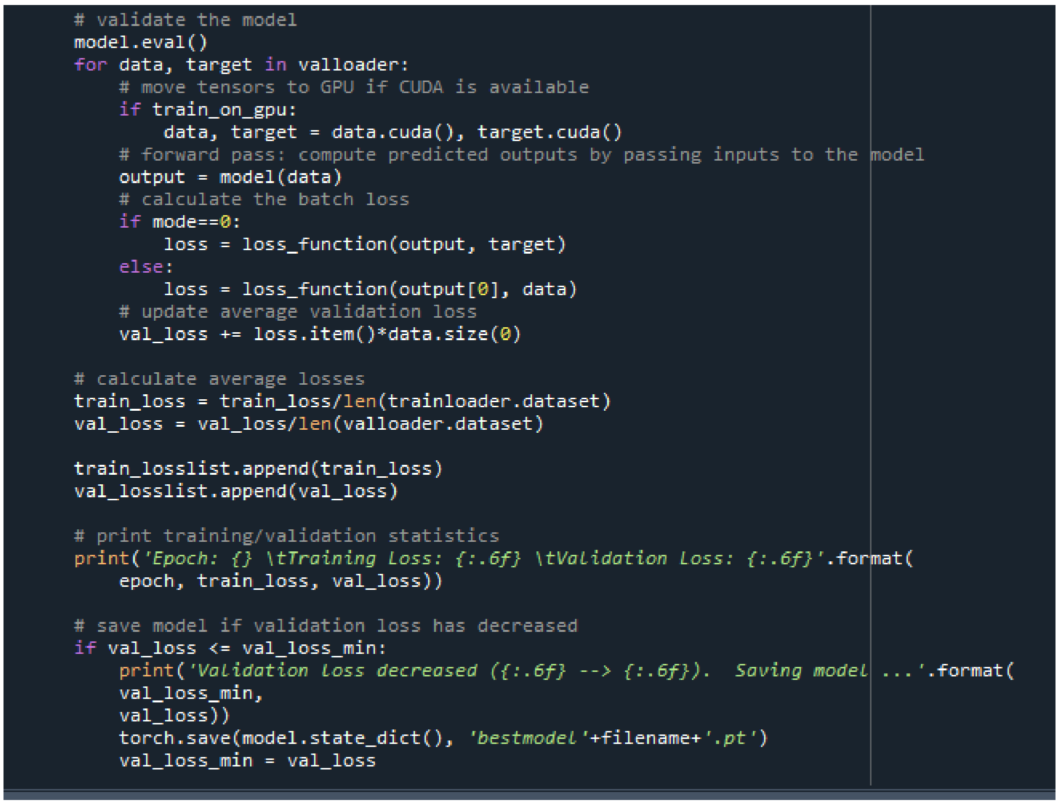
## 9.1 Model1



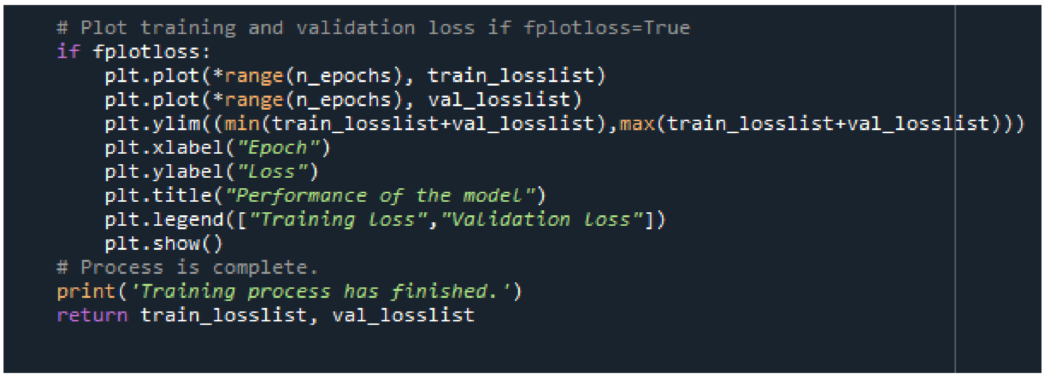
```python
import numpy as np
import os
import torch
from torch import nn
import torchvision as tv
from torchvision.datasets import CIFAR10
from torchvision import transforms
import matplotlib.pyplot as plt

torch.manual_seed(42)
dataset = CIFAR10(os.getcwd(), download=True, transform=transforms.ToTensor())


train_data,val_data = torch.utils.data.random_split(dataset,[40000,10000])
trainloader = torch.utils.data.DataLoader(train_data, batch_size=10, shuffle=True)
valloader = torch.utils.data.DataLoader(val_data, batch_size=10, shuffle=True)



########################Background work#############################
#-----------------------------------------------------------
#                        Training
def train_model(optimiser, model, loss_function, trainloader, valloader,
                n_epochs=10, fplotloss=True, fdraw=False, filename="", mode=0):

    # move tensors to GPU if CUDA is available
    train_on_gpu = torch.cuda.is_available()
    if train_on_gpu:
        print("GPU available! Train model on GPU.")
        model.cuda()

    train_losslist = []
    val_losslist = []


    # Track the best model so far (evaluated on validation set)
    val_loss_min = np.Inf

    print("Entering training cycles.")
    for epoch in [*range(n_epochs)]:

        # keep track of training and validation loss
        train_loss = 0.0
        val_loss = 0.0
```



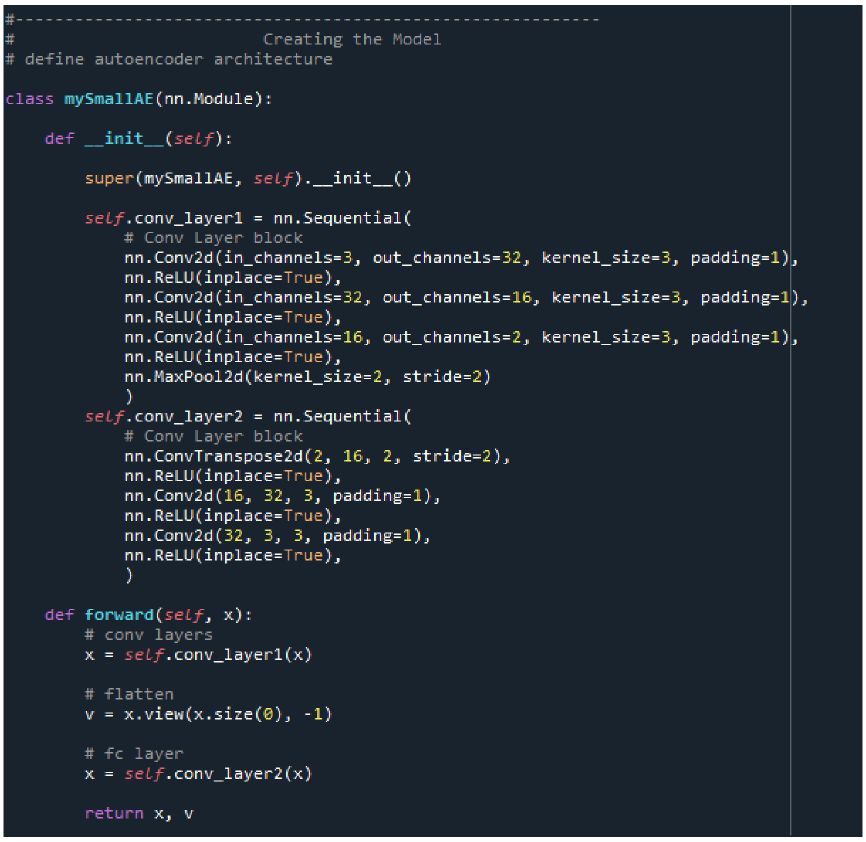
```python
        # train the model, one training cycle
        model.train()
        for data, target in trainloader:
            # move tensors to GPU if CUDA is available
            if train_on_gpu:
                data, target = data.cuda(), target.cuda()
            # clear the gradients of all optimized variables
            optimiser.zero_grad()
            # forward pass: compute predicted outputs by passing inputs to the model
            output = model(data)
            # calculate the batch loss
            if mode==0:
                loss = loss_function(output, target)
            else: # Be careful with the AE mode training
                loss = loss_function(output[0], data)
            # backward pass: compute gradient of the loss with respect to model parameters
            loss.backward()
            # perform a single optimization step (parameter update)
            optimiser.step()
            # update training loss
            train_loss += loss.item()*data.size(0)
```



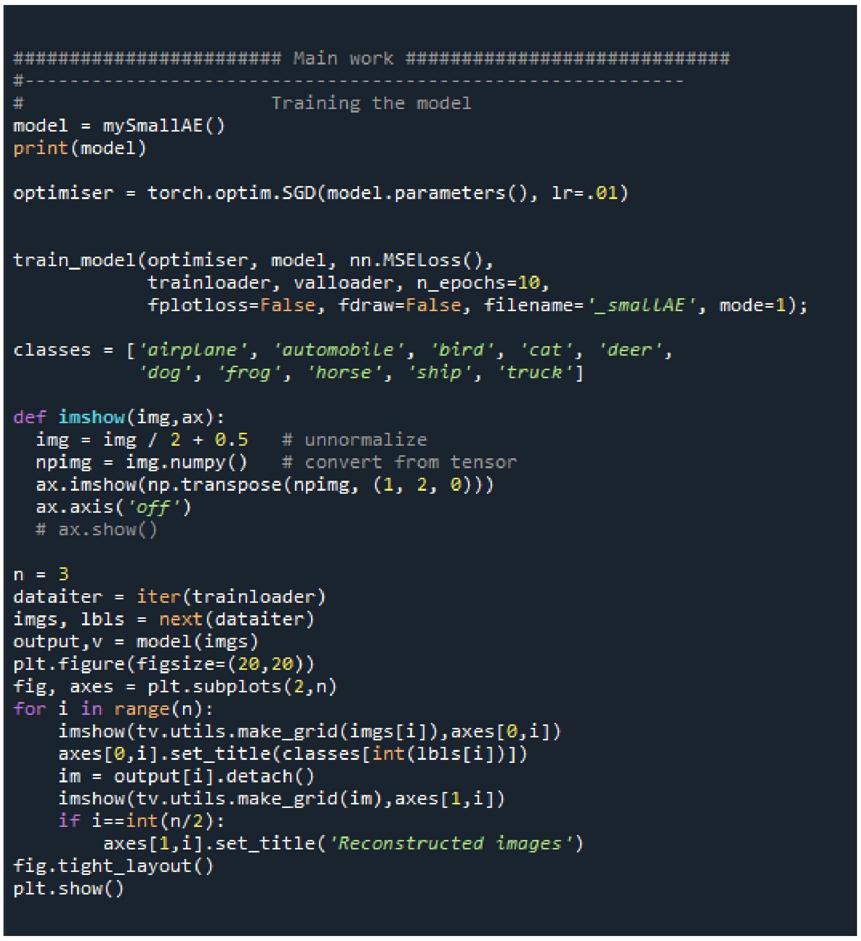
```
        # validate the model
        model.eval()
        for data, target in valloader:
            # move tensors to GPU if CUDA is available
            if train_on_gpu:
                data, target = data.cuda(), target.cuda()
            # forward pass: compute predicted outputs by passing inputs to the model
            output = model(data)
            # calculate the batch loss
            if mode==0:
                loss = loss_function(output, target)
            else:
                loss = loss_function(output[0], data)
            # update average validation loss
            val_loss += loss.item()*data.size(0)

        # calculate average losses
        train_loss = train_loss/len(trainloader.dataset)
        val_loss = val_loss/len(valloader.dataset)

        train_losslist.append(train_loss)
        val_losslist.append(val_loss)

        # print training/validation statistics
        print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
            epoch, train_loss, val_loss))

        # save model if validation loss has decreased
        if val_loss <= val_loss_min:
            print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(
            val_loss_min,
            val_loss))
            torch.save(model.state_dict(), 'bestmodel'+filename+'.pt')
            val_loss_min = val_loss
```



```
    # Plot training and validation loss if fplotloss=True
    if fplotloss:
        plt.plot(*range(n_epochs), train_losslist)
        plt.plot(*range(n_epochs), val_losslist)
        plt.ylim((min(train_losslist+val_losslist),max(train_losslist+val_losslist)))
        plt.xlabel("Epoch")
        plt.ylabel("Loss")
        plt.title("Performance of the model")
        plt.legend(["Training Loss","Validation Loss"])
        plt.show()
    # Process is complete.
    print('Training process has finished.')
    return train_losslist, val_losslist
```



```python
#------------------------------------------------------------
#                          Creating the Model
# define autoencoder architecture

class mySmallAE(nn.Module):

    def __init__(self):

        super(mySmallAE, self).__init__()

        self.conv_layer1 = nn.Sequential(
            # Conv Layer block
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=32, out_channels=16, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(in_channels=16, out_channels=2, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
            )
        self.conv_layer2 = nn.Sequential(
            # Conv Layer block
            nn.ConvTranspose2d(2, 16, 2, stride=2),
            nn.ReLU(inplace=True),
            nn.Conv2d(16, 32, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(32, 3, 3, padding=1),
            nn.ReLU(inplace=True),
            )

    def forward(self, x):
        # conv layers
        x = self.conv_layer1(x)

        # flatten
        v = x.view(x.size(0), -1)

        # fc layer
        x = self.conv_layer2(x)

        return x, v
```



```python
######################## Main work #############################
#------------------------------------------------------------
#                          Training the model
model = mySmallAE()
print(model)

optimiser = torch.optim.SGD(model.parameters(), lr=.01)


train_model(optimiser, model, nn.MSELoss(),
            trainloader, valloader, n_epochs=10,
            fplotloss=False, fdraw=False, filename='_smallAE', mode=1);

classes = ['airplane', 'automobile', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck']

def imshow(img,ax):
  img = img / 2 + 0.5   # unnormalize
  npimg = img.numpy()   # convert from tensor
  ax.imshow(np.transpose(npimg, (1, 2, 0)))
  ax.axis('off')
  # ax.show()

n = 3
dataiter = iter(trainloader)
imgs, lbls = next(dataiter)
output,v = model(imgs)
plt.figure(figsize=(20,20))
fig, axes = plt.subplots(2,n)
for i in range(n):
    imshow(tv.utils.make_grid(imgs[i]),axes[0,i])
    axes[0,i].set_title(classes[int(lbls[i])])
    im = output[i].detach()
    imshow(tv.utils.make_grid(im),axes[1,i])
    if i==int(n/2):
        axes[1,i].set_title('Reconstructed images')
fig.tight_layout()
plt.show()
```

## 9.2 Model2

This is the GitHub link to the source code for model2.
https://github.com/takyamamoto/CIFAR-ConvolutionalAutoEncoder-Chainer