Programming Assignment 2: Convolutional Neural Networks

Version 1.5

- Fixed the bug in the compute_loss function in part A

Version Release Date: 2022-02-05

Due Date: Friday, Feb. 18, at 11:59pm

Based on an assignment by Lisa Zhang

For CSC413/2516 in Winter 2022 with Professors Jimmy Ba and Bo Wang

Submission: You must submit two files through MarkUs: a PDF file containing your writeup, titled *a2-writeup.pdf*, and your code file *a2-code.ipynb*. Your writeup must be typeset.

The programming assignments are individual work. See the Course Syllabus for detailed policies.

Introduction:
This assignment will focus on the applications of convolutional neural networks in various image processing tasks. First, we will train a convolutional neural network for a task known as image colourization. Given a greyscale image, we will predict the colour at each pixel. This a difficult problem for many reasons, one of which being that it is ill-posed: for a single greyscale image, there can be multiple, equally valid colourings.

In the second half of the assignment, we switch gears and perform object detection by fine-tuning a pre-trained model. Specifically, we use the YOLOv3 (Redmon and Farhadi, 2018) pre-trained model and fine-tune it on the COCO (Lin et al., 2014) dataset.

Colab FAQ and Using GPU

For some basic overview and features offered in Colab notebooks, check out: Overview of Colaboratory Features.

You need to use the Colab GPU for this assignment by selecting:

> **Runtime $\rightarrow$ Change runtime type $\rightarrow$ Hardware Accelerator: GPU**

Download CIFAR and Colour dictionary

We will use the CIFAR-10 data set, which consists of images of size 32x32 pixels. For most of the questions we will use a subset of the dataset. To make the problem easier, we will only use the “Horse” category from this data set. Now let’s learn to colour some horses!

The data loading script is included below. It can take up to a couple of minutes to download everything the first time.

All files are stored at `/content/csc413/a2/data/` folder.

Helper code

You can ignore the restart warning.

```
######################################################################
# Setup working directory
######################################################################
%mkdir -p /content/csc413/a2/
%cd /content/csc413/a2

######################################################################
# Helper functions for loading data
######################################################################
# adapted from
# https://github.com/fchollet/keras/blob/master/keras/datasets/cifar10.py

import os
import pickle
import sys
import tarfile

import numpy as np
from PIL import Image
from six.moves.urllib.request import urlretrieve


def get_file(fname, origin, untar=False, extract=False, archive_format="auto", cache_dir="data"):
    datadir = os.path.join(cache_dir)
    if not os.path.exists(datadir):
        os.makedirs(datadir)

    if untar:
        untar_fpath = os.path.join(datadir, fname)
        fpath = untar_fpath + ".tar.gz"
    else:
        fpath = os.path.join(datadir, fname)

    print("File path: %s" % fpath)
    if not os.path.exists(fpath):
        print("Downloading data from", origin)

        error_msg = "URL fetch failure on {}: {} -- {}"
        try:
            try:
                urlretrieve(origin, fpath)
            except URLError as e:
                raise Exception(error_msg.format(origin, e.errno, e.reason))
            except HTTPError as e:
                raise Exception(error_msg.format(origin, e.code, e.msg))
        except (Exception, KeyboardInterrupt) as e:
            if os.path.exists(fpath):
                os.remove(fpath)
            raise

    if untar:
        if not os.path.exists(untar_fpath):
            print("Extracting file.")
            with tarfile.open(fpath) as archive:
                archive.extractall(datadir)
        return untar_fpath

    if extract:
        _extract_archive(fpath, datadir, archive_format)

    return fpath


def load_batch(fpath, label_key="labels"):
    """Internal utility for parsing CIFAR data.
    # Arguments
        fpath: path the file to parse.
        label_key: key for label data in the retrieve
            dictionary.
    # Returns
        A tuple `(data, labels)`.
    """
    f = open(fpath, "rb")
    if sys.version_info < (3,):
        d = pickle.load(f)
    else:
        d = pickle.load(f, encoding="bytes")
        # decode utf8
        d_decoded = {}
        for k, v in d.items():
            d_decoded[k.decode("utf8")] = v
        d = d_decoded
    f.close()
    data = d["data"]
    labels = d[label_key]

    data = data.reshape(data.shape[0], 3, 32, 32)
    return data, labels


def load_cifar10(transpose=False):
    """Loads CIFAR10 dataset.
    # Returns
        Tuple of Numpy arrays: `(x_train, y_train), (x_test, y_test)`.
    """
    dirname = "cifar-10-batches-py"
    origin = "http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz"
    path = get_file(dirname, origin=origin, untar=True)
    num_train_samples = 50000

    x_train = np.zeros((num_train_samples, 3, 32, 32), dtype="uint8")
    y_train = np.zeros((num_train_samples,), dtype="uint8")

    for i in range(1, 6):
        fpath = os.path.join(path, "data_batch_" + str(i))
        data, labels = load_batch(fpath)
        x_train[(i - 1) * 10000 : i * 10000, :, :, :] = data
        y_train[(i - 1) * 10000 : i * 10000] = labels

    fpath = os.path.join(path, "test_batch")
    x_test, y_test = load_batch(fpath)

    y_train = np.reshape(y_train, (len(y_train), 1))
    y_test = np.reshape(y_test, (len(y_test), 1))

    if transpose:
        x_train = x_train.transpose(0, 2, 3, 1)
        x_test = x_test.transpose(0, 2, 3, 1)
    return (x_train, y_train), (x_test, y_test)
```

```
/content/csc413/a2
```

Download files

This may take 1 or 2 mins for the first time.

```
# Download cluster centers for k-means over colours
colours_fpath = get_file(
    fname="colours", origin="http://www.cs.toronto.edu/~jba/kmeans_colour_a2.tar.gz", untar=True
)
# Download CIFAR dataset
m = load_cifar10()
```

```
File path: data/colours.tar.gz
File path: data/cifar-10-batches-py.tar.gz
```

Image Colourization as Classification

We will select a subset of 24 colours and frame colourization as a pixel-wise classification problem, where we label each pixel with one of 24 colours. The 24 colours are selected using k-means clustering over colours, and selecting cluster centers.

This was already done for you, and cluster centers are provided in http://www.cs.toronto.edu/~jba/kmeans_colour_a2.tar.gz, which was downloaded by the helper functions above. For simplicity, we will measure distance in RGB space. This is not ideal but reduces the software dependencies for this assignment.

Helper code

```
"""
Colourization of CIFAR-10 Horses via classification.
"""
import argparse
import math
import time

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as npr
import scipy.misc
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable

# from load_data import load_cifar10

HORSE_CATEGORY = 7
```

Data related code

```
def get_rgb_cat(xs, colours):
    """
    Get colour categories given RGB values. This function doesn't
    actually do the work, instead it splits the work into smaller
    chunks that can fit into memory, and calls helper function
    _get_rgb_cat

    Args:
      xs: float numpy array of RGB images in [B, C, H, W] format
      colours: numpy array of colour categories and their RGB values
    Returns:
      result: int numpy array of shape [B, 1, H, W]
    """
    if np.shape(xs)[0] < 100:
        return _get_rgb_cat(xs)
    batch_size = 100
    nexts = []
    for i in range(0, np.shape(xs)[0], batch_size):
        next = _get_rgb_cat(xs[i : i + batch_size, :, :, :], colours)
        nexts.append(next)
    result = np.concatenate(nexts, axis=0)
    return result
```

```
def _get_rgb_cat(xs, colours):
    """
    Get colour categories given RGB values. This is done by choosing
    the colour in `colours` that is the closest (in RGB space) to
    each point in the image `xs`. This function is a little memory
    intensive, and so the size of `xs` should not be too large.

    Args:
      xs: float numpy array of RGB images in [B, C, H, W] format
      colours: numpy array of colour categories and their RGB values
    Returns:
      result: int numpy array of shape [B, 1, H, W]
    """
    num_colours = np.shape(colours)[0]
    xs = np.expand_dims(xs, 0)
    cs = np.reshape(colours, [num_colours, 1, 3, 1, 1])
    dists = np.linalg.norm(xs - cs, axis=2)  # 2 = colour axis
    cat = np.argmin(dists, axis=0)
    cat = np.expand_dims(cat, axis=1)
    return cat


def get_cat_rgb(cats, colours):
    """
    Get RGB colours given the colour categories

    Args:
      cats: integer numpy array of colour categories
      colours: numpy array of colour categories and their RGB values
    Returns:
      numpy tensor of RGB colours
    """
    return colours[cats]


def process(xs, ys, max_pixel=256.0, downsize_input=False):
    """
    Pre-process CIFAR10 images by taking only the horse category,
    shuffling, and have colour values be bound between 0 and 1

    Args:
      xs: the colour RGB pixel values
      ys: the category labels
      max_pixel: maximum pixel value in the original data
    Returns:
      xs: value normalized and shuffled colour images
      grey: greyscale images, also normalized so values are between 0 and 1
    """
    xs = xs / max_pixel
    xs = xs[np.where(ys == HORSE_CATEGORY)[0], :, :, :]
    npr.shuffle(xs)

    grey = np.mean(xs, axis=1, keepdims=True)

    if downsize_input:
        downsize_module = nn.Sequential(
            nn.AvgPool2d(2),
            nn.AvgPool2d(2),
            nn.Upsample(scale_factor=2),
            nn.Upsample(scale_factor=2),
        )
        xs_downsized = downsize_module.forward(torch.from_numpy(xs).float())
        xs_downsized = xs_downsized.data.numpy()
        return (xs, xs_downsized)
    else:
        return (xs, grey)


def get_batch(x, y, batch_size):
    """
    Generated that yields batches of data

    Args:
      x: input values
      y: output values
      batch_size: size of each batch
    Yields:
      batch_x: a batch of inputs of size at most batch_size
      batch_y: a batch of outputs of size at most batch_size
    """
    N = np.shape(x)[0]
    assert N == np.shape(y)[0]
    for i in range(0, N, batch_size):
        batch_x = x[i : i + batch_size, :, :, :]
        batch_y = y[i : i + batch_size, :, :, :]
        yield (batch_x, batch_y)
```

Torch helper

```
def get_torch_vars(xs, ys, gpu=False):
    """
    Helper function to convert numpy arrays to pytorch tensors.
    If GPU is used, move the tensors to GPU.

    Args:
      xs (float numpy tenosor): greyscale input
      ys (int numpy tenosor): categorical labels
      gpu (bool): whether to move pytorch tensor to GPU
    Returns:
      Variable(xs), Variable(ys)
    """
    xs = torch.from_numpy(xs).float()
    ys = torch.from_numpy(ys).long()
    if gpu:
        xs = xs.cuda()
        ys = ys.cuda()
    return Variable(xs), Variable(ys)


def compute_loss(criterion, outputs, labels, batch_size, num_colours):
    """
    Helper function to compute the loss. Since this is a pixelwise
    prediction task we need to reshape the output and ground truth
    tensors into a 2D tensor before passing it in to the loss criteron.

    Args:
      criterion: pytorch loss criterion
      outputs (pytorch tensor): predicted labels from the model
      labels (pytorch tensor): ground truth labels
      batch_size (int): batch size used for training
      num_colours (int): number of colour categories
    Returns:
      pytorch tensor for loss
    """
    batch = outputs.size(0)
    loss_out = outputs.transpose(1, 3).contiguous().view([batch * 32 * 32, num_colours])
    loss_lab = labels.transpose(1, 3).contiguous().view([batch * 32 * 32])
    return criterion(loss_out, loss_lab)


def run_validation_step(
    cnn,
    criterion,
    test_grey,
    test_rgb_cat,
    batch_size,
    colours,
    plotpath=None,
    visualize=True,
    downsize_input=False
):
    correct = 0.0
    total = 0.0
    losses = []
    num_colours = np.shape(colours)[0]
    for i, (xs, ys) in enumerate(get_batch(test_grey, test_rgb_cat, batch_size)):
        images, labels = get_torch_vars(xs, ys, args.gpu)
        outputs = cnn(images)

        val_loss = compute_loss(
            criterion, outputs, labels, batch_size=args.batch_size, num_colours=num_colours
        )
        losses.append(val_loss.data.item())

        _, predicted = torch.max(outputs.data, 1, keepdim=True)
        total += labels.size(0) * 32 * 32
        correct += (predicted == labels.data).sum()

    if plotpath:  # only plot if a path is provided
        plot(
            xs,
            ys,
            predicted.cpu().numpy(),
            colours,
            plotpath,
            visualize=visualize,
            compare_bilinear=downsize_input,
        )
    val_loss = np.mean(losses)
    val_acc = 100 * correct / total
    return val_loss, val_acc
```

Visualization

```
def plot(input, gtlabel, output, colours, path, visualize, compare_bilinear=False):
    """
    Generate png plots of input, ground truth, and outputs

    Args:
      input: the greyscale input to the colourization CNN
      gtlabel: the grouth truth categories for each pixel
      output: the predicted categories for each pixel
      colours: numpy array of colour categories and their RGB values
      path: output path
      visualize: display the figures inline or save the figures in path
    """
    grey = np.transpose(input[:10, :, :, :], [0, 2, 3, 1])
    gtcolor = get_cat_rgb(gtlabel[:10, 0, :, :], colours)
    predcolor = get_cat_rgb(output[:10, 0, :, :], colours)

    img_stack = [np.hstack(np.tile(grey, [1, 1, 1, 3])), np.hstack(gtcolor), np.hstack(predcolor)]

    if compare_bilinear:
        downsize_module = nn.Sequential(
            nn.AvgPool2d(2),
            nn.AvgPool2d(2),
            nn.Upsample(scale_factor=2, mode="bilinear"),
            nn.Upsample(scale_factor=2, mode="bilinear"),
        )
        gt_input = np.transpose(
            gtcolor,
            [
                0,
                3,
                1,
                2
            ],
        )
        color_bilinear = downsize_module.forward(torch.from_numpy(gt_input).float())
        color_bilinear = np.transpose(color_bilinear.data.numpy(), [0, 2, 3, 1])
        img_stack = [
            np.hstack(np.transpose(input[:10, :, :, :], [0, 2, 3, 1])),
            np.hstack(gtcolor),
            np.hstack(predcolor),
            np.hstack(color_bilinear),
        ]
    img = np.vstack(img_stack)

    plt.grid(None)
    plt.imshow(img, vmin=0.0, vmax=1.0)
    if visualize:
        plt.show()
    else:
        plt.savefig(path)


def toimage(img, cmin, cmax):
    return Image.fromarray((img.clip(cmin, cmax) * 255).astype(np.uint8))


def plot_activation(args, cnn):
    # LOAD THE COLOURS CATEGORIES
    colours = np.load(args.colours, allow_pickle=True)[0]
    num_colours = np.shape(colours)[0]

    (x_train, y_train), (x_test, y_test) = load_cifar10()
    test_rgb, test_grey = process(x_test, y_test, downsize_input=args.downsize_input)
    test_rgb_cat = get_rgb_cat(test_rgb, colours)

    # Take the idnex of the test image
    id = args.index
    outdir = "outputs/" + args.experiment_name + "/act" + str(id)
    if not os.path.exists(outdir):
        os.makedirs(outdir)
    images, labels = get_torch_vars(
        np.expand_dims(test_grey[id], 0), np.expand_dims(test_rgb_cat[id], 0)
    )
    cnn.cpu()
    outputs = cnn(images)
    _, predicted = torch.max(outputs.data, 1, keepdim=True)
    predcolor = get_cat_rgb(predicted.cpu().numpy()[0, 0, :, :], colours)
    img = predcolor
    toimage(predcolor, cmin=0, cmax=1).save(os.path.join(outdir, "output_%d.png" % id))

    if not args.downsize_input:
        img = np.tile(np.transpose(test_grey[id], [1, 2, 0]), [1, 1, 3])
    else:
        img = np.transpose(test_grey[id], [1, 2, 0])
    toimage(img, cmin=0, cmax=1).save(os.path.join(outdir, "input_%d.png" % id))

    img = np.transpose(test_rgb[id], [1, 2, 0])
    toimage(img, cmin=0, cmax=1).save(os.path.join(outdir, "input_%d_gt.png" % id))

    def add_border(img):
        return np.pad(img, 1, "constant", constant_values=1.0)

    def draw_activations(path, activation, imgwidth=4):
        img = np.vstack(
            [
                np.hstack(
                    [
                        add_border(filter)
                        for filter in activation[i * imgwidth : (i + 1) * imgwidth, :, :]
                    ]
                )
                for i in range(activation.shape[0] // imgwidth)
            ]
        )
        scipy.misc.imsave(path, img)
    for i, tensor in enumerate([cnn.out1, cnn.out2, cnn.out3, cnn.out4, cnn.out5]):
        draw_activations(
            os.path.join(outdir, "conv%d_out_%d.png" % (i + 1, id)), tensor.data.cpu().numpy()[0]
        )
    print("visualization results are saved to %s" % outdir)
```

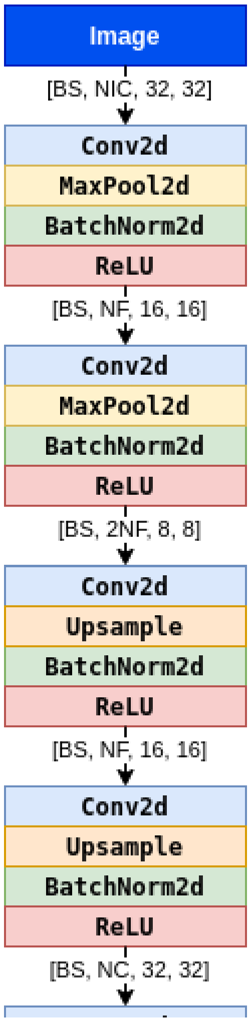
Training

```
class AttrDict(dict):
    def __init__(self, *args, **kwargs):
        super(AttrDict, self).__init__(*args, **kwargs)
        self.__dict__ = self


def train(args, cnn=None):
    # Set the maximum number of threads to prevent crash in Teaching Labs
    # TODO: necessary?
    torch.set_num_threads(5)
    # Numpy random seed
    npr.seed(args.seed)

    # Save directory
    save_dir = "outputs/" + args.experiment_name

    # LOAD THE COLOURS CATEGORIES
    colours = np.load(args.colours, allow_pickle=True, encoding="bytes")[0]
    num_colours = np.shape(colours)[0]
    # INPUT CHANNEL
    num_in_channels = 1 if not args.downsize_input else 3
    # LOAD THE MODEL
    if cnn is None:
        Net = globals()[args.model]
        cnn = Net(args.kernel, args.num_filters, num_colours, num_in_channels)

    # LOSS FUNCTION
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(cnn.parameters(), lr=args.learn_rate)

    # DATA
    print("Loading data...")
    (x_train, y_train), (x_test, y_test) = load_cifar10()

    print("Transforming data...")
    train_rgb, train_grey = process(x_train, y_train, downsize_input=args.downsize_input)
    train_rgb_cat = get_rgb_cat(train_rgb, colours)
    test_rgb, test_grey = process(x_test, y_test, downsize_input=args.downsize_input)
    test_rgb_cat = get_rgb_cat(test_rgb, colours)

    # Create the outputs folder if not created already
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)

    print("Beginning training ...")
    if args.gpu:
        cnn.cuda()
    start = time.time()

    train_losses = []
    valid_losses = []
    valid_accs = []
    for epoch in range(args.epochs):
        # Train the Model
        cnn.train()  # Change model to 'train' mode
        losses = []
        for i, (xs, ys) in enumerate(get_batch(train_grey, train_rgb_cat, args.batch_size)):
            images, labels = get_torch_vars(xs, ys, args.gpu)
            # Forward + Backward + Optimize
            optimizer.zero_grad()
            outputs = cnn(images)

            loss = compute_loss(
                criterion, outputs, labels, batch_size=args.batch_size, num_colours=num_colours
            )
            loss.backward()
            optimizer.step()
            losses.append(loss.data.item())

        # plot training images
        if args.plot:
            _, predicted = torch.max(outputs.data, 1, keepdim=True)
            plot(
                xs,
                ys,
                predicted.cpu().numpy(),
                colours,
                save_dir + "/train_%d.png" % epoch,
                args.visualize,
                args.downsize_input,
            )

        # plot training images
        avg_loss = np.mean(losses)
        train_losses.append(avg_loss)
        time_elapsed = time.time() - start
        print(
            "Epoch [%d/%d], Loss: %.4f, Time (s): %d"
            % (epoch + 1, args.epochs, avg_loss, time_elapsed)
        )

        # Evaluate the model
        cnn.eval()  # Change model to 'eval' mode (BN uses moving mean/var).
        val_loss, val_acc = run_validation_step(
            cnn,
            criterion,
            test_grey,
            test_rgb_cat,
            args.batch_size,
            colours,
            save_dir + "/test_%d.png" % epoch,
            args.visualize,
            args.downsize_input,
        )
        time_elapsed = time.time() - start
        valid_losses.append(val_loss)
        valid_accs.append(val_acc)
        print(
            "Epoch [%d/%d], Val Loss: %.4f, Val Acc: %.1f%%, Time(s): %.2f"
            % (epoch + 1, args.epochs, val_loss, val_acc, time_elapsed)
        )

    # Plot training curve
    plt.figure()
    plt.plot(train_losses, "ro-", label="Train")
    plt.plot(valid_losses, "go-", label="Validation")
    plt.legend()
    plt.title("Loss")
    plt.xlabel("Epochs")
    plt.savefig(save_dir + "/training_curve.png")

    if args.checkpoint:
        print("Saving model...")
        torch.save(cnn.state_dict(), args.checkpoint)

    return cnn
```

Part A: Pooling and Upsampling (2 pts)

Question 1

Complete the `PoolUpsampleNet` CNN model following the architecture described in the assignment handout.



In the diagram above, we denote the number of filters as **NF**. Further layers double the number of filters, denoted as **2NF**. In the final layers, the number of filters will be equivalent to the number of colour classes, denoted as **NC**. Consequently, your constructed neural network should define the number of input/output layers with respect to the variables `num_filters` and `num_colours`, as opposed to a constant value.

The specific modules to use are listed below. If parameters are not otherwise specified, use the default PyTorch parameters.

- `nn.Conv2d` — The number of input filters should match the second dimension of the *input* tensor (e.g. the first `nn.Conv2d` layer has **NIC** input filters). The number of output filters should match the second dimension of the *output* tensor (e.g. the first `nn.Conv2d` layer has **NF** output filters). Set kernel size to parameter `kernel`. Set padding to the `padding` variable included in the starter code.
- `nn.MaxPool2d` — Use `kernel_size=2` for all layers.
- `nn.BatchNorm2d` — The number of features is specified after the hyphen in the diagram as a multiple of **NF** or **NC**.
- `nn.Upsample` — Use `scaling_factor=2` for all layers.
- `nn.ReLU`

We recommend grouping each block of operations (those adjacent without whitespace in the diagram) into `nn.Sequential` containers. Grouping up relevant operations will allow for easier implementation of the `forward` method.

```
class PoolUpsampleNet(nn.Module):
    def __init__(self, kernel, num_filters, num_colours, num_in_channels):
        super().__init__()

        # Useful parameters
        padding = kernel // 2

        ############### YOUR CODE GOES HERE ###############
        self.block1 = nn.Sequential(
            nn.Conv2d(in_channels=num_in_channels, out_channels=num_filters,
                      kernel_size=kernel, padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.block2 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters, out_channels=2*num_filters,
                      kernel_size=kernel, padding=padding),
            nn.MaxPool2d(kernel_size=2),
            nn.BatchNorm2d(num_features=2*num_filters),
            nn.ReLU()
        )
        self.block3 = nn.Sequential(
            nn.Conv2d(in_channels=2*num_filters, out_channels=num_filters,
                      kernel_size=kernel, padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.block4 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters, out_channels=num_colours,
                      kernel_size=kernel, padding=padding),
            nn.Upsample(scale_factor=2),
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU()
        )
        self.block5 = nn.Conv2d(in_channels=num_colours,
                                out_channels=num_colours, kernel_size=kernel,
                                padding=padding)
        ###################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = self.block5(x)
        return x
        ####################################################
```

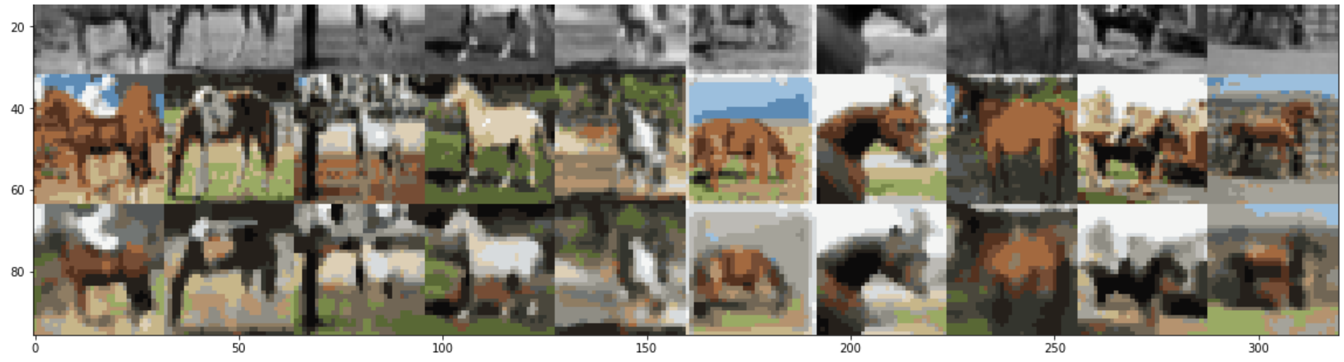
Question 2

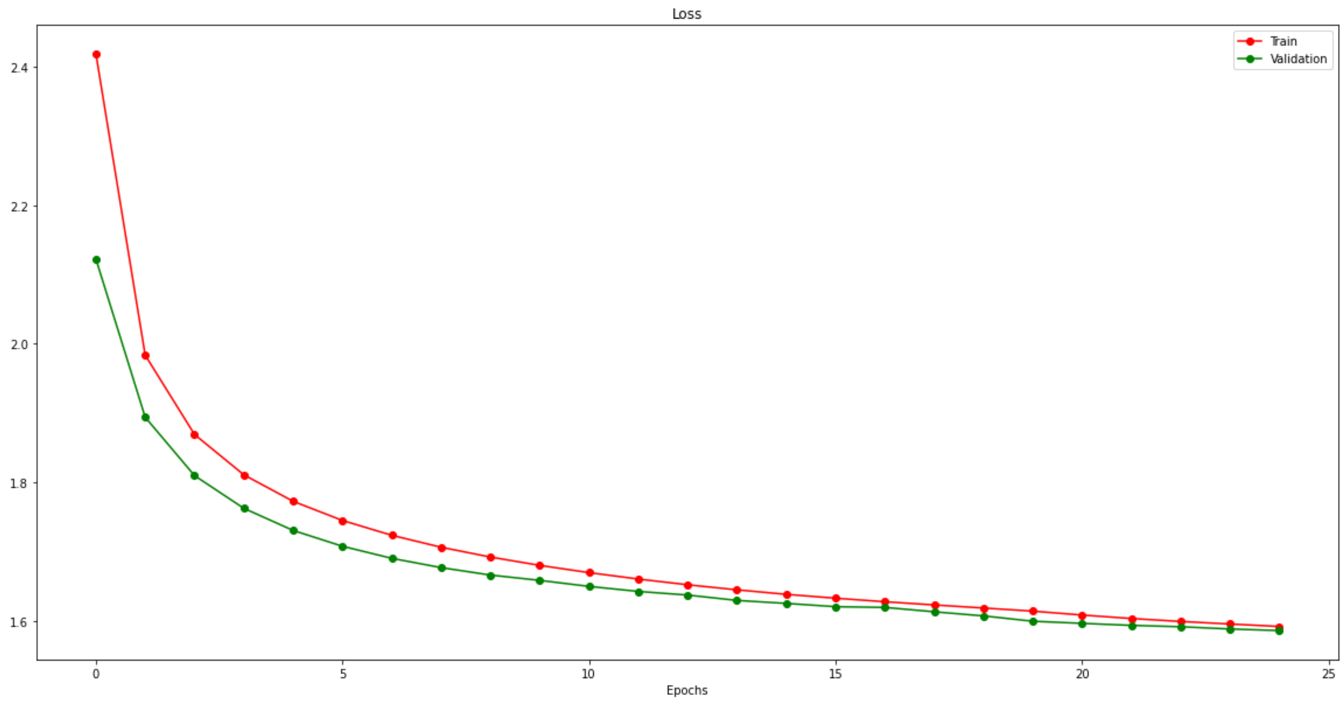
Run main training loop of `PoolUpsampleNet`. This will train the CNN for a few epochs using the cross-entropy objective. It will generate some images showing the trained result at the end. Do these results look good to you? Why or why not?

```
plt.rcParams['figure.figsize'] = [20,10]
```

```
args = AttrDict()
args_dict = {
    "gpu": True,
    "valid": False,
    "checkpoint": "",
    "colours": "./data/colours/colour_kmeans24_cat7.npy",
    "model": "PoolUpsampleNet",
    "kernel": 3,
    "num_filters": 32,
    'learn_rate':0.001,
    "batch_size": 100,
    "epochs": 25,
    "seed": 0,
    "plot": True,
    "experiment_name": "colourization_cnn",
    "visualize": False,
    "downsize_input": False,
}
args.update(args_dict)
cnn = train(args)
```

```
Loading data...
File path: data/cifar-10-batches-py.tar.gz
Transforming data...
Beginning training ...
Epoch [1/25], Loss: 2.4184, Time (s): 2
Epoch [1/25], Val Loss: 2.1222, Val Acc: 28.0%, Time(s): 2.97
Epoch [2/25], Loss: 1.9834, Time (s): 5
Epoch [2/25], Val Loss: 1.8938, Val Acc: 33.9%, Time(s): 5.92
Epoch [3/25], Loss: 1.8690, Time (s): 8
Epoch [3/25], Val Loss: 1.8096, Val Acc: 35.7%, Time(s): 9.01
Epoch [4/25], Loss: 1.8108, Time (s): 11
Epoch [4/25], Val Loss: 1.7624, Val Acc: 36.9%, Time(s): 12.23
Epoch [5/25], Loss: 1.7725, Time (s): 14
Epoch [5/25], Val Loss: 1.7305, Val Acc: 37.7%, Time(s): 15.61
Epoch [6/25], Loss: 1.7449, Time (s): 18
Epoch [6/25], Val Loss: 1.7077, Val Acc: 38.2%, Time(s): 19.14
Epoch [7/25], Loss: 1.7235, Time (s): 21
Epoch [7/25], Val Loss: 1.6903, Val Acc: 38.6%, Time(s): 22.84
Epoch [8/25], Loss: 1.7063, Time (s): 25
Epoch [8/25], Val Loss: 1.6769, Val Acc: 38.9%, Time(s): 26.71
Epoch [9/25], Loss: 1.6921, Time (s): 29
Epoch [9/25], Val Loss: 1.6661, Val Acc: 39.1%, Time(s): 30.70
Epoch [10/25], Loss: 1.6801, Time (s): 33
Epoch [10/25], Val Loss: 1.6585, Val Acc: 39.3%, Time(s): 34.90
Epoch [11/25], Loss: 1.6696, Time (s): 38
Epoch [11/25], Val Loss: 1.6498, Val Acc: 39.5%, Time(s): 39.28
Epoch [12/25], Loss: 1.6604, Time (s): 42
Epoch [12/25], Val Loss: 1.6425, Val Acc: 39.7%, Time(s): 43.77
Epoch [13/25], Loss: 1.6521, Time (s): 47
Epoch [13/25], Val Loss: 1.6373, Val Acc: 39.8%, Time(s): 48.43
Epoch [14/25], Loss: 1.6448, Time (s): 51
Epoch [14/25], Val Loss: 1.6295, Val Acc: 40.1%, Time(s): 53.24
Epoch [15/25], Loss: 1.6383, Time (s): 56
Epoch [15/25], Val Loss: 1.6252, Val Acc: 40.1%, Time(s): 58.22
Epoch [16/25], Loss: 1.6327, Time (s): 62
Epoch [16/25], Val Loss: 1.6205, Val Acc: 40.2%, Time(s): 63.80
Epoch [17/25], Loss: 1.6276, Time (s): 67
Epoch [17/25], Val Loss: 1.6194, Val Acc: 40.2%, Time(s): 69.06
Epoch [18/25], Loss: 1.6230, Time (s): 72
Epoch [18/25], Val Loss: 1.6131, Val Acc: 40.4%, Time(s): 74.76
Epoch [19/25], Loss: 1.6186, Time (s): 78
Epoch [19/25], Val Loss: 1.6072, Val Acc: 40.6%, Time(s): 80.61
Epoch [20/25], Loss: 1.6140, Time (s): 84
Epoch [20/25], Val Loss: 1.5995, Val Acc: 40.8%, Time(s): 86.37
Epoch [21/25], Loss: 1.6084, Time (s): 90
Epoch [21/25], Val Loss: 1.5964, Val Acc: 40.8%, Time(s): 92.30
Epoch [22/25], Loss: 1.6034, Time (s): 96
Epoch [22/25], Val Loss: 1.5935, Val Acc: 40.9%, Time(s): 98.48
Epoch [23/25], Loss: 1.5992, Time (s): 102
Epoch [23/25], Val Loss: 1.5915, Val Acc: 40.9%, Time(s): 104.81
Epoch [24/25], Loss: 1.5955, Time (s): 108
Epoch [24/25], Val Loss: 1.5883, Val Acc: 41.0%, Time(s): 111.22
Epoch [25/25], Loss: 1.5919, Time (s): 115
Epoch [25/25], Val Loss: 1.5860, Val Acc: 41.0%, Time(s): 117.76
```





ANSWER: The results do not look too good since a lot of the pixels are still gray and the outline of the objects are very blurry. The overall colors that are filled in are also very dull (not vibrant).

```
# This is formatted as code
```

Question 3

See assignment handout.

Total Number of Weights:

$$k^2(NIC \cdot NF + 4NF^2 + NF \cdot NC + NC^2)$$

- Block1: Conv2d = $k^2 NIC \cdot NF$; MaxPool2d = 0; Relu = 0
- Block2: Conv2d = $k^2 NF \cdot 2NF$; MaxPool2d = 0; Relu = 0
- Block3: Conv2d = $k^2 2NF \cdot NF$; Upsample = 0; Relu = 0

- Block4: Conv2d = $k^2 NF \cdot NC$; Upsample = 0; Relu = 0
- Block5: Conv2d = $k^2 NC \cdot NC$

Total Number of Outputs:

$2880NF + 3328NC$

- Block1: Conv2d = $32^2 NF$; MaxPool2d = $16^2 NF$; Relu = $16^2 NF$
- Block2: Conv2d = $16^2 2NF$; MaxPool2d = $8^2 2NF$; Relu = $8^2 2NF$
- Block3: Conv2d = $8^2 NF$; Upsample = $16^2 NF$; Relu = $16^2 NF$
- Block4: Conv2d = $16^2 NC$; Upsample = $32^2 NC$; Relu = $32^2 NC$
- Block5: Conv2d = $32^2 NC$

Total Number of Connections:

$1024k^2 NIC \cdot NF + 640k^2 NF^2 + 256k^2 NF \cdot NC$

$+1024k^2 NC^2 + 2432NF + 2048NC$

- Block1: Conv2d = $32^2 k^2 NIC \cdot NF$; MaxPool2d = $4 \cdot 16^2 NF$; Relu = $16^2 NF$
- Block2: Conv2d = $16^2 k^2 NF \cdot 2NF$; MaxPool2d = $4 \cdot 8^2 2NF$; Relu = $8^2 2NF$
- Block3: Conv2d = $8^2 k^2 2NF \cdot NF$; Upsample = $16^2 NF$; Relu = $16^2 NF$
- Block4: Conv2d = $16^2 k^2 NF \cdot NC$; Upsample = $32^2 NC$; Relu = $32^2 NC$
- Block5: Conv2d = $32^2 k^2 NC \cdot NC$

Total Number of Weights (doubled input dimensions):

$k^2(NIC \cdot NF + 4NF^2 + NF \cdot NC + NC^2)$

- Block1: Conv2d = $k^2 NIC \cdot NF$; MaxPool2d = 0; Relu = 0
- Block2: Conv2d = $k^2 NF \cdot 2NF$; MaxPool2d = 0; Relu = 0
- Block3: Conv2d = $k^2 2NF \cdot NF$; Upsample = 0; Relu = 0
- Block4: Conv2d = $k^2 NF \cdot NC$; Upsample = 0; Relu = 0
- Block5: Conv2d = $k^2 NC \cdot NC$

Total Number of Outputs (doubled input dimensions):

$4(2880NF + 3328NC)$

- Block1: Conv2d = $(2 \cdot 32)^2 NF$; MaxPool2d = $(2 \cdot 16)^2 NF$; Relu = $(2 \cdot 16)^2 NF$
- Block2: Conv2d = $(2 \cdot 16)^2 2NF$; MaxPool2d = $(2 \cdot 8)^2 2NF$; Relu = $(2 \cdot 8)^2 2NF$
- Block3: Conv2d = $(2 \cdot 8)^2 NF$; Upsample = $(2 \cdot 16)^2 NF$; Relu = $(2 \cdot 16)^2 NF$
- Block4: Conv2d = $(2 \cdot 16)^2 NC$; Upsample = $(2 \cdot 32)^2 NC$; Relu = $(2 \cdot 32)^2 NC$
- Block5: Conv2d = $(2 \cdot 32)^2 NC$

Total Number of Connections (doubled input dimensions):

$4(1024k^2 NIC \cdot NF + 640k^2 NF^2 + 256k^2 NF \cdot NC$

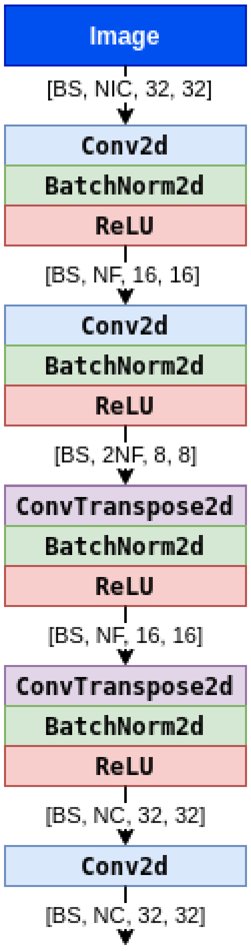
$+1024k^2NC^2 + 2432NF + 2048NC)$

- Block1: Conv2d = $(2 \cdot 32)^2 k^2 NIC \cdot NF$; MaxPool2d = $4 \cdot (2 \cdot 16)^2 NF$; Relu = $(2 \cdot 16)^2 NF$
- Block2: Conv2d = $(2 \cdot 16)^2 k^2 NF \cdot 2NF$; MaxPool2d = $4 \cdot (2 \cdot 8)^2 2NF$; Relu = $(2 \cdot 8)^2 2NF$
- Block3: Conv2d = $(2 \cdot 8)^2 k^2 2NF \cdot NF$; Upsample = $(2 \cdot 16)^2 NF$; Relu = $(2 \cdot 16)^2 NF$
- Block4: Conv2d = $(2 \cdot 16)^2 k^2 NF \cdot NC$; Upsample = $(2 \cdot 32)^2 NC$; Relu = $(2 \cdot 32)^2 NC$
- Block5: Conv2d = $(2 \cdot 32)^2 k^2 NC \cdot NC$

Part B: Strided and Transposed Dilated Convolutions (3 pts)

Question 1

Complete the `ConvTransposeNet` CNN model following the architecture described in the assignment handout.



An excellent visualization of convolutions and transposed convolutions with strides can be found here: https://github.com/vdumoulin/conv_arithmetic.

The specific modules to use are listed below. If parameters are not otherwise specified, use the default PyTorch parameters.

- `nn.Conv2d` — The number of input and output filters, and the kernel size, should be set in the same way as Part A. For the first two `nn.Conv2d` layers, set `stride` to 2 and set `padding` to 1.
- `nn.BatchNorm2d` — The number of features should be specified in the same way as for Part A.
- `nn.ConvTranspose2d` — The number of input filters should match the second dimension of the *input* tensor. The number of output filters should match the second dimension of the *output* tensor. Set `kernel_size` to parameter `kernel`. Set `stride` to 2, set `dilation` to 1, and set both `padding` and `output_padding` to 1.
- `nn.ReLU`

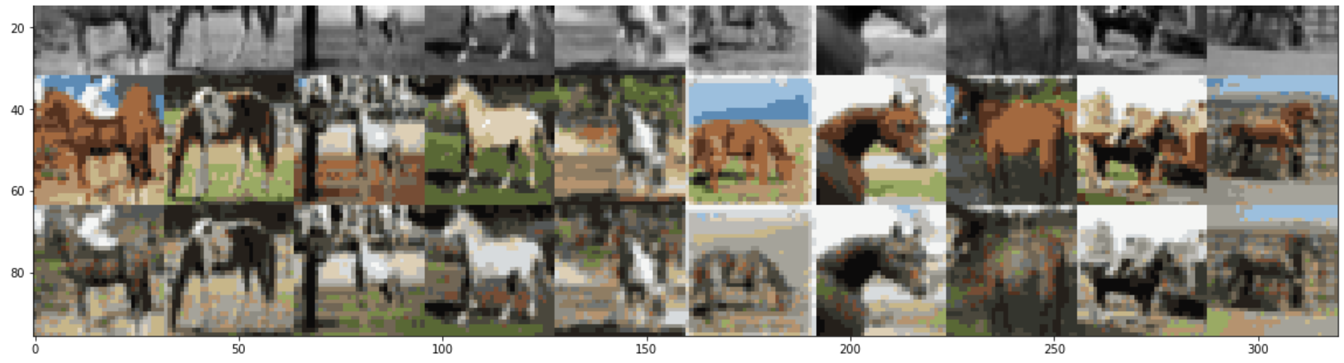
```
class ConvTransposeNet(nn.Module):
    def __init__(self, kernel, num_filters, num_colours, num_in_channels):
        super().__init__()

        # Useful parameters
        stride = 2
        padding = kernel // 2
        output_padding = 1

        ############### YOUR CODE GOES HERE ###############
        self.block1 = nn.Sequential(
            nn.Conv2d(in_channels=num_in_channels, out_channels=num_filters,
                      kernel_size=kernel, padding=1, stride=2),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.block2 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters, out_channels=2*num_filters,
                      kernel_size=kernel, padding=1, stride=2),
            nn.BatchNorm2d(num_features=2*num_filters),
            nn.ReLU()
        )
        self.block3 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=2*num_filters,
                               out_channels=num_filters, kernel_size=kernel,
                               padding=1, stride=2, output_padding=1,
                               dilation=1),
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.block4 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=num_filters,
                               out_channels=num_colours, kernel_size=kernel,
                               padding=1, stride=2, output_padding=1,
                               dilation=1),
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU()
        )
        self.block5 = nn.Conv2d(in_channels=num_colours,
                                out_channels=num_colours,
                                kernel_size=kernel, padding=padding)
        ###################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = self.block5(x)
        return x
        ###################################################
```

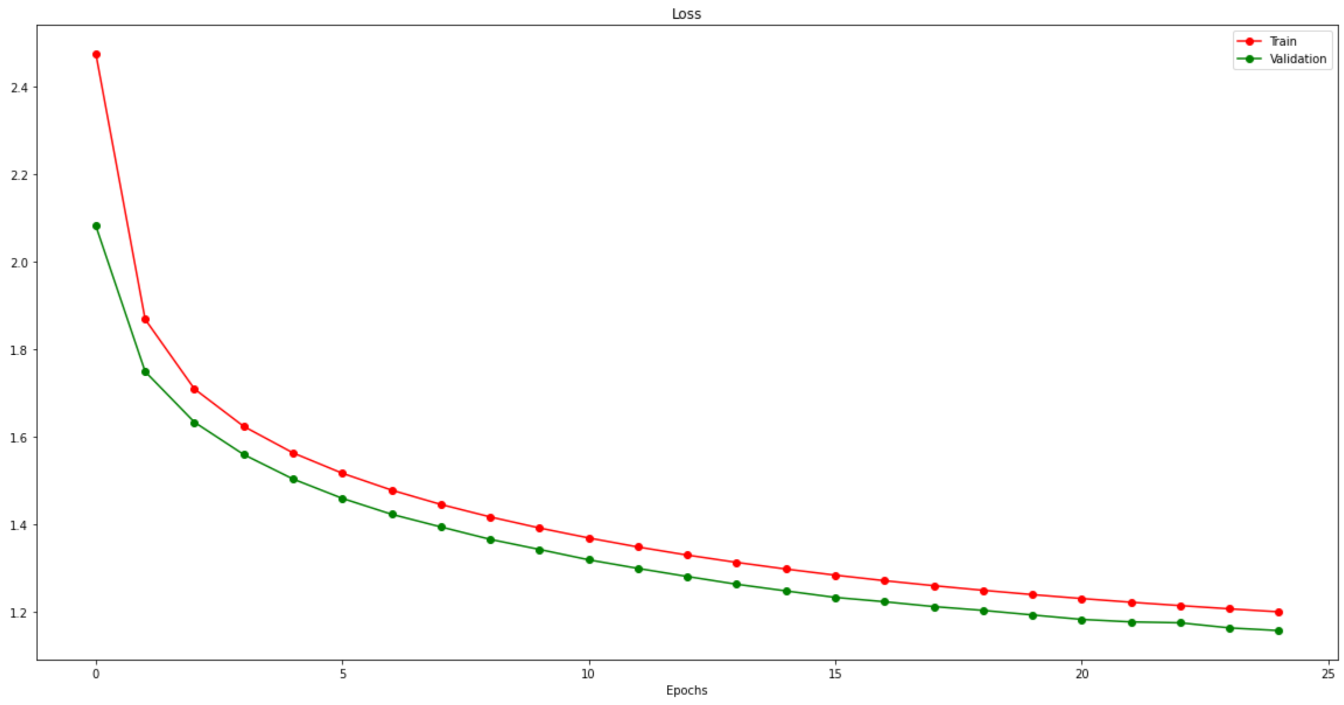
Question 2

Train the model for at least 25 epochs using a batch size of 100 and a kernel size of 3. Plot the training curve, and include this plot in your write-up. How do the results compare to the previous model?

```
args = AttrDict()
args_dict = {
    "gpu": True,
    "valid": False,
    "checkpoint": "",
    "colours": "./data/colours/colour_kmeans24_cat7.npy",
    "model": "ConvTransposeNet",
    "kernel": 3,
    "num_filters": 32,
    'learn_rate':0.001,
    "batch_size": 100,
    "epochs": 25,
    "seed": 0,
    "plot": True,
    "experiment_name": "colourization_cnn",
    "visualize": False,
    "downsize_input": False,
}
args.update(args_dict)
cnn = train(args)
```

```
Loading data...
File path: data/cifar-10-batches-py.tar.gz
Transforming data...
Beginning training ...
Epoch [1/25], Loss: 2.4758, Time (s): 2
Epoch [1/25], Val Loss: 2.0819, Val Acc: 31.4%, Time(s): 3.28
Epoch [2/25], Loss: 1.8682, Time (s): 6
Epoch [2/25], Val Loss: 1.7487, Val Acc: 38.1%, Time(s): 6.72
Epoch [3/25], Loss: 1.7089, Time (s): 9
Epoch [3/25], Val Loss: 1.6328, Val Acc: 40.9%, Time(s): 10.30
Epoch [4/25], Loss: 1.6233, Time (s): 13
Epoch [4/25], Val Loss: 1.5589, Val Acc: 42.8%, Time(s): 14.03
Epoch [5/25], Loss: 1.5627, Time (s): 17
Epoch [5/25], Val Loss: 1.5029, Val Acc: 44.4%, Time(s): 17.91
Epoch [6/25], Loss: 1.5161, Time (s): 21
Epoch [6/25], Val Loss: 1.4586, Val Acc: 45.6%, Time(s): 21.95
Epoch [7/25], Loss: 1.4775, Time (s): 25
Epoch [7/25], Val Loss: 1.4222, Val Acc: 46.6%, Time(s): 26.15
Epoch [8/25], Loss: 1.4447, Time (s): 29
Epoch [8/25], Val Loss: 1.3933, Val Acc: 47.4%, Time(s): 30.50
Epoch [9/25], Loss: 1.4162, Time (s): 33
Epoch [9/25], Val Loss: 1.3649, Val Acc: 48.3%, Time(s): 34.99
Epoch [10/25], Loss: 1.3908, Time (s): 38
Epoch [10/25], Val Loss: 1.3417, Val Acc: 49.0%, Time(s): 39.63
Epoch [11/25], Loss: 1.3680, Time (s): 43
Epoch [11/25], Val Loss: 1.3180, Val Acc: 49.7%, Time(s): 44.43
Epoch [12/25], Loss: 1.3474, Time (s): 48
Epoch [12/25], Val Loss: 1.2983, Val Acc: 50.4%, Time(s): 49.40
Epoch [13/25], Loss: 1.3288, Time (s): 53
Epoch [13/25], Val Loss: 1.2798, Val Acc: 51.0%, Time(s): 54.52
Epoch [14/25], Loss: 1.3121, Time (s): 58
Epoch [14/25], Val Loss: 1.2621, Val Acc: 51.5%, Time(s): 59.80
Epoch [15/25], Loss: 1.2969, Time (s): 63
Epoch [15/25], Val Loss: 1.2469, Val Acc: 52.0%, Time(s): 65.29
Epoch [16/25], Loss: 1.2830, Time (s): 69
Epoch [16/25], Val Loss: 1.2320, Val Acc: 52.6%, Time(s): 70.86
Epoch [17/25], Loss: 1.2703, Time (s): 74
Epoch [17/25], Val Loss: 1.2221, Val Acc: 52.8%, Time(s): 76.56
Epoch [18/25], Loss: 1.2588, Time (s): 80
Epoch [18/25], Val Loss: 1.2110, Val Acc: 53.2%, Time(s): 82.47
Epoch [19/25], Loss: 1.2483, Time (s): 86
Epoch [19/25], Val Loss: 1.2023, Val Acc: 53.5%, Time(s): 88.53
Epoch [20/25], Loss: 1.2385, Time (s): 92
Epoch [20/25], Val Loss: 1.1919, Val Acc: 53.8%, Time(s): 94.70
Epoch [21/25], Loss: 1.2294, Time (s): 99
Epoch [21/25], Val Loss: 1.1817, Val Acc: 54.1%, Time(s): 101.05
Epoch [22/25], Loss: 1.2210, Time (s): 105
Epoch [22/25], Val Loss: 1.1760, Val Acc: 54.3%, Time(s): 107.61
Epoch [23/25], Loss: 1.2132, Time (s): 112
Epoch [23/25], Val Loss: 1.1739, Val Acc: 54.3%, Time(s): 114.39
Epoch [24/25], Loss: 1.2059, Time (s): 118
Epoch [24/25], Val Loss: 1.1622, Val Acc: 54.7%, Time(s): 121.29
Epoch [25/25], Loss: 1.1990, Time (s): 125
Epoch [25/25], Val Loss: 1.1562, Val Acc: 54.9%, Time(s): 128.35
```





Questions 3 – 5

Q3)

ANSWER: This model has better performance than the part A model. This model has lower validation loss of 1.1505 compared to 1.5812 for part A model and higher validation accuracy of 55.0% compared to 41.2% for part A model. This might be because this model uses

ConvTranspose2D which is essentially a convolution with trainable parameters instead of part A model's Upsample layer which uses nearest interpolation. It may also be because this model uses a stride of 2 during the convolution instead of using max pooling (again, convolution has parameters that can be learned, whereas max pooling is a fixed operation).

Q4)

ANSWER: If k=4, then padding=1 and output_padding=0. If k=5, then padding=2 and output_padding=1.

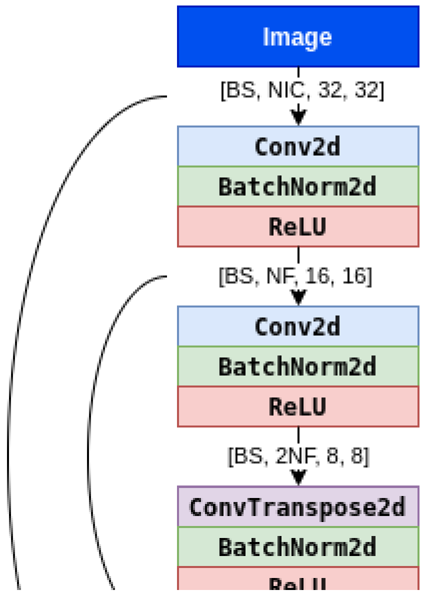
Q5)

ANSWEER: For a fixed number of epochs, larger batchsize leads to higher validation loss and worse performance.

Part C. Skip Connections (1 pts)

A skip connection in a neural network is a connection which skips one or more layer and connects to a later layer. We will introduce skip connections to our previous model.

Question 1



In this question, we will be adding a skip connection from the first layer to the last, second layer to the second last, etc. That is, the final convolution should have both the output of the previous layer and the initial greyscale input as input. This type of skip-connection is introduced by Ronneberger et al.[2015], and is called a "UNet".

Just like the `ConvTransposeNet` class that you have completed in the previous part, complete the `__init__` and `forward` methods methods of the `UNet` class below.

Hint: You will need to use the function `torch.cat`.

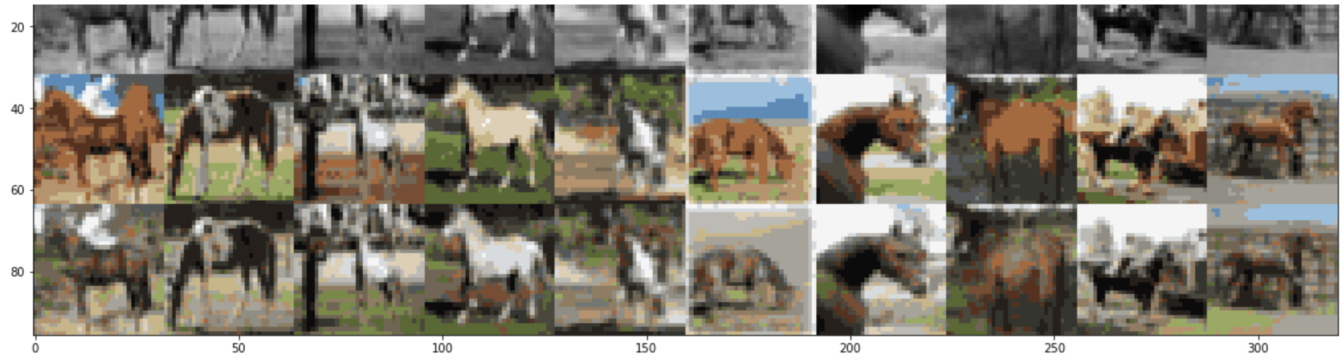
```
class UNet(nn.Module):
    def __init__(self, kernel, num_filters, num_colours, num_in_channels):
        super().__init__()

        # Useful parameters
        stride = 2
        padding = kernel // 2
        output_padding = 1

        ############### YOUR CODE GOES HERE ###############
        self.block1 = nn.Sequential(
            nn.Conv2d(in_channels=num_in_channels, out_channels=num_filters, kernel_size=kernel, padding=1, s
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.block2 = nn.Sequential(
            nn.Conv2d(in_channels=num_filters, out_channels=2*num_filters, kernel_size=kernel, padding=1, str
            nn.BatchNorm2d(num_features=2*num_filters),
            nn.ReLU()
        )
        self.block3 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=2*num_filters, out_channels=num_filters, kernel_size=kernel, paddi
            nn.BatchNorm2d(num_features=num_filters),
            nn.ReLU()
        )
        self.block4 = nn.Sequential(
            nn.ConvTranspose2d(in_channels=num_filters + num_filters, out_channels=num_colours, kernel_size=k
            nn.BatchNorm2d(num_features=num_colours),
            nn.ReLU()
        )
        self.block5 = nn.Conv2d(in_channels=num_colours + num_in_channels, out_channels=num_colours, kernel_s
        ###################################################

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        x_1 = self.block1(x)
        x_2 = self.block2(x_1)
        x_3 = self.block3(x_2)
        x_3 = torch.cat([x_3, x_1], 1)
        x_4 = self.block4(x_3)
        x_4 = torch.cat([x_4, x], 1)
        x_5 = self.block5(x_4)
        return x_5
        ###################################################
```

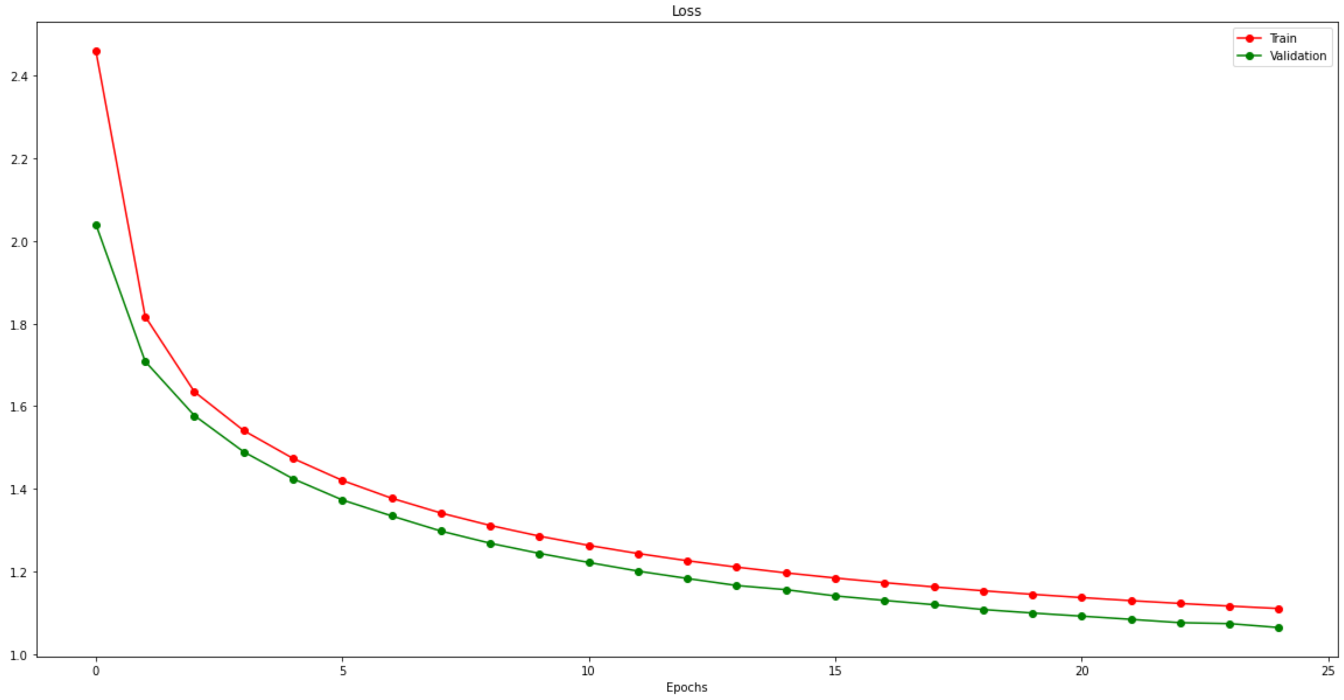
Question 2

Train the model for at least 25 epochs using a batch size of 100 and a kernel size of 3. Plot the training curve, and include this plot in your write-up.

```
args = AttrDict()
args_dict = {
    "gpu": True,
    "valid": False,
    "checkpoint": "",
    "colours": "./data/colours/colour_kmeans24_cat7.npy",
    "model": "UNet",
    "kernel": 3,
    "num_filters": 32,
    'learn_rate':0.001,
    "batch_size": 100,
    "epochs": 25,
    "seed": 0,
    "plot": True,
    "experiment_name": "colourization_cnn",
    "visualize": False,
    "downsize_input": False,
}
args.update(args_dict)
cnn = train(args)
```

```
Loading data...
File path: data/cifar-10-batches-py.tar.gz
Transforming data...
Beginning training ...
Epoch [1/25], Loss: 2.4597, Time (s): 3
Epoch [1/25], Val Loss: 2.0400, Val Acc: 32.6%, Time(s): 3.67
Epoch [2/25], Loss: 1.8170, Time (s): 6
Epoch [2/25], Val Loss: 1.7083, Val Acc: 38.2%, Time(s): 7.46
Epoch [3/25], Loss: 1.6346, Time (s): 10
Epoch [3/25], Val Loss: 1.5771, Val Acc: 41.9%, Time(s): 11.41
Epoch [4/25], Loss: 1.5408, Time (s): 14
Epoch [4/25], Val Loss: 1.4894, Val Acc: 44.9%, Time(s): 15.50
Epoch [5/25], Loss: 1.4735, Time (s): 18
Epoch [5/25], Val Loss: 1.4243, Val Acc: 47.1%, Time(s): 19.74
Epoch [6/25], Loss: 1.4203, Time (s): 23
Epoch [6/25], Val Loss: 1.3731, Val Acc: 48.7%, Time(s): 24.16
Epoch [7/25], Loss: 1.3773, Time (s): 27
Epoch [7/25], Val Loss: 1.3345, Val Acc: 49.9%, Time(s): 28.72
Epoch [8/25], Loss: 1.3415, Time (s): 32
Epoch [8/25], Val Loss: 1.2978, Val Acc: 51.1%, Time(s): 33.43
Epoch [9/25], Loss: 1.3114, Time (s): 37
Epoch [9/25], Val Loss: 1.2683, Val Acc: 52.0%, Time(s): 38.27
Epoch [10/25], Loss: 1.2855, Time (s): 42
Epoch [10/25], Val Loss: 1.2437, Val Acc: 52.7%, Time(s): 43.35
Epoch [11/25], Loss: 1.2631, Time (s): 47
Epoch [11/25], Val Loss: 1.2218, Val Acc: 53.4%, Time(s): 48.51
Epoch [12/25], Loss: 1.2434, Time (s): 52
Epoch [12/25], Val Loss: 1.2009, Val Acc: 54.1%, Time(s): 53.84
Epoch [13/25], Loss: 1.2260, Time (s): 58
Epoch [13/25], Val Loss: 1.1831, Val Acc: 54.6%, Time(s): 60.03
Epoch [14/25], Loss: 1.2106, Time (s): 64
Epoch [14/25], Val Loss: 1.1661, Val Acc: 55.2%, Time(s): 65.66
Epoch [15/25], Loss: 1.1967, Time (s): 70
Epoch [15/25], Val Loss: 1.1559, Val Acc: 55.5%, Time(s): 71.87
Epoch [16/25], Loss: 1.1842, Time (s): 76
Epoch [16/25], Val Loss: 1.1409, Val Acc: 55.9%, Time(s): 77.78
Epoch [17/25], Loss: 1.1730, Time (s): 82
Epoch [17/25], Val Loss: 1.1300, Val Acc: 56.2%, Time(s): 83.86
Epoch [18/25], Loss: 1.1627, Time (s): 88
Epoch [18/25], Val Loss: 1.1197, Val Acc: 56.5%, Time(s): 90.10
Epoch [19/25], Loss: 1.1533, Time (s): 94
Epoch [19/25], Val Loss: 1.1079, Val Acc: 56.9%, Time(s): 96.50
Epoch [20/25], Loss: 1.1448, Time (s): 101
Epoch [20/25], Val Loss: 1.0995, Val Acc: 57.2%, Time(s): 103.07
Epoch [21/25], Loss: 1.1368, Time (s): 107
Epoch [21/25], Val Loss: 1.0920, Val Acc: 57.3%, Time(s): 109.75
Epoch [22/25], Loss: 1.1295, Time (s): 114
Epoch [22/25], Val Loss: 1.0842, Val Acc: 57.5%, Time(s): 116.61
Epoch [23/25], Loss: 1.1227, Time (s): 121
Epoch [23/25], Val Loss: 1.0762, Val Acc: 57.7%, Time(s): 123.61
Epoch [24/25], Loss: 1.1163, Time (s): 128
Epoch [24/25], Val Loss: 1.0738, Val Acc: 57.7%, Time(s): 130.77
Epoch [25/25], Loss: 1.1105, Time (s): 135
Epoch [25/25], Val Loss: 1.0644, Val Acc: 58.0%, Time(s): 138.02
```





Question 3

ANSWER:

The skip-connection model had improved performance compared to the previous model. It has a validation loss of 1.0644 and accuracy of 58% compared to the previous model B's validation loss of 1.1602 and accuracy of 55%. The output images are also a little bit more clear and accurate that the previou ones.

This may be because the skip-connection model allows the model to reuse features learned in previous layers as well as allowing for more paramters to optimize (increases model complexity).

Object Detection as Regression and Classification - the YOLO approach

In the previous two parts, we worked on training models for image colourization. Now we will switch gears and perform object detection by fine-tuning a pre-trained model.

For the following, you are not expected to read the referenced papers, though the writing is very entertaining (by academic paper standards) and it may help provide additional context.

We use the YOLO (You Only Look Once) approach, as laid out in the the original paper by Redmon et al. YOLO uses a single neural network to predict bounding boxes (4 coordinates describing the corners of the box bounding a particular object) and class probabilities (what object is in the bounding box) based on a single pass over an image. It first divides the image into a grid, and for each grid cell predicts bounding boxes, confidence for those boxes, and conditional class probabilities.

For the YOLOv3 model, which we use here, we draw from their YOLOv3 paper which also builds on the previous YOLO9000 paper.

We use the pretrained YOLOv3 model weights and fine-tune it on the COCO (Lin et al., 2014) dataset.

Setup

```
!git clone https://github.com/Silent-Zebra/2022
```

```
fatal: destination path '2022' already exists and is not an empty directory.
```

Rerun the cd command below if you restart the runtime (but everything should work fine without restarting the runtime anyway)

```
%cd 2022/assets/assignments/pa2-q4-files
```

```
/content/csc413/a2/2022/assets/assignments/pa2-q4-files
```

```
def notebook_init():
    # For  notebooks
    print('Checking setup...')
    from IPython import display  # to display images and clear console output

    from utils.general import emojis
    from utils.torch_utils import select_device  # imports

    display.clear_output()
    select_device(newline=False)
    print(emojis('Setup complete ✅'))
    return display

display = notebook_init()
```

```
YOLOv3 🚀 46dad08 torch 1.10.0+cu111 CUDA:0 (Tesla K80, 11441MiB)
Setup complete ✅
```

In my experience you don't have to restart the runtime after the below installation

```
!pip install -r requirements.txt
```

```
Requirement already satisfied: matplotlib>=3.2.2 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: opencv-python>=4.1.2 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.7/dist-packages (
Collecting PyYAML>=5.3.1
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_1
     |████████████████████████████████| 596 kB 11.9 MB/s
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: torch>=1.7.0 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: torchvision>=0.8.1 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: tensorboard>=2.4.1 in /usr/local/lib/python3.7/dist-packa
Collecting wandb
  Downloading wandb-0.12.10-py2.py3-none-any.whl (1.7 MB)
     |████████████████████████████████| 1.7 MB 47.9 MB/s
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.7/dist-packages
Collecting thop
  Downloading thop-0.0.31.post2005241907-py3-none-any.whl (8.7 kB)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/li
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: protobuf>=3.6.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/p
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: grpcio>=1.24.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from absl-
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages
Collecting shortuuid>=0.5.0
  Downloading shortuuid-1.0.8-py3-none-any.whl (9.5 kB)
Collecting GitPython>=1.0.0
  Downloading GitPython-3.1.26-py3-none-any.whl (180 kB)
     |████████████████████████████████| 180 kB 53.4 MB/s
Requirement already satisfied: promise<3,>=2.0 in /usr/local/lib/python3.7/dist-packages
```

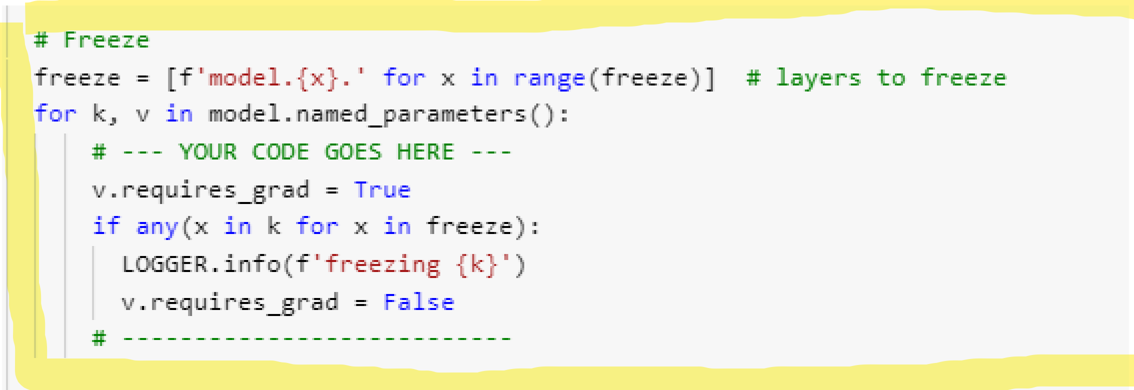
Part D.1: Freezing Parameters

A common practice in computer vision tasks is to take a pre-trained model trained on a large datset and finetune only parts of the model for a specific usecase. This can be helpful, for example, for preventing overfitting if the dataset we fine-tune on is small.

In this notebook, we are finetuning on the COCO dataset, and freezing model parameters is not strictly necessary here. However, it still allows for faster training and is meant to be instructive.

Fill in the section in `train.py` (2022/assets/assignments/pa2-q4-files/train.py) for freezing model parameters (line 129). The key idea here is to set `requires_grad` to be `False` for all frozen layers `v` and `True` for all other layers `v`. Hint: it might be helpful to check a condition such as: `any(x in k for x in freeze)`.

```
    Created wheel for pathtools: filename=pathtools-0.1.2-py3-none-any.whl size=8806 sha2
```



```
# Freeze
freeze = [f'model.{x}.' for x in range(freeze)]  # layers to freeze
for k, v in model.named_parameters():
    # --- YOUR CODE GOES HERE ---
    v.requires_grad = True
    if any(x in k for x in freeze):
      LOGGER.info(f'freezing {k}')
      v.requires_grad = False
    # ---------------------------
```

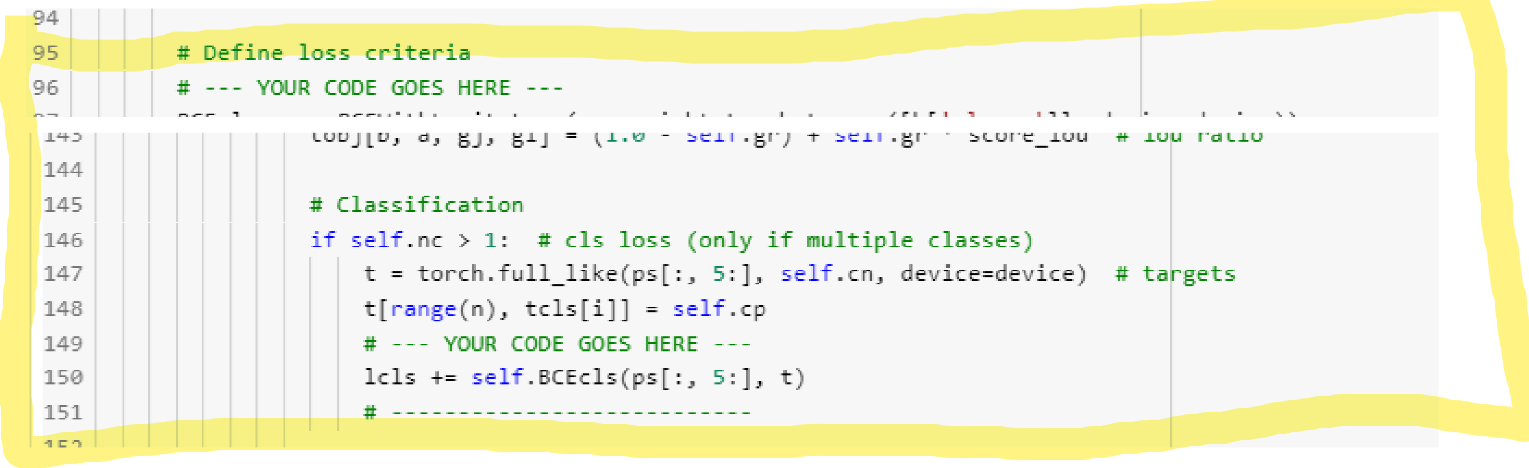
RESTART RUNTIME

Part D.2: Classification Loss

The YOLO model's loss function consists of several components, including a regression loss for the bounding box as well as a classification loss for the object in the bounding box. We will consider only the classification loss here.

For the classification loss, we will work in `utils/loss.py`.

First define in line 97 the `BCEcls` by calling `nn.BCEWithLogitsLoss` (see PyTorch documentation [here](#) for reference) using `h['cls_pw']` as the positive weight for the classification, and passing in `device` as well. Then, in line 150, add to `lcls` the loss using `self.BCEcls` called on the respective parts of the prediction related to the classification component `(ps[:, 5:])` and the target `t`.



```
    # Define loss criteria
    # --- YOUR CODE GOES HERE ---
```

```
            tobj[b, a, gj, gi] = (1.0 - self.gr) + self.gr * score_iou  # iou ratio

            # Classification
            if self.nc > 1:  # cls loss (only if multiple classes)
                t = torch.full_like(ps[:, 5:], self.cn, device=device)  # targets
                t[range(n), tcls[i]] = self.cp
                # --- YOUR CODE GOES HERE ---
                lcls += self.BCEcls(ps[:, 5:], t)
                # ---------------------------
```

Training

Train the YOLOv3 model on COCO128 for 5 epochs, freezing 10 layers, by running the below cell.

You can set up an account for wandb (click on the output area at the flashing cursor where it asks you to enter your choice, and type whatever input number you like, followed by hitting enter), which provides lots of cool visualizations (during training you will see live updates at https://wandb.ai/home). For the purpose of this assignment, you can enter 3 (skipping wandb).

NOTE: This cell below should take around 6 minutes. If it is taking much longer, please double check your work on Part D.1 (freezing the model parameters)

```
!python train.py --img 640 --batch 16 --epochs 5 --data coco128.yaml --weights yolov3.pt --cache --freeze 10
```

```
                bird        128         16      0.965          1      0.995       0.72
                 cat        128          4       0.85          1      0.995       0.90
                 dog        128          9      0.852          1      0.984       0.77
               horse        128          2      0.682          1      0.995       0.79
            elephant        128         17      0.919      0.882       0.94       0.77
                bear        128          1       0.67          1      0.995       0.89
               zebra        128          4      0.873          1      0.995       0.97
             giraffe        128          9      0.984          1      0.995       0.75
            backpack        128          6      0.817      0.667      0.688       0.45
            umbrella        128         18      0.827      0.833      0.882       0.52
             handbag        128         19        0.8      0.421      0.538       0.31
                 tie        128          7       0.94      0.857      0.857       0.62
            suitcase        128          4      0.785          1      0.995       0.72
             frisbee        128          5       0.68        0.8      0.761       0.68
                skis        128          1      0.714          1      0.995       0.69
```

```
           snowboard        128          7      0.964      0.857      0.896       0.66
         sports ball        128          6      0.798      0.833       0.78       0.46
                kite        128         10      0.623      0.664      0.641       0.21
        baseball bat        128          4      0.589       0.75      0.856       0.36
      baseball glove        128          7      0.894      0.714      0.718       0.41
          skateboard        128          5      0.699      0.937      0.898       0.52
       tennis racket        128          7      0.584      0.604       0.67       0.36
              bottle        128         18      0.749      0.829      0.742        0.4
          wine glass        128         16      0.781       0.67      0.821       0.51
                 cup        128         36      0.862       0.87      0.898       0.61
                fork        128          6      0.955        0.5      0.688       0.43
               knife        128         16      0.754      0.768      0.864       0.54
               spoon        128         22      0.721      0.587      0.637       0.44
                bowl        128         28      0.831      0.703      0.779       0.61
              banana        128          1      0.738          1      0.995       0.49
            sandwich        128          2      0.111      0.111      0.448       0.42
              orange        128          4      0.719          1      0.895       0.64
            broccoli        128         11      0.746      0.268      0.426       0.32
              carrot        128         24      0.736      0.708      0.762       0.52
             hot dog        128          2      0.565          1      0.663       0.66
               pizza        128          5        0.8          1      0.995       0.73
               donut        128         14      0.814          1      0.965       0.88
                cake        128          4      0.748          1      0.995       0.87
               chair        128         35       0.73      0.771      0.719       0.42
               couch        128          6          1      0.826      0.942       0.68
        potted plant        128         14      0.822      0.786      0.858       0.58
                 bed        128          3       0.51      0.353      0.585       0.39
        dining table        128         13      0.718      0.615       0.61       0.34
              toilet        128          2      0.648          1      0.995       0.94

                  tv        128          2      0.475          1      0.995       0.92
              laptop        128          3      0.801      0.333      0.665       0.34
               mouse        128          2          1          0      0.398       0.14
              remote        128          8      0.842       0.75      0.812       0.66
          cell phone        128          8      0.677        0.5      0.605       0.32
           microwave        128          3      0.737          1      0.995        0.8
                oven        128          5      0.396        0.6      0.501       0.38
                sink        128          6      0.467        0.5      0.505       0.34
        refrigerator        128          5      0.722        0.8      0.872       0.71
                book        128         29      0.764      0.335      0.475       0.25
               clock        128          9      0.876          1      0.995       0.83
                vase        128          2      0.473          1      0.995       0.94
            scissors        128          1          1          0     0.0149     0.0059
```

Visualization

Set up the visualization by running the below cell. Note that if you ran the training loop multiple times, you would have additional folder exp2, exp3, etc.

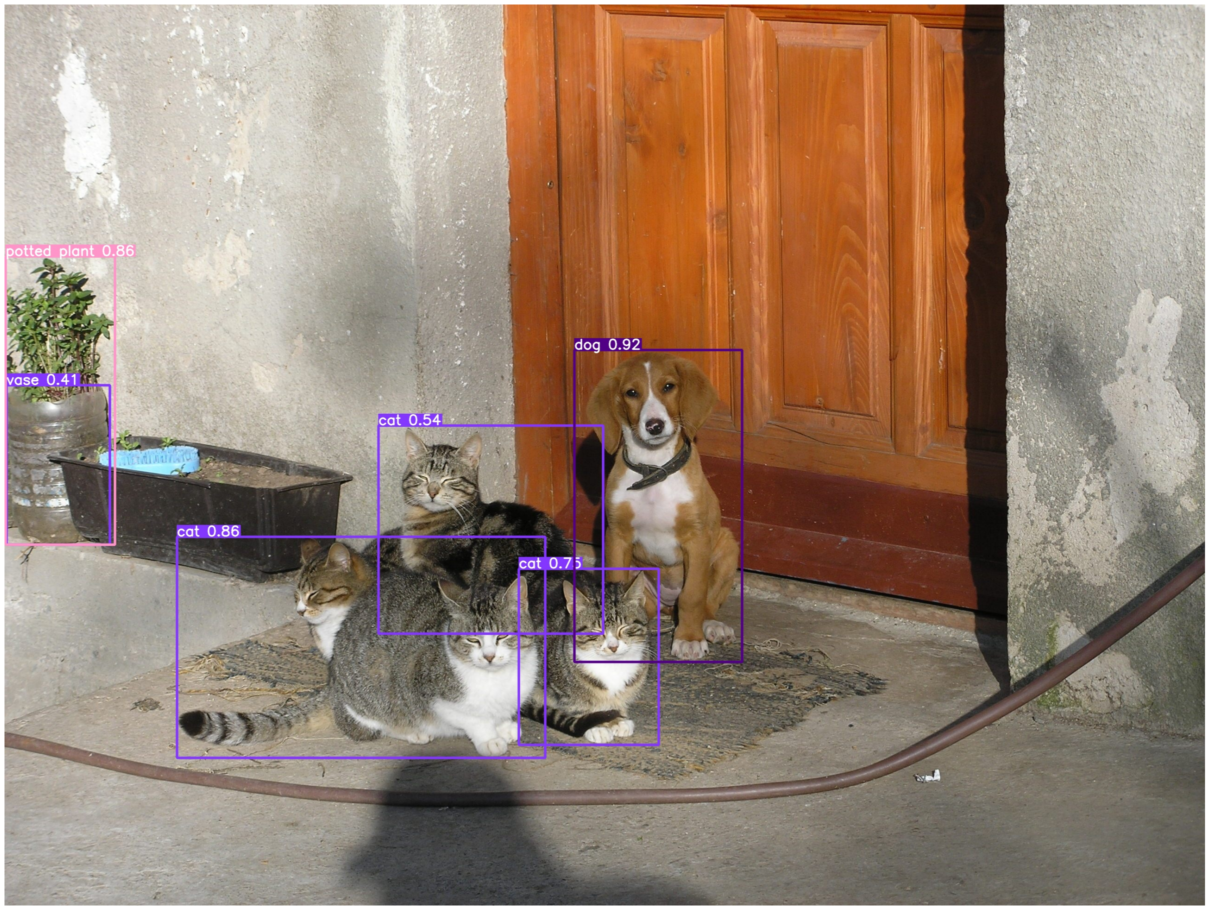
```
!python detect.py --weights runs/train/exp/weights/best.pt --img 640 --conf 0.25 --source data/images
```

```
    detect: weights=['runs/train/exp/weights/best.pt'], source=data/images, imgsz=[640, 640
```

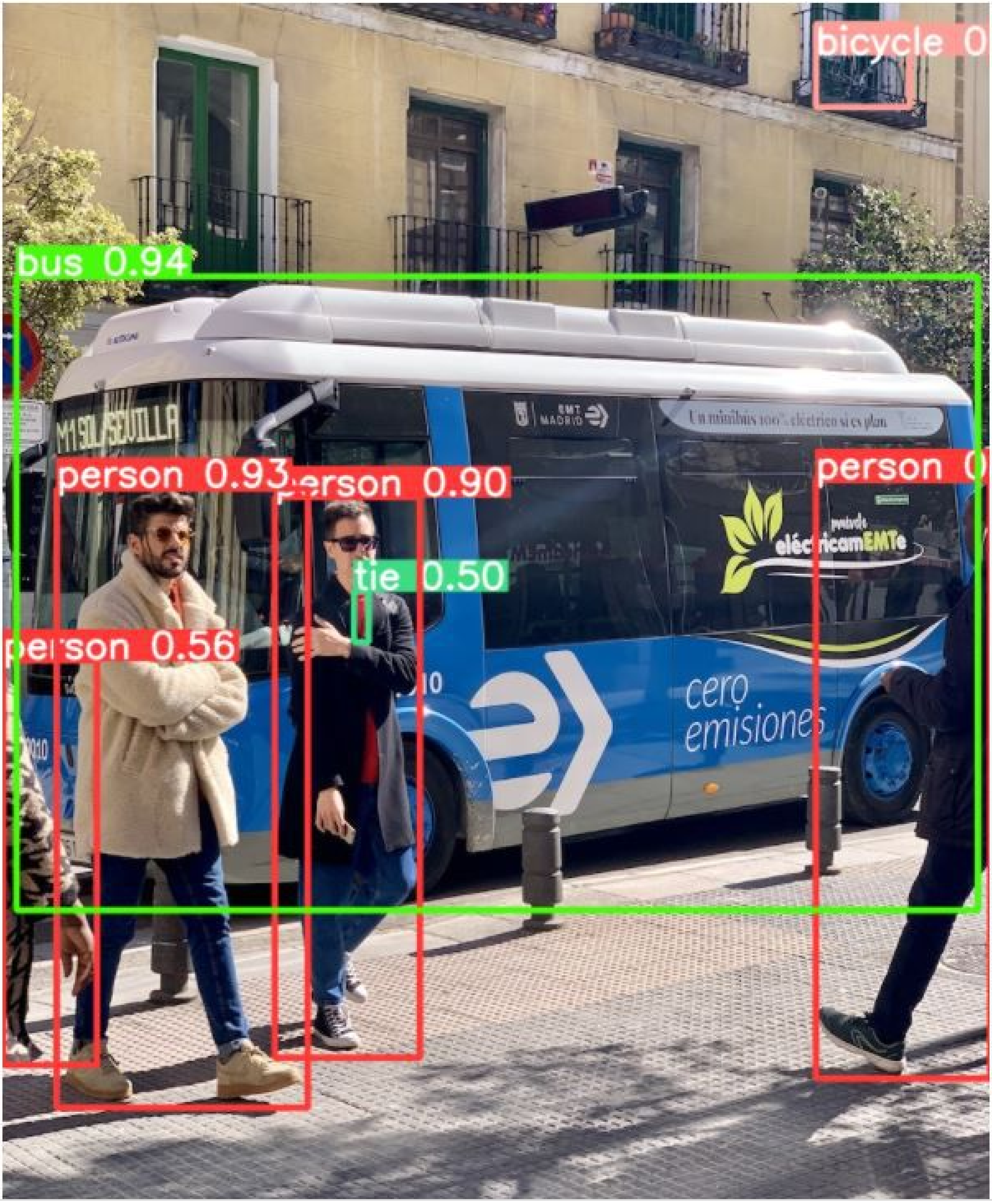
```
YOLOv3 🚀 46dad08 torch 1.10.0+cu111 CUDA:0 (Tesla K80, 11441MiB)

Fusing layers...
Model Summary: 261 layers, 61922845 parameters, 0 gradients, 156.1 GFLOPs
image 1/2 /content/csc413/a2/2022/assets/assignments/pa2-q4-files/data/images/Cats_and_
image 2/2 /content/csc413/a2/2022/assets/assignments/pa2-q4-files/data/images/bus.jpg: 
Speed: 0.5ms pre-process, 144.7ms inference, 1.6ms NMS per image at shape (1, 3, 640, 6
Results saved to runs/detect/exp
```

Display Results

```
display.Image(filename='runs/detect/exp/Cats_and_dog.jpg', width=600)
```



```
display.Image(filename='runs/detect/exp/bus.jpg', width=600)
```


bicycle 0
bus 0.94
person 0.93
person 0.90
person 0
tie 0.50
person 0.56
M1 SOL/SEVILLA
EMT MADRID
eléctricamEMTe
cero emisiones