# 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** → **Change runtime type** → **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.p
y

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
Downloading data from
http://www.cs.toronto.edu/~jba/kmeans_colour_a2.tar.gz
Extracting file.
File path: data/cifar-10-batches-py.tar.gz
Downloading data from http://www.cs.toronto.edu/~kriz/cifar-10-
python.tar.gz
Extracting file.
```

## 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
```

```python
    _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*

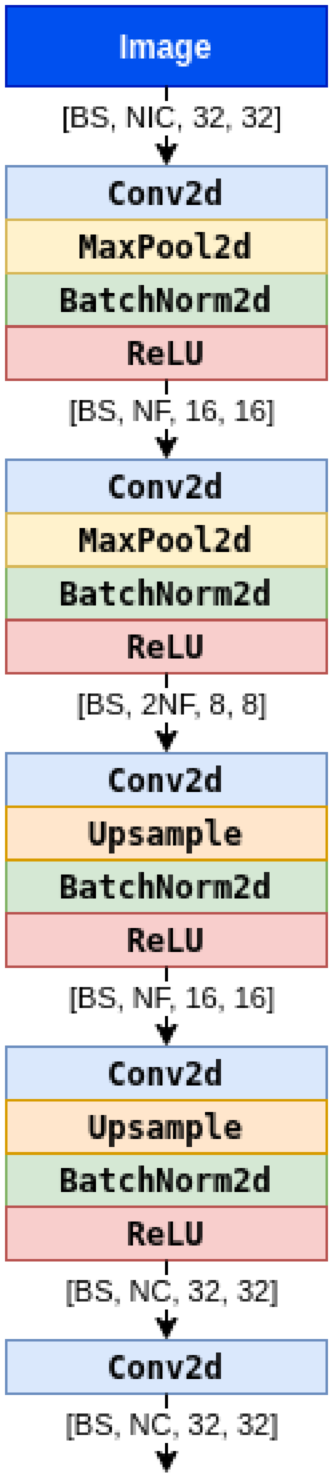
```
class AttrDict(dict):
    def __init__(self, *args, **kwargs):
        super(AttrDict, self).__init__(*args, **kwargs)
        self.__dict__ = self


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.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(num_in_channels, num_filters,
kernel_size=kernel, padding = padding),
            torch.nn.MaxPool2d(kernel_size = 2),
            torch.nn.BatchNorm2d(num_filters),
            torch.nn.ReLU()
            )

        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(num_filters, 2 * num_filters,
kernel_size=kernel, padding = padding),
            torch.nn.MaxPool2d(kernel_size = 2),
            torch.nn.BatchNorm2d(2 * num_filters),
            torch.nn.ReLU()
            )

        self.layer3 = torch.nn.Sequential(
            torch.nn.Conv2d(2 * num_filters, num_filters,
kernel_size=kernel, padding = padding),
            torch.nn.Upsample(scale_factor = 2),
            torch.nn.BatchNorm2d(num_filters),
            torch.nn.ReLU()
            )

        self.layer4 = torch.nn.Sequential(
            torch.nn.Conv2d(num_filters, num_colours,
kernel_size=kernel, padding = padding),
            torch.nn.Upsample(scale_factor = 2),
            torch.nn.BatchNorm2d(num_colours),
            torch.nn.ReLU()
            )

        self.layer5 = torch.nn.Conv2d(num_colours, num_colours,
kernel_size=kernel, padding = padding)


    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        ###################################################
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.layer5(x)
        return x
```

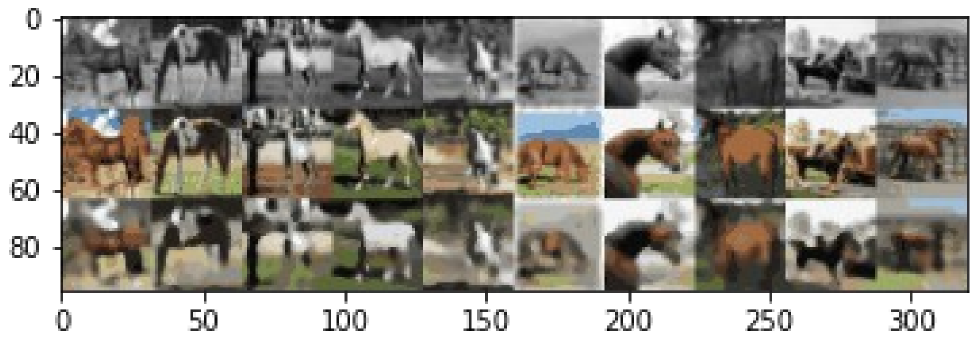
## Question 2

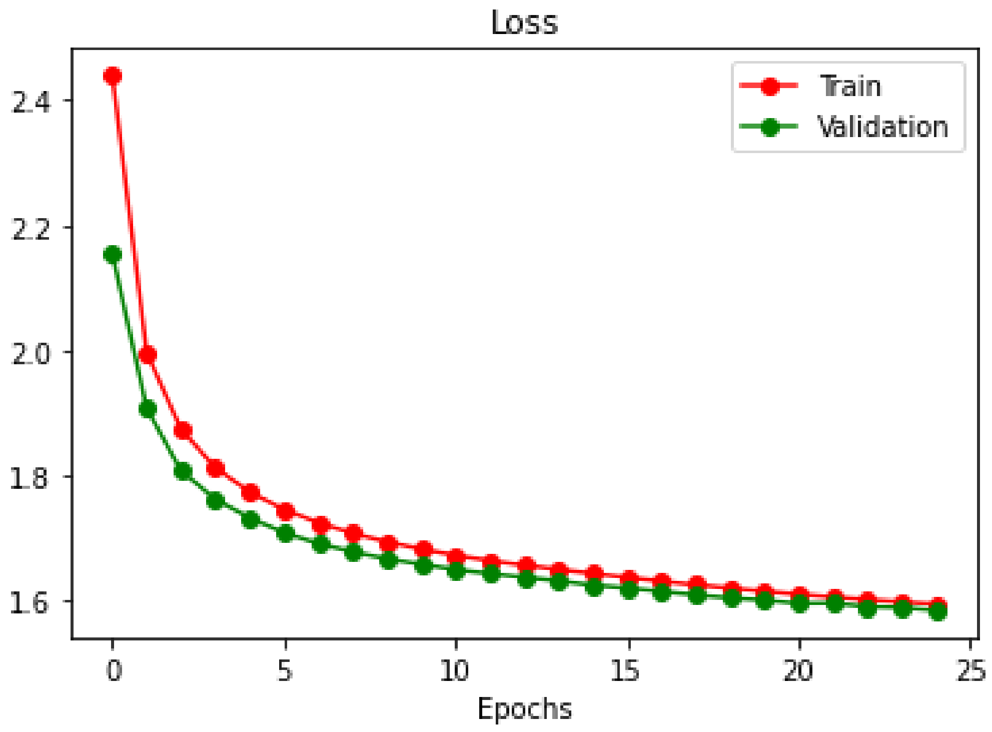
Run main training loop of `PoolUpsampleNet`. This will train the CNN for a few epochs using the cross-entropy objective. It will generate some images showing the trained result at the end. Do these results look good to you? Why or why not?

```
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: 2.4397, Time (s): 2
Epoch [1/25], Val Loss: 2.1542, Val Acc: 26.3%, Time(s): 2.50
Epoch [2/25], Loss: 1.9967, Time (s): 4
Epoch [2/25], Val Loss: 1.9079, Val Acc: 33.0%, Time(s): 4.93
Epoch [3/25], Loss: 1.8744, Time (s): 7
Epoch [3/25], Val Loss: 1.8081, Val Acc: 35.8%, Time(s): 7.40
Epoch [4/25], Loss: 1.8130, Time (s): 9
Epoch [4/25], Val Loss: 1.7627, Val Acc: 36.9%, Time(s): 9.92
Epoch [5/25], Loss: 1.7743, Time (s): 12
Epoch [5/25], Val Loss: 1.7331, Val Acc: 37.6%, Time(s): 12.48
Epoch [6/25], Loss: 1.7465, Time (s): 14
Epoch [6/25], Val Loss: 1.7094, Val Acc: 38.2%, Time(s): 15.08
Epoch [7/25], Loss: 1.7253, Time (s): 17
Epoch [7/25], Val Loss: 1.6919, Val Acc: 38.7%, Time(s): 17.72
Epoch [8/25], Loss: 1.7084, Time (s): 19
Epoch [8/25], Val Loss: 1.6786, Val Acc: 39.0%, Time(s): 20.42
Epoch [9/25], Loss: 1.6947, Time (s): 22
Epoch [9/25], Val Loss: 1.6678, Val Acc: 39.2%, Time(s): 23.17
Epoch [10/25], Loss: 1.6831, Time (s): 25
Epoch [10/25], Val Loss: 1.6588, Val Acc: 39.4%, Time(s): 25.94
Epoch [11/25], Loss: 1.6733, Time (s): 28
Epoch [11/25], Val Loss: 1.6504, Val Acc: 39.6%, Time(s): 28.88
Epoch [12/25], Loss: 1.6647, Time (s): 31
Epoch [12/25], Val Loss: 1.6439, Val Acc: 39.8%, Time(s): 32.13
Epoch [13/25], Loss: 1.6573, Time (s): 34
Epoch [13/25], Val Loss: 1.6380, Val Acc: 40.0%, Time(s): 35.06
Epoch [14/25], Loss: 1.6506, Time (s): 37
Epoch [14/25], Val Loss: 1.6325, Val Acc: 40.1%, Time(s): 38.03
Epoch [15/25], Loss: 1.6441, Time (s): 40
Epoch [15/25], Val Loss: 1.6254, Val Acc: 40.2%, Time(s): 41.04
Epoch [16/25], Loss: 1.6379, Time (s): 43
Epoch [16/25], Val Loss: 1.6209, Val Acc: 40.4%, Time(s): 44.11
Epoch [17/25], Loss: 1.6318, Time (s): 46
Epoch [17/25], Val Loss: 1.6153, Val Acc: 40.5%, Time(s): 47.23
Epoch [18/25], Loss: 1.6260, Time (s): 49
Epoch [18/25], Val Loss: 1.6099, Val Acc: 40.6%, Time(s): 50.36
Epoch [19/25], Loss: 1.6207, Time (s): 52
Epoch [19/25], Val Loss: 1.6052, Val Acc: 40.8%, Time(s): 53.55
Epoch [20/25], Loss: 1.6156, Time (s): 56
Epoch [20/25], Val Loss: 1.6016, Val Acc: 40.9%, Time(s): 56.78
Epoch [21/25], Loss: 1.6108, Time (s): 59
Epoch [21/25], Val Loss: 1.5963, Val Acc: 41.0%, Time(s): 60.36
Epoch [22/25], Loss: 1.6065, Time (s): 62
Epoch [22/25], Val Loss: 1.5967, Val Acc: 40.9%, Time(s): 63.67
Epoch [23/25], Loss: 1.6025, Time (s): 66
Epoch [23/25], Val Loss: 1.5912, Val Acc: 41.1%, Time(s): 67.01
Epoch [24/25], Loss: 1.5988, Time (s): 69
Epoch [24/25], Val Loss: 1.5898, Val Acc: 41.1%, Time(s): 70.59
Epoch [25/25], Loss: 1.5952, Time (s): 73
Epoch [25/25], Val Loss: 1.5852, Val Acc: 41.2%, Time(s): 74.29
```





## Question 3

*Original weight input dimension (width/height)*

**Number of Weights** in 5 convolution layers in their respective order is $k^2 * \left[ NIC * NF + 2 * NF^2 + 2 * NF^2 + NF * NC + NC^2 \right]$

$= k^2 * \left[ NIC * NF + 4 NF^2 + NF * NC + NC^2 \right]$

Number of weights in max pooling layers is 0 as there are no trainable parameters for retrieving maximum. Number of weights in upsampling layers is also 0 as it is used to double the dimension.

Therefore, the total number of weights in the model is $k^2 * [NIC * NF + 4NF^2 + NF * NC + NC^2]$

**Number of Outputs**

The outputs from 4 maxpooling/upscaling layers are $16^2 NF$, $2(8^2) NF$, $16^2 NF$, and $32^2 NC$ respectively.

The outputs from the convolution layers are $32^2 NF$, $(2) 16^2 NF$,$2(8^2) NF$, $16^2 NC$ and $32^2 NC$

**Number of Connections**

Number of connections between the first convolution layer and the first maxpooling layer is $(32k)^2 NIC * NF$. Number of connections between the first maxpooling layer and ReLu is $(16)^2 * NF$.

Number of connections between the second convolution layer and the second maxpooling layer is $(16k)^2 (2) NF^2$. Number of connections between the second maxpooling layer and ReLu is $(8)^2 * (2) NF$.

Number of connections between the third convolution layer and the first upsampling layer is $(8k)^2 (2) NF^2$. Number of connections between the first upsampling layer and ReLu is $(16)^2 * NF$.

Number of connections between the fourth convolution layer and the second upsampling layer is $(16k)^2 NF * NC$. Number of connections between the first upsampling layer and ReLu is $(32)^2 * NC$.

Number of connections between the last ReLu and the fifth convolution layer is $(32k * NC)^2$

*Each weight input dimension is doubled*

**Number of Weights** remains the same as the original weight input as the number of weights for convolutional layers does not depend on width or height. Therefore, the total number of weights in the model is still $k^2 * [NIC * NF + 4NF^2 + NF * NC + NC^2]$

In terms of **Number of Outputs**, as both width and height are doubled, the number of outputs will be 4 times the number of output of the original dimension. Therefore, the outputs from 4 maxpooling/upscaling layers are $(4) 16^2 NF$, $(8^3) NF$, $(4) 16^2 NF$, and $(4) 32^2 NC$ respectively.

The outputs from the convolution layers are $(4) 32^2 NF$, $(8) 16^2 NF$,$(8^3) NF$, $(4) 16^2 NC$ and $(4) 32^2 NC$

Similarly, the **Number of Connections** will be 4 times the original dimension.

Thus, number of connections between the first convolution layer and the first maxpooling layer is $(4)(32k)^2 NIC*NF$. Number of connections between the first maxpooling layer and ReLu is $(4)(16)^2*NF$.

Number of connections between the second convolution layer and the second maxpooling layer is $(4)(16k)^2(2)NF^2$. Number of connections between the second maxpooling layer and ReLu is $(8)^3 NF$.

Number of connections between the third convolution layer and the first upsampling layer is $(4)(8k)^2(2)NF^2$. Number of connections between the first upsampling layer and ReLu is $(4)(16)^2*NF$.

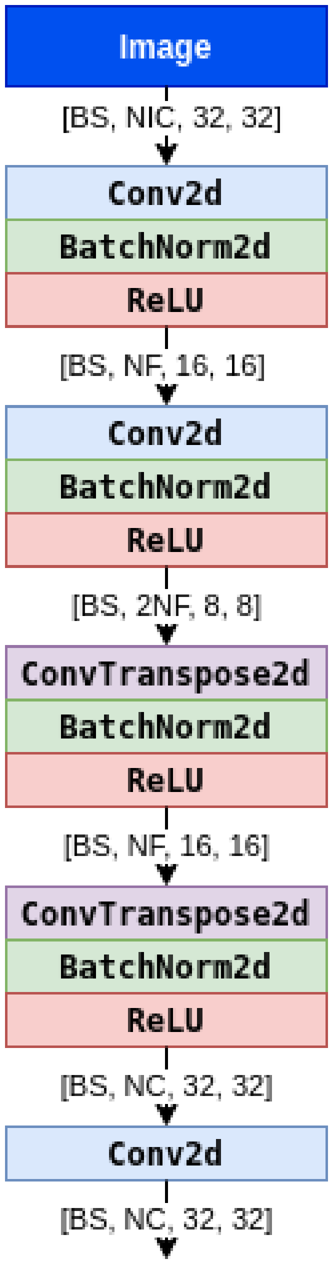
Number of connections between the fourth convolution layer and the second upsampling layer is $(4)(16k)^2 NF*NC$. Number of connections between the first upsampling layer and ReLu is $(4)(32)^2*NC$.

Number of connections between the last ReLu and the fifth convolution layer is $(4)(32k*NC)^2$

## 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`

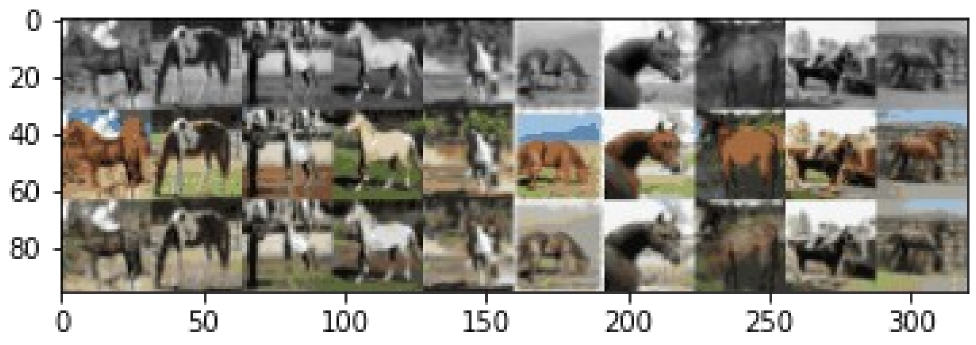
```
class ConvTransposeNet(nn.Module):
    def __init__(self, kernel, num_filters, num_colours,
num_in_channels):
        super().__init__()

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

        ############### YOUR CODE GOES HERE ###############
        ###################################################

        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(num_in_channels, num_filters,
kernel_size=kernel, padding = 1, stride = 2),
            torch.nn.BatchNorm2d(num_filters),
            torch.nn.ReLU()
            )

        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(num_filters, 2 * num_filters,
kernel_size=kernel, padding = 1, stride = 2),
            torch.nn.BatchNorm2d(2 * num_filters),
            torch.nn.ReLU()
            )

        self.layer3 = torch.nn.Sequential(
            torch.nn.ConvTranspose2d(2 * num_filters, num_filters,
kernel_size = kernel, stride = 2, dilation = 1, padding = 1,
output_padding = 1),
            torch.nn.BatchNorm2d(num_filters),
            torch.nn.ReLU()
            )

        self.layer4 = torch.nn.Sequential(
            torch.nn.ConvTranspose2d(num_filters, num_colours,
kernel_size = kernel, stride = 2, dilation = 1, padding = 1,
output_padding = 1),
            torch.nn.BatchNorm2d(num_colours),
            torch.nn.ReLU()
            )

        self.layer5 = torch.nn.Conv2d(num_colours, num_colours,
kernel_size = kernel, padding = padding)


    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        ###################################################
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)
        x = self.layer5(x)
        return x
```

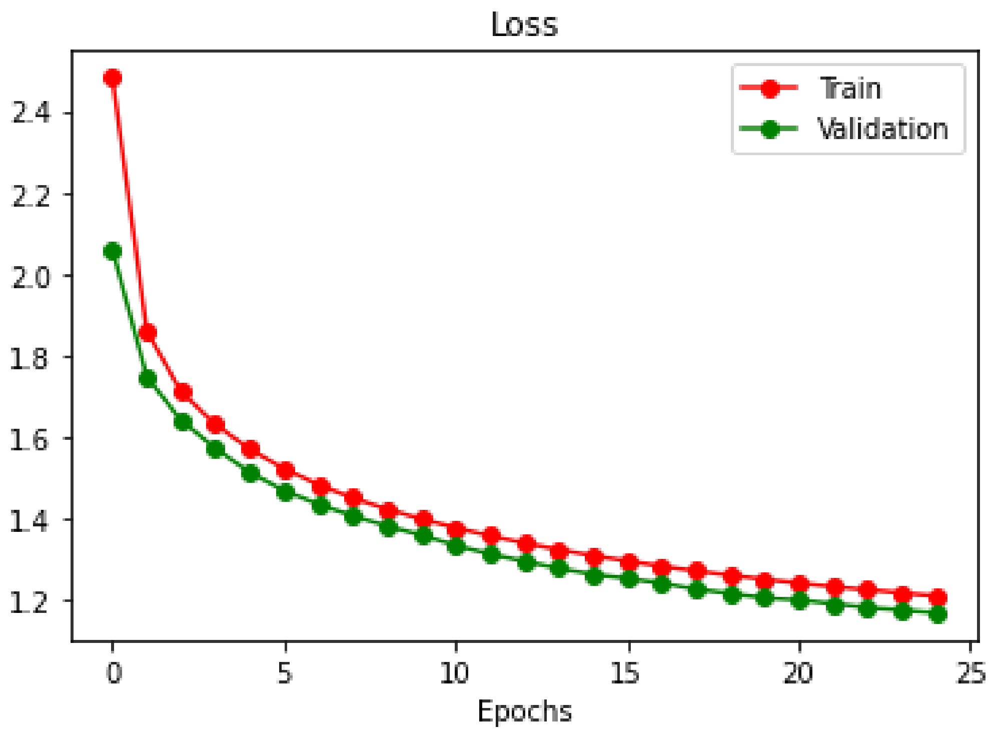
### Question 2

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: 2.4820, Time (s): 2
Epoch [1/25], Val Loss: 2.0605, Val Acc: 30.9%, Time(s): 2.32
Epoch [2/25], Loss: 1.8601, Time (s): 4
Epoch [2/25], Val Loss: 1.7481, Val Acc: 37.6%, Time(s): 4.51
Epoch [3/25], Loss: 1.7111, Time (s): 6
Epoch [3/25], Val Loss: 1.6406, Val Acc: 40.3%, Time(s): 6.77
Epoch [4/25], Loss: 1.6305, Time (s): 8
Epoch [4/25], Val Loss: 1.5723, Val Acc: 42.0%, Time(s): 9.06
Epoch [5/25], Loss: 1.5703, Time (s): 11
Epoch [5/25], Val Loss: 1.5130, Val Acc: 43.9%, Time(s): 11.41
Epoch [6/25], Loss: 1.5221, Time (s): 13
Epoch [6/25], Val Loss: 1.4685, Val Acc: 45.2%, Time(s): 13.78
Epoch [7/25], Loss: 1.4828, Time (s): 15
Epoch [7/25], Val Loss: 1.4350, Val Acc: 46.1%, Time(s): 16.20
Epoch [8/25], Loss: 1.4506, Time (s): 18
Epoch [8/25], Val Loss: 1.4063, Val Acc: 46.9%, Time(s): 18.68
Epoch [9/25], Loss: 1.4230, Time (s): 20
Epoch [9/25], Val Loss: 1.3814, Val Acc: 47.6%, Time(s): 21.20
Epoch [10/25], Loss: 1.3986, Time (s): 23
Epoch [10/25], Val Loss: 1.3593, Val Acc: 48.3%, Time(s): 23.78
Epoch [11/25], Loss: 1.3771, Time (s): 25
Epoch [11/25], Val Loss: 1.3340, Val Acc: 49.2%, Time(s): 26.40
Epoch [12/25], Loss: 1.3575, Time (s): 28
Epoch [12/25], Val Loss: 1.3128, Val Acc: 49.9%, Time(s): 29.06
Epoch [13/25], Loss: 1.3398, Time (s): 31
Epoch [13/25], Val Loss: 1.2947, Val Acc: 50.5%, Time(s): 31.77
Epoch [14/25], Loss: 1.3236, Time (s): 33
Epoch [14/25], Val Loss: 1.2778, Val Acc: 51.0%, Time(s): 34.50
Epoch [15/25], Loss: 1.3089, Time (s): 36
Epoch [15/25], Val Loss: 1.2631, Val Acc: 51.4%, Time(s): 37.28
Epoch [16/25], Loss: 1.2954, Time (s): 39
Epoch [16/25], Val Loss: 1.2527, Val Acc: 51.8%, Time(s): 40.11
Epoch [17/25], Loss: 1.2829, Time (s): 42
Epoch [17/25], Val Loss: 1.2416, Val Acc: 52.1%, Time(s): 42.94
Epoch [18/25], Loss: 1.2714, Time (s): 45
Epoch [18/25], Val Loss: 1.2262, Val Acc: 52.6%, Time(s): 45.82
Epoch [19/25], Loss: 1.2607, Time (s): 48
Epoch [19/25], Val Loss: 1.2159, Val Acc: 52.9%, Time(s): 48.76
Epoch [20/25], Loss: 1.2507, Time (s): 51
Epoch [20/25], Val Loss: 1.2053, Val Acc: 53.2%, Time(s): 51.73
Epoch [21/25], Loss: 1.2415, Time (s): 54
Epoch [21/25], Val Loss: 1.1999, Val Acc: 53.4%, Time(s): 54.76
Epoch [22/25], Loss: 1.2328, Time (s): 57
Epoch [22/25], Val Loss: 1.1906, Val Acc: 53.7%, Time(s): 57.83
Epoch [23/25], Loss: 1.2247, Time (s): 60
Epoch [23/25], Val Loss: 1.1812, Val Acc: 54.0%, Time(s): 60.96
Epoch [24/25], Loss: 1.2172, Time (s): 63
Epoch [24/25], Val Loss: 1.1754, Val Acc: 54.2%, Time(s): 64.15
Epoch [25/25], Loss: 1.2101, Time (s): 66
Epoch [25/25], Val Loss: 1.1684, Val Acc: 54.4%, Time(s): 67.42
```





## Question 3

ConvTransposeNet has lower validaton loss, higher validation accuracy and shorter computational time than PoolUpsampleNet. It has a higher accuracy as it is built up from a low resolution image to an image with higher resolution. Therefore, after training, the network has the ability to identify images with rough resolution and modifies them to have higher resolution for prediction and/or classification. These steps allow the model to make better predictions, which results in lower loss.

## Question 4

The padding parameter for the convolution layers will increase. As we increase the kernel size, the output layer's receptive field is larger. Therefore, the output layer would have a smaller size. Therefore, if we want to maintain the output size, padding needs to be increased.

### Question 5

When batch size is 100, the validation loss is 1.1566, the validation accuracy is 54.9% and it takes about 70 seconds to train.

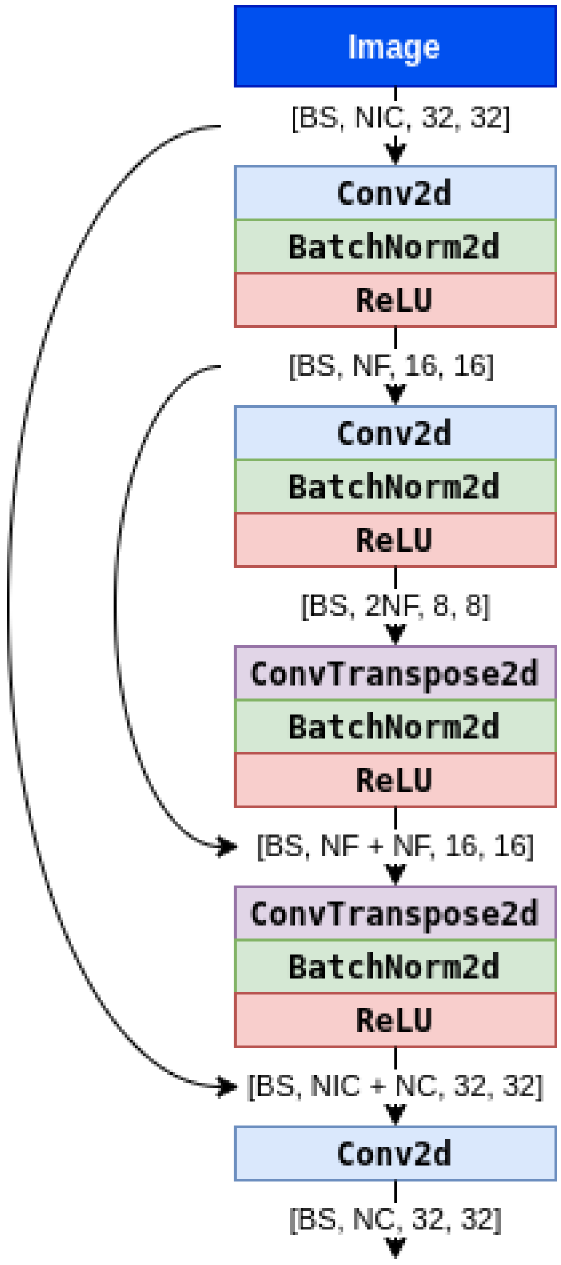
When batch size is 32, validation loss decreases and validation accuracy rises to 56.9% but it takes 5 more seconds for the program to execute. With a smaller batch size, the predicted pixel can closely resemble the color of the original image. This means the predicted image is very similar to the original RGB image.

On the other hand, if batch size is 128, the validation loss is higher and the validation accuracy drops to 53.4%. However, the training finished in 66 seconds. With larger batch size, the predicted pixel is more similar to the black and white image and the image is less colourful.

## 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`.

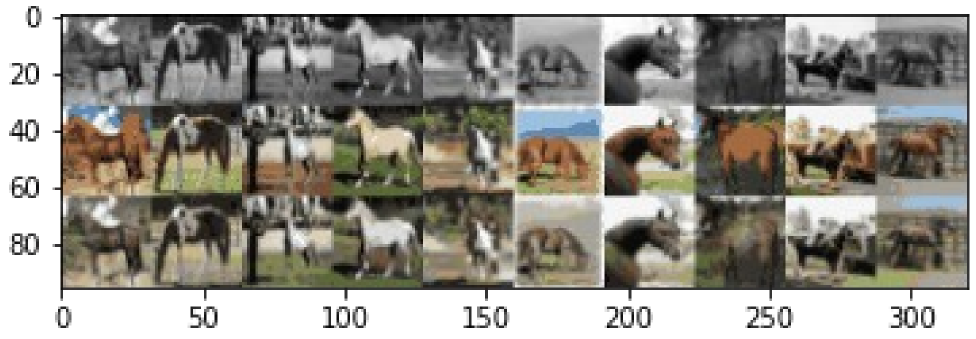
```
class UNet(nn.Module):
    def __init__(self, kernel, num_filters, num_colours,
num_in_channels):
        super().__init__()

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

        ############### YOUR CODE GOES HERE ###############
        ###################################################

        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(num_in_channels, num_filters,
kernel_size=kernel, padding = 1, stride = 2),
            torch.nn.BatchNorm2d(num_filters),
            torch.nn.ReLU()
            )

        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(num_filters, 2 * num_filters,
kernel_size=kernel, padding = 1, stride = 2),
            torch.nn.BatchNorm2d(2 * num_filters),
            torch.nn.ReLU()
            )

        self.layer3 = torch.nn.Sequential(
            torch.nn.ConvTranspose2d(2 * num_filters, num_filters,
kernel_size = kernel, stride = 2, dilation = 1, padding = 1,
output_padding = 1),
            torch.nn.BatchNorm2d(num_filters),
            torch.nn.ReLU()
            )

        # the input dim. is 2 * num_filters because layer 3_prime has
2 * num_filters input units
        self.layer4 = torch.nn.Sequential(
            torch.nn.ConvTranspose2d(2 * num_filters, num_colours,
kernel_size = kernel, stride = stride, dilation = 1, padding = 1,
output_padding = 1),
            torch.nn.BatchNorm2d(num_colours),
            torch.nn.ReLU()
            )

        self.layer5 = torch.nn.Conv2d(num_colours, num_colours,
kernel_size = kernel, padding = padding)

    def forward(self, x):
        ############### YOUR CODE GOES HERE ###############
        ###################################################
        x1 = self.layer1(x)
        x2 = self.layer2(x1)
        x3 = self.layer3(x2)
        x3_prime = torch.cat((x1, x3), 1)
        x4 = self.layer4(x3_prime)
        x4_prime = torch.cat((x, x4), 1)
        x5 = self.layer5(x4)
        return x5
```

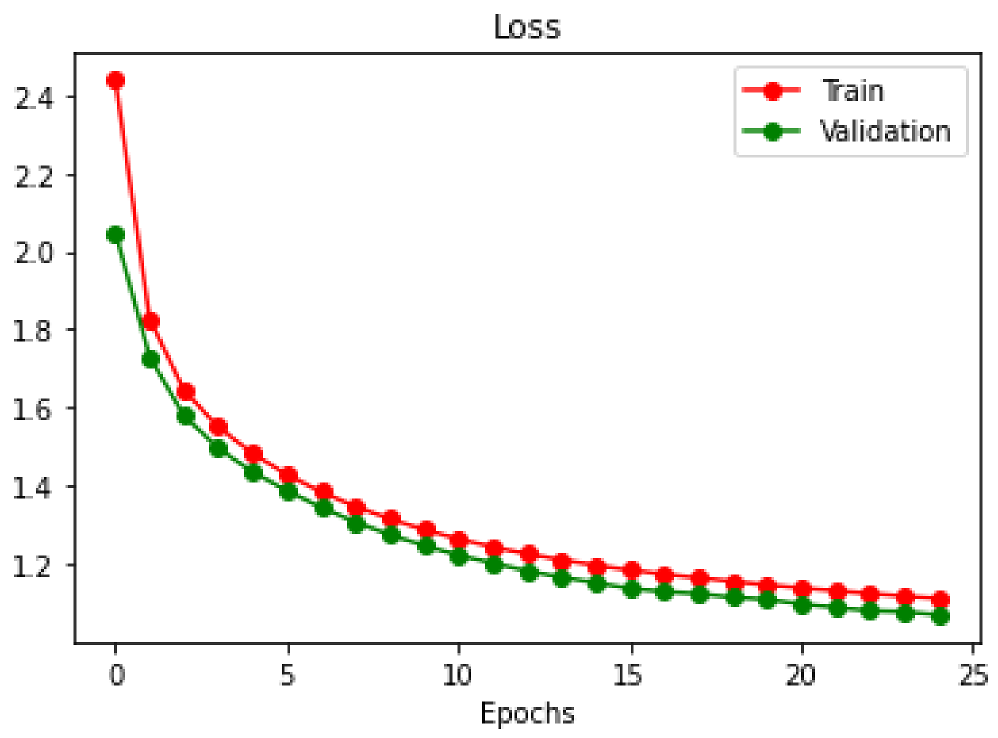
## Question 2

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: 2.4398, Time (s): 2
Epoch [1/25], Val Loss: 2.0491, Val Acc: 32.6%, Time(s): 2.43
Epoch [2/25], Loss: 1.8276, Time (s): 4
Epoch [2/25], Val Loss: 1.7302, Val Acc: 37.4%, Time(s): 4.75
Epoch [3/25], Loss: 1.6447, Time (s): 6
Epoch [3/25], Val Loss: 1.5793, Val Acc: 42.2%, Time(s): 7.13
Epoch [4/25], Loss: 1.5504, Time (s): 9
Epoch [4/25], Val Loss: 1.4961, Val Acc: 44.6%, Time(s): 9.57
Epoch [5/25], Loss: 1.4825, Time (s): 11
Epoch [5/25], Val Loss: 1.4349, Val Acc: 46.5%, Time(s): 12.03
Epoch [6/25], Loss: 1.4283, Time (s): 14
Epoch [6/25], Val Loss: 1.3872, Val Acc: 47.8%, Time(s): 14.52
Epoch [7/25], Loss: 1.3839, Time (s): 16
Epoch [7/25], Val Loss: 1.3441, Val Acc: 49.2%, Time(s): 17.07
Epoch [8/25], Loss: 1.3466, Time (s): 19
Epoch [8/25], Val Loss: 1.3065, Val Acc: 50.5%, Time(s): 19.66
Epoch [9/25], Loss: 1.3146, Time (s): 21
Epoch [9/25], Val Loss: 1.2743, Val Acc: 51.5%, Time(s): 22.30
Epoch [10/25], Loss: 1.2871, Time (s): 24
Epoch [10/25], Val Loss: 1.2469, Val Acc: 52.5%, Time(s): 25.29
Epoch [11/25], Loss: 1.2635, Time (s): 27
Epoch [11/25], Val Loss: 1.2219, Val Acc: 53.3%, Time(s): 27.99
Epoch [12/25], Loss: 1.2431, Time (s): 30
Epoch [12/25], Val Loss: 1.2013, Val Acc: 54.0%, Time(s): 30.76
Epoch [13/25], Loss: 1.2255, Time (s): 33
Epoch [13/25], Val Loss: 1.1821, Val Acc: 54.6%, Time(s): 33.58
Epoch [14/25], Loss: 1.2098, Time (s): 35
Epoch [14/25], Val Loss: 1.1649, Val Acc: 55.2%, Time(s): 36.45
Epoch [15/25], Loss: 1.1962, Time (s): 38
Epoch [15/25], Val Loss: 1.1514, Val Acc: 55.5%, Time(s): 39.35
Epoch [16/25], Loss: 1.1838, Time (s): 41
Epoch [16/25], Val Loss: 1.1365, Val Acc: 56.0%, Time(s): 42.28
Epoch [17/25], Loss: 1.1730, Time (s): 44
Epoch [17/25], Val Loss: 1.1282, Val Acc: 56.2%, Time(s): 45.27
Epoch [18/25], Loss: 1.1637, Time (s): 47
Epoch [18/25], Val Loss: 1.1232, Val Acc: 56.3%, Time(s): 48.30
Epoch [19/25], Loss: 1.1544, Time (s): 50
Epoch [19/25], Val Loss: 1.1140, Val Acc: 56.6%, Time(s): 51.38
Epoch [20/25], Loss: 1.1458, Time (s): 53
Epoch [20/25], Val Loss: 1.1081, Val Acc: 56.8%, Time(s): 54.49
Epoch [21/25], Loss: 1.1379, Time (s): 56
Epoch [21/25], Val Loss: 1.0965, Val Acc: 57.2%, Time(s): 57.62
Epoch [22/25], Loss: 1.1305, Time (s): 60
Epoch [22/25], Val Loss: 1.0877, Val Acc: 57.5%, Time(s): 61.32
Epoch [23/25], Loss: 1.1234, Time (s): 63
Epoch [23/25], Val Loss: 1.0795, Val Acc: 57.8%, Time(s): 64.55
Epoch [24/25], Loss: 1.1170, Time (s): 67
Epoch [24/25], Val Loss: 1.0770, Val Acc: 57.8%, Time(s): 67.83
Epoch [25/25], Loss: 1.1110, Time (s): 70
Epoch [25/25], Val Loss: 1.0689, Val Acc: 58.0%, Time(s): 71.19
```





### Question 3

The skip connection model performs better than the previous 2 models. The validation loss is lower, the validation accuracy is higher but the training time is about the same as the other 2 models. The predictive image is also more colourful than the previous 2 networks.

The skip connection model merges information from previous layers with the current layer. This recovers some information lost during downsampling. This ensures the flow between each input and output layer is maintained at a maximum level. On the other hand, reusing previous features also stabilize training and convergence.

## 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
```

```
[Errno 2] No such file or directory: '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-packages (from -r requirements.txt (line
4)) (3.2.2)
Requirement already satisfied: numpy>=1.18.5 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
5)) (1.21.5)
```

```
Requirement already satisfied: opencv-python>=4.1.2 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
6)) (4.1.2.30)
Requirement already satisfied: Pillow>=7.1.2 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
7)) (7.1.2)
Requirement already satisfied: PyYAML>=5.3.1 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
8)) (6.0)
Requirement already satisfied: requests>=2.23.0 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
9)) (2.23.0)
Requirement already satisfied: scipy>=1.4.1 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
10)) (1.4.1)
Requirement already satisfied: torch>=1.7.0 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
11)) (1.10.0+cu111)
Requirement already satisfied: torchvision>=0.8.1 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
12)) (0.11.1+cu111)
Requirement already satisfied: tqdm>=4.41.0 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
13)) (4.62.3)
Requirement already satisfied: tensorboard>=2.4.1 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
16)) (2.8.0)
Requirement already satisfied: wandb in /usr/local/lib/python3.7/dist-
packages (from -r requirements.txt (line 17)) (0.12.10)
Requirement already satisfied: pandas>=1.1.4 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
20)) (1.3.5)
Requirement already satisfied: seaborn>=0.11.0 in
/usr/local/lib/python3.7/dist-packages (from -r requirements.txt (line
21)) (0.11.2)
Requirement already satisfied: thop in /usr/local/lib/python3.7/dist-
packages (from -r requirements.txt (line 36)) (0.0.31.post2005241907)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.2->-r
requirements.txt (line 4)) (0.11.0)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.2->-r
requirements.txt (line 4)) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!
=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from
matplotlib>=3.2.2->-r requirements.txt (line 4)) (3.0.7)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.7/dist-packages (from matplotlib>=3.2.2->-r
requirements.txt (line 4)) (1.3.2)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1
```

```
in /usr/local/lib/python3.7/dist-packages (from requests>=2.23.0->-r
requirements.txt (line 9)) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in
/usr/local/lib/python3.7/dist-packages (from requests>=2.23.0->-r
requirements.txt (line 9)) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests>=2.23.0->-r
requirements.txt (line 9)) (2021.10.8)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests>=2.23.0->-r
requirements.txt (line 9)) (3.0.4)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.7/dist-packages (from torch>=1.7.0->-r
requirements.txt (line 11)) (3.10.0.2)
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in
/usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r
requirements.txt (line 16)) (0.4.6)
Requirement already satisfied: setuptools>=41.0.0 in
/usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r
requirements.txt (line 16)) (57.4.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in
/usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r
requirements.txt (line 16)) (1.8.1)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r
requirements.txt (line 16)) (3.3.6)
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0
in /usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r
requirements.txt (line 16)) (0.6.1)
Requirement already satisfied: protobuf>=3.6.0 in
/usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r
requirements.txt (line 16)) (3.17.3)
Requirement already satisfied: werkzeug>=0.11.15 in
/usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r
requirements.txt (line 16)) (1.0.1)
Requirement already satisfied: wheel>=0.26 in
/usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r
requirements.txt (line 16)) (0.37.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r
requirements.txt (line 16)) (1.35.0)
Requirement already satisfied: grpcio>=1.24.3 in
/usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r
requirements.txt (line 16)) (1.43.0)
Requirement already satisfied: absl-py>=0.4 in
/usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4.1->-r
requirements.txt (line 16)) (1.0.0)
Requirement already satisfied: pytz>=2017.3 in
/usr/local/lib/python3.7/dist-packages (from pandas>=1.1.4->-r
requirements.txt (line 20)) (2018.9)
```

```
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-
packages (from absl-py>=0.4->tensorboard>=2.4.1->-r requirements.txt
(line 16)) (1.15.0)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard>=2.4.1->-r requirements.txt (line 16)) (0.2.8)
Requirement already satisfied: rsa<5,>=3.1.4 in
/usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard>=2.4.1->-r requirements.txt (line 16)) (4.8)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in
/usr/local/lib/python3.7/dist-packages (from google-auth<3,>=1.6.3-
>tensorboard>=2.4.1->-r requirements.txt (line 16)) (4.2.4)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.7/dist-packages (from google-auth-
oauthlib<0.5,>=0.4.1->tensorboard>=2.4.1->-r requirements.txt (line
16)) (1.3.1)
Requirement already satisfied: importlib-metadata>=4.4 in
/usr/local/lib/python3.7/dist-packages (from markdown>=2.6.8-
>tensorboard>=2.4.1->-r requirements.txt (line 16)) (4.11.0)
Requirement already satisfied: zipp>=0.5 in
/usr/local/lib/python3.7/dist-packages (from importlib-metadata>=4.4-
>markdown>=2.6.8->tensorboard>=2.4.1->-r requirements.txt (line 16))
(3.7.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in
/usr/local/lib/python3.7/dist-packages (from pyasn1-modules>=0.2.1-
>google-auth<3,>=1.6.3->tensorboard>=2.4.1->-r requirements.txt (line
16)) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.7/dist-packages (from requests-oauthlib>=0.7.0-
>google-auth-oauthlib<0.5,>=0.4.1->tensorboard>=2.4.1->-r
requirements.txt (line 16)) (3.2.0)
Requirement already satisfied: Click!=8.0.0,>=7.0 in
/usr/local/lib/python3.7/dist-packages (from wandb->-r
requirements.txt (line 17)) (7.1.2)
Requirement already satisfied: pathtools in
/usr/local/lib/python3.7/dist-packages (from wandb->-r
requirements.txt (line 17)) (0.1.2)
Requirement already satisfied: yaspin>=1.0.0 in
/usr/local/lib/python3.7/dist-packages (from wandb->-r
requirements.txt (line 17)) (2.1.0)
Requirement already satisfied: GitPython>=1.0.0 in
/usr/local/lib/python3.7/dist-packages (from wandb->-r
requirements.txt (line 17)) (3.1.26)
Requirement already satisfied: psutil>=5.0.0 in
/usr/local/lib/python3.7/dist-packages (from wandb->-r
requirements.txt (line 17)) (5.4.8)
Requirement already satisfied: sentry-sdk>=1.0.0 in
/usr/local/lib/python3.7/dist-packages (from wandb->-r
requirements.txt (line 17)) (1.5.5)
Requirement already satisfied: promise<3,>=2.0 in
```

```
/usr/local/lib/python3.7/dist-packages (from wandb->-r
requirements.txt (line 17)) (2.3)
Requirement already satisfied: docker-pycreds>=0.4.0 in
/usr/local/lib/python3.7/dist-packages (from wandb->-r
requirements.txt (line 17)) (0.4.0)
Requirement already satisfied: shortuuid>=0.5.0 in
/usr/local/lib/python3.7/dist-packages (from wandb->-r
requirements.txt (line 17)) (1.0.8)
Requirement already satisfied: gitdb<5,>=4.0.1 in
/usr/local/lib/python3.7/dist-packages (from GitPython>=1.0.0->wandb-
>-r requirements.txt (line 17)) (4.0.9)
Requirement already satisfied: smmap<6,>=3.0.1 in
/usr/local/lib/python3.7/dist-packages (from gitdb<5,>=4.0.1-
>GitPython>=1.0.0->wandb->-r requirements.txt (line 17)) (5.0.0)
Requirement already satisfied: termcolor<2.0.0,>=1.1.0 in
/usr/local/lib/python3.7/dist-packages (from yaspin>=1.0.0->wandb->-r
requirements.txt (line 17)) (1.1.0)
```

## 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)`.

## 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`.

## 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
```

```
wandb: (1) Create a W&B account
wandb: (2) Use an existing W&B account
wandb: (3) Don't visualize my results
wandb: Enter your choice: (30 second timeout) 3
wandb: You chose 'Don't visualize my results'
train: weights=yolov3.pt, cfg=, data=coco128.yaml,
hyp=data/hyps/hyp.scratch.yaml, epochs=5, batch_size=16, imgsz=640,
rect=False, resume=False, nosave=False, noval=False,
noautoanchor=False, evolve=None, bucket=, cache=ram,
image_weights=False, device=, multi_scale=False, single_cls=False,
adam=False, sync_bn=False, workers=8, project=runs/train, name=exp,
exist_ok=False, quad=False, linear_lr=False, label_smoothing=0.0,
patience=100, freeze=10, save_period=-1, local_rank=-1, entity=None,
upload_dataset=False, bbox_interval=-1, artifact_alias=latest
github: skipping check (not a git repository), for updates see
https://github.com/ultralytics/yolov3
YOLOv3 □ 46dad08 torch 1.10.0+cu111 CUDA:0 (Tesla K80, 11441MiB)

hyperparameters: lr0=0.01, lrf=0.1, momentum=0.937,
weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8,
warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0, obj=1.0,
obj_pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015,
hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5,
shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0,
mixup=0.0, copy_paste=0.0
Weights & Biases: run 'pip install wandb' to automatically track and
visualize YOLOv3 □ runs (RECOMMENDED)
TensorBoard: Start with 'tensorboard --logdir runs/train', view at
http://localhost:6006/

                 from  n    params  module
arguments
  0                -1  1       928  models.common.Conv
```

```
[3, 32, 3, 1]
  1                -1  1     18560  models.common.Conv
[32, 64, 3, 2]
  2                -1  1     20672  models.common.Bottleneck
[64, 64]
  3                -1  1     73984  models.common.Conv
[64, 128, 3, 2]
  4                -1  2    164608  models.common.Bottleneck
[128, 128]
  5                -1  1    295424  models.common.Conv
[128, 256, 3, 2]
  6                -1  8   2627584  models.common.Bottleneck
[256, 256]
  7                -1  1   1180672  models.common.Conv
[256, 512, 3, 2]
  8                -1  8  10498048  models.common.Bottleneck
[512, 512]
  9                -1  1   4720640  models.common.Conv
[512, 1024, 3, 2]
 10                -1  4  20983808  models.common.Bottleneck
[1024, 1024]
 11                -1  1   5245952  models.common.Bottleneck
[1024, 1024, False]
 12                -1  1    525312  models.common.Conv
[1024, 512, [1, 1]]
 13                -1  1   4720640  models.common.Conv
[512, 1024, 3, 1]
 14                -1  1    525312  models.common.Conv
[1024, 512, 1, 1]
 15                -1  1   4720640  models.common.Conv
[512, 1024, 3, 1]
 16                -2  1    131584  models.common.Conv
[512, 256, 1, 1]
 17                -1  1         0
torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 18           [-1, 8]  1         0  models.common.Concat
[1]
 19                -1  1   1377792  models.common.Bottleneck
[768, 512, False]
 20                -1  1   1312256  models.common.Bottleneck
[512, 512, False]
 21                -1  1    131584  models.common.Conv
[512, 256, 1, 1]
 22                -1  1   1180672  models.common.Conv
[256, 512, 3, 1]
 23                -2  1     33024  models.common.Conv
[256, 128, 1, 1]
 24                -1  1         0
torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 25           [-1, 6]  1         0  models.common.Concat
```

```
[1]
 26                -1  1    344832  models.common.Bottleneck
[384, 256, False]
 27                -1  2    656896  models.common.Bottleneck
[256, 256, False]
 28      [27, 22, 15]  1    457725  models.yolo.Detect
[80, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90,
156, 198, 373, 326]], [256, 512, 1024]]
Model Summary: 333 layers, 61949149 parameters, 61949149 gradients,
156.3 GFLOPs

Transferred 439/439 items from yolov3.pt
freezing model.0.conv.weight
freezing model.0.bn.weight
freezing model.0.bn.bias
freezing model.1.conv.weight
freezing model.1.bn.weight
freezing model.1.bn.bias
freezing model.2.cv1.conv.weight
freezing model.2.cv1.bn.weight
freezing model.2.cv1.bn.bias
freezing model.2.cv2.conv.weight
freezing model.2.cv2.bn.weight
freezing model.2.cv2.bn.bias
freezing model.3.conv.weight
freezing model.3.bn.weight
freezing model.3.bn.bias
freezing model.4.0.cv1.conv.weight
freezing model.4.0.cv1.bn.weight
freezing model.4.0.cv1.bn.bias
freezing model.4.0.cv2.conv.weight
freezing model.4.0.cv2.bn.weight
freezing model.4.0.cv2.bn.bias
freezing model.4.1.cv1.conv.weight
freezing model.4.1.cv1.bn.weight
freezing model.4.1.cv1.bn.bias
freezing model.4.1.cv2.conv.weight
freezing model.4.1.cv2.bn.weight
freezing model.4.1.cv2.bn.bias
freezing model.5.conv.weight
freezing model.5.bn.weight
freezing model.5.bn.bias
freezing model.6.0.cv1.conv.weight
freezing model.6.0.cv1.bn.weight
freezing model.6.0.cv1.bn.bias
freezing model.6.0.cv2.conv.weight
freezing model.6.0.cv2.bn.weight
freezing model.6.0.cv2.bn.bias
freezing model.6.1.cv1.conv.weight
freezing model.6.1.cv1.bn.weight
freezing model.6.1.cv1.bn.bias
freezing model.6.1.cv2.conv.weight
freezing model.6.1.cv2.bn.weight
freezing model.6.1.cv2.bn.bias
freezing model.6.2.cv1.conv.weight
freezing model.6.2.cv1.bn.weight
freezing model.6.2.cv1.bn.bias
freezing model.6.2.cv2.conv.weight
freezing model.6.2.cv2.bn.weight
freezing model.6.2.cv2.bn.bias
freezing model.6.3.cv1.conv.weight
freezing model.6.3.cv1.bn.weight
freezing model.6.3.cv1.bn.bias
freezing model.6.3.cv2.conv.weight
freezing model.6.3.cv2.bn.weight
freezing model.6.3.cv2.bn.bias
freezing model.6.4.cv1.conv.weight
freezing model.6.4.cv1.bn.weight
freezing model.6.4.cv1.bn.bias
freezing model.6.4.cv2.conv.weight
freezing model.6.4.cv2.bn.weight
freezing model.6.4.cv2.bn.bias
freezing model.6.5.cv1.conv.weight
freezing model.6.5.cv1.bn.weight
freezing model.6.5.cv1.bn.bias
freezing model.6.5.cv2.conv.weight
freezing model.6.5.cv2.bn.weight
freezing model.6.5.cv2.bn.bias
freezing model.6.6.cv1.conv.weight
freezing model.6.6.cv1.bn.weight
freezing model.6.6.cv1.bn.bias
freezing model.6.6.cv2.conv.weight
freezing model.6.6.cv2.bn.weight
freezing model.6.6.cv2.bn.bias
freezing model.6.7.cv1.conv.weight
freezing model.6.7.cv1.bn.weight
freezing model.6.7.cv1.bn.bias
freezing model.6.7.cv2.conv.weight
freezing model.6.7.cv2.bn.weight
freezing model.6.7.cv2.bn.bias
freezing model.7.conv.weight
freezing model.7.bn.weight
freezing model.7.bn.bias
freezing model.8.0.cv1.conv.weight
freezing model.8.0.cv1.bn.weight
freezing model.8.0.cv1.bn.bias
freezing model.8.0.cv2.conv.weight
freezing model.8.0.cv2.bn.weight
freezing model.8.0.cv2.bn.bias
freezing model.8.1.cv1.conv.weight
freezing model.8.1.cv1.bn.weight
freezing model.8.1.cv1.bn.bias
freezing model.8.1.cv2.conv.weight
freezing model.8.1.cv2.bn.weight
freezing model.8.1.cv2.bn.bias
freezing model.8.2.cv1.conv.weight
freezing model.8.2.cv1.bn.weight
freezing model.8.2.cv1.bn.bias
freezing model.8.2.cv2.conv.weight
freezing model.8.2.cv2.bn.weight
freezing model.8.2.cv2.bn.bias
freezing model.8.3.cv1.conv.weight
freezing model.8.3.cv1.bn.weight
freezing model.8.3.cv1.bn.bias
freezing model.8.3.cv2.conv.weight
freezing model.8.3.cv2.bn.weight
freezing model.8.3.cv2.bn.bias
freezing model.8.4.cv1.conv.weight
freezing model.8.4.cv1.bn.weight
freezing model.8.4.cv1.bn.bias
freezing model.8.4.cv2.conv.weight
freezing model.8.4.cv2.bn.weight
freezing model.8.4.cv2.bn.bias
freezing model.8.5.cv1.conv.weight
freezing model.8.5.cv1.bn.weight
freezing model.8.5.cv1.bn.bias
freezing model.8.5.cv2.conv.weight
freezing model.8.5.cv2.bn.weight
freezing model.8.5.cv2.bn.bias
freezing model.8.6.cv1.conv.weight
freezing model.8.6.cv1.bn.weight
freezing model.8.6.cv1.bn.bias
freezing model.8.6.cv2.conv.weight
freezing model.8.6.cv2.bn.weight
freezing model.8.6.cv2.bn.bias
freezing model.8.7.cv1.conv.weight
freezing model.8.7.cv1.bn.weight
freezing model.8.7.cv1.bn.bias
freezing model.8.7.cv2.conv.weight
freezing model.8.7.cv2.bn.weight
freezing model.8.7.cv2.bn.bias
freezing model.9.conv.weight
freezing model.9.bn.weight
freezing model.9.bn.bias
Scaled weight_decay = 0.0005
optimizer: SGD with parameter groups 72 weight, 75 weight (no decay),
75 bias
albumentations: version 1.0.3 required by YOLOv3, but version 0.1.12
is currently installed
train: Scanning '../datasets/coco128/labels/train2017.cache' images
```

```
and labels... 128 found, 0 missing, 2 empty, 0 corrupted: 100% 128/128
[00:00<?, ?it/s]
train: Caching images (0.1GB ram): 100% 128/128 [00:00<00:00,
267.11it/s]
val: Scanning '../datasets/coco128/labels/train2017.cache' images and
labels... 128 found, 0 missing, 2 empty, 0 corrupted: 100% 128/128
[00:00<?, ?it/s]
val: Caching images (0.1GB ram): 100% 128/128 [00:01<00:00,
105.91it/s]
Plotting labels to runs/train/exp3/labels.jpg...

AutoAnchor: 4.27 anchors/target, 0.994 Best Possible Recall (BPR).
Current anchors are a good fit to dataset ✅
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to runs/train/exp3
Starting training for 5 epochs...

     Epoch   gpu_mem       box       obj       cls    labels  img_size
       0/4     3.99G   0.03622   0.05623   0.01037       250
640: 100% 8/8 [00:29<00:00,  3.65s/it]
               Class     Images     Labels          P          R
mAP@.5 mAP@.5:.95: 100% 4/4 [00:13<00:00,  3.48s/it]
                 all        128        929      0.688       0.79
0.805      0.586

     Epoch   gpu_mem       box       obj       cls    labels  img_size
       1/4      8.5G    0.0353   0.05446  0.009651       219
640: 100% 8/8 [00:27<00:00,  3.40s/it]
               Class     Images     Labels          P          R
mAP@.5 mAP@.5:.95: 100% 4/4 [00:13<00:00,  3.47s/it]
                 all        128        929      0.702      0.784
0.811      0.588

     Epoch   gpu_mem       box       obj       cls    labels  img_size
       2/4      8.5G   0.03818   0.05692    0.0118       280
640: 100% 8/8 [00:27<00:00,  3.38s/it]
               Class     Images     Labels          P          R
mAP@.5 mAP@.5:.95: 100% 4/4 [00:13<00:00,  3.47s/it]
                 all        128        929      0.697      0.786
0.811      0.591

     Epoch   gpu_mem       box       obj       cls    labels  img_size
       3/4      8.5G   0.03935   0.05555   0.01291       243
640: 100% 8/8 [00:26<00:00,  3.37s/it]
               Class     Images     Labels          P          R
mAP@.5 mAP@.5:.95: 100% 4/4 [00:13<00:00,  3.46s/it]
                 all        128        929       0.71      0.785
0.815      0.593
```

```
     Epoch   gpu_mem       box       obj       cls    labels  img_size
       4/4      8.5G   0.03757   0.05091   0.01077       188       
640: 100% 8/8 [00:27<00:00,  3.38s/it]
               Class     Images     Labels          P          R      
mAP@.5 mAP@.5:.95: 100% 4/4 [00:13<00:00,  3.45s/it]
                 all        128        929      0.747      0.762      
0.816      0.594

5 epochs completed in 0.062 hours.
Optimizer stripped from runs/train/exp3/weights/last.pt, 124.4MB
Optimizer stripped from runs/train/exp3/weights/best.pt, 124.4MB

Validating runs/train/exp3/weights/best.pt...
Fusing layers...
Model Summary: 261 layers, 61922845 parameters, 0 gradients, 156.1
GFLOPs
               Class     Images     Labels          P          R      
mAP@.5 mAP@.5:.95: 100% 4/4 [00:16<00:00,  4.02s/it]
                 all        128        929      0.748      0.762      
0.816      0.594
              person        128        254      0.854      0.783      
0.853      0.626
             bicycle        128          6      0.655        0.5      
0.659       0.38
                 car        128         46      0.831      0.543      
0.624      0.312
          motorcycle        128          5      0.913          1      
0.995      0.779
            airplane        128          6      0.908          1      
0.995      0.727
                 bus        128          7      0.882      0.714      
0.843      0.757
               train        128          3       0.86          1      
0.995      0.895
               truck        128         12      0.676      0.583      
0.645       0.43
                boat        128          6      0.671        0.5      
0.711      0.451
       traffic light        128         14          1      0.428      
0.566      0.291
           stop sign        128          2      0.649          1      
0.995      0.821
               bench        128          9          1      0.755      
0.841      0.422
                bird        128         16      0.966          1      
0.995      0.684
                 cat        128          4      0.893          1      
0.995      0.937
                 dog        128          9      0.889      0.889      
0.984      0.778
```

```
               horse        128          2      0.548          1
0.995      0.796
            elephant        128         17      0.927      0.941
0.936      0.797
                bear        128          1      0.671          1
0.995      0.895
               zebra        128          4      0.865          1
0.995      0.971
             giraffe        128          9      0.868          1
0.984      0.781
            backpack        128          6      0.779        0.5
0.652      0.394
            umbrella        128         18      0.871      0.889
0.917      0.625
             handbag        128         19      0.707      0.383
0.574      0.339
                 tie        128          7      0.888      0.857
0.857      0.665
            suitcase        128          4      0.751          1
0.995      0.722
             frisbee        128          5      0.722        0.8
0.761      0.661
                skis        128          1      0.698          1
0.995      0.796
           snowboard        128          7      0.881      0.714
0.848      0.613
         sports ball        128          6      0.817      0.751
0.809      0.483
                kite        128         10      0.541        0.8
0.664      0.213
        baseball bat        128          4      0.525      0.563
0.503      0.246
      baseball glove        128          7       0.58      0.714
0.718       0.41
          skateboard        128          5      0.816      0.899
0.962      0.434
       tennis racket        128          7      0.635      0.571
0.603      0.378
              bottle        128         18       0.61      0.778
0.75      0.488
          wine glass        128         16      0.775      0.861
0.873      0.529
                 cup        128         36      0.868      0.861
0.915      0.628
                fork        128          6          1       0.33
0.706      0.445
               knife        128         16      0.764      0.812
0.811      0.545
               spoon        128         22      0.786      0.591
0.631      0.441
```

```
                bowl        128         28      0.833      0.713
0.758      0.627
              banana        128          1      0.704          1
0.995      0.597
            sandwich        128          2          1          0
0.638       0.61
              orange        128          4      0.563          1
0.995      0.678
            broccoli        128         11      0.483      0.364
0.411      0.315
              carrot        128         24      0.751       0.75
0.802      0.516
             hot dog        128          2      0.576          1
0.995      0.995
               pizza        128          5      0.787          1
0.995      0.729
               donut        128         14      0.771          1
0.941       0.85
                cake        128          4      0.735          1
0.995      0.902
               chair        128         35      0.776      0.695
0.778      0.465
               couch        128          6      0.692      0.833
0.942      0.588
        potted plant        128         14      0.796      0.857
0.897      0.618
                 bed        128          3          1      0.464
0.863      0.659
        dining table        128         13      0.627      0.519
0.571      0.344
              toilet        128          2      0.681          1
0.995      0.945
                  tv        128          2      0.501          1
0.995      0.821
              laptop        128          3      0.429      0.333
0.597      0.295
               mouse        128          2          1          0
0.662      0.217
              remote        128          8      0.752      0.625
0.665      0.587
          cell phone        128          8      0.694      0.625
0.635      0.389
           microwave        128          3      0.682          1
0.995      0.864
                oven        128          5      0.476        0.6
0.477      0.374
                sink        128          6      0.409        0.5
0.565      0.379
        refrigerator        128          5      0.623        0.8
0.855      0.705
```

```
                 book        128          29       0.626       0.276
0.372      0.205
                clock        128           9       0.915           1
0.995      0.857
                 vase        128           2       0.381           1
0.995      0.945
             scissors        128           1       0.499           1
0.497     0.0995
           teddy bear        128          21       0.946       0.841
0.907      0.609
           toothbrush        128           5       0.838           1
0.995      0.777
Results saved to runs/train/exp3
```

## Visualization

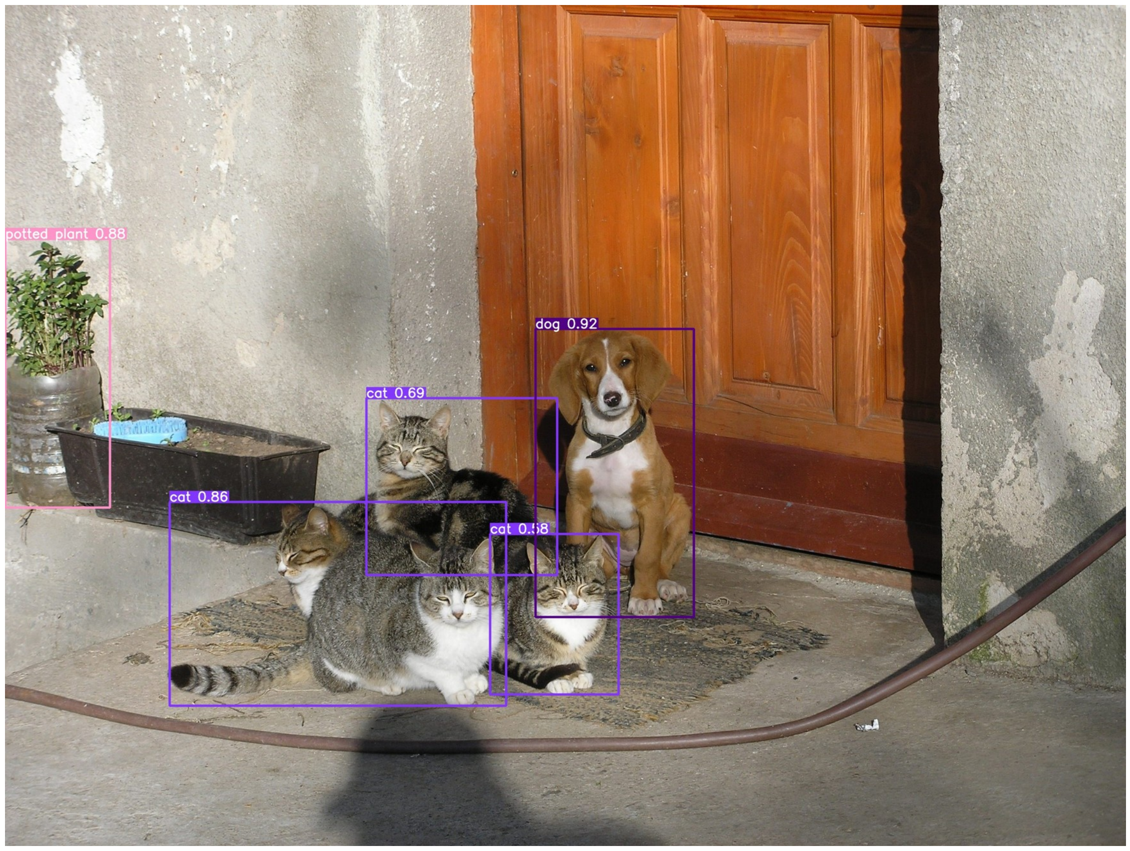
Set up the visualization by running the below cell. Note that if you ran the training loop multiple times, you would have additional folder exp2, exp3, etc.

```
!python detect.py --weights runs/train/exp3/weights/best.pt --img 640
--conf 0.25 --source data/images
```

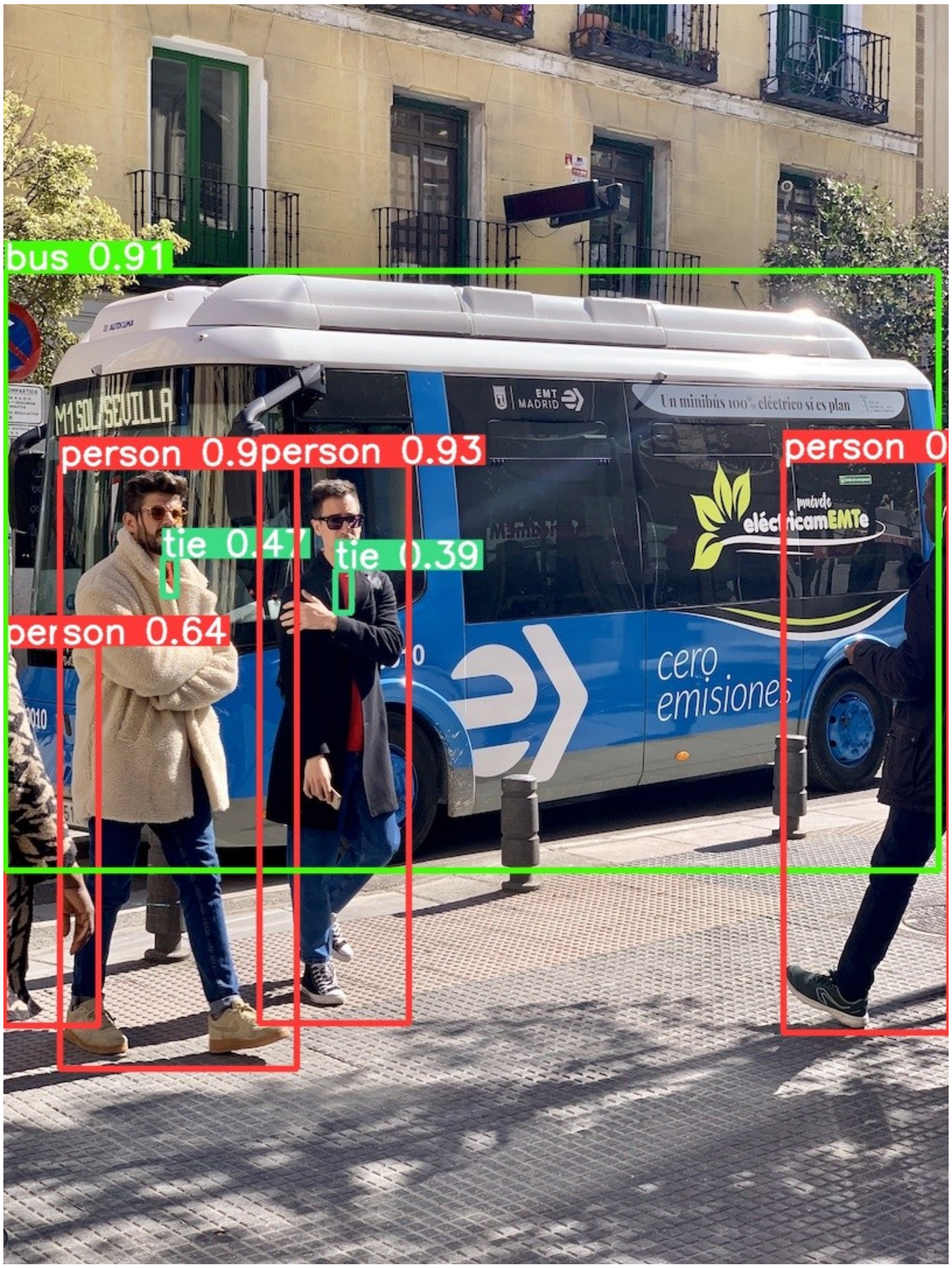
```
detect: weights=['runs/train/exp3/weights/best.pt'],
source=data/images, imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45,
max_det=1000, device=, view_img=False, save_txt=False,
save_conf=False, save_crop=False, nosave=False, classes=None,
agnostic_nms=False, augment=False, visualize=False, update=False,
project=runs/detect, name=exp, exist_ok=False, line_thickness=3,
hide_labels=False, hide_conf=False, half=False, dnn=False
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_dog.jpg: 480x640 3 cats, 1 dog, 1 potted plant, Done.
(0.149s)
image 2/2
/content/csc413/a2/2022/assets/assignments/pa2-q4-files/data/images/
bus.jpg: 640x480 4 persons, 1 bus, 2 ties, Done. (0.151s)
Speed: 0.6ms pre-process, 150.0ms inference, 1.9ms NMS per image at
shape (1, 3, 640, 640)
Results saved to runs/detect/exp4
```

Display Results

```
display.Image(filename='runs/detect/exp4/Cats_and_dog.jpg', width=600)
```



```
display.Image(filename='runs/detect/exp4/bus.jpg', width=600)
```


bus 0.91
person 0.9
person 0.93
person 0
tie 0.47
tie 0.39
person 0.64
M1 SOL/SEVILLA
EMT MADRID
Un minibús 100% eléctrico si es plan
muévete eléctricamEMTe
cero emisiones