HW4

February 14, 2025

Please complete the `NotImplemented` parts of the code cells and write your answers in the markdown cells designated for your response to any questions asked. The tag `# AUTOGRADED` (all caps, with a space after `#`) should be at the beginning of each autograded code cell, so make sure that you do not change that. You are also not allowed to import any new package other than the ones already imported. Doing so will prevent the autograder from grading your code.

For the code submission, run the last cell in the notebook to create the submission zip file. If you are working in Colab, make sure to download and then upload a copy of the completed notebook itself to its working directory to be included in the zip file. Finally, submit the zip file to Gradescope.

After you finish the assignment and fill in your code and response where needed (all cells should have been run), save the notebook as a PDF using the `jupyter nbconvert --to pdf HW4.ipynb` command (via a notebook code cell or the command line directly) and submit the PDF to Gradescope under the PDF submission item. If you cannot get this to work locally, you can upload the notebook to Google Colab and create the PDF there. You can find the notebook containing the instruction for this on Canvas.

If you are running the notebook locally, make sure you have created a virtual environment (using `conda` for example) and have the proper packages installed. We are working with `python=3.10` and `torch>=2`.

Files to be included in submission:

- `HW4.ipynb`
- `model_config.yaml`
- `train_config.yaml`
- `experiments.xlsx`

[1]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

1 Implement and Train a Convolutional ResNet on CIFAR-10

[2]:
```python
"""
DO NOT CHANGE THIS CELL OR ADD ANY IMPORTS ANYWHERE IN THE NOTEBOOK!
"""


# utilities
```

```python
import os
from typing import Sequence
import sys
sys.path.append('/content/drive/MyDrive')

# for interactive plotting
try:
    from google.colab import drive
    drive.mount("/content/drive/")
    # save a copy of the notebook to the drive
except ImportError:
    print("Not in Colab")
os.system('pip install openpyxl -qq')

import pandas as pd
pd.set_option('display.expand_frame_repr', False)
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

# provided code you can or should use:
from HW4_utils import save_yaml, load_yaml, zip_files, Tracker, train

import torch
from torch import nn
import torch.nn.functional as F

from torchvision import datasets
from torchvision.transforms import v2

if torch.cuda.is_available():
    Device = 'cuda'
elif torch.backends.mps.is_available():
    Device = 'mps'
else:
    Device = 'cpu'

print(f'Device is {Device}')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call
drive.mount("/content/drive/", force_remount=True).
Device is cuda
```

1.1 Implement a ResNet model (50)

First, you have to implement a Convolutional ResNet. You can add additional code cells and test your model with random inputs of the correct shape, dtype, and device, to make sure it runs without any errors before using it in the actual training. You can also print the shape through the model to check if it processes the data as intended.

Below you can find a simple illustration of a ResNet Block that you have to implement. Putting aside the batch dimension (number of samples in a batch), the input is of shape `(C, H, W)` where `C, H, W` stand for the number of input channels, height, and width respectively. Each block has an expansion factor of `e`, meaning that the number of channels will be multiplied by `e` after `conv1`, and stay the same number throughout the rest of the block. The spatial dimensions will be divided by `e` by the strided convolution of `conv1`. If `e == 1`, the shape of the tensor is the same throughout the block, and the shortcut (if the block is residual) is just the identity (`nn.Identity()`). However, if `e > 1`, the input and the output are not the same shape and cannot be added together. To get around this, the shortcut has to apply a convolutional layer of the right configuration to change the shape of the input tensor (do not forget batchnorm if we are using it). The batchnorm layers (`bn`) will be optional, and whether to use batchnorm is determined by the boolean flag `batchnorm` of the constructor of the block. The shortcut connection is also optional, determined by the boolean flag `residual` passed to the constructor.

After you implement the ResNet Block in `Block`, you will use it to implement the `Model`. The Model starts with an input layer called `layer0` (a convolutional layer, an optional batchnorm, and an activation) that does not change the spatial shape and creates `base_channels` feature maps from the input. Then there are several residual blocks, and finally a fully connected layer at the end. The output of the last block has to go through a global average pooling layer before the fully connected layer. Think about which of the pooling functions you should use and how. The global average pooling layer should thake the average of all pixels (channelwise), regardless of the input shape. The number of blocks is decided by the length of the `expands` arguments, which is a `list` containing the expansion factor of each block.

You should use `kernel_size = 3` for all `nn.Conv2d` modules. However, you have to decide the correct value for `stride` and `padding` to keep the shapes as intended.

Is there any other module or activation needed after the final linear layer? If so, include that as well. Keep in mind that this is a multi-class classification task and the output of the model is going to be passed to `nn.CrossEntropyLoss()`, so read its documentation to see what the model's output has to be.

Your grade depends on your implementation being well written based on the constructor arguments. You will lose points if you call a module that does not exist, or define modules that are not needed and unnecessarily take extra memory. For example, if `batchnorm` is `False`, the block and the model should not have any batchnorm modules. If `expand` is `1`, the residual connection should not change the input. Otherwise, the shortcut has to consist of a convolution, and batchnorm (only if `batchnorm` is `True`) and an activation. If `residual` is `False`, there should not be any shortcut connection. For nonparameteric transformations like the activation, use calls from `torch` or `torch.nn.functional` (imported as `F`) directly in the `forward` method.

After you are done, clean your code from print statements and parts that were there only for debugging.

[3]:
```
class Block(nn.Module):

    def __init__(
            self,
            in_channels: int,
            expand: int = 1,
            activation: str = 'relu', # name of the function in https://pytorch.
↪org/docs/stable/nn.functional.html#non-linear-activation-functions
            batchnorm: bool = False,
            residual: bool = False,
            ):
        super().__init__()

        # you can call self.act in the forward method
        self.activation = F.__getattribute__(activation)

        self.batchnorm = batchnorm
        self.residual = residual

        out_channels = in_channels * expand

        # --- Main path ---
        # 1st conv: expands channels by 'expand' and applies stride=expand
        self.conv1 = nn.Conv2d(
            in_channels,
            out_channels,
            kernel_size=3,
            stride=expand,
            padding=1,
            bias=(not batchnorm)
        )
        if batchnorm:
            self.bn1 = nn.BatchNorm2d(out_channels)

        # 2nd conv: same shape as output of conv1
        self.conv2 = nn.Conv2d(
            out_channels,
            out_channels,
            kernel_size=3,
            stride=1,
            padding=1,
            bias=(not batchnorm)
        )
        if batchnorm:
            self.bn2 = nn.BatchNorm2d(out_channels)

        # --- Shortcut / skip connection ---
        # Only needed if residual=True.
        # If expand=1, shapes match => identity.
        # If expand>1, shapes differ => need conv (stride=expand) to match␣
↪path.
        if residual:
            if expand == 1:
                self.shortcut_conv = nn.Identity()
            else:
                self.shortcut_conv = nn.Conv2d(
                    in_channels,
                    out_channels,
                    kernel_size=1,
                    stride=expand,
                    padding=0,
                    bias=(not batchnorm)
                )
                if batchnorm:
                    self.shortcut_bn = nn.BatchNorm2d(out_channels)
        else:
            self.shortcut_conv = None

    def forward(
            self,
            x: torch.FloatTensor, # (B, C, H, W)
            ) -> torch.FloatTensor: # (B, C*e, H//e, W//e)

        # Main path
        out = self.conv1(x)
        if self.batchnorm:
            out = self.bn1(out)
        out = self.activation(out)

        out = self.conv2(out)
        if self.batchnorm:
            out = self.bn2(out)

        # Optional residual
        if self.residual:
            if isinstance(self.shortcut_conv, nn.Identity):
                skip = x
            else:
                skip = self.shortcut_conv(x)
                if self.batchnorm:
                    skip = self.shortcut_bn(skip)
            out = out + skip

        # Final activation
        out = self.activation(out)
        return out



class Model(nn.Module):
```

```python
    def __init__(
            self,
            base_channels: int,
            expands: Sequence[int],
            activation: str = 'relu',
            batchnorm: bool = False,
            residual: bool = False,
            num_classes=10,
            ):
        super().__init__()

        self.activation=F.__getattribute__(activation)
        self.batchnorm = batchnorm

        # --- layer0 ---
        self.layer0_conv = nn.Conv2d(
            3,
            base_channels,
            kernel_size=3,
            stride=1,
            padding=1,
            bias=(not batchnorm)
        )
        if batchnorm:
            self.layer0_bn = nn.BatchNorm2d(base_channels)

        # --- Stack of blocks ---
        blocks = []
        in_channels = base_channels
        for e in expands:
            block = Block(
                in_channels=in_channels,
                expand=e,
                activation=activation,
                batchnorm=batchnorm,
                residual=residual
            )
            blocks.append(block)
            in_channels = in_channels * e  # updated for the next block
        self.blocks = nn.Sequential(*blocks)

        # --- Classification head ---
        self.fc = nn.Linear(in_channels, num_classes)

    def forward(
            self,
            x: torch.FloatTensor, # (batch_size, 3, height, width)
            ) -> torch.FloatTensor: # (batch_size, num_classes)

        # layer0
        x = self.layer0_conv(x)
        if self.batchnorm:
            x = self.layer0_bn(x)
        x = self.activation(x)

        # main blocks
        x = self.blocks(x)

        # global average pooling: from (B, C, H, W) to (B, C, 1, 1)
        x = F.adaptive_avg_pool2d(x, (1, 1))
        # flatten to (B, C)
        x = x.view(x.size(0), -1)

        # final linear for classification => logits
        x = self.fc(x)
        return x
```

1.2 Data Augmentation (5)

Apply at least three types of data augmentation. You can change the directory that the data is stored at. It can be a folder in your local machine or Google drive if you are using Colab.

[9]:
```
data_path = 'data'

train_data = datasets.CIFAR10(
    root=data_path,
    train=True,
    download=True,
    transform=v2.Compose([
        v2.ToImage(),

        v2.RandomCrop(32, padding=4),           # 1) Random Crop
        v2.RandomHorizontalFlip(p=0.5),         # 2) Random Horizontal Flip
        v2.ColorJitter(
            brightness=0.2,
            contrast=0.2,
            saturation=0.2,
            hue=0.05
        ),                                      # 3) Color Jitter

        v2.ToDtype(torch.float32, scale=True), # To [0, 1]
        v2.Lambda(lambda x: x - 0.5),           # To [-0.5, 0.5]
    ])
```

```
)

test_data = datasets.CIFAR10(
    root=data_path,
    train=False,
    download=True,
    transform=v2.Compose([
        v2.ToImage(),
        v2.ToDtype(torch.float32, scale=True),  # To [0, 1]
        v2.Lambda(lambda x: x - 0.5),           # To [-0.5, 0.5]
    ])
)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to
data/cifar-10-python.tar.gz

100%|      | 170M/170M [00:03<00:00, 44.5MB/s]

Extracting data/cifar-10-python.tar.gz to data
Files already downloaded and verified
```

1.3 Training and Hyperparameter Tuning (20)

Next, you are going to explore the model hyperparameters. Later, you are going to verify the effectiveness of batchnorm and shortcut connections. To perform a fair comparison, first find a good configuration for the base model with batchnorm and residual connection. Then, we will do an ablation study by training a version of your successful model without those components. You will form your conclusions based on the results you observe. If you get the best grade at your first try, you have great luck, but **you have to run at least 5 experiments** with different configurations to get 5 points of this part. Try at least one model with `batchnorm = False` and one with `residual = False` and one with both `False`. The goal is to see the effect of these models. We suggest you find a successful model with both `True`, and perform an ablation study to see their effect by excluding them from the model.

In this assignment, we have to keep track of the previous experiments to compare the results later in a table. For that purpose, we will save the information about each experiment in a folder to access it later. The code is set up so that it counts the numbder of folders, and names them as their index in the order that they were conducted. The first experiment will create the first folder called `00`, the next one will be `01`, and so on.

To keep track of your past experiments more efficiently, you will have to use `pandas` to create a dataframe that collects the information about your past experiments. 15 points of your grade depend on your dataframe. The dataframe you create should be displayed in the notebook in your submission. The rows should correspond to different experiments, and different columns are different hyperparameters (including both the model and the training) and the metrics (loss and accuracy on train and test dataset).

15 points of your grade depend on the best test accuracy you achieve:

$acc \geq 90\% \rightarrow 25$ points (10 bonus)

$85\% \leq acc < 90\% \rightarrow 20$ points (5 bonus)

$80\% \leq acc < 85\% \rightarrow 15$ points

$75\% \leq acc < 80\% \rightarrow 10$ points

$70\% \leq acc < 75\% \rightarrow 5$ points

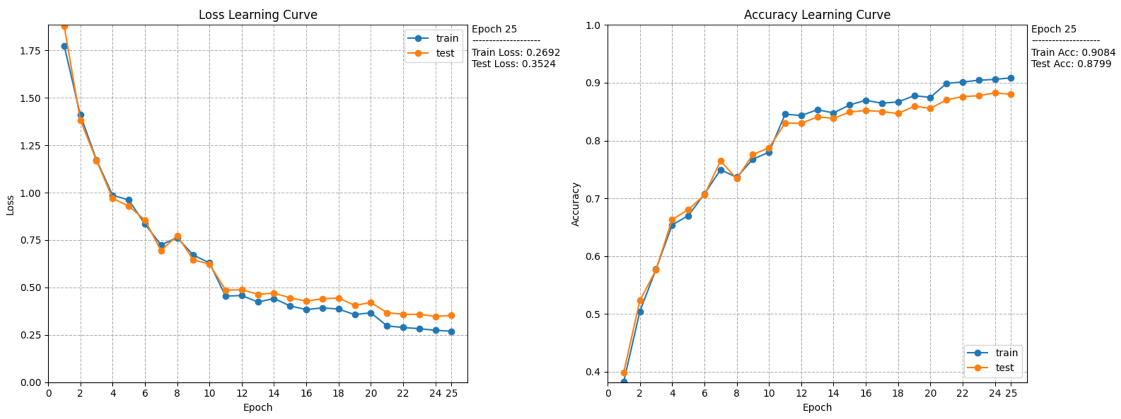
$acc < 70\% \rightarrow 0$ points

[5]:
```
"""
Choose the folder to save model checkpoints and results.
In Colab, this is a folder in your drive (somewhere in /content/drive/MyDrive/..
 ↪.) where you have your course content
"""
results_dir = 'results'
os.makedirs(results_dir, exist_ok=True)
```

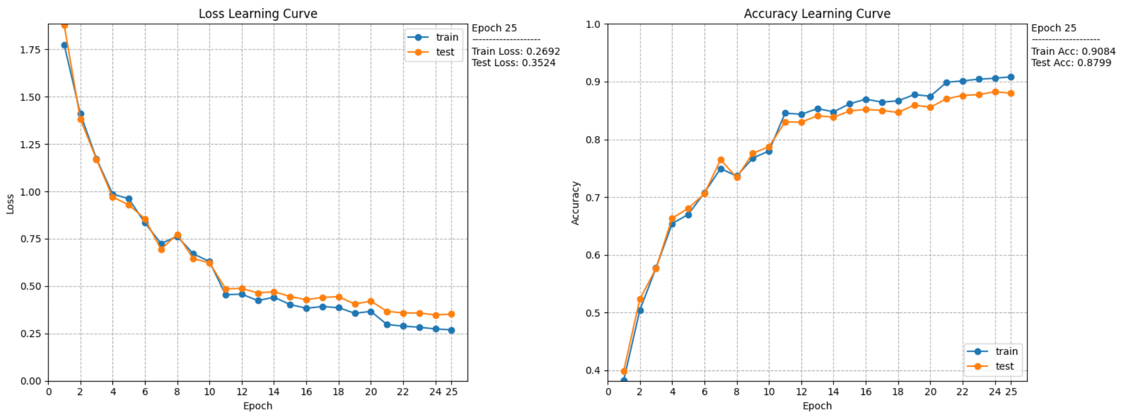
[ ]:
```
"""
Find a good model config and train config.

You can run this cell as many times as you want.
Each time, a new experiment will run and the information will be saved in the␣
 ↪save_path folder.
"""

model_config = dict(
    base_channels = 16,
    expands =  [1,1,1,1,1,1,1,1,
                2,1,1,1,1,1,1,1,
                2,1,1,1,1,1,1,1,
                4,1,1,1,1,1,1,1,]
    ,
    activation = "relu",
    batchnorm = True,
    residual = True,
    )

train_config = dict(
    optim_name = "Adam", # name from optim
    optim_config = dict(lr = 1.65e-3, weight_decay = 3e-4), # dict, then L2 reg
    lr_scheduler_name = "StepLR", # name from lr_scheduler
    lr_scheduler_config = dict( step_size=10, gamma=0.3), # dict
    # lr_scheduler_config = dict(gamma=0.95), Tried ExponentialLR once, but␣
 ↪quickly removed
    n_epochs = 25, # int
    batch_size = 32, # int
    )
```

```
'''
Experiments
First experiments before losing the results to Colab disconnection
1. ResNet-16
  Similar-ish to HW3 with gamma of 0.3 and starting with weight_decay L2␣
 ↪regularization
  - Epochs: 25 should take 45mins
2. Larger model with 15 blocks rather than 9 blocks ResNet-32
3. 21 blocks ResNet-44
4. Weight_decay of 5e-4 and LR of .0015
5. Trying smaller batch size for generalization at 64 to improve test accuracy

Generally started in the low-mid 70s and got to the high 70s on test accuracy

2nd Experiments
1. Equivalent to 1st Ex: number 5
2. Adjusted to a step_size of 5 and gamma of 0.2. Worse performance, but likely␣
 ↪due to the gamma
3. ResNet upped to 4 expansions rather than 3like implementation without␣
 ↪changing to 2 fully connected layers, but adding an expansion and doing 6␣
 ↪blocks per stage
- So it's more like a ResNet49
- Also upped the Gamma to be 0.4 and step_size = 10
- Result: Clear overfitting from epochs 12-17 as training loss diverges from␣
 ↪test loss
4. Increasing the base channels to 32, increased block by 1 per stage,␣
 ↪increased LR to 2e-3, weight_decay to 1e-3 and Gamma to 0.5
  - 4-expansion ResNet57
  - The 32 seemed fine, but from the loss curve it seems that the learning rate␣
 ↪was too high in the middle and later epochs
5. LR to 1.8e-3, weight_decay at 5e-3 and Gamma at 0.4 and step_size of 8 to␣
 ↪make sure the learning rate gets adjusted before the last epoch
  - For whatever reason, the NN seems to have trouble at these hyperparameters␣
 ↪with so many channels and struggles with overfitting, but
    each epoch takes so long to run that it's necessary to bring it to 25␣
 ↪epochs max
  - I believe that there are too many layers and too aggressive a LR decay to␣
 ↪try and reach a maximum accuracy at 25 epochs
6. Back to Experiment 5 from first experiments on a new Colab GPU
7. Attempting an entirely new line with less blocks and larger expansions to␣
 ↪try something I found online that hits larger parameters.
- Additionally, I think removing weight decay is important here because the␣
 ↪larger weighted features could be making it invariant to shifts
  and changes which I've been missing by using weight decay.
- This didn't work either, and did pretty bad. So really, I believe it's my␣
 ↪data transforms
```

```
8. Back to a normal model except this time I did a 4 expand for the last one to␣
 ↪boost representational learning, went back to weight decay
    since the previous model did so poorly compared to training accuracy which␣
 ↪was great actually. So this told me that weight decay was
    good for the generalization like I originally thought. I also lowered my␣
 ↪batch size to 32 which increased my epoch time by 50%. Then,
    finally, I had one last ditch effort to change my data augmentations.␣
 ↪Rather than my 3rd augmentation be a rotation, I did the popular
    ColorJitter instead.
- Best model yet, so it was clearly a data augmentation issue though I believe␣
 ↪the 4 expand at the end helped as well, and I didn't have
  a chance to tune the batch_size to see how it affected the generalization,␣
 ↪but my test accuracy was very close to my train accuracy
'''


n_experiments = len(os.listdir(results_dir))
name = f'{n_experiments:02d}'
save_path = f'{results_dir}/{name}'
os.makedirs(save_path, exist_ok=False) # Not Overwrite

save_yaml(model_config, f'{save_path}/model_config.yaml')
save_yaml(train_config, f'{save_path}/train_config.yaml')

train(
    save_path = save_path,
    model = Model(**model_config).to(Device),
    train_data = train_data,
    test_data = test_data,
    loss_fn = nn.CrossEntropyLoss(),
    device = Device,
    train_pbar = False,
    val_pbar = False,
    plot_freq = 1, # plot the curve every how many epochs
    save_freq = 1, # save the model state dict every how many epochs
    **train_config,
    )
```



```
epochs: 100%|      | 25/25 [1:29:40<00:00, 215.22s/epoch]
```



1.4 Create a table from the experiments to summarize the results (15)

Create a `pandas` dataframe from the saved information about the experiments.

The rows should be the different experiments, with the index being called `name`, which is the name of the experiment folder as a two digit format.

The columns should be the hyperparameters for the model and training, with the same name as the keys in the `model_config` and `train_config` dictionaries.

The columns should also include the following:

- `train_loss`: the final training loss
- `test_loss`: the final test loss
- `train_acc`: the final training accuracy
- `test_acc`: the final test accuracy

You can retrieve these values using the Tracker's `load_results` method, and access them through the tracker.

After you are done, save the dataframe as an excel file and display it in the notebook.

For the PDF submission, assign the pages corresponding to the output of the following cell for the table, and the cell output for the results, to be graded. If you do not assign the correct pages of the PDF, you will be penalized.

[7]:
```
import os
import pandas as pd
import pickle
import yaml
from typing import Dict, Any

table_rows = []

results_dir = "/home/jrob/cmu_grad/deep_learning/HW4/results"

# Make sure it exists (create if needed)
os.makedirs(results_dir, exist_ok=True)
# 1) List all experiment subfolders in results_dir
experiment_folders = sorted(
    [d for d in os.listdir(results_dir) if os.path.isdir(os.path.
 ↪join(results_dir, d))]
)

for exp_name in experiment_folders:
    exp_path = os.path.join(results_dir, exp_name)

    # 2) Load model_config.yaml
    model_config_path = os.path.join(exp_path, 'model_config.yaml')
    if os.path.isfile(model_config_path):
        with open(model_config_path, 'r') as f:
            model_config = yaml.safe_load(f)
    else:
        model_config = {}

    # 3) Load train_config.yaml
    train_config_path = os.path.join(exp_path, 'train_config.yaml')
    if os.path.isfile(train_config_path):
        with open(train_config_path, 'r') as f:
            train_config = yaml.safe_load(f)
    else:
        train_config = {}

    # 4) Load Tracker results (results.pkl)
    results_pkl_path = os.path.join(exp_path, 'results.pkl')
    if os.path.isfile(results_pkl_path):
        with open(results_pkl_path, 'rb') as f:
            tracker_data = pickle.load(f)
        # The tracker keys include:
        # 'train_losses', 'test_losses', 'train_accs', 'test_accs', 'epoch',␣
 ↪'n_epochs'

        # final epoch index
        final_idx = tracker_data['epoch'] - 1  # Because epoch is 1-based
        train_loss = tracker_data['train_losses'][final_idx]
        test_loss = tracker_data['test_losses'][final_idx]
        train_acc = tracker_data['train_accs'][final_idx]
        test_acc = tracker_data['test_accs'][final_idx]
    else:
        # No results found
        train_loss = None
        test_loss = None
        train_acc = None
        test_acc = None

    # Combine all info into a single row
    row = {}
    # Copy the model hyperparams
    for k, v in model_config.items():
        row[k] = v
    # Copy the training hyperparams
    for k, v in train_config.items():
        # Note that train_config['optim_config'] might be a dict, etc.
        # Flatten them if needed or just store them as is
        row[k] = v

    # Add final metrics
    row['train_loss'] = train_loss
    row['test_loss'] = test_loss
    row['train_acc'] = train_acc
    row['test_acc'] = test_acc

    # We'll use the folder name as the index
    row['name'] = exp_name
    table_rows.append(row)

# Create DataFrame
table = pd.DataFrame(table_rows)
# Set index to be the experiment folder name
table = table.set_index('name')

# Save as Excel
```

```
table.to_excel(f'{results_dir}/experiments.xlsx')

# Find best experiment
best_experiment = table['test_acc'].idxmax()
print(f'Best experiment was {best_experiment}'
      f"with test accuracy of {table.loc[best_experiment, 'test_acc']:.2%}")

# Display
table
```

```
Best experiment was 05with test accuracy of 87.99%
```

[7]:

```
      base_channels                                         expands  \
name
00               16  [1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, …
01               16  [1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, …
02               16  [1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, …
03               32  [1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, …
04               32  [1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, …
05               16  [1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, …

     activation  batchnorm  residual optim_name  \
name
00         relu       True      True       Adam
01         relu       True      True       Adam
02         relu       True      True       Adam
03         relu       True      True       Adam
04         relu       True      True       Adam
05         relu       True      True       Adam

                                    optim_config lr_scheduler_name  \
name
00     {'lr': 0.0017, 'weight_decay': 0.0005}           StepLR
01     {'lr': 0.0017, 'weight_decay': 0.0005}           StepLR
02     {'lr': 0.0015, 'weight_decay': 0.0005}           StepLR
03       {'lr': 0.002, 'weight_decay': 0.001}           StepLR
04      {'lr': 0.0015, 'weight_decay': 0.003}           StepLR
05    {'lr': 0.00165, 'weight_decay': 0.0003}           StepLR

                  lr_scheduler_config  n_epochs  batch_size  train_loss  \
name
00     {'step_size': 7, 'gamma': 0.3}        25          64    0.328124
01     {'step_size': 5, 'gamma': 0.2}        25          64    0.434584
02    {'step_size': 10, 'gamma': 0.4}        25          64    0.340612
03    {'step_size': 10, 'gamma': 0.5}        25          64    0.457017
04     {'step_size': 7, 'gamma': 0.4}        25          64    0.657786
05    {'step_size': 10, 'gamma': 0.3}        25          32    0.269210
```

```
      test_loss  train_acc  test_acc
name
00     0.717561    0.88654    0.7754
01     0.891322    0.84846    0.7136
02     0.627355    0.88388    0.7995
03     0.756820    0.84210    0.7458
04     1.043306    0.77202    0.6513
05     0.352446    0.90838    0.8799
```

[ ]:
```
"""
Optional: Write code that deletes the folders you want to discard (if you hate␣
 ↪them so much),
and renames the remaining ones to have sequential numbers with format {n:02d}␣
 ↪again.

You can use os.listdir, os.rename, os.remove, os.path.join, ...
"""
```

1.5 Conclusion (10)

Explain your findings from your hyperparameter search. What were the most and least effective factors?

RESPONSE:

Experiments First experiments before losing the results to Colab disconnection 1. ResNet-16 Similar-ish to HW3 with gamma of 0.3 and starting with weight_decay L2 regularization - Epochs: 25 should take 45mins 2. Larger model with 15 blocks rather than 9 blocks ResNet-32 3. 21 blocks ResNet-44 4. Weight_decay of 5e-4 and LR of .0015 5. Trying smaller batch size for generalization at 64 to improve test accuracy

Generally started in the low-mid 70s and got to the high 70s on test accuracy

2nd Experiments 1. Equivalent to 1st Ex: number 5 2. Adjusted to a step_size of 5 and gamma of 0.2. Worse performance, but likely due to the gamma 3. ResNet upped to 4 expansions rather than 3like implementation without changing to 2 fully connected layers, but adding an expansion and doing 6 blocks per stage - So it's more like a ResNet49 - Also upped the Gamma to be 0.4 and step_size = 10 - Result: Clear overfitting from epochs 12-17 as training loss diverges from test loss 4. Increasing the base channels to 32, increased block by 1 per stage, increased LR to 2e-3, weight_decay to 1e-3 and Gamma to 0.5 - 4-expansion ResNet57 - The 32 seemed fine, but from the loss curve it seems that the learning rate was too high in the middle and later epochs 5. LR to 1.8e-3, weight_decay at 5e-3 and Gamma at 0.4 and step_size of 8 to make sure the learning rate gets adjusted before the last epoch - For whatever reason, the NN seems to have trouble at these hyperparameters with so many channels and struggles with overfitting, but each epoch takes so long to run that it's necessary to bring it to 25 epochs max - I believe that there are too many layers and too aggressive a LR decay to try and reach a maximum accuracy at 25 epochs 6. Back to Experiment 5 from first experiments on a new Colab GPU 7. Attempting an entirely new line with less blocks and larger expansions to try something I found online that hits larger parameters. -

Additionally, I think removing weight decay is important here because the larger weighted features could be making it invariant to shifts and changes which I've been missing by using weight decay. - This didn't work either, and did pretty bad. So really, I believe it's my data transforms 8. Back to a normal model except this time I did a 4 expand for the last one to boost representational learning, went back to weight decay since the previous model did so poorly compared to training accuracy which was great actually. So this told me that weight decay was good for the generalization like I originally thought. I also lowered my batch size to 32 which increased my epoch time by 50%. Then, finally, I had one last ditch effort to change my data augmentations. Rather than my 3rd augmentation be a rotation, I did the popular ColorJitter instead. - Best model yet, so it was clearly a data augmentation issue though I believe the 4 expand at the end helped as well, and I didn't have a chance to tune the batch_size to see how it affected the generalization, but my test accuracy was very close to my train accuracy

2 Zip submission files

You can run the following cell to zip the generated files for submission.

If you are on Colab, make sure to download and then upload a completed copy of the notebook to the working directory so the code can detect and include it in the zip file for submission.

[8]:
```python
# copying the best model to the working directory
# load table from excel
table = pd.read_excel(f'{results_dir}/experiments.xlsx', index_col='name')

best_experiment = table['test_acc'].idxmax()
os.system(f'cp {results_dir}/{best_experiment:02d}/model_config.yaml .')
os.system(f'cp {results_dir}/{best_experiment:02d}/train_config.yaml .')

# copying the excel files to the working directory
os.system(f'cp {results_dir}/experiments.xlsx .')

# creating the zip file consisting of the notebook and the above files
files_to_zip = ['HW4.ipynb', 'model_config.yaml', 'train_config.yaml',␣
 ↪'experiments.xlsx']
zip_files('HW4_submission.zip', *files_to_zip)
```

```
sh: 0: getcwd() failed: No such file or directory
cp: cannot create regular file './model_config.yaml': No such file or directory
sh: 0: getcwd() failed: No such file or directory
cp: cannot create regular file './train_config.yaml': No such file or directory
sh: 0: getcwd() failed: No such file or directory
cp: cannot create regular file './experiments.xlsx': No such file or directory
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 14
     12 # creating the zip file consisting of the notebook and the above files
     13 files_to_zip = ['HW4.ipynb', 'model_config.yaml', 'train_config.yaml', 'experiments.xlsx']
---> 14 zip_files('HW4_submission.zip', *files_to_zip)

NameError: name 'zip_files' is not defined
```