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 Implement and Train a Convolutional ResNet on CIFAR-10

[3]:
```python
"""
DO NOT CHANGE THIS CELL OR ADD ANY IMPORTS ANYWHERE IN THE NOTEBOOK!
"""

# utilities
import os
from typing import Sequence

# 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}')
```

```
Not in Colab
Device is cpu
```

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.

[5]:
```python
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.act = F.__getattribute__(activation)
        self.batchnorm = batchnorm
        self.residual = residual
        out_channels = in_channels * expand

        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3,␣
↪stride=expand, padding=1, bias=not batchnorm)
        self.bn1 = nn.BatchNorm2d(out_channels) if batchnorm else nn.Identity()

        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3,␣
↪stride=1, padding=1, bias=not batchnorm)
        self.bn2 = nn.BatchNorm2d(out_channels) if batchnorm else nn.Identity()

        # short cut
        if self.residual and expand > 1:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1,␣
↪stride=expand, bias=False),
                nn.BatchNorm2d(out_channels) if batchnorm else nn.Identity()
            )
        else:
            self.shortcut = nn.Identity()

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

        shortcut = self.shortcut(x) if self.residual else x

        x = self.conv1(x)
        x = self.bn1(x)
        x = self.act(x)
        x = self.conv2(x)
        x = self.bn2(x)

        return self.act(x + shortcut) if self.residual else self.act(x)

class Model(nn.Module):

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

        self.act = F.__getattribute__(activation)
        self.layer0 = nn.Sequential(
            nn.Conv2d(3, base_channels, kernel_size=3, stride=1, padding=1,␣
↪bias=not batchnorm),
            nn.BatchNorm2d(base_channels) if batchnorm else nn.Identity(),
            nn.ReLU(inplace=True)
        )

        layers = []
        in_channels = base_channels
        for expand in expands:
            layers.append(Block(in_channels, expand, activation, batchnorm,␣
↪residual))
            in_channels *= expand
        self.res_layers = nn.Sequential(*layers)
        self.global_avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Linear(in_channels, 10) # 10 classes

    def forward(
            self,
            x: torch.FloatTensor, # (batch_size, 3, height, width)
            ) -> torch.FloatTensor: # (batch_size, num_classes)
        x = self.layer0(x)
        x = self.res_layers(x)
        x = self.global_avg_pool(x)
        x = torch.flatten(x, 1)
        return self.fc(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.

[7]:
```
data_path = 'data'

train_data = datasets.CIFAR10(
    root = data_path,
    train = True,
    download = True,
    transform = v2.Compose([
        v2.ToImage(),
        v2.RandomHorizontalFlip(p=0.5),
        v2.RandomRotation(degrees=15),
        v2.RandomResizedCrop(size=(32, 32), scale=(0.8, 1.0)),
        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]
        ])
        )
```

```
Files already downloaded and verified
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

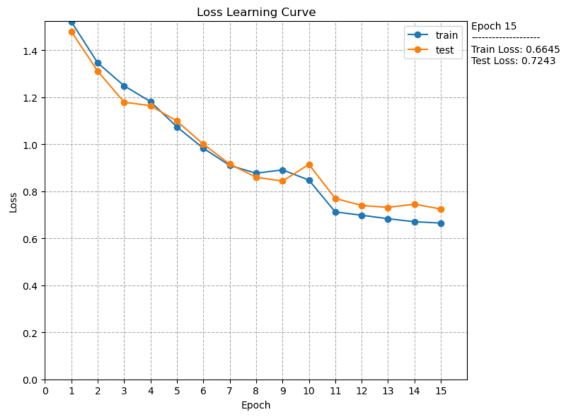
$70\% \leq acc < 75\% \rightarrow$ 5 points

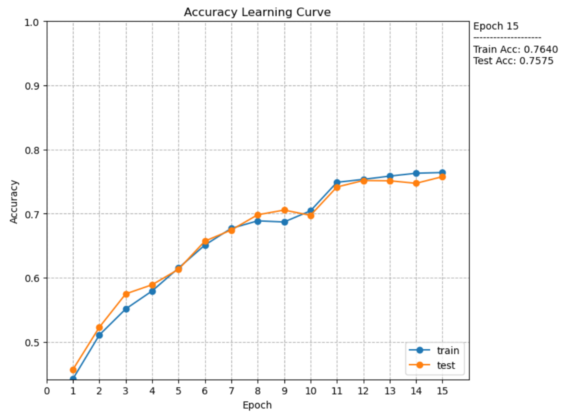
$acc < 70\% \rightarrow$ 0 points

[9]:
```
"""
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)
```

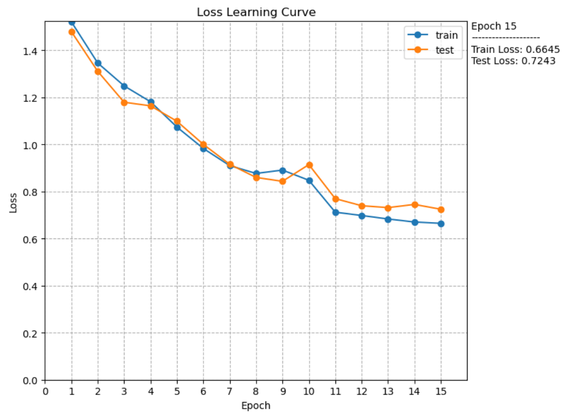
[49]:
```
"""
Find a good model config and train config.

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 = 32,
    expands = [1, 2, 2, 4],#2, 2, 4],
    activation = "relu",
    batchnorm = False,
    residual = False,
    )

train_config = dict(
    optim_name = "Adam", # name from optim
    optim_config = {"lr": 0.001}, # dict
    lr_scheduler_name = "StepLR", # name from lr_scheduler
    lr_scheduler_config = {"step_size": 10, "gamma": 0.1}, # dict
    n_epochs = 15, # int
    batch_size = 32, # int
    )

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')
```

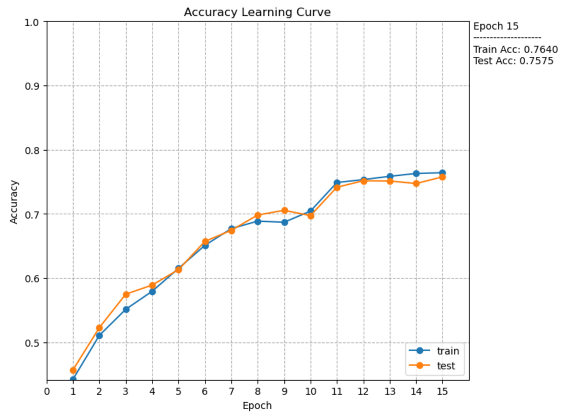
```python
train(
    save_path = save_path,
    model = Model(**model_config).to(Device),
    train_data = train_data,
    test_data = test_data,
    loss_fn = nn.CrossEntropyLoss(),
    device = Device,
    train_pbar = False,
    val_pbar = False,
    plot_freq = 1, # plot the curve every how many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





```
epochs:
100%|                                      |
15/15 [35:12<00:00, 140.86s/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.

[70]:
```
experiment_folders = sorted(os.listdir(results_dir))
experiment_data = []
for exp_name in experiment_folders:
    exp_path = os.path.join(results_dir, exp_name)

    model_config_path = os.path.join(exp_path, "model_config.yaml")
    train_config_path = os.path.join(exp_path, "train_config.yaml")
    results_pkl_path = os.path.join(exp_path, "results.pkl")
    if not (os.path.exists(model_config_path) and os.path.
 ↪exists(train_config_path) and os.path.exists(results_pkl_path)):
        print(f"Skipping {exp_name}: Missing files")
        continue

    with open(model_config_path, "r") as file:
        model_config = yaml.safe_load(file)
    with open(train_config_path, "r") as file:
        train_config = yaml.safe_load(file)
    with open(results_pkl_path, "rb") as file:
        tracker_results = pickle.load(file)

    train_loss = tracker_results["train_losses"][-1] if "train_losses" in␣
 ↪tracker_results else None
    test_loss = tracker_results["test_losses"][-1] if "test_losses" in␣
 ↪tracker_results else None
```

```
    train_acc = tracker_results["train_accs"][-1] if "train_accs" in␣
↪tracker_results else None
    test_acc = tracker_results["test_accs"][-1] if "test_accs" in␣
↪tracker_results else None

    experiment_result = {
        "name": exp_name,
        **model_config,
        **train_config,
        "train_loss": train_loss,
        "test_loss": test_loss,
        "train_acc": train_acc,
        "test_acc": test_acc,
    }

    experiment_data.append(experiment_result)



#table: pd.DataFrame = NotImplemented
table = pd.DataFrame(experiment_data)

# save to excel
table.to_excel(f'{results_dir}/experiments.xlsx')

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

# display in notebook
table
```

```
Skipping 00: Missing files
Skipping 11: Missing files
Best experiment was 4 with test accuracy of 86.93%
```

[70]:
```
  name  base_channels          expands activation  batchnorm  residual
optim_name   optim_config lr_scheduler_name              lr_scheduler_config
n_epochs  batch_size  train_loss  test_loss  train_acc  test_acc
0   01             32           [1, 2]       relu       True      True
Adam  {'lr': 0.001}            StepLR  {'step_size': 10, 'gamma': 0.1}        15
32    0.736013   0.750488    0.74428    0.7360
1   02             32        [1, 2, 4]       relu       True      True
Adam  {'lr': 0.001}            StepLR  {'step_size': 10, 'gamma': 0.1}        15
32    0.450642   0.507859    0.84436    0.8283
2   03             32     [1, 2, 2, 4]       relu       True      True
```

```
Adam  {'lr': 0.001}            StepLR  {'step_size': 10, 'gamma': 0.1}        15
32    0.321476   0.430590    0.88702    0.8549
3   04               32  [1, 2, 2, 2, 4]        relu        True       True
Adam  {'lr': 0.001}            StepLR  {'step_size': 10, 'gamma': 0.1}        15
32    0.361834   0.437692    0.87198    0.8542
4   05               64  [1, 2, 2, 2, 4]        relu        True       True
Adam  {'lr': 0.001}            StepLR  {'step_size': 10, 'gamma': 0.1}        15
32    0.294366   0.383444    0.89776    0.8693
5   06               32  [1, 2, 2, 2, 4]        relu        True       True
Adam   {'lr': 0.01}            StepLR  {'step_size': 10, 'gamma': 0.1}        15
32    0.417341   0.453868    0.85468    0.8454
6   07               32  [1, 2, 2, 2, 4]        relu        True       True
Adam   {'lr': 0.01}            StepLR   {'step_size': 7, 'gamma': 0.1}        15
32    0.475693   0.507486    0.83190    0.8277
7   08               32     [1, 2, 2, 2, 4]     relu       False       True
Adam  {'lr': 0.001}            StepLR  {'step_size': 10, 'gamma': 0.1}        15
32    0.537311   0.677839    0.81154    0.7767
8   09               32     [1, 2, 2, 2, 4]     relu        True      False
Adam  {'lr': 0.001}            StepLR  {'step_size': 10, 'gamma': 0.1}        15
32    0.336019   0.417317    0.88334    0.8564
9   10               32     [1, 2, 2, 2, 4]     relu       False      False
Adam  {'lr': 0.001}            StepLR  {'step_size': 10, 'gamma': 0.1}        15
32    0.664540   0.724252    0.76402    0.7575
```

[68]:
```
# """
# 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, ...
# """
# import shutil

# # Identify folders to delete
# folders_to_delete = {f"{i:02d}" for i in range(6)}

# # Remove unwanted folders
# for folder in experiment_folders:
#     if folder in folders_to_delete:
#         folder_path = os.path.join(results_dir, folder)
#         shutil.rmtree(folder_path, ignore_errors=True)
#         print(f"Deleted folder: {folder}")

# # Refresh the list after deletion
# experiment_folders = sorted(os.listdir(results_dir))
# # Rename remaining folders sequentially
# for new_idx, old_name in enumerate(experiment_folders):
#     new_name = f"{new_idx:02d}"
#     old_path = os.path.join(results_dir, old_name)
#     new_path = os.path.join(results_dir, new_name)

#     if old_path != new_path:  # Avoid renaming if it's already correctly named
#         os.rename(old_path, new_path)
#         print(f"Renamed {old_name} → {new_name}")
```

```
Deleted folder: 00
Deleted folder: 01
Deleted folder: 02
Deleted folder: 04
Deleted folder: 05
Renamed .ipynb_checkpoints → 00
Renamed 06 → 01
Renamed 07 → 02
Renamed 08 → 03
Renamed 09 → 04
Renamed 10 → 05
Renamed 11 → 06
Renamed 12 → 07
Renamed 14 → 08
Renamed 15 → 09
Renamed 16 → 10
Renamed experiments.xlsx → 11
```

1.5 Conclusion (10)

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

RESPONSE:

From the above results, we see that deeper channels seem to preform better. As the number of expansions increase, so does that accuracy. However, there wasn't such a big jump going from [1, 2, 2, 4] to [1, 2, 2, 2, 4] in test accuracy showing diminishing returns. Also, larger base_channels of 54 compared to 32 had higher results meaning increasing the size of the model helped. The results also show a learning rate of 0.001 out preformed a learning rate of 0.01 with a gamma of 0.1 and both a step size of 10 and 7. Overall, shallow models underpreformed compared to deeper models, although there is a point were performance seems to not increase as much. Additionally, lower learning rates seem to work better then smaller learning rates.

In terms of batchnorm and residual, when keeping all other variables constant (i.e, base channels, expands, optimizer, learning rate etc.), when batchnorm is set to false there is a drop in test accuracy from .8549 to .7767. However when only residual is set to fall there is actually a small increase from .8549 to 0.8564 although this might not be statistically significant. But we notice when both are set to false the test accuracy falls to a low of .7575. This means that not including batchnorm has a significant impact where as not including residual does not, but not including both creates the worst outcome.

Also while setting residual tto false might not greatly affect the test accuracy, train accuracy increases significantly showing that the model is overfitting. This is also true with making the model larger and deeper with trial 4.

[87]:
```
import os

source_file = "results/05/model_config.yaml"
destination_file = "model_config.yaml"

# Check file permissions
print(f"Checking permissions for {source_file}:")
os.system(f"ls -l {source_file}")

# Try copying using sudo (if on Linux/macOS)
os.system(f"cp {source_file} {destination_file}")
```

```
Checking permissions for results/05/model_config.yaml:
```

[87]:
```
1
```

[91]:
```
results_dir = "results"
best_experiment = 5  # Example
source_file = f"{results_dir}/{best_experiment:02d}/model_config.yaml"
print(f"Trying to copy: {source_file}")
```

```
Trying to copy: results/05/model_config.yaml
```

[95]:
```
import os

cmd = f'cp {results_dir}/{best_experiment:02d}/model_config.yaml .'
status = os.system(cmd)

if status != 0:
    print(f"Copy command failed with exit code: {status}")
else:
    print("Copy command executed successfully.")

# Verify if the file was copied
if os.path.exists("model_config.yaml"):
    print("Copy successful.")
else:
    print("Copy failed.")
```

```
Copy command failed with exit code: 1
Copy failed.
```

[107]:
```
os.system("dir results\\05")
```

```
[107]: 0
```

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.

```
[93]: # 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)
```

```
5
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[93], line 15
     13 # creating the zip file consisting of the notebook and the above files
     14 files_to_zip = ['HW4.ipynb', 'model_config.yaml', 'train_config.yaml',␣
 ↪'experiments.xlsx']
---> 15 zip_files('HW4_submission.zip', *files_to_zip)

File ~\Deep Learning\HW4\HW4_utils.py:37, in zip_files(output_filename,␣
 ↪*file_paths)
     35 with zipfile.ZipFile(output_filename, 'w') as zipf:
     36     for file_path in file_paths:
---> 37         zipf.write(file_path, os.path.basename(file_path))

File ~\anaconda3\Lib\zipfile\__init__.py:1841, in ZipFile.write(self, filename,␣
 ↪arcname, compress_type, compresslevel)
   1836 if self._writing:
   1837     raise ValueError(
   1838         "Can't write to ZIP archive while an open writing handle exists
   1839     )
-> 1841 zinfo = ZipInfo.from_file(filename, arcname,
   1842                                  strict_timestamps=self._strict_timestamps)
   1844 if zinfo.is_dir():
   1845     zinfo.compress_size = 0

File ~\anaconda3\Lib\zipfile\__init__.py:557, in ZipInfo.from_file(cls,␣
  ↪filename, arcname, strict_timestamps)
    555 if isinstance(filename, os.PathLike):
    556     filename = os.fspath(filename)
--> 557 st = os.stat(filename)
    558 isdir = stat.S_ISDIR(st.st_mode)
    559 mtime = time.localtime(st.st_mtime)

FileNotFoundError: [WinError 2] The system cannot find the file specified:␣
  ↪'model_config.yaml'
```

```
[ ]: Cant zip, downloaded each file seperately and zipped myself
```