# DS542 Midterm Challenge Report

Declan Young

Created on March 25 | Last edited on March 28

## Model Description

### Part 1

The simple CNN model created for this section has the following structure:

```
Conv2d(3, 32, (5,5))
MaxPool2d((2,2))
Conv2d(32, 64, (5,5))
MaxPool2d((2,2))
Linear(64*5*5, 128)
Linear(128, 100)
```

As this part required a relatively simple CNN, a CNN that has only 4 hidden layers was created. It is comprised of 2 convolutional layers with max pooling layers after it followed by 2 fully collected linear layers. The input layer is a convolutional layer with input channels 3, while the output layer is fully connected layer with output size 100.

The reason why a kernel size of 5 was used for both convolutional layers, as opposed to a kernel size of 3, is due to the fact the larger the kernel size, the more generic the features extracted by the kernel would be. Considering the facts that we are trying to classify images into over a hundred categories and that the images are quite small, it seems more reasonable to find general features.

Apart from this, the number of neurons in all of the hidden layer outputs were kept to relatively small powers of 2 due to convention and the relatively small size of the input images (32x32).

Max pooling layers were also used as by convention, max pooling layers are used after convolutional layers to reduce trainable parameters whilst keeping the most useful amount of information.

## Part 2

The ResNet18 model architecture from torchvision with a modified output layer was used for this part of the challenge:

```
model = torchvision.models.resnet18(pretrained=False)
model.fc = nn.Linear(512, 100)
```

The output layer was modified to ensure that the model would output the correct amount of features for expected for the Cifar-100 and OOD dataset. By default, the ResNet18 model outputs 1000 features as that corresponds with the ImageNet dataset it's trained on in torchvision.

The ResNet18 model was chosen for this part as it had the highest clean test accuracy on the CIFAR-100 dataset compared to the rest of the torchvision models. The initial reason ResNet was chosen for testing was because of its relatively fast training time as well as the fact that its able to circumvent the vanishing gradient problem through its use of skip connections.

## Part 3

The ResNeXt50 model architecture from torchvision with a modified input and output layer was used for this part of the challenge:

```
model = torchvision.models.resnext50_32x4d(pretrained=True)
model.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
model.fc = nn.Linear(2048, 100)
```

In addition to the above mentioned modification of the output layer, the input layer was also modified for this model. According to Torchvision's documentation, their pretrained models all expect images of of at least 224x224 pixels. However, as the images within the CIFAR-100 dataset are of size 32x32, the input layer's kernel and padding sizes were reduced to 5, 1, and 1 from a size of 7, 1 and 3 respectively. Thus, preserving the already small amount of information. The padding and stride values were chosen arbitrarily.

The ResNext50 model was chosen for this part as it had the highest clean test accuracy on the CIFAR-100 dataset compared to the rest of the torchvision models. Initially, ResNet18 was tested since the only major difference between part 2 and 3 is the inclusion of the pretrained weights. However, as mentioned, this was switched to ResNext50 after it was found to have higher test accuracy on CIFAR-100 and due to it being a "highly modularised network architecture for image classification" according to the paper.

## Hyperparameter Tuning

As both parts 1 and 2 required only a rather simplistic model that does not necessarily have to reach the specific threshold of 0.397, no hyperparameter tuning was conducted for them.

For part 3, however, some very rudimentary hyperparameter tuning consisting of changing ResNext's first layer's kernel size was conducted. Specifically, the kernel sizes of 3 and 5 were

tested. As the kernel of size 5 had a higher clean test accuracy on CIFAR-100 and the OOD, it was chosen.

## Regularisation Techniques

Although not strictly considered regularisation, normalisation with 2 separate means and standard deviations were tested. The two notable values used for normalization were ImageNet's and CIFAR-100's normalization values. Initially for part3, the normalization values for ImageNet were used as opposed to CIFAR-100 due to the fact that the model was initially trained on images from ImageNet. However, after extensive experimentation and after seeing that the OOD dataset was normalised with the use of the CIFAR-100 dataset normalisation values, the normalisation values were switched to that of the CIFAR-100 dataset.

Apart from normalisation and the regularisation 1 found in the model architectures of ResNet18 and ResNext50, no other regularisation techniques were explicitly added.

## Data Augmentation Strategy

No data augmentation techniques were used for parts 1 and 2.

The use of the random horizontal flip data augmentation technique was used to improve the model's performance for part3. This data augmentation technique randomly flips the image with a given probability. In this particular case, the probability was set to 0.5. Other data augmentation techniques such as random crop, and colour jitter were also tested, but it ended up not being used because it lowered the overall clean test set accuracy on the CIFAR-100 dataset.

## Results Analysis

The final model for part 1 achieved an accuracy of 22.09% on the OOD on Kaggle. During training, it achieved an accuracy of 36.46% on the training set and 30.94% on the validation set. The small

difference in its training and validation accuracy lends credence to the notion that it is rather well generalised. Considering that this model is a simple CNN without any data augmentation or specific normalisation used to establish a baseline accuracy to, it is understandable why its accuracy is quite low.

The final model for part 2 achieved an accuracy of 31.67% on the OOD on Kaggle. During training, it achieved an accuracy of 47.57% on the training set and 37.89% on the validation set. The small difference in its training and validation accuracy demonstrates that it is rather well generalised.

The final model for part 3 managed to achieve an accuracy of 48.17% on the OOD on Kaggle, as well as 61.08% on the CIFAR-100 dataset when fine-tuned locally. During training, it achieved a 80.73% on the training set, and 61.14% on the validation set. The rather similar training and testing accuracy for the model demonstrates that the model generalises pretty well. Additionally, the use of pretrained weights from ImageNet helped the model generalise rather well to the CIFAR-100 and OOD dataset, with only about a 17.87% difference between their respective accuracies, despite only being fine tuned for 5 epochs. However, it is worth noting that the difference between its training and validation accuracies is the largest out of the final three models at 19.59%.

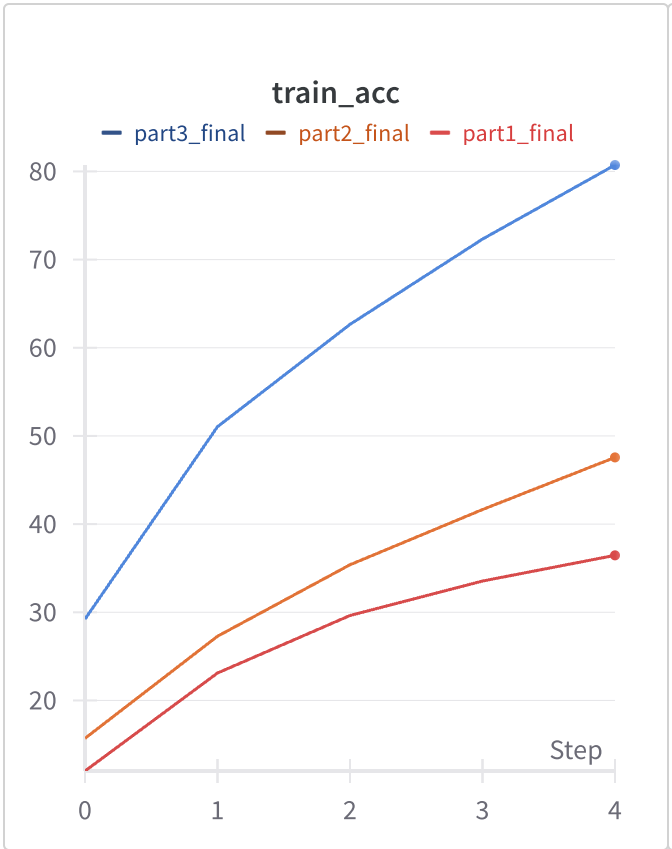
As the models in parts 1 and 2 were primarily used to establish baseline performances, only the final model in part 3 will be further discussed.

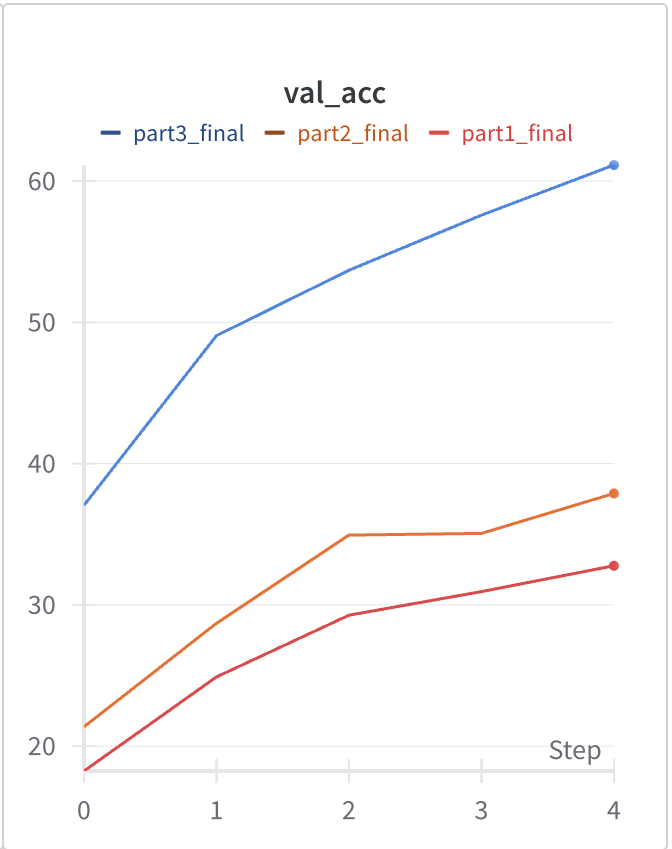
Some strengths of the final model in part 3 are that it most likely generalizes quite well across datasets, and effectively extracts early layer features. Firstly, it most likely generalizes well due to the fact that it achieved 61.08% and 48.17 % on CIFAR-100 and the OOD dataset respectively by fine tuning a ImageNet trained ResNext50 model. As such, it lends credence to the notion the model effectively utilises transfer learning to a pretty high degree. Next, by changing the kernel size of the first layer in the ResNeXt to a 3x3 as opposed to 7x7 in the first layer, it increases the receptive

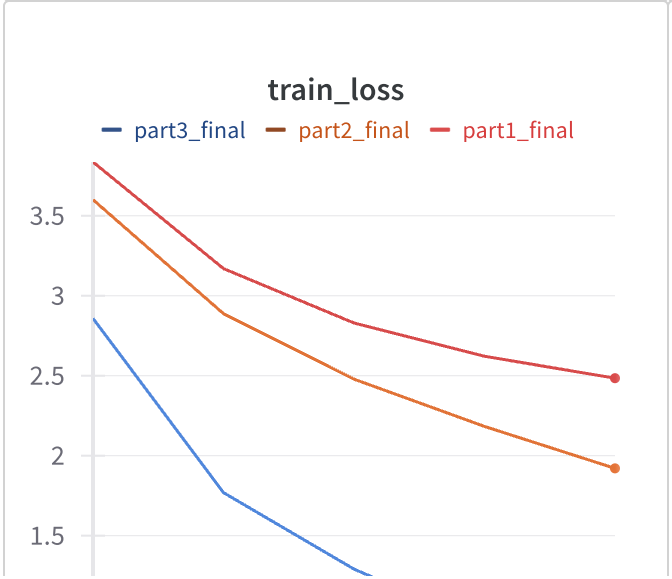
field early in the model and most likely helps it enhance feature extraction in smaller images such as CIFAR-100.

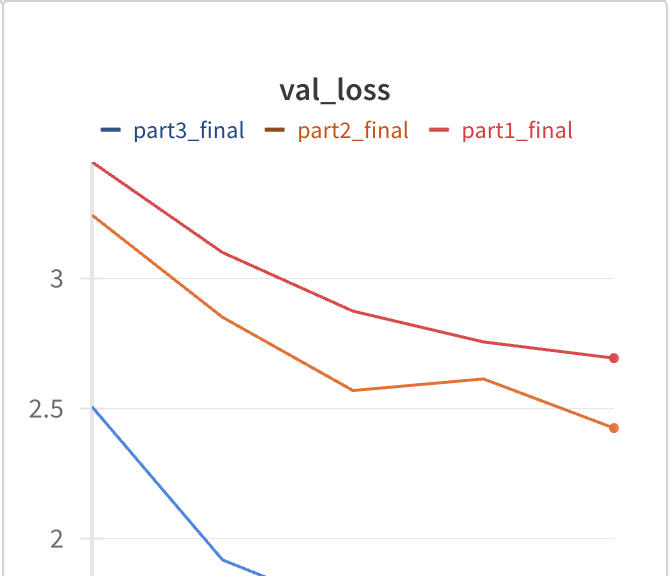
However, despite these rather positive results, the results still show that there are still some weaknesses of the model. Namely the fact that the model is slightly overfitting to the data, as evidenced by the aforementioned training and validation accuracies. This can most likely be attributed to the fact that ResNeXt50 is a rather deep complex model. Thus, it is more prone to overfitting smaller datasets. Additionally, the lack of hyperparamter tuning for stride and padding in the modified first convolutional layer means that the model could still possibly be optimised.

Some ways in which the model can be improved are through the use of more data augmentation, more comprehensive hyperparameter tuning, and the use of more regularisation techniques. Firstly, more rigorous data augmentation could be conducted as CIFAR-100 only consists of about 60000 images. Hence, its training set only has about 48000 images (by the 80-20 split). By introducing more data augmentation techniques, it could help with better fine tuning the ImageNet weights to be more in line with both the CIFAR-100 and OOD dataset. Next, more hyperparameter tuning can be used to improve the overall accuracy of the model. In this specific case, the a learning rate of 0.001 for the Adam optimiser. This number was chosen arbitrarily, as such, it is possible another set of untested hyperparameters could be used to improve the overall accuracy of the model. Although a cosine annealing learning rate scheduler was used to adapt the learning rate during training, it still could have been refined further. Additionally, other hyperparameters such as stride padding were not extensively considered as well. Finally, more regularization could be added to make the model more well generalized. As previously mentioned, the difference between the model's traning and validation accuracies are at 19.59%, demonstrating that the model is most likely overfitting to a low degree. Thus, by adding more regualrization techniques to the model, it can potentially increase the potency of the model on unseen images.

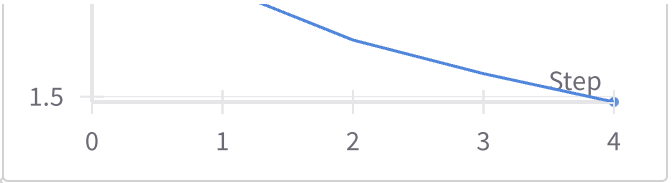

train_acc
part3_final
part2_final
part1_final
80
70
60
50
40
30
20
Step
0
1
2
3
4

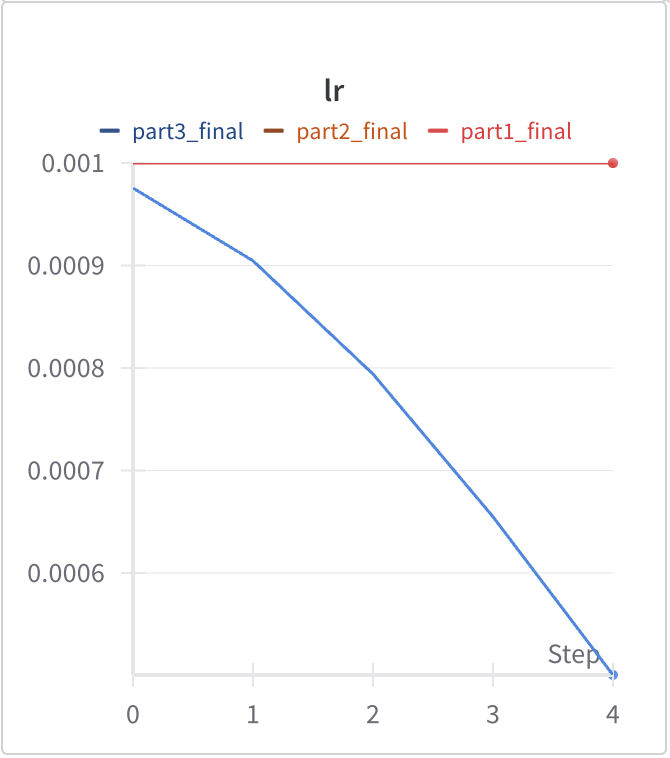

val_acc
part3_final
part2_final
part1_final
60
50
40
30
20
Step
0
1
2
3
4


train_loss
part3_final
part2_final
part1_final
3.5
3
2.5
2
1.5


val_loss
part3_final
part2_final
part1_final
3
2.5
2


1
Step
0
1
2
3
4


1.5
Step
0
1
2
3
4


lr
part3_final
part2_final
part1_final
0.001
0.0009
0.0008
0.0007
0.0006
Step
0
1
2
3
4

Run set 3

## AI Disclosure

AI was used to figure out how to change the final layer of the ResNet18 and ResNext50 models as well as to define a scheduler to use for the three models. Specifically the following equivalent lines of code:

```
#part1.py
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

#part2.py
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)
----------------------------------------------------------------------------------------
```

```
model.fc = nn.Linear(512, 100)

#part3.py
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=10, eta_min=1e-6)
-----------------------------------------------------------------------------------------
model.fc = nn.Linear(2048, 100)
```

Created with ❤️ on Weights & Biases.

https://wandb.ai/declanyg-boston-university/-sp25-ds542-challenge/reports/DS542-Midterm-Challenge-Report--VmlldzoxMTk3MDUxMA