MSc in Artificial Intelligence and Machine Learning

CS6482 – Deep Reinforcement Learning

Assignment 1: Sem2 AY 23/24 - Convolutional Neural Networks (CNNs)

Pratik Verma – 23007575

Siddharth Prince – 23052058

Table of contents

1. Introduction

Need to write something or might just skip to the Dataset.

2. Dataset

For the dataset, we zeroed in on the Caltech-101 dataset. It consists of images belonging to 101 object categories, ranging from everyday objects like airplanes and cars to animals like frogs and insects. This had the perfect mix of enough samples and classification categories while not being too much for our machines to handle. In other words, it felt like it hit the sweet spot between the mini-ImageNet and ImageNet datasets.

2.1 Visualisation

However, when exploring the data, we did find out that the samples are not evenly distributed. There are about six classes with a substantial number of images compared to the rest of the 95 classes that have 50 or less images each.

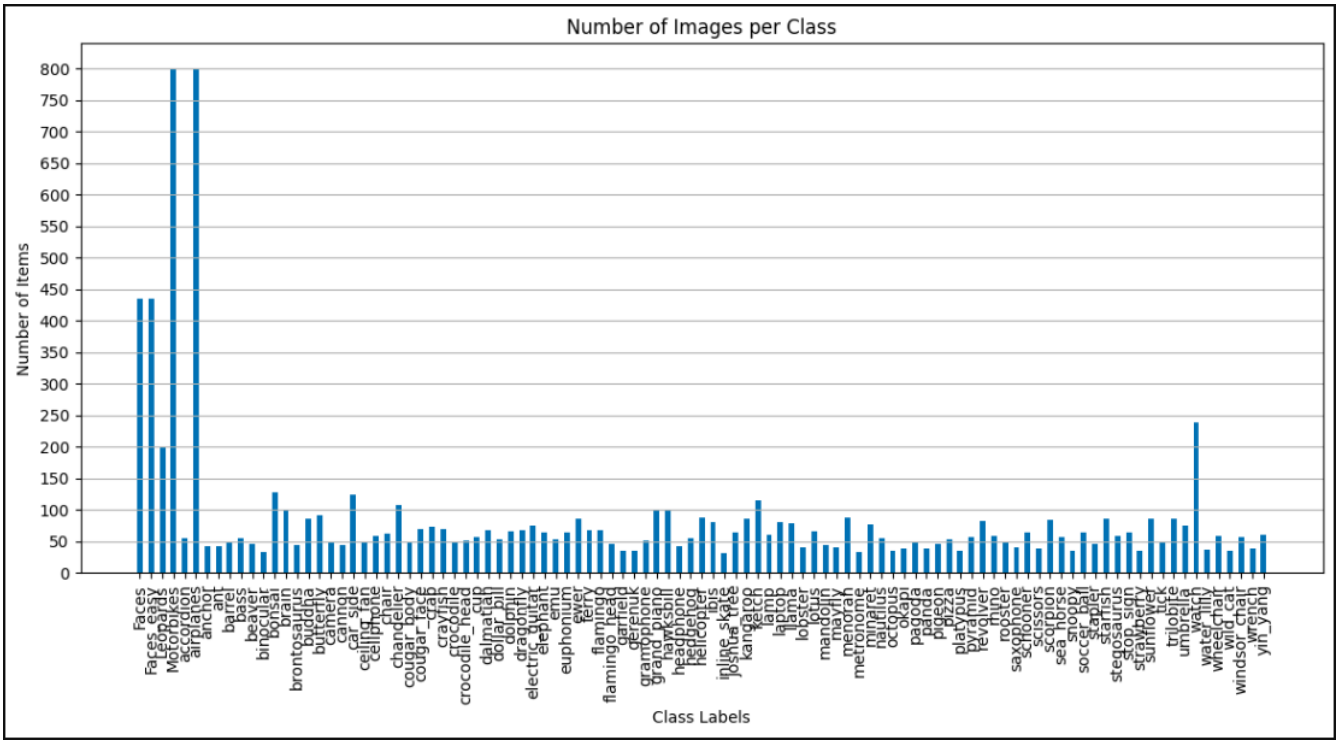


Fig 2.1: Data distribution bar graph of the Caltech-101 dataset

This does seem to be a problem especially when randomising the data. The class imbalance can negatively impact model training if certain classes are underrepresented in the training data.

3. Data Pre-processing

Before training the model, we performed the standard pre-processing steps such as normalisation and resizing all images to the 224x224 dimensions for compatibility with the ZFNet architecture.

3.1 Normalisation

We initially used the gold standard for the mean and standard deviation values from ImageNet that we found is frequently used [1][2] and accepted.

```
[8]: # Transformations
transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

Fig 3.1: Code snippet from notebook showing the data transformations

However, it made sense to compute these values for the Caltech-101 dataset from scratch and then use those to normalise the data because we are training a model from scratch and not using an ImageNet based pre-trained model. The mean and standard deviation values across the three colour channels that we calculated for the dataset are as follows:

- Mean: [0.5459, 0.5288, 0.5022]
- Standard deviation: [0.1811, 0.1777, 0.1851]

The code to compute the mean and standard deviation was taken from a PyTorch forum thread [4].

4. CNN Architecture

We adopted the ZFNet architecture, which is an improved version of AlexNet. ZFNet's key feature is the fine-tuning of convolutional layers to enhance feature map resolution, thus improving the model's ability to capture intricate patterns within the images.

4.1 ZFNet Architecture

ZFnet uses a combination of fully connected layers and CNNs. ZF Net was invented by Dr. Rob Fergus and his PhD student at that moment, Dr. Matthew D. Zeiler in NYU. It was the ILSVRC 2013 winner. The network has relatively fewer parameters than AlexNet, but still outperforms it on ILSVRC 2012 classification task by achieving top accuracy with only 1000 images per class. ZFnet is an improved version of AlexNet. Tweaking the architecture hyperparameters, in

particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.

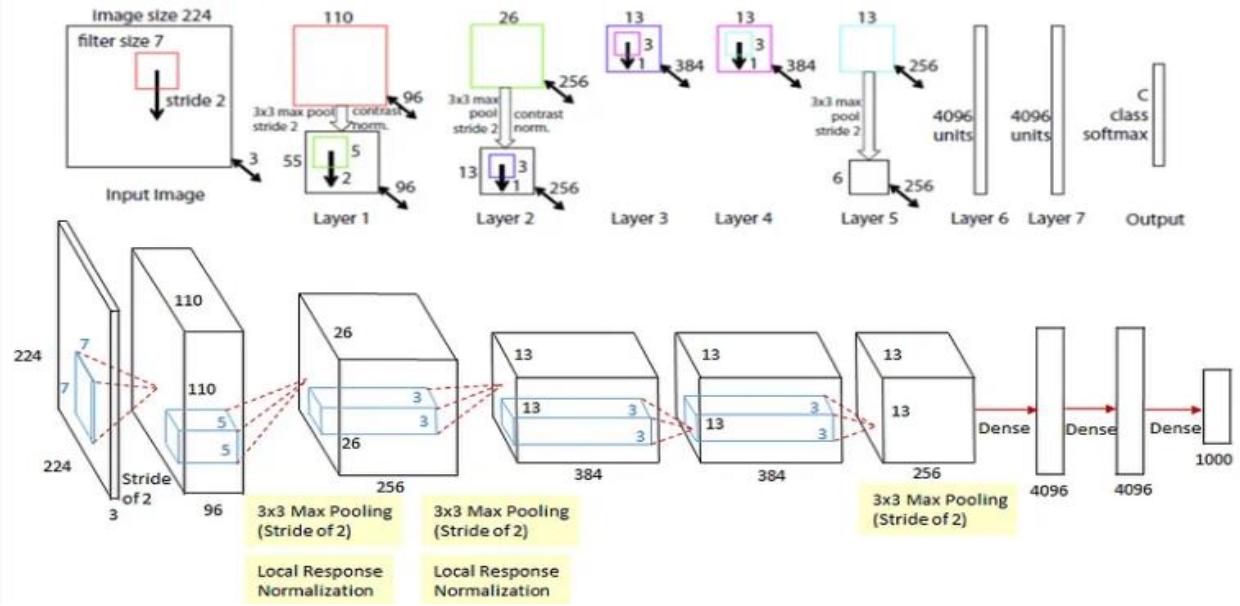


Fig 4.1: ZFnet Architecture

ZFNet is redrawn as the same style of AlexNet for the ease of comparison. To solve the two problems observed in layer 1 and layer 2, ZFNet makes two changes [3].

4.2 What makes ZFnet different from Alexnet?

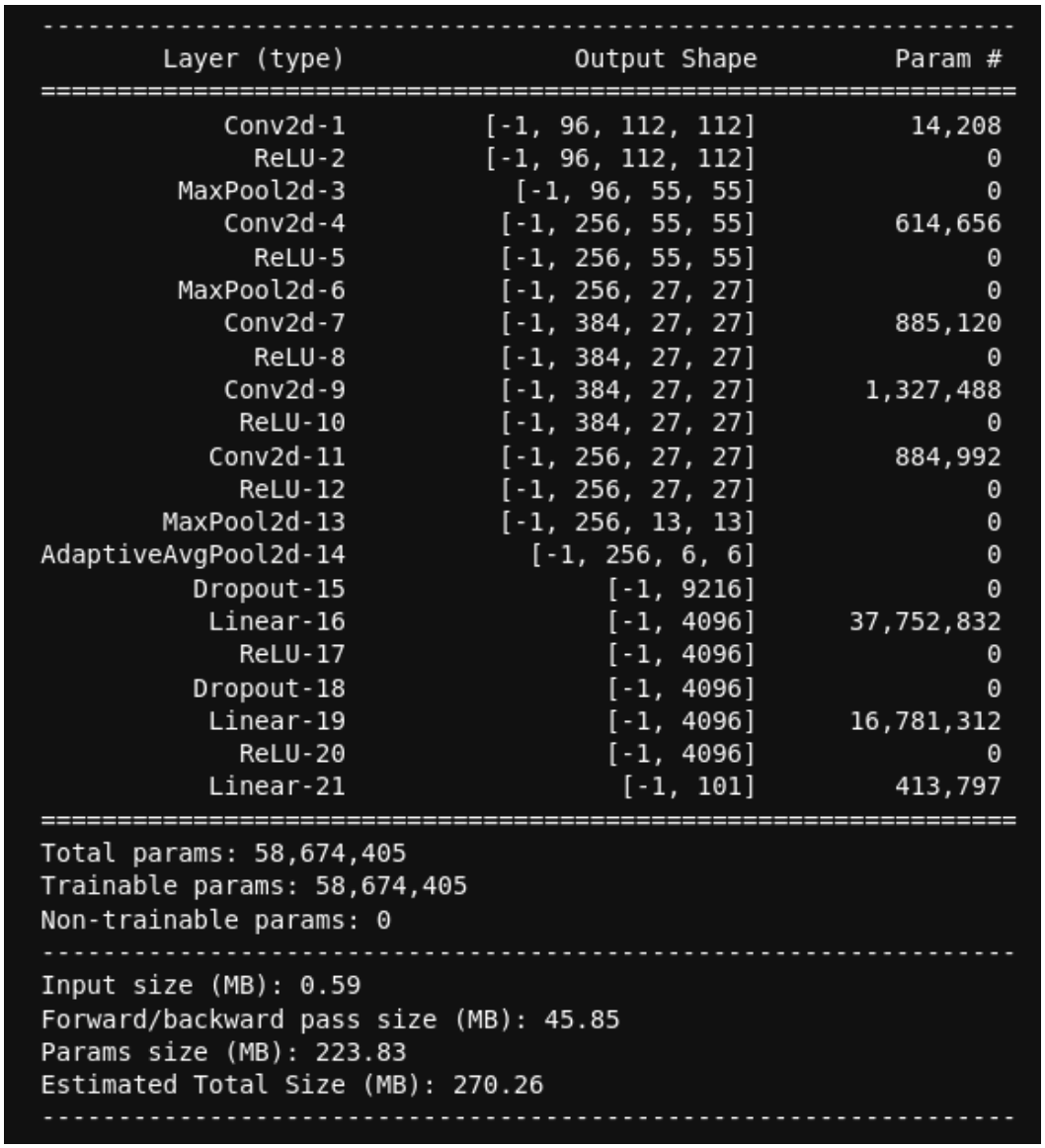
- Architecture Modifications: The main difference is in the changes ZFNet made to AlexNet's architecture for better understanding and visualization. ZFNet modified the first convolutional layer of AlexNet, making the filter size larger (from 11x11 with a stride of 4 in AlexNet to 7x7 with a stride of 2 in ZFNet) and altering the number of filters. These changes were aimed at capturing more image detail by reducing the aggressiveness of the initial convolutional operation, thereby making the model easier to visualize and understand.
- Purpose: While AlexNet was primarily designed to push the boundaries of image classification performance on ImageNet, ZFNet aimed at understanding the inner workings of CNNs, specifically what features and patterns a trained network learns to recognize.

4.3 Impact:

Performance: Although ZFNet made modest adjustments to the architecture, it managed to improve upon AlexNet's performance in the ILSVRC competition, showcasing that even minor architectural tweaks can have significant impacts on model efficacy.

Understanding CNNs: ZFNet's visualization techniques have had a lasting impact on the field, leading to better understanding and interpretability of convolutional networks. This has spurred further research into network architecture design and visualization techniques to make AI models more transparent.

4.4 Neural Network Structure



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 96, 112, 112]          14,208
              ReLU-2         [-1, 96, 112, 112]               0
         MaxPool2d-3           [-1, 96, 55, 55]               0
            Conv2d-4          [-1, 256, 55, 55]         614,656
              ReLU-5          [-1, 256, 55, 55]               0
         MaxPool2d-6          [-1, 256, 27, 27]               0
            Conv2d-7          [-1, 384, 27, 27]         885,120
              ReLU-8          [-1, 384, 27, 27]               0
            Conv2d-9          [-1, 384, 27, 27]       1,327,488
             ReLU-10          [-1, 384, 27, 27]               0
           Conv2d-11          [-1, 256, 27, 27]         884,992
             ReLU-12          [-1, 256, 27, 27]               0
        MaxPool2d-13          [-1, 256, 13, 13]               0
AdaptiveAvgPool2d-14            [-1, 256, 6, 6]               0
          Dropout-15                 [-1, 9216]               0
           Linear-16                 [-1, 4096]      37,752,832
             ReLU-17                 [-1, 4096]               0
          Dropout-18                 [-1, 4096]               0
           Linear-19                 [-1, 4096]      16,781,312
             ReLU-20                 [-1, 4096]               0
           Linear-21                  [-1, 101]         413,797
================================================================
Total params: 58,674,405
Trainable params: 58,674,405
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.59
Forward/backward pass size (MB): 45.85
Params size (MB): 223.83
Estimated Total Size (MB): 270.26
----------------------------------------------------------------
```

Fig 4.2: ZFNet Implementation Layer Summary

Here's a breakdown of the architecture (layer explanation in the architecture code's comments):

Conv2d-1: Convolutional layer with output shape [-1, 96, 112, 112] and 14,208 parameters.

ReLU-2: Rectified Linear Unit activation function with no additional parameters.

MaxPool2d-3: Max pooling layer with no additional parameters.

Conv2d-5: Another convolutional layer with 614,656 parameters.

ReLU-6: Another ReLU activation function.

MaxPool2d-7: Another max pooling layer.

Conv2d-9: Convolutional layer with 885,120 parameters.

ReLU-10: ReLU activation.

Conv2d-11: Convolutional layer with 1,327,488 parameters.

ReLU-12: ReLU activation.

Conv2d-13: Convolutional layer with 884,992 parameters.

ReLU-14: ReLU activation.

MaxPool2d-15: Max pooling layer.

AdaptiveAvgPool2d-16: Adaptive average pooling layer.

Dropout-17: Dropout layer for regularization.

Linear-18: Fully connected layer with 37,752,832 parameters.

ReLU-19: ReLU activation.

Dropout-20: Another dropout layer.

Linear-21: Fully connected layer with 16,781,312 parameters.

ReLU-22: ReLU activation.

Linear-23: Final fully connected layer with 413,797 parameters that likely corresponds to the number of classes the network is meant to predict.

The architecture consists of convolutional layers, interspersed with ReLU activation functions to add non-linearity, max pooling for spatial invariance and dimension reduction, local response normalization for normalization between layers, adaptive average pooling to reduce spatial dimensions to a fixed size, dropout layers for regularization to prevent overfitting, and fully connected layers to combine features for classification. The output shapes change as the data passes through the layers, indicating changes in spatial resolution and the number of feature maps. As is evident from the network summary in Fig. 4.2, the total number of trainable weights for this model comes to 58,674,405.

5. Loss Function

Given that the machine learning task here is multi-class classification, we utilized the cross-entropy loss function as it is widely used in for such problems. It calculates the difference between the predicted probabilities for each class and the actual distribution of labels in the training data telling us how far off we are from the ground truth. Keeping with the usage of PyTorch libraries, we used the *nn.CrossEntropyLoss()* function which we pass to the

train_loop() method. This is the starting point for the training and evaluation loops over the defined number of epochs in the code.

```
[13]: # Define the loss function and the optimizer
      loss_fn = nn.CrossEntropyLoss()
```

Fig 5.1: Code snippet from notebook showing loss function definition

Minimizing the cross-entropy loss during training pushes the model towards generating more accurate class predictions which we do using our optimiser of choice.

```
'''
The loss_fn (function passed as a parameter here) takes the predicted output and the actual output and returns
the difference according to the loss function specified. This is how we can optimise to minimise the error.
'''
loss = loss_fn(pred, y)

# This is the backprop set up.
loss.backward()
```

Fig 5.2: Code snippet from notebook showing the calculation of loss during training

6. Optimiser

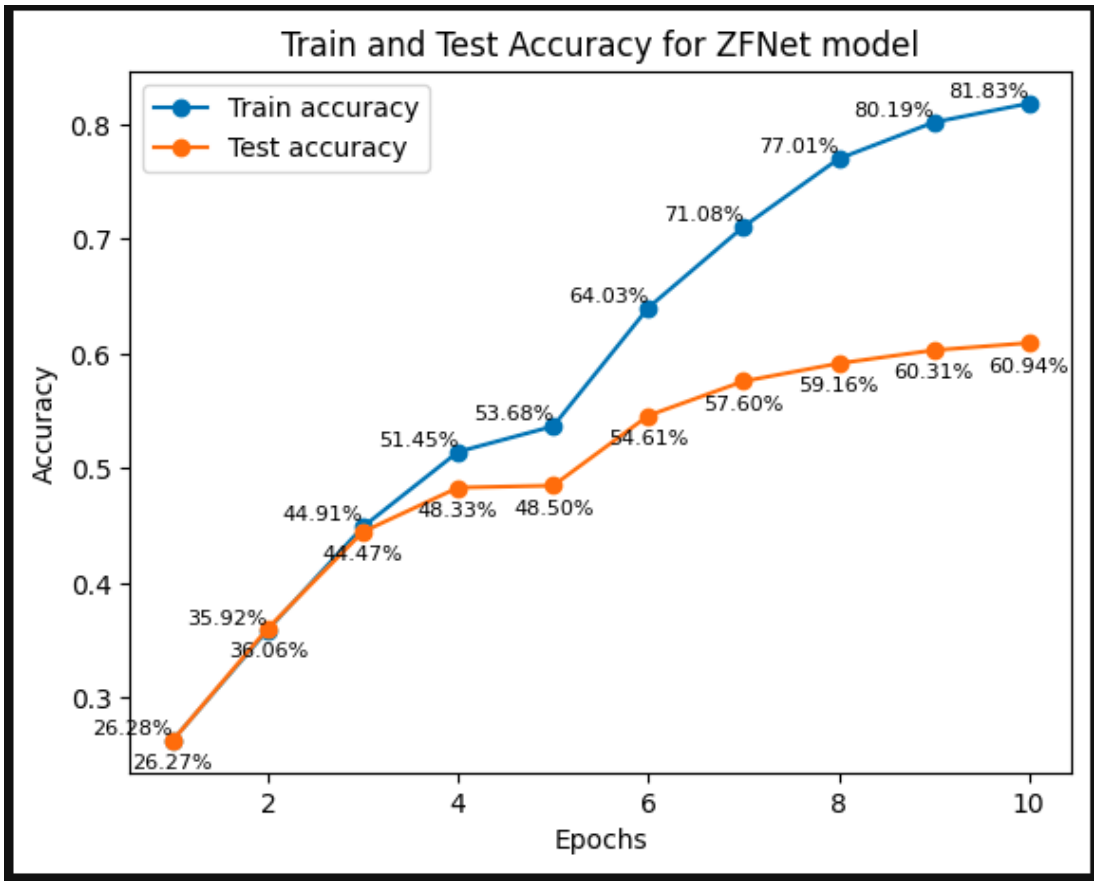
We chose the Adam optimiser for its adaptive learning rate capabilities over Stochastic Gradient Descent (SGD) which has a fixed learning rate. This adaptability can be beneficial for problems with complex loss surfaces, like those encountered in deep neural networks. In the context of the Caltech-101 dataset, Adam can potentially help the model navigate the challenges posed by the class imbalance by adjusting learning rates for parameters associated with underrepresented classes.

7. Results

The application of ZFNet to the Caltech-101 dataset yielded promising results. Despite the initial class imbalance, our pre-processing and data augmentation strategies enabled the model to achieve a high classification accuracy, demonstrating ZFNet's potential in image classification tasks.

7.1. ZFNet Model

The highest testing accuracy score achieved by just directly using the ZFNet model's architecture was 60.94% with a training accuracy of 81.83%.

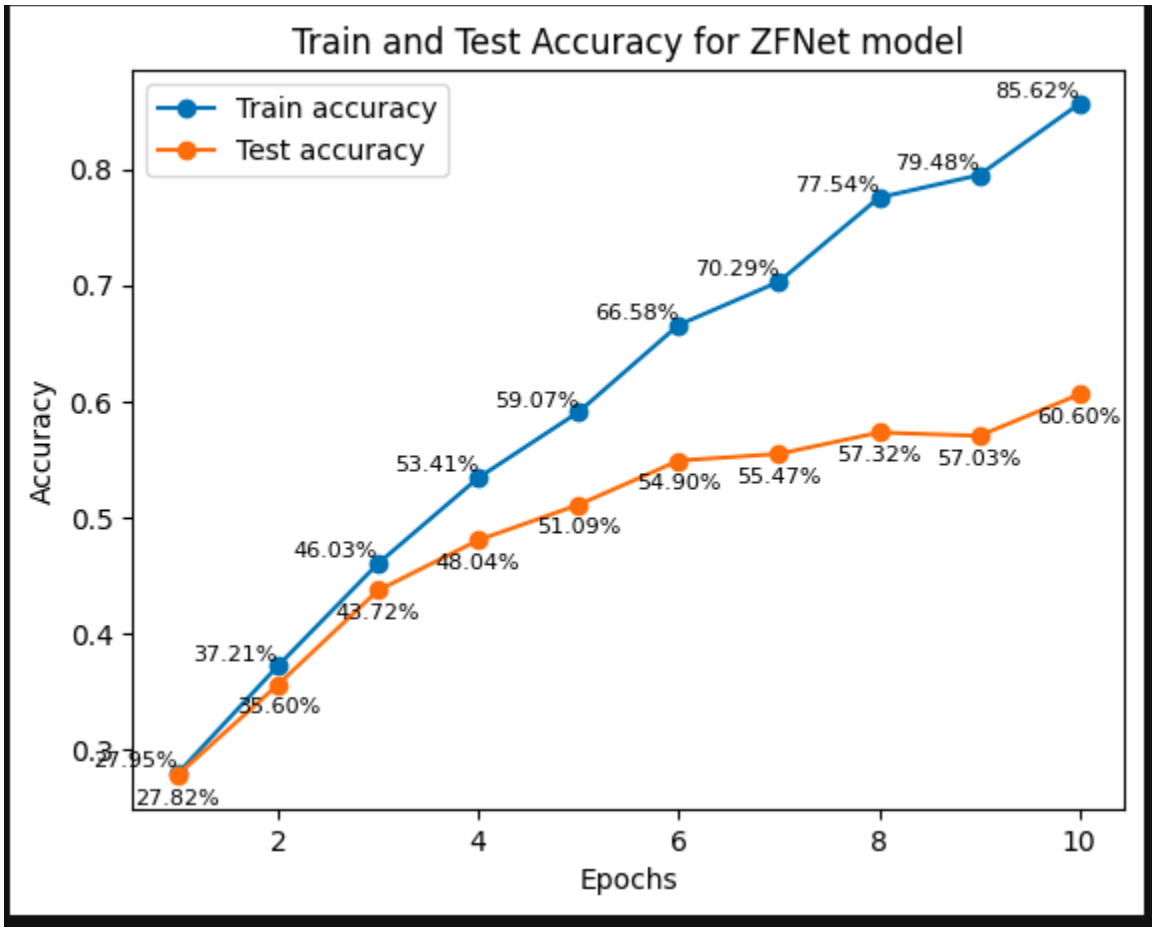


The model has a steady curve with both training and testing accuracies aligning until the 3rd epoch. After this, there is a small disparity between the two till epoch #5. But from here on out, it is obvious that the model is overfitting to the training data as the testing accuracy is not growing nearly as much as the training accuracy. Thus, we already have signs of clear over-fitting (without having to manually engineer it).

7.2. Results from hyper-parameter tuning

To try and improve the model, we tried a couple of things with respect to hyperparameter tuning. One was the normalisation change mentioned in section 3.1. The second was to tweak the drop-off rates in both the dropout layers in the model to try and curb overfitting.

7.2.1 Normalisation changes



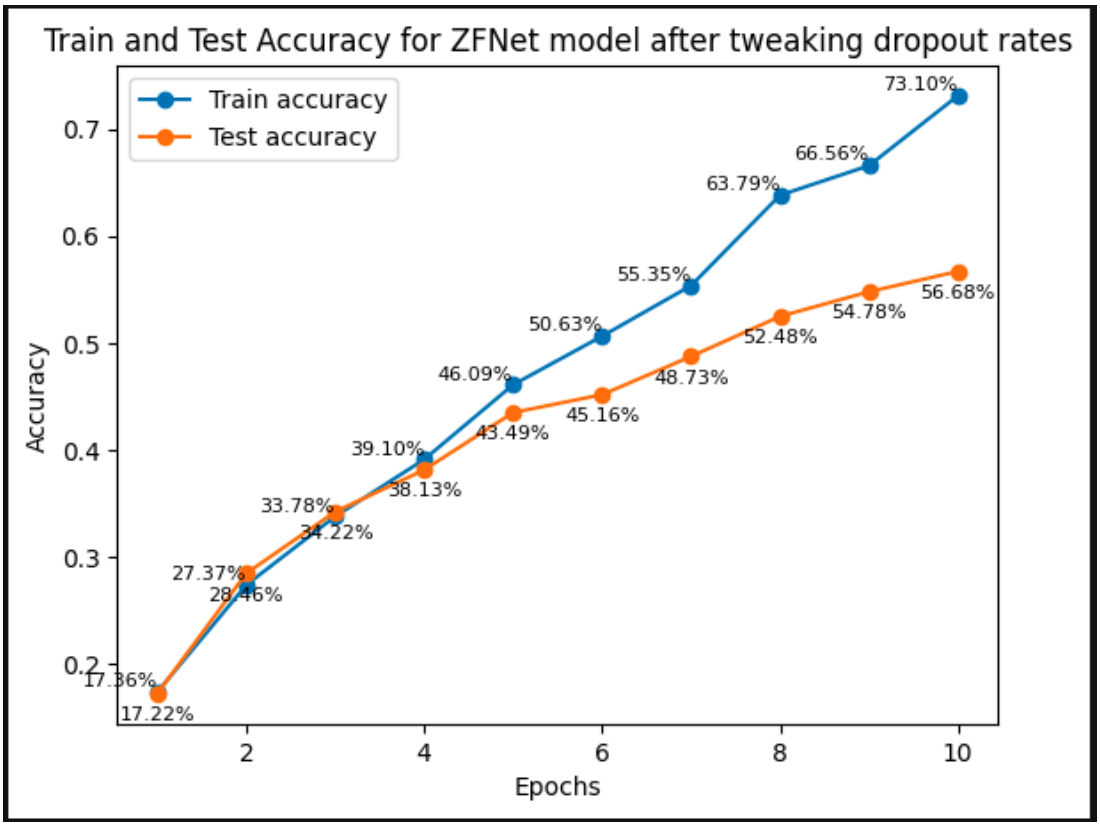
While the training accuracy figures have increased over the same number of epochs to 85.62%, the testing accuracy seems to have fallen a little shy of the maximum accuracy achieved with the run using ImageNet's mean and std deviation figures for normalisation. However, the curves are smoother and seems to indicate a steady climb in its learning. The slight reduction in gap also suggests slight improvement with respect to overfitting.

7.2.2 Tweaking drop-off rate

In the first dropout layer we changed the probability to 0.45 and in the second we changed it to 0.55.

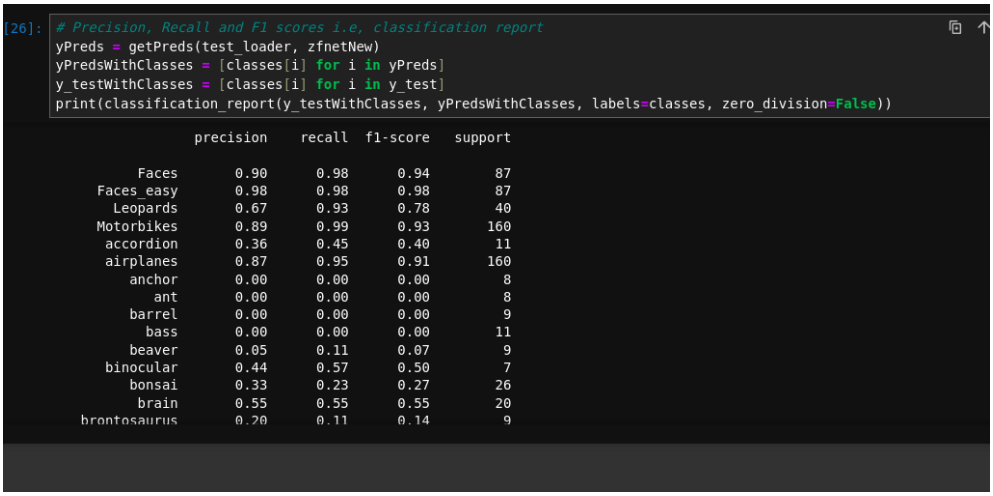
```
self.classifier = nn.Sequential(
    nn.Dropout(p = 0.45), # Decreasing the dropout probablilty to 0.45 instead of default 0.5
    nn.Linear(256 * 6 * 6, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(p = 0.55), # Increasing the dropout probablilty to 0.55 instead of default 0.5
    nn.Linear(4096, 4096),
    nn.ReLU(inplace=True),
    nn.Linear(4096, num_classes),
```

We tried different combinations, but this yielded the best in terms of decrease in overfitting.



As we can clearly see, the gap between the training and testing accuracy curves are visibly decreased. Though this gives an overall reduced training and testing accuracy, it still proves to generalise to new data better than the previous iterations of the model.

7.3. Result evaluation



```
[26]: # Precision, Recall and F1 scores i.e, classification report
yPreds = getPreds(test_loader, zfnetNew)
yPredsWithClasses = [classes[i] for i in yPreds]
y_testWithClasses = [classes[i] for i in y_test]
print(classification_report(y_testWithClasses, yPredsWithClasses, labels=classes, zero_division=False))
```

	precision	recall	f1-score	support
Faces	0.90	0.98	0.94	87
Faces_easy	0.98	0.98	0.98	87
Leopards	0.67	0.93	0.78	40
Motorbikes	0.89	0.99	0.93	160
accordion	0.36	0.45	0.40	11
airplanes	0.87	0.95	0.91	160
anchor	0.00	0.00	0.00	8
ant	0.00	0.00	0.00	8
barrel	0.00	0.00	0.00	9
bass	0.00	0.00	0.00	11
beaver	0.05	0.11	0.07	9
binocular	0.44	0.57	0.50	7
bonsai	0.33	0.23	0.27	26
brain	0.55	0.55	0.55	20
brontosaurus	0.20	0.11	0.14	9

We finally did further evaluation with the metrics and found out that some of the classes have none of the images correctly classified. Examples are classes like anchor, ant, barrel, and bass. They seem to have suffered from an imbalanced number of training examples.

anchor	0.00	0.00	0.00	8
ant	0.00	0.00	0.00	8
barrel	0.00	0.00	0.00	9
bass	0.00	0.00	0.00	11

With prior data engineering like augmenting the data with duplicates of images in different orientations for example and balancing the data samples better, we could have had better results.

8. References

1. https://www.kaggle.com/code/ahmeddheyaa/caltech-101-image-classification?scriptVersionId=153412198&cellId=16
2. https://stackoverflow.com/a/57533806/5584011
3. Review: ZFNet — Winner of ILSVRC 2013 (Image Classification) | by Sik-Ho Tsang | Coinmonks | Medium
4. https://discuss.pytorch.org/t/computing-the-mean-and-std-of-dataset/34949/31
5. https://github.com/vigneshthakkar/Deep-Nets/blob/master/ZFNet.py