

CS7015: Deep Learning (for Computer Vision)

Name: Anoubhav Agarwaal

Roll no: BE16B002

Experiment – 1 (Transfer Learning)

Network architecture modification

- The fully connected layers of the ResNet-18 model were modified.
- ResNet-18 predicts labels for the 1000 classes in ImageNet.
- Whereas, in the CIFAR-10 dataset, we have only ten classes.
- The fully connected layers were removed, and in place of it, a single linear layer was added, which mapped the convolution base output directly to the ten classes.
- This was done to reduce training time and not compromise the performance by more than a few % points.
- Thus, a stack of fully connected layers was not used as this increases the number of trainable parameters (and therefore training time) significantly.

Input image size

- The experiments were first performed on 32x32, i.e., the original size of CIFAR-10 images. This was done because the **epoch duration decreased by 6-7 times** compared to 224x224 input, i.e., the size of the images used for training ResNet-18.
 - With 32x32, there is an **observed drop in performance**. But a broader set of hyperparameters can be explored. The model was trained on 224x224 images using the best performing parameters (based on test accuracy) on 32x32 image input.
-

1. Train the network from scratch with CIFAR-10 (No Pretrained weights)

Parameters used for 32x32

- Epochs = 40
- Learning_rate = [0.01]
- Batch_size = [32]
- Weight_decay = [0.001]
- Pretrained_weights = [[False]]

Results

- Without pretrained weights a maximum **test accuracy of 70.5%** was achieved.
- **Overfitting % was significant**, as seen by the difference in the train and test accuracies in the plot.

Parameters used for 224x224

- Same as 32x32 except epochs were reduced to 20 (due to training time).
- Different combinations of learning rates and weight decay were tried for four epochs each. The learning rate and weight decay, with the best test accuracy, was chosen.
- Coincidentally, they were the same as those determined for 32x32 input.

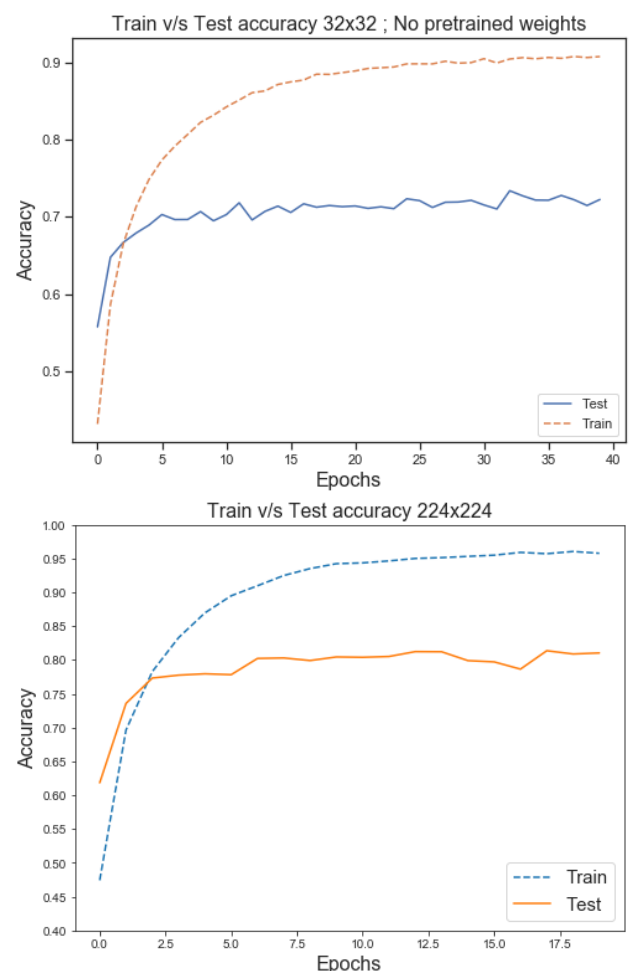


Fig 1. Shows the Train vs. Test accuracy for 32x32 input (top) and 224x224 input (bottom); No pretrained weights.

Results

- Without pretrained weights, maximum **test accuracy of 81.36%** was achieved.
- A massive increase of about **11%** is observed in switching the input image dimensions from 32x32 to 224x224 (keeping all other hyperparameters constant).
- Overfitting % was significant**, as seen by the difference in the train and test accuracies in the plot above.

2. Train the network from scratch with CIFAR-10 (With Pretrained weights)

Parameters used for 32x32 images

- Epochs = 15
- Learning_rate = [0.01, 0.001, 0.0001]
- Batch_size = [32]
- Weight_decay = [0.001, 0.005, 0.01]
- Pretrained_weights = [[True, 'all'], [True, 40], [True, 20], [True, 10]]

All permutations of the above parameters were tried, i.e., $3 \times 3 \times 4 = 36$ total runs. Pretrained_weights parameter takes a list of lists. The first item in the list of lists is a flag variable. It indicates whether to use the pretrained weights or not. If True, the second item represents how many of the last K layers need to be unfreezed/fine-tuned.

Thus, [True, 10] means, use the pretrained weights and only fine-tune the last 10 layers. [True, 'all'] uses the pretrained weights, and all layers are fine-tuned, i.e., all layers have requires_grad = True.

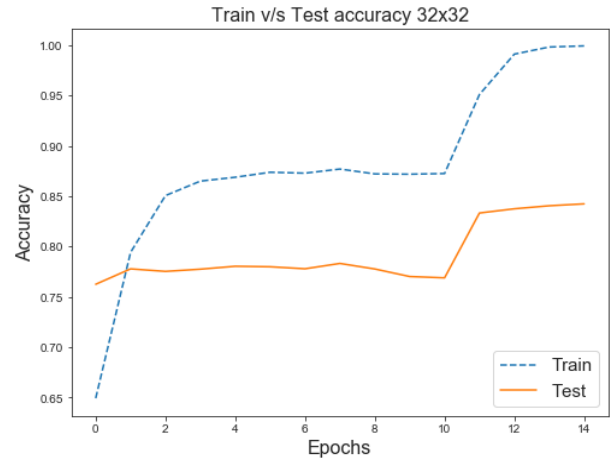


Fig 2. shows the Train vs Test accuracy for 32x32 input with Pretrained weights

Analysis of results for 32x32

1) Best network configurations:

Test accuracy of **84.24%** using a learning rate of 0.001 and a weight decay of 0.01. It was achieved by fine-tuning all the layers. The Overfitting percentage was high.

run	epoch	loss	train_accuracy	test_accuracy	lr	weight_decay	pretrained	overfit%
13	15	0.028581	0.99934	0.8424	0.001	0.01	[True, 'all']	15.694

Fig 3. shows the parameter set for the best test accuracy (for 32x32 images) obtained using pretrained weights of ResNet18.

2) Pretrained_weights:

From Fig4., we observe that when more layers are fine-tuned, higher test accuracy is observed. Unfreezing only the final layers (in the case of [True, 20] and [True, 10]) leads to a significant drop in accuracy. The model has around 60 layers (Conv-Relu-Pool layers are counted separated). On fine-tuning the last 40 out of 60 layers, we get better performance than fine-tuning all the layers. Thus, there is a successful transfer of knowledge captured in the initial 20 layers of the convolutional base. These features are generic and transferable to the CIFAR-10 dataset.

	loss	train_accuracy	test_accuracy	overfit%
pretrained				
[True, 'all']	0.513512	0.82764	0.718524	11.8463
[True, 40]	0.420542	0.86163	0.728659	14.064
[True, 20]	0.541619	0.819787	0.649081	17.8038
[True, 10]	0.871884	0.702904	0.541493	16.7037

Fig 4. The variation of the mean loss, train, and test accuracy and overfit % with pretrained_weights.

3) Weight decay and Learning rate:

The **overfit% drops** significantly with an **increase in weight_decay**. The mean test accuracy across all weight_decay terms is similar. The learning rate of **0.001 performed the best**.

weight_decay					lr				
loss	train_accuracy	test_accuracy	overfit%		loss	train_accuracy	test_accuracy	overfit%	
0.001	0.456761	0.846646	0.665128	18.8411	0.0001	0.552353	0.822263	0.627109	20.2713
0.005	0.608996	0.795252	0.658161	14.3816	0.001	0.365282	0.876621	0.686825	19.5955
0.01	0.694911	0.767073	0.655029	12.0906	0.01	0.843033	0.710087	0.664383	5.44658

Fig 5. The variation of the **mean** loss, train, and test accuracy and overfit % with weight_decay and learning rate.

The performance of the ResNet18 model on 32x32 input images is poor. It only achieved a test accuracy of 84% with around 11 million parameters. In contrast, the best CNN model, built in assignment 2, achieved a **test accuracy of 90%** with only 4.4 million parameters.

run	epoch	loss	train_accuracy	test_accuracy	total parameters	lr	weight_decay	overfit%
6	19	0.03989	0.98764	0.9018	4403518	0.001	0.003	8.584

Fig 6. shows the parameter set for the best test accuracy (32x32 images) obtained using a fewer-layer CNN built earlier in assn-2.

Parameters used for 224x224 images

- Epochs = 15
- Learning_rate = [0.001, 0.0001]
- Batch_size = [32]
- Weight_decay = [0.001]
- Pretrained_weights = [[True, 'all']]

We pick fewer parameters compared to 32x32 due to training time. These are picked based on the results of previous experimentation as explained below.

From Fig. 4, we see that [True, 'all'] and [True, '40'] have superior performance compared to unfreezing only the final 10/20 layers. Hence, we pick only [True, 'all'] in the pretrained weights parameter set.

From Fig. 5, it is evident that **weight decay of 0.001 performed best** on 32x32 images. Hence, only this is used for 224x224.

We **neglect 0.01** learning rate and pick the other two lower learning rates as we are fine-tuning ALL the layers and would not want large gradient updates.

Also, Fig 5. hides the variation of the performance when considering the learning rate AND pretrained weights. This can be seen in Fig 7.

For pretrained = [True, 'all'], the learning rate of 0.01 performs the worst. Thus, the other two learning rates were picked.

		loss	train_accuracy	test_accuracy	overfit%
pretrained	lr				
	0.0001	0.348419	0.890744	0.699156	20.1366
[True, 'all']	0.001	0.296068	0.899163	0.788227	11.8424
	0.01	0.896049	0.693015	0.668189	3.55987
	0.0001	0.942858	0.686085	0.539782	15.183
[True, 10]	0.001	0.571654	0.808491	0.526056	28.6218
	0.01	1.10114	0.614134	0.558642	6.3064
	0.0001	0.540097	0.830339	0.601551	23.5431
[True, 20]	0.001	0.325966	0.88925	0.665544	22.9889
	0.01	0.758794	0.739772	0.680147	6.87947
	0.0001	0.378038	0.881885	0.667949	22.2223
[True, 40]	0.001	0.267439	0.90958	0.767473	14.9292
	0.01	0.616149	0.793426	0.750556	5.04058

Fig 7. The variation of the **mean** loss, train, and test accuracy and overfit % with pretrained_weights and learning rate.

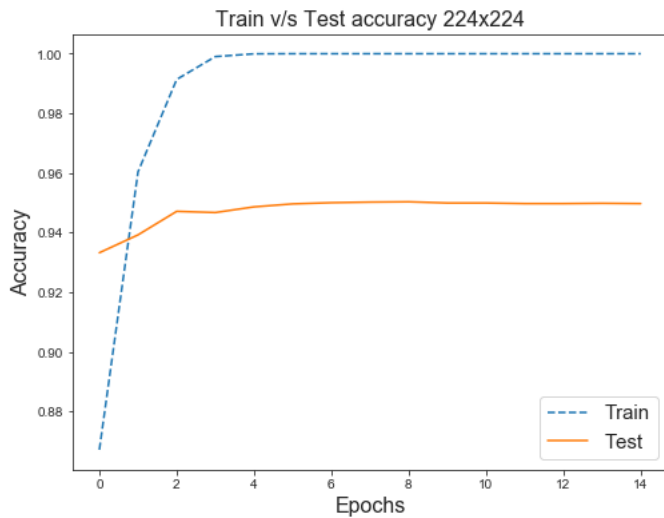
Analysis of results for 224x224

1) Best network configurations:

Test accuracy of 95.03% using a learning rate of 0.001 and a weight decay of 0.001. It was achieved by fine-tuning all the layers. We obtain an overfitting percentage of 5%. A massive increase of about **11%** is observed in switching the input image dimensions from 32x32 to 224x224. (Coincidentally, same as the case with no pre-trained weights).

run	epoch	loss	train_accuracy	test_accuracy	lr	weight_decay	pretrained	overfit%
2	9	0.002082	1.0	0.9503	0.001	0.001	[True, 'all']	4.97

Fig 8. shows the parameter set for the best test accuracy (for 224x224 images) obtained using pretrained weights of ResNet18



	loss	train_accuracy	test_accuracy	overfit%
lr				
0.0001	0.207009	0.941321	0.92334	3.0356
0.001	0.0412242	0.987853	0.947573	4.46893

Fig 9. shows the Train vs. Test accuracy for 224x224 input with Pretrained weights (left) and the effect of learning rate (right)

The learning rate of 0.001 performs better than 0.0001 on average (in terms of test accuracy), as seen in Fig 9(right). The train and test accuracy plateau early on at around the fourth epoch at 100% and 95%, respectively.

Experiment – 2 (Data Augmentation)

2.1 Train the network from scratch with Tiny-CIFAR-10 (No Pretrained weights)

2.2 Train the network with Tiny-CIFAR-10 (Using Pretrained weights)

The **same network architecture as experiment 1** is used to answer questions 2.1 and 2.2. This is for making a fair comparison between the performance in the two datasets (CIFAR-10 and Tiny-CIFAR-10). The main difference is that in this experiment, the **size of the dataset is 1/10th of what was used in Experiment-1.**

Parameters used for 32x32 image input (2.1)

- Epochs = 80
- Learning_rate = [0.01]
- Batch_size = [32]
- Weight_decay = [0.001]
- Pretrained_weights = [[False]]

Three augmentation schemes were tried, as seen in **Fig 10 (below)**. The general idea being that Aug 3 is more complex (in terms of no. of transforms) than 2, which is more complex than 1. The bold text represents the changes.

Fig 12. shows the train vs. test accuracy for the best augmentation scheme. Surprisingly, it was observed that the simplest augmentation, **Aug 1, performed the best** (as seen in **Fig 11**).

From **Fig. 11 and Fig. 12**, we observe that without pretrained weights and an image input of 32x32, training on the tiny-CIFAR-10 dataset gives us a **maximum test accuracy of 55.61%** using augmentation scheme 1. The overfit% is massive (almost 45% for the best case).

We observed a test accuracy of 70.5% (for 32x32 input; No pretrained weights) on using the entire CIFAR-10 dataset. With tiny-CIFAR-10 (10% of CIFAR-10) and data augmentation, we achieved 55.61%. There is a **vast 15% gap in test accuracy**, which we couldn't compensate using data augmentation.

Fig 10. The three image augmentation schemes tried to increase the size of the Tiny-CIFAR-10 dataset artificially.

<pre>train_transform = transforms.Compose([transforms.Resize(32), transforms.RandomHorizontalFlip(p=0.5), transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=.05, hue=.05), transforms.ToTensor()])</pre>	Aug 1
<pre>train_transform = transforms.Compose([transforms.Resize(40), transforms.RandomCrop(32), transforms.RandomHorizontalFlip(p=0.5), transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=.05, hue=.05), transforms.ToTensor()])</pre>	Aug 2
<pre>train_transform = transforms.Compose([transforms.Resize(40), transforms.RandomCrop(32), transforms.RandomHorizontalFlip(p=0.5), transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=.05, hue=.05), transforms.RandomRotation(20, resample=PIL.Image.BILINEAR), transforms.ToTensor()])</pre>	Aug 3

Type	train_accuracy		test_accuracy	
	mean	max	mean	max
aug1	0.824485	0.9988	0.485316	0.5561
aug2	0.795754	1	0.392362	0.4374
aug3	0.574892	0.7498	0.37953	0.433

Fig 11. The mean train and test accuracies for the three image augmentation schemes to increase the size of the Tiny-CIFAR-10 dataset artificially; No pretrained weights; 32 x 32 input.

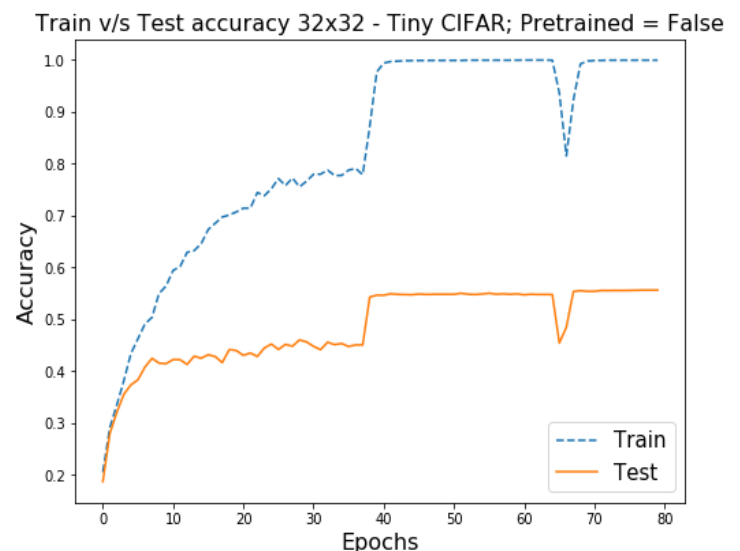


Fig 12. shows the Train vs. Test accuracy for the best augmentation scheme, i.e., **Aug 1**;

The huge overfit% suggests that the augmentations made on the reduced training set could not capture the underlying distribution of the test set. Also, the size of the training set is too small, and the model complexity of ResNet-18 with almost 12 million parameters is too large. This led to an almost perfect train accuracy of 100% and loss of generalization to the test set.

My hypothesis for why Aug 3 performs worse than Aug 1 is that rotated images are mostly not present in the test set. Thus, **introducing augmentations that are not present in the test set worsens the performance of the model.**

I chose not to do any **test time augmentation** (TTA) (e.g., combining prediction of K random crops for each test image) as the effect of augmenting the training dataset would be tougher to observe. Test time augmentation would improve accuracy even further. However, it defeats the purpose of the experiment.

Due to the inferior performance of 32x32 input images on Tiny-CIFAR-10, time constraints, and overall superior performance of 224x224 inputs, **I decided not to experiment with 32x32 image input for questions 2.2 and 2.3.** Thus, I only used 224x224 input for questions 2.2 and 2.3.

Parameters used for 224x224 (2.1 and 2.2)

- Epochs = 80
- Learning_rate = [0.01, 0.001]
- Batch_size = [64]
- Weight_decay = [0.005, 0.001, 0.0005]
- Pretrained_weights = [[False], [True, 40], [True, 'all']]

The pretrained_weights parameter set consists of both False (2.1) and True (2.2) case. **Out of memory errors** was obtained after executing the script for a long time. Thus, **not all permutations** of the above parameters were tried. Results of only 10 out of the 18 possible permutations were obtained.

Also, based on the results of the 32x32 inputs, it was decided to test **only Aug 1 scheme** (refer Fig. 10) for image augmentation as Aug 1 scheme had an almost **10% higher test accuracy** compared to the other two schemes.

Analysis of results for 224x224

1) Best network configurations:

run	epoch	loss	train_accuracy	test_accuracy	lr	weight_decay	pretrained	overfit%
1	25	0.013943	1.0	0.598	0.01	0.005	[False]	40.2

run	epoch	loss	train_accuracy	test_accuracy	lr	weight_decay	pretrained	overfit%
2	22	0.005633	1.0	0.8985	0.01	0.005	[True, 'all']	10.15

Fig 13. shows the parameter set for the best test accuracy (for 224x224 images) obtained **from scratch (top; 2.1)** and **using pretrained weights (bottom; 2.2)** on the Tiny-CIFAR-10 dataset with image augmentation.

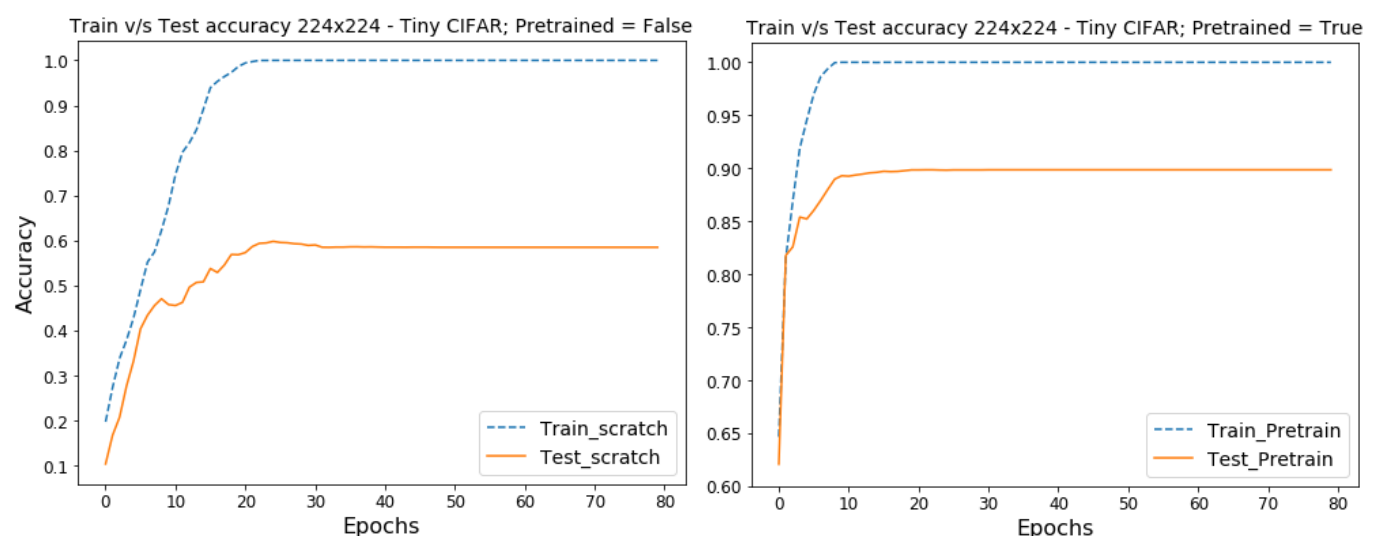


Fig 14. shows the train vs. test accuracy (for 224x224 images) obtained **from scratch (left)** and **using pretrained weights (right)** on the Tiny-CIFAR-10 dataset with image augmentation. Note: the y-axis ranges are different.

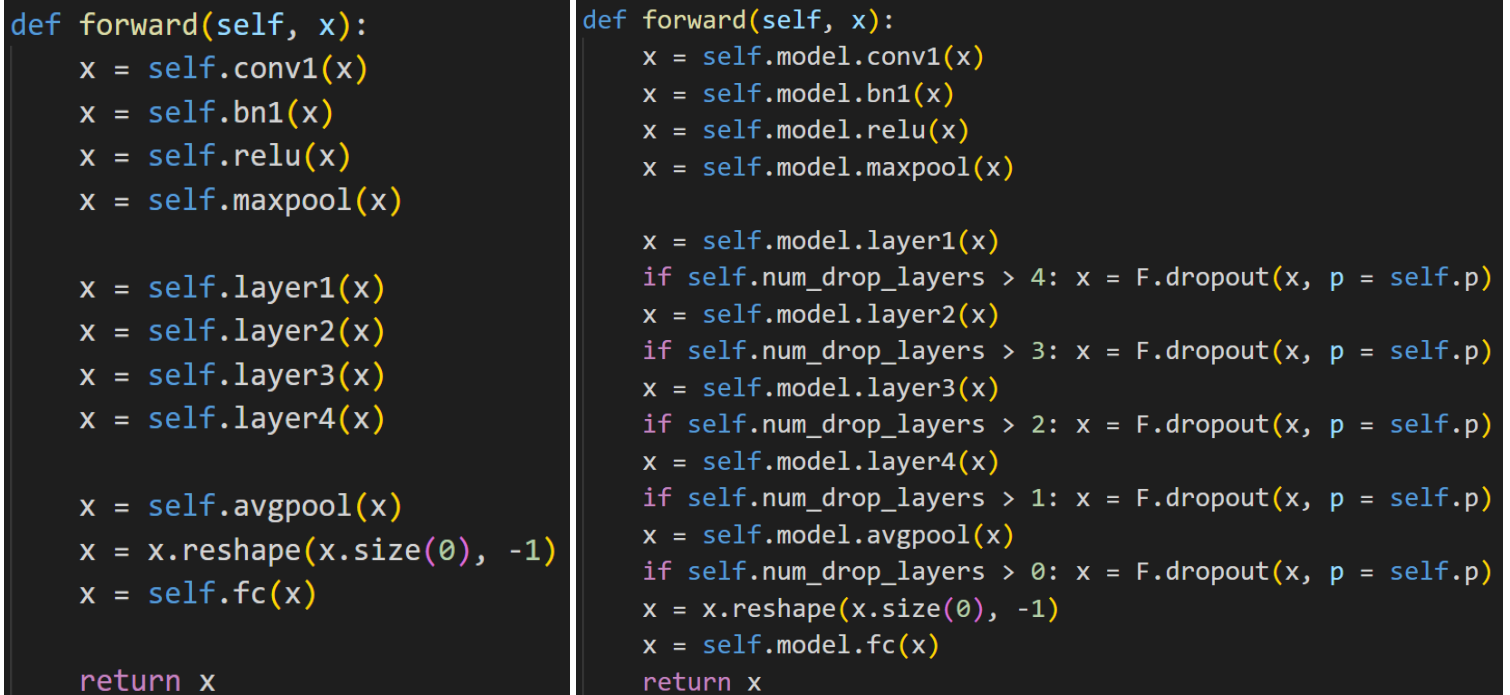
From **Fig. 13 and 14**, It is evident that having pretrained weights plays a critical role in obtaining a high-test accuracy. Thus, **transfer learning has immense significance while training on the Tiny-CIFAR-10 dataset.** There is a **30% difference in test accuracies** when the model is trained from scratch (60%) and using pretrained weights (90%).

Without pretrained weights and only image augmentation, we achieve a maximum test accuracy of **60% in Tiny-CIFAR-10**. It is inferior compared to the **84% obtained using no pretrained weights on CIFAR-10**.

Also, with pretrained weights + image augmentation, we achieve a test accuracy of **90% in Tiny-CIFAR-10**. Whereas, with only pretrained weights trained on the **CIFAR-10 dataset**, we achieved **95% test accuracy**. Thus, even though the size of the dataset was diminished by 10X times, image augmentation was able to reduce **the difference in performance to only 5%**. On employing TTA, this gap can be further reduced.

2.3 Trying Dropout after different layers

Fig 15. (left) The forward function in **resnet.py** looks like the following. It was modified to include **at most five dropout layers** (user-defined parameter), each having the same **dropout rate of p** (user-defined parameter).



```
def forward(self, x):
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)

    x = self.avgpool(x)
    x = x.reshape(x.size(0), -1)
    x = self.fc(x)

    return x
```

```
def forward(self, x):
    x = self.model.conv1(x)
    x = self.model.bn1(x)
    x = self.model.relu(x)
    x = self.model.maxpool(x)

    x = self.model.layer1(x)
    if self.num_drop_layers > 4: x = F.dropout(x, p = self.p)
    x = self.model.layer2(x)
    if self.num_drop_layers > 3: x = F.dropout(x, p = self.p)
    x = self.model.layer3(x)
    if self.num_drop_layers > 2: x = F.dropout(x, p = self.p)
    x = self.model.layer4(x)
    if self.num_drop_layers > 1: x = F.dropout(x, p = self.p)
    x = self.model.avgpool(x)
    if self.num_drop_layers > 0: x = F.dropout(x, p = self.p)
    x = x.reshape(x.size(0), -1)
    x = self.model.fc(x)

    return x
```

Fig 15. (left) Shows the unchanged forward function in **resnet.py**, **(right)** shows the modification in the forward function to incorporate Dropout after different layers. The number of dropout layers and the dropout rate can be set using the user-defined parameters **num_drop_layers** and **p**, respectively.

Parameters used for 224x224 (2.3)

- Epochs = 30
- Learning_rate = [0.001]
- Batch_size = [32]
- Weight_decay = [0.001]
- Pretrained_weights = [[False], [True, 'all']]
- Num_drop_layers = [1, 2, 3, 4, 5]
- Dropout_rate = [0.1, 0.2, 0.3]

The **learning rate and weight decay** were experimentally determined by trying out combinations of learning rates and weight decay for 4 epochs each. The learning rate and weight decay with the best test accuracy was chosen. This was done to reduce the total number of permutations of the parameter sets.

Due to the Out of Memory (OOM) errors faced in experiment 2.1 and 2.2, the **batch size** was restricted to 32. For the **Pretrained_weights** parameter, we try [False] and [True, 'all'] due to the significant role played by pretrained weights in the previous experiment (where we observed a 30% difference in test accuracy).

Aug 1 image augmentation scheme is used.

The number of dropout layers in the forward pass depends on the **Num_drop_layers** parameter. When set to five, all dropout layers are used. When set to three, the last three dropout layers are used. (Refer Fig 15 (right)).

The dropout rate for each layer is controlled by the **Dropout_rate** parameter. We test the rates of 0.1, 0.2, and 0.3. With high dropout rates (>0.7), it was observed that learning barely occurred, with the test accuracy hovering around 10-20% (i.e., close to random guessing). Thus, only the lower dropout rates were tried.

All permutations lead to a total of $5 \times 3 = 15$ parameter sets for this experiment.

Analysis of results for 224x224

1) Best network configurations:

Test accuracy of 90.56% is obtained using Dropout + image augmentation. This is a **0.71% increase** compared to just using image augmentation (experiment 2.2). The best configuration uses two dropout layers, each having a dropout rate of 0.1.

Overfit% was high. This is a recurring event on using the Tiny-CIFAR-10 dataset for training and the original test set for testing. After trying out heavy regularization using Image augmentation and Dropout, the only possible explanation for this overfit% is that the training set does not capture a significant proportion of the examples observed in the test set. In simple terms, a model trained in detecting blue cars might not perform well on detecting red ones. Thus, the training set might not contain enough information to predict some examples in the test set accurately. Thus, I hypothesize that **the issue is not in the model complexity per se; it is instead in the data input to the model.** The underlying distribution of the training set and test set are different.

We obtained a **95.03% test accuracy** using pretrained weights on the **CIFAR-10 dataset** with no image augmentation or dropout. The learning rate, weight decay, and batch sizes for both the best runs, i.e., (top) Tiny-CIFAR-10 and (bottom) CIFAR-10, are the same.

There is a **4.5% test accuracy gap** between the two experiments (**2.3 and 1.2**). The top was trained on only 10% of the dataset of the bottom one.

run	epoch	loss	train_accuracy	test_accuracy	p	num_drop_layers	pretrained	overfit%
2	25	0.00029	1.0	0.9056	0.1	2	[True, 'all']	9.44

run	epoch	loss	train_accuracy	test_accuracy	lr	weight_decay	pretrained	overfit%
2	9	0.002082	1.0	0.9503	0.001	0.001	[True, 'all']	4.97

Fig 16. (top; 2.3) shows the parameter set for the best test accuracy (for 224x224 images) obtained using pretrained weights on the Tiny-CIFAR-10 dataset with image augmentation + Dropout. **(bottom; 1.2)** shows the best test accuracy obtained using pretrained weights on the CIFAR-10 dataset with no image augmentation or Dropout

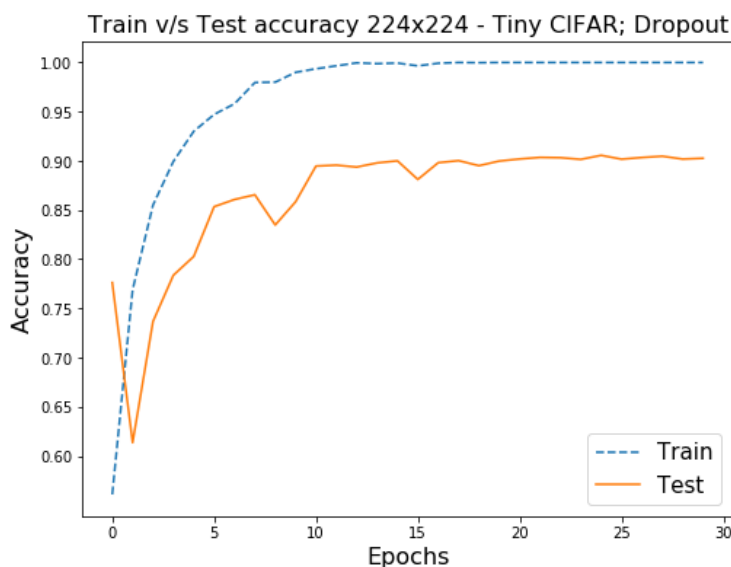


Fig 17. Shows the train vs. test accuracy (for the best run) obtained using pretrained weights on the Tiny-CIFAR-10 dataset with image augmentation + Dropout.

2) Pretrained_weights:

From **Fig. 18**, It is once again evident that having pretrained weights plays a critical role in obtaining a high-test accuracy. Thus, **transfer learning has immense significance while training on the Tiny-CIFAR-10 dataset**. There is an almost **50% difference** in test accuracies when the model is trained from **scratch (41%)** and using **pretrained weights (90.5%)**.

	train_accuracy		test_accuracy		overfit%	
	mean	max	mean	max	mean	max
pretrained						
[False]	0.395038	0.6614	0.31469	0.4094	8.17533	31.43
[True, 'all']	0.949471	1	0.850896	0.9056	10.5671	18.75

Fig 18. shows the effect of using **pretrained weights** on the Tiny-CIFAR-10 dataset with image augmentation + Dropout.

3) Number of dropout layers:

From **Fig. 18**, We observe the stark difference in test accuracies when pretrained weights are used/neglected.

In the mean train_accuracy column, we observe that on increasing the num_drop_layers from 1 to 5, there is a drop-in train accuracy as expected. However, that drop is not gained in the test accuracies.

When **pretrained = False**, the model is unable to learn after a certain point. The mean test accuracies for the different number of dropout layers are stuck around 31%.

The model gets stuck in local minima when pretrained is set to False. Hence, the mean training accuracy lies in the 35-44% range.

pretrained	num_drop_layers	train_accuracy	test_accuracy	overfit%
[False]	1	0.437358	0.317889	12.052
	2	0.427887	0.322452	10.722
	3	0.392809	0.310332	8.33022
	4	0.368133	0.309381	5.94544
	5	0.349004	0.313393	3.827
[True, 'all']	1	0.958789	0.863848	10.2064
	2	0.957478	0.862017	10.2388
	3	0.950647	0.851899	10.6506
	4	0.944433	0.844872	10.6783
	5	0.936007	0.831844	11.0614

Fig 19. Shows the effect of using **pretrained weights and num_drop_layers** on the Tiny-CIFAR-10 dataset with image augmentation + Dropout. This effect is observed using the **mean** train accuracy, test accuracy, and overfit%.

This is not attributed to low model complexity as we can see the model performing well when pretrained weights are used. Thus, **the issue is in the optimization and entrapment in local minima when Pretrained = False**.

When **pretrained = True**, on increasing the number of dropout layers, both the mean train and test accuracies drop steadily. Thus, the overfit% is not reduced.

4) Dropout rate

Due to the poor performance of the model when Pretrained is set to False, we observe the effect of dropout rate only for Pretrain is True, i.e., pretrained weights are used.

From **Fig. 20**, We see that the **dropout rate of 0.1 performs the best**. From **Fig. 21**, We see that for a fixed number of dropout layers, the mean train and test accuracies decrease on increasing the dropout rate.

In all cases, the **mean overfit% is significant ~9-11%**. Also, the overfit% does not follow any decreasing trend on increasing the dropout rate or the number of dropout layers.

As stated earlier, I hypothesize that **the issue is not in the model complexity per se; it is instead in the data input to the model** and its mismatch with the test dataset.

p	train_accuracy		test_accuracy		overfit%	
	mean	max	mean	max	mean	max
0.1	0.957212	1	0.86152	0.9056	10.2852	18.75
0.2	0.950721	1	0.8531	0.9046	10.3754	17.49
0.3	0.940479	1	0.838068	0.9005	11.0407	16.45

Fig 20. Shows the effect of using different dropout rates using pretrained weights on the Tiny-CIFAR-10 dataset with image augmentation + Dropout.

num_drop_layers	p	train_accuracy		test_accuracy		overfit%
1	0.1		0.96112	0.868497	9.86367	
	0.2		0.95946	0.865167	10.1603	
	0.3		0.955787	0.85788	10.5953	
2	0.1		0.961753	0.865727	10.3207	
	0.2		0.958187	0.866527	9.616	
	0.3		0.952493	0.853797	10.7797	
3	0.1		0.958633	0.861863	10.4423	
	0.2		0.9499	0.85284	10.445	
	0.3		0.943407	0.840993	11.0643	
4	0.1		0.954527	0.861187	10.0733	
	0.2		0.945307	0.84623	10.601	
	0.3		0.933467	0.8272	11.3607	
5	0.1		0.950027	0.850327	10.726	
	0.2		0.940753	0.834737	11.0547	
	0.3		0.91724	0.81047	11.4037	

Fig 21. Shows the effect of using a different number of dropout layers and dropout rates using pretrained weights on the Tiny-CIFAR-10 dataset with image augmentation + Dropout.

Summary			Best test accuracy (%)	
Experiment	Dataset	Description	32x32	224x224
1.1	CIFAR - 10	From scratch	70.5	81.36
1.2	CIFAR - 10	Pretrained	84.24	95.93
2.1	Tiny-CIFAR-10	From scratch + DA	55.61	59.8
2.2	Tiny-CIFAR-10	Pretrained + DA	-	89.95
2.3	Tiny-CIFAR-10	Pretrained + DA + DO	-	90.56

Fig 22. Shows the best test accuracies for each experiment.

THE END