Tiny Imagenet -

1. **Network Design (best accuracy)**

5 convolutional layers with ReLU activation, kernel_sizes of 5 and 2 with stride=1, padding=2 followed by a batch normalization layer and a max-pooling layer as shown in architecture below except for the last conv layer.
Also, there are 2 fully connected layers - first with dropout of 0.5 and ReLU activation followed by batch normalization

```
self.conv_1 = nn.Conv2d(3, 128, kernel_size=5, stride=1, padding=2)
self.relu_1 = nn.ReLU(True);
self.batch_norm_1 = nn.BatchNorm2d(128);
self.pool_1 = nn.MaxPool2d(kernel_size=2, stride=2)


self.conv_2 = nn.Conv2d(128, 128, kernel_size=5, stride=1, padding=2)
self.relu_2 = nn.ReLU(True);
self.batch_norm_2 = nn.BatchNorm2d(128);
self.pool_2 = nn.MaxPool2d(kernel_size=2, stride=2)

self.conv_3 = nn.Conv2d(128, 128, kernel_size=5, stride=1, padding=2)
self.relu_3 = nn.ReLU(True);
self.batch_norm_3 = nn.BatchNorm2d(128);
self.pool_3 = nn.MaxPool2d(kernel_size=2, stride=2)

self.conv_4 = nn.Conv2d(128, 128, kernel_size=2, stride=1, padding=2)
self.relu_4 = nn.ReLU(True);
self.batch_norm_4 = nn.BatchNorm2d(128);
self.pool_4 = nn.MaxPool2d(kernel_size=2, stride=2)

self.conv_5 = nn.Conv2d(128, 128, kernel_size=5, stride=1, padding=2)
self.relu_5 = nn.ReLU(True);
self.batch_norm_5 = nn.BatchNorm2d(128);

self.fc_1 = nn.Linear(3200, 1024);
self.relu_6 = nn.ReLU(True);
self.batch_norm_6 = nn.BatchNorm1d(1024);
self.dropout_1 = nn.Dropout(p=0.5);

self.fc_2 = nn.Linear(1024, 200);
```

Following data augmentations used:

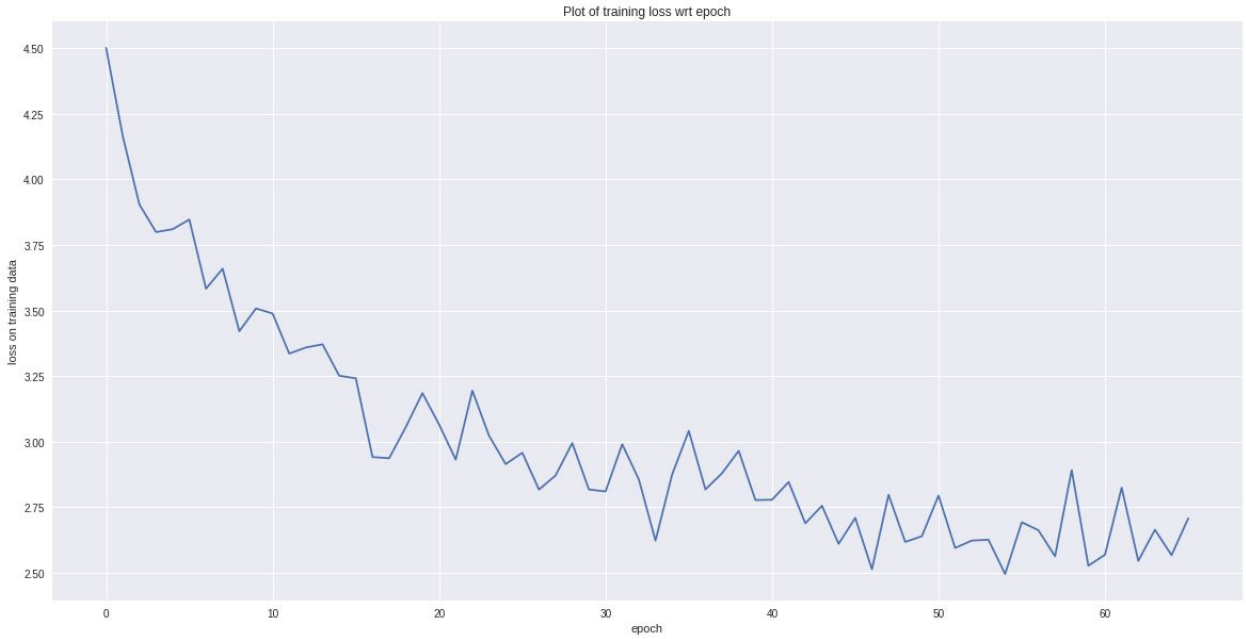
Train data - ToPILImage(),
RandomResizedCrop(64),
RandomHorizontalFlip(),
RandomRotation(18),
ToTensor(),
Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

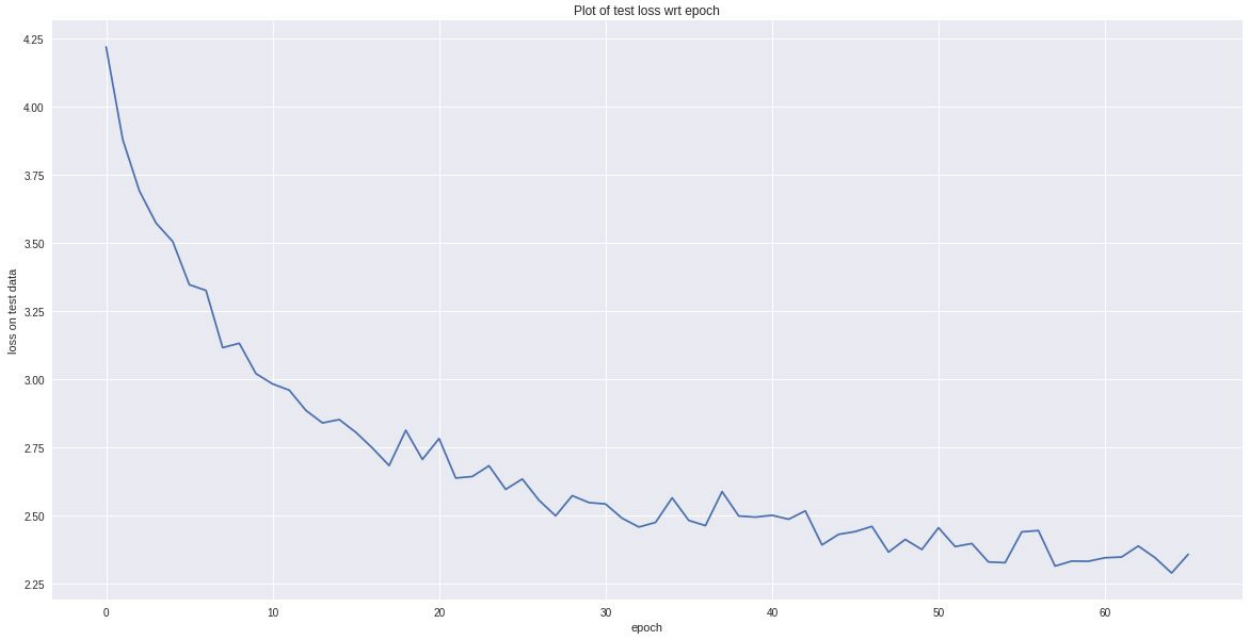
Test data - ToPILImage(),
Resize(64),
ToTensor(),
Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

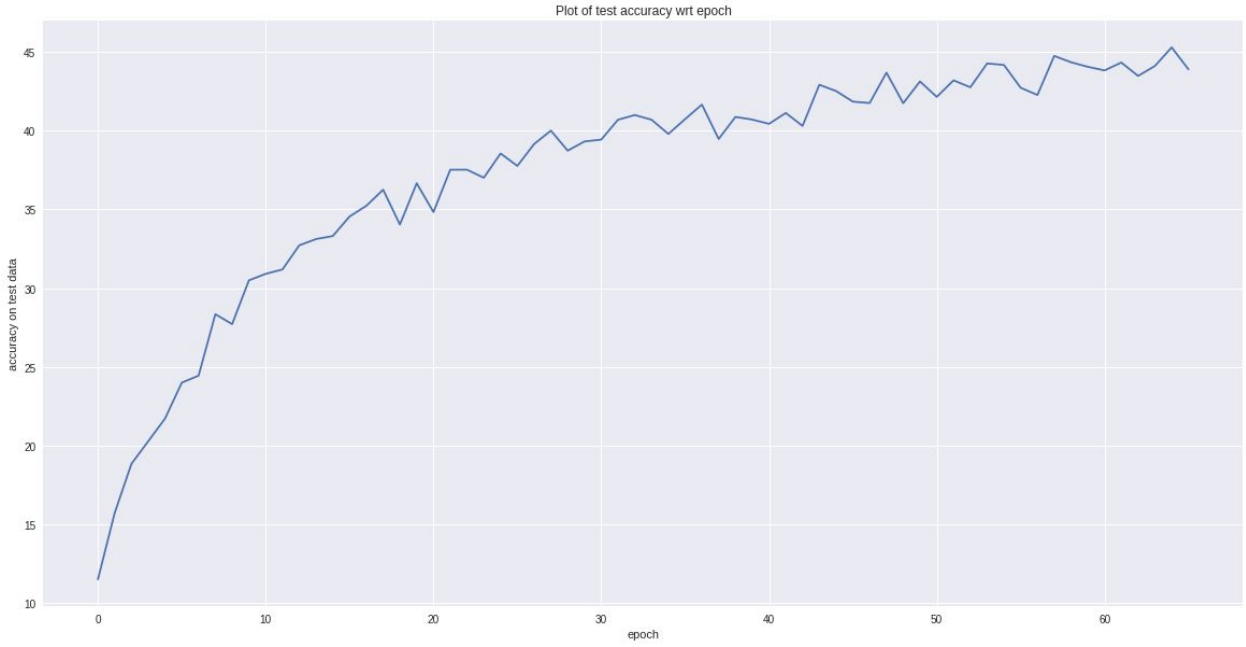
Hyperparameters -

BATCH_SIZE = 256
EPOCHS = 200 (ended at 65 due to spurious environment reset)
LEARNING_RATE = 0.01
MOMENTUM = 0.9
WEIGHT_DECAY = 0.0005
Loss - cross entropy loss
Optimizer - stochastic gradient descent

After 65 epochs,
Training loss = 2.706642
Test loss = 2.3579
Test Accuracy = 43.89%







2. <u>Network design (with less accuracy) -</u>

5 convolutional layers with ReLU activation, kernel_size of 5 with stride=1, padding=2 followed by a batch normalization layer and a max-pooling layer as shown in architecture below except for the last conv layer.
Also, there are 2 fully connected layers - first with dropout of 0.5 and Softmax activation followed by batch normalization

```
self.conv_1 = nn.Conv2d(3, 64, kernel_size=5, stride=1, padding=2)
self.relu_1 = nn.ReLU(True);
self.batch_norm_1 = nn.BatchNorm2d(64);
self.pool_1 = nn.MaxPool2d(kernel_size=2, stride=2)


self.conv_2 = nn.Conv2d(64, 128, kernel_size=5, stride=1, padding=2)
self.relu_2 = nn.ReLU(True);
self.batch_norm_2 = nn.BatchNorm2d(128);
self.pool_2 = nn.MaxPool2d(kernel_size=2, stride=2)

self.conv_3 = nn.Conv2d(128, 128, kernel_size=5, stride=1, padding=2)
self.relu_3 = nn.ReLU(True);
self.batch_norm_3 = nn.BatchNorm2d(128);
self.pool_3 = nn.MaxPool2d(kernel_size=2, stride=2)

self.conv_4 = nn.Conv2d(128, 128, kernel_size=5, stride=1, padding=2)
self.relu_4 = nn.ReLU(True);
self.batch_norm_4 = nn.BatchNorm2d(128);
self.pool_4 = nn.MaxPool2d(kernel_size=2, stride=2)

self.conv_5 = nn.Conv2d(128, 128, kernel_size=5, stride=1, padding=2)
self.relu_5 = nn.ReLU(True);
self.batch_norm_5 = nn.BatchNorm2d(128);

self.fc_1 = nn.Linear(512, 400);
self.relu_6 = nn.Softmax(1);
self.batch_norm_6 = nn.BatchNorm1d(400);
self.dropout_1 = nn.Dropout(p=0.5);

self.fc_2 = nn.Linear(400, 200);
```

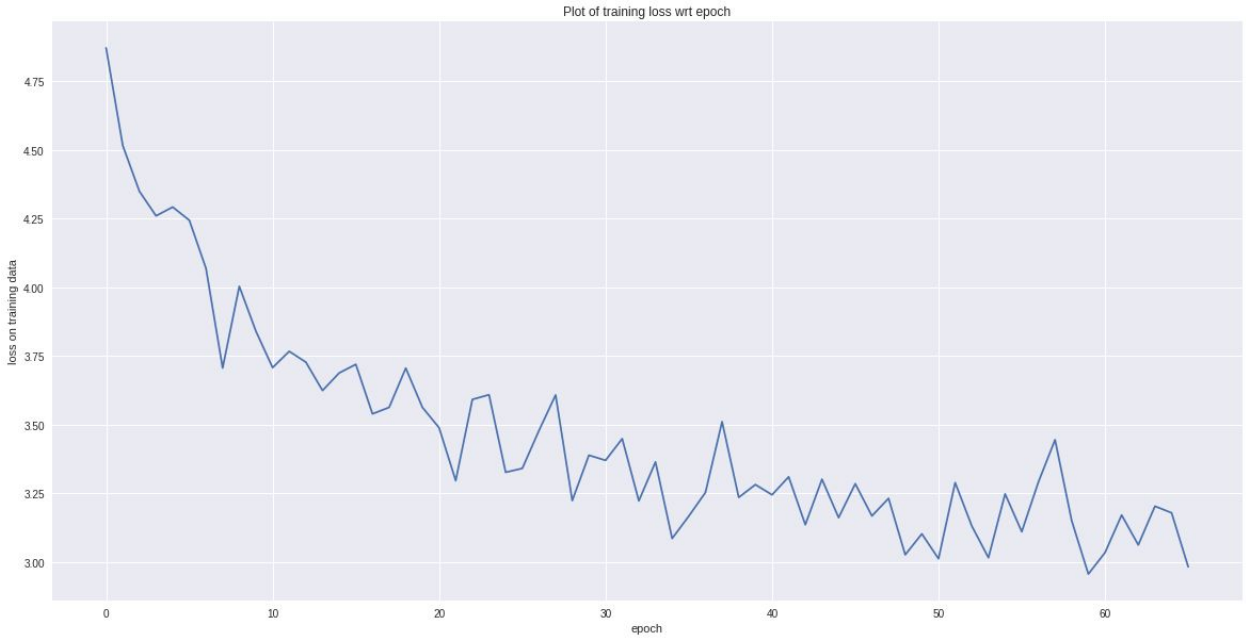
Augmentation - not used any

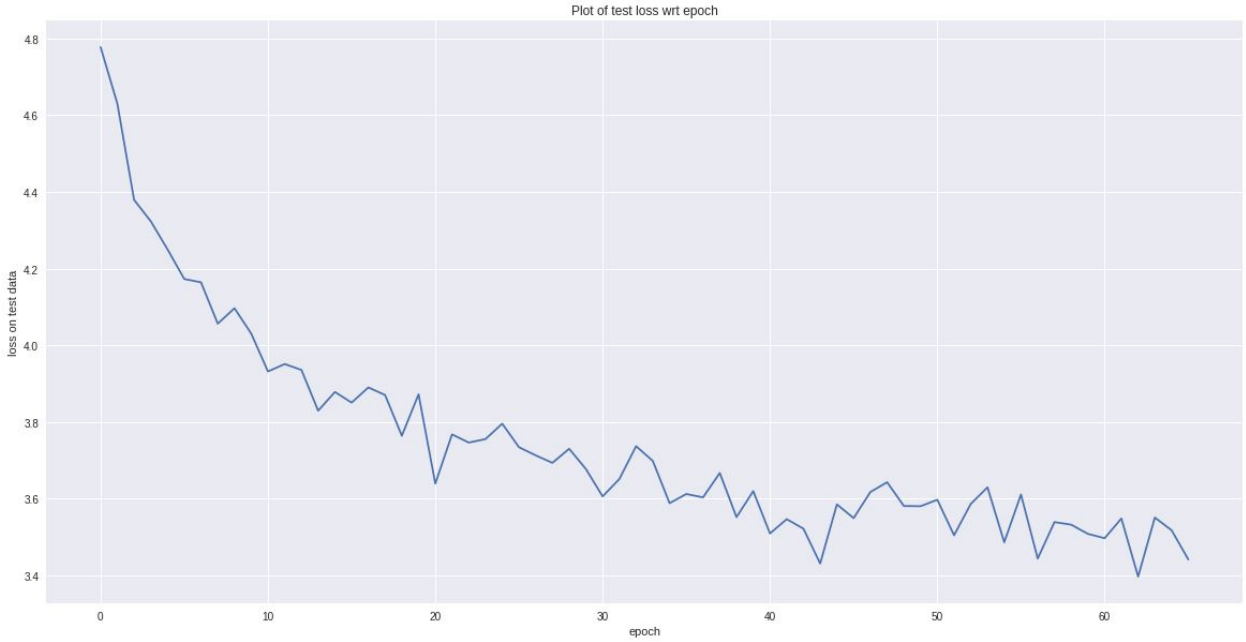
Hyperparameters -

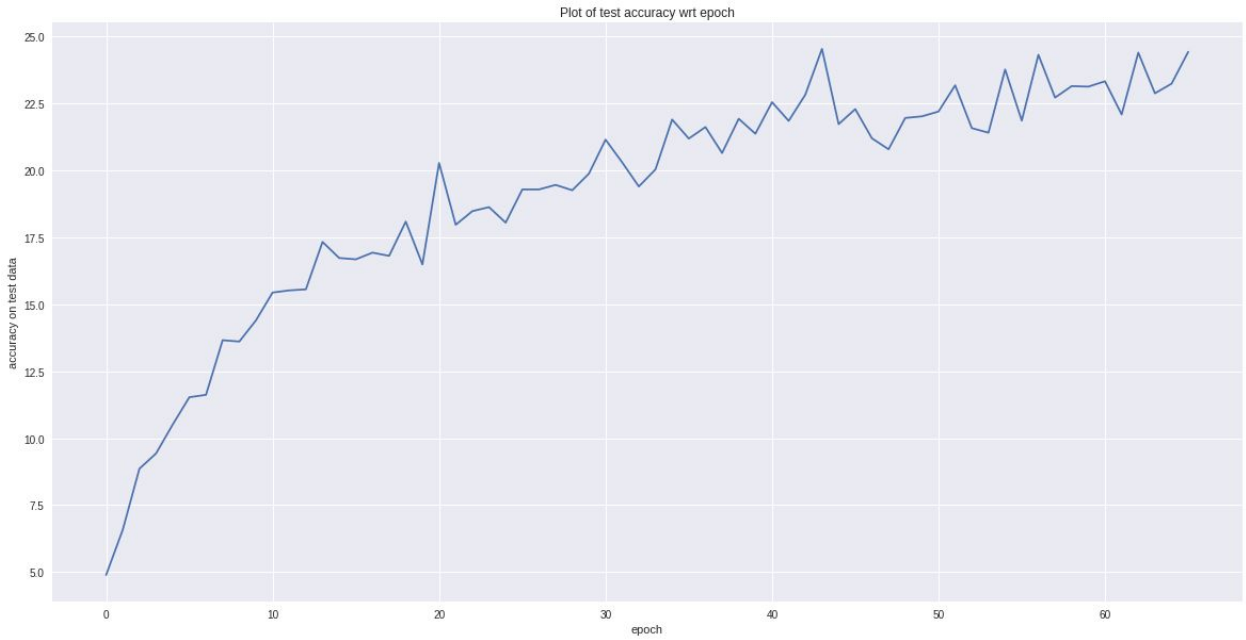
BATCH_SIZE = 256
EPOCHS = 200 (ended at 65 due to spurious environment reset)
LEARNING_RATE = 0.01
MOMENTUM = 0.9
WEIGHT_DECAY = 0.0005
Loss - cross entropy loss

Optimizer - stochastic gradient descent

After 65 epochs,
Training loss = 2.982553
Test loss = 3.4409
Test Accuracy = 24.43%







3. <u>Why 2nd worked badly and 1st worked well?</u>

The 1st network uses data augmentation with cropping, random rotation & normalization. So, for a given input image, multiple copies of the same image are generated and passed into the shuffled input stream which leads to a better extraction of image features. Also, I've used variable kernel sizes for max-pooling layer depending on its depth which helps in better downsampling while preserving the information.

<u>Full Imagenet</u> -

1. **<u>Network Design (best accuracy)</u>**

I used a standard Resent-18 model for this task & has the following detailed architecture-

Layer0 - nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False) with batch normalization, RELU activation & max-pooling layer (kernel_size=3, stride=2, padding=1)

Layer1 - Basic block with 2 conv layers (3*3 conv, 64) with RELU activation & batch normalization

Layer2 - Basic block with 2 conv layers (3*3 conv, 128) with RELU activation & batch normalization

Layer3 - Basic block with 2 conv layers (3*3 conv, 256) with RELU activation & batch normalization

Layer4 - Basic block with 2 conv layers (3*3 conv, 512) with RELU activation & batch normalization, followed by an Average pooling layer & a fully connected layer (512*4, 1000)

Following data augmentations used:

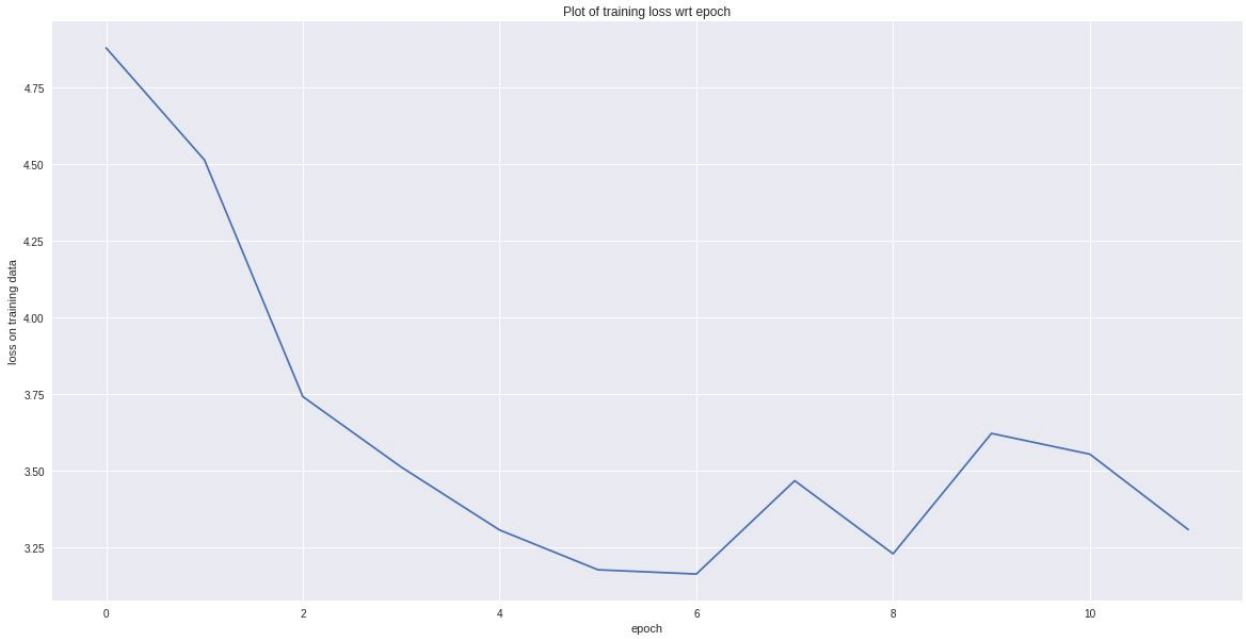
Train data - ToPILImage(),
RandomResizedCrop(128),
RandomHorizontalFlip(),
RandomRotation(18),
ToTensor(),
Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

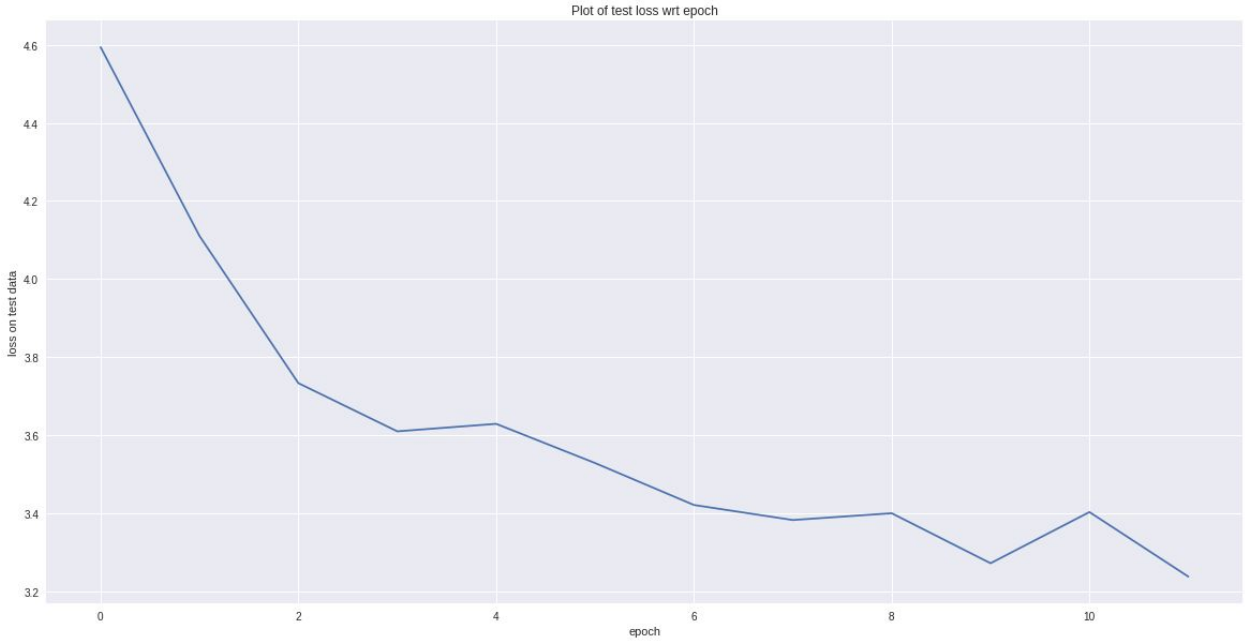
Test data - ToPILImage(),
Resize(128),
ToTensor(),
Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

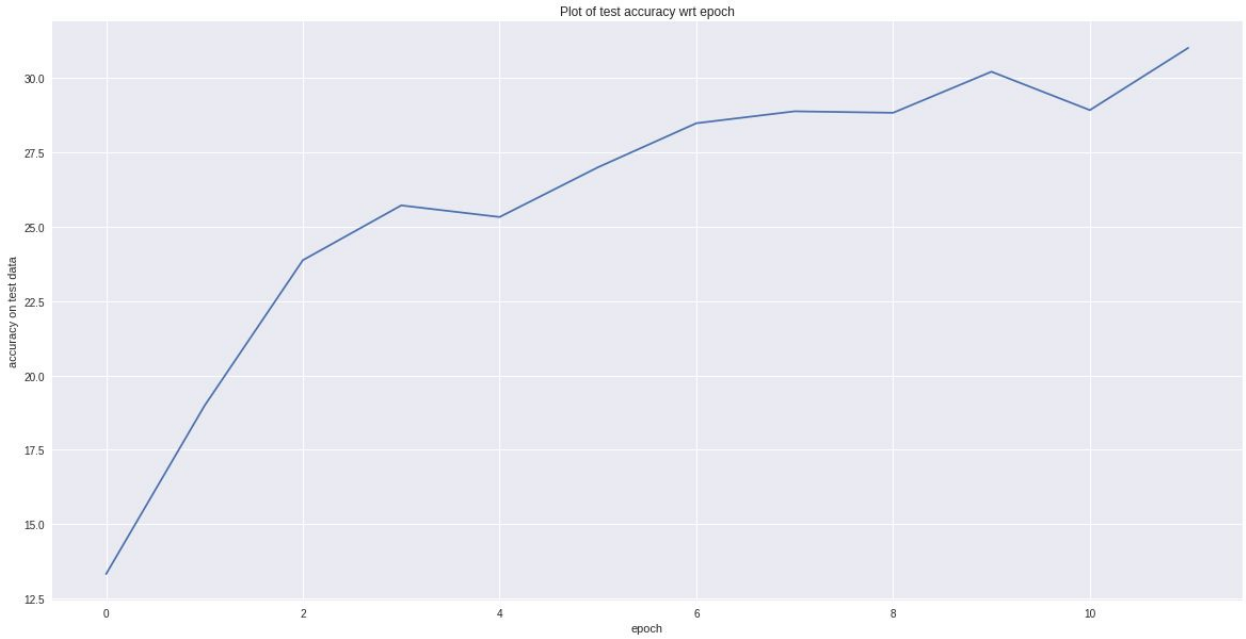
Hyperparameters -

BATCH_SIZE = 64
EPOCHS = 200 (ended at 12 due to spurious environment reset)
LEARNING_RATE = 0.01
MOMENTUM = 0.9
WEIGHT_DECAY = 0.0005, Loss - cross entropy loss
Optimizer - stochastic gradient descent

After 12 epochs,
Training loss = 3.307224
Test loss = 3.2387, Test Accuracy = 31.00%



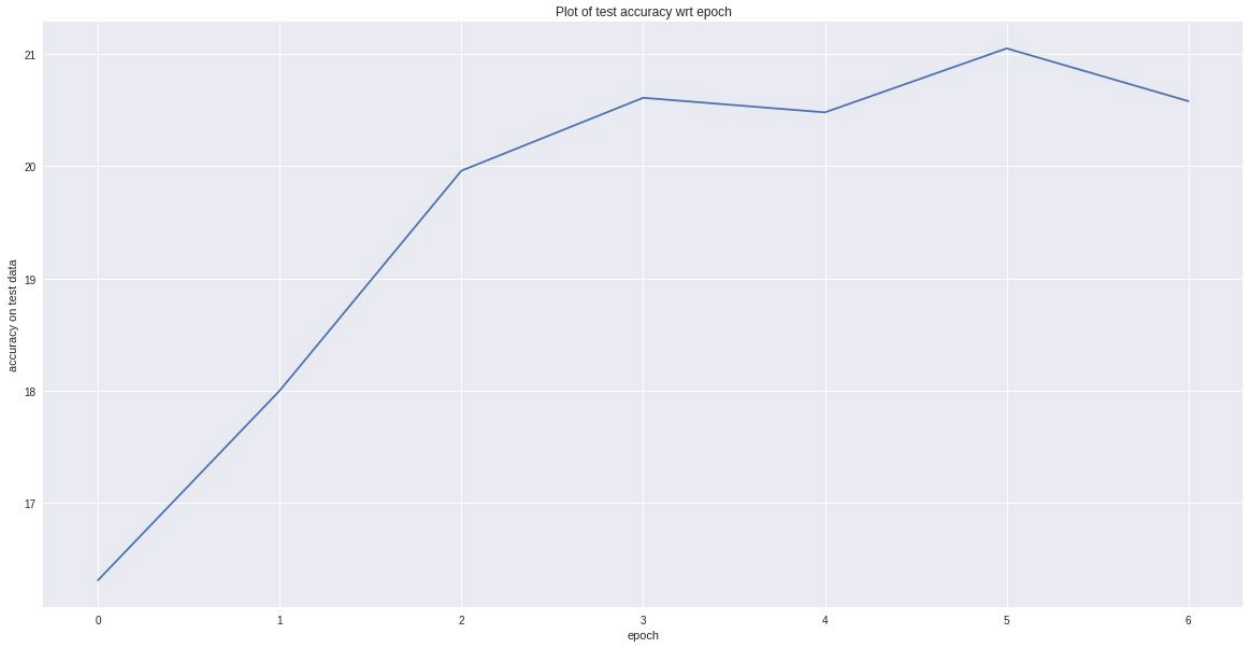




2. Deeper networks for full imagenet

Yes, the Resnet model that I tried is deeper (9 conv layers as compared to 5 conv layers in tiny imagenet model). The accuracy of full imagenet on the tiny model gave me the following test

accuracy plot:



Clearly, the delta increase in accuracy wrt epoch was slow in this case as we had larger images & bigger training data size. The model would have taken time to converge, also the accuracy is less since the model is not able to extract local features using only 5 conv layers given larger images. So, increasing the depth of the network makes sense in order to learn/extract robust global + local features and for faster convergence.

3.

For larger images -

If the same network would be used, the model would take more time to converge as well as the accuracy (train & test) could be lower than the previously standard image_size (128*128*3) training since that network would have been optimized to extract local features for that size robustly.

In order to accommodate larger images training, we can do the following things:

- Increasing the stride of convolution & max-pooling
- Increasing number of channels of intermediate convolutional layers as it will help in extracting the local relevant features without adding up to the parameters
- Using smaller batch size to take care of memory limitation
- Stacking up extra layers before the fully connected (dense) layers can help in learning of fine grained features
- Implementing data augmentation techniques like flipping, flipping + cropping can help create multiple copies of same image & will assist in better learning of local features
- Using an Average pooling layer before the dense layer

For smaller images -

If the same network would be used, the model would have tendency to overfit quickly (i.e. the training accuracy will improve but the test accuracy will start dropping). Hence by using early stopping criterion, dynamically changing the weight decay rate & learning rate while monitoring the test accuracy, we can stop the learning/training at a particular epoch and manage to get a good accuracy.

In order to accommodate larger images training, we can do the following things:

- Decreasing the stride of convolution & max-pooling will help with proper down-sampling while preserving the limited information (in image pixels)
- Implementing data augmentation techniques like flipping, flipping + cropping can help create multiple copies of same image & will assist in better learning of local features
- Deciding on an early stopping criterion for learning
- Changing the weight decay rate & learning rate while monitoring the test accuracy
- Using less number of max-pooling layers with optimal (less) stride