Computer Vision Task

Bilal Yağız Gündeğer

Image Classification:

Image classification task very crucial for several application in real life. In this task Tiny ImageNet is given which contains,

Training data set → 100,000 images,

Validation data set → 10,000 images,

Test data set → 10,000 images.

Total 200 different classes of object exist in these data sets. The images are RGB and their size is 64*64 [Figure 1].

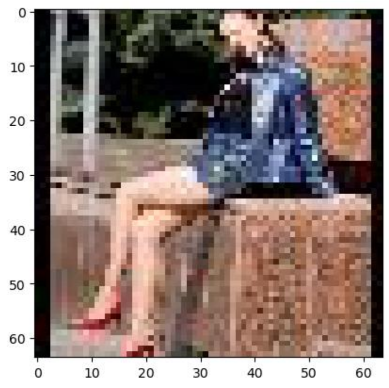
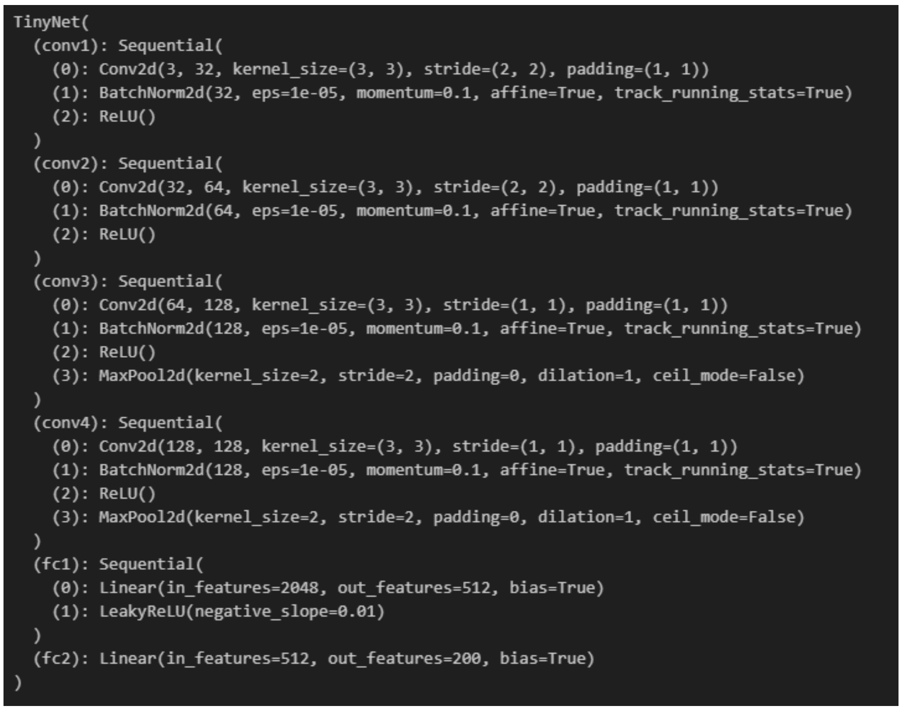


Figure 1 Sample image.

So, for this purpose I implement several deep learning models and techniques to predict these class. All implementation is in Pytorch framework.

Model architecture:

For the initial attempt, I have developed a CNN-based deep learning model. I will further enhance this model by exploring modifications to the architecture and hyperparameters. Below, you are seeing my first model architecture:



```
TinyNet(
  (conv1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Sequential(
    (0): Linear(in_features=2048, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.01)
  )
  (fc2): Linear(in_features=512, out_features=200, bias=True)
)
```

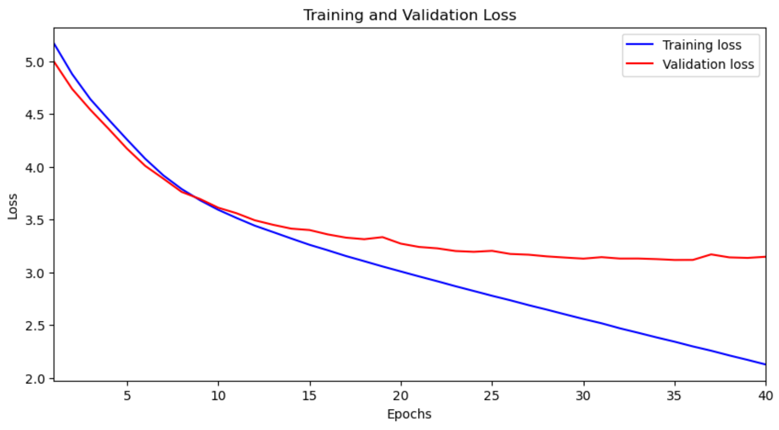
Figure 2 First Version of the model.

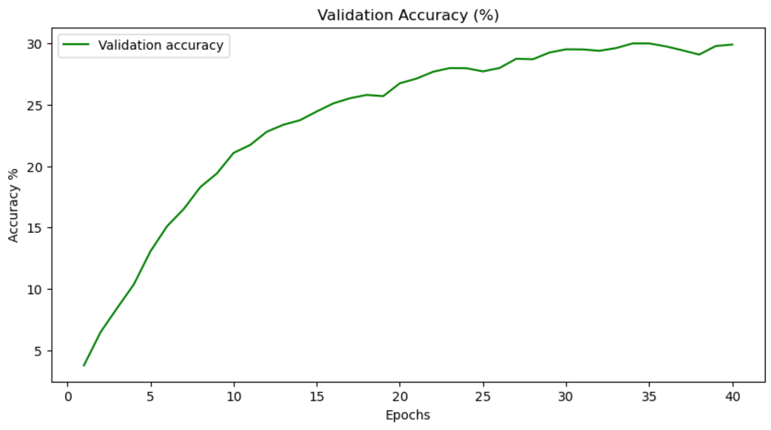
The model architecture includes four convolutional layers followed by two fully connected layers, designed to effectively extract features from the input images and perform accurate classification. By increasing the number of channels in each convolution layer, I aim to learn richer features from the images. To enhance the training process and improve convergence speed, batch normalization layers are integrated after each convolutional layer. By using the ReLU function as an activation, I introduce nonlinearity to the model to learn more complex features from the training data.

For training, I utilized the SGD optimizer with a learning rate of 0.001. The loss function employed for multiclassification is cross-entropy. Initially, I will attempt to enhance the model and explore different data preprocessing techniques.

To prevent overfitting and halt the learning process when further improvement is not observed, I implemented early stopping. The 'patience' parameter is also a hyperparameter, set to 5. This means that if the validation loss does not decrease for five consecutive epochs, the learning process is stopped. This helps to detect when the model reaches a plateau, indicating that it is no longer learning effectively.

The first iteration of training for 50 epochs with early stopping:





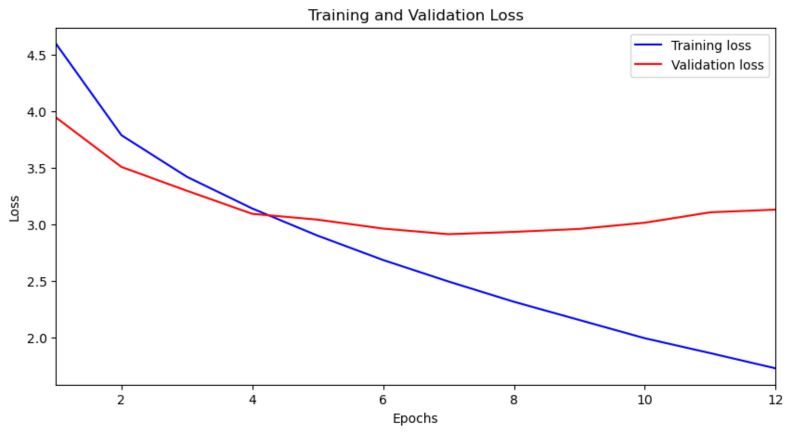
Accuracy: %29.9

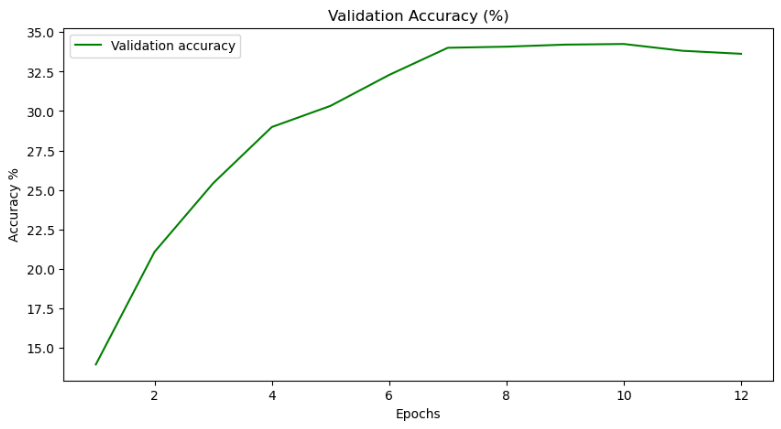
As we can see from the loss graph, the gap between the training and validation loss is increasing. Therefore, to prevent overfitting, we may need to try a different strategy. As a first step, I have taken the following actions:

- Data augmentation was implemented with the following settings:
    - Horizontal flip (applied with a probability of 0.5)
- Color jitter to adjust the brightness, contrast, saturation, and hue values of images.
- L2 regularization was introduced with a weight decay parameter of 10^-5.

Additionally, I switched the optimizer function to Adam for faster convergence.

The second iteration of training for 50 epochs with early stopping:

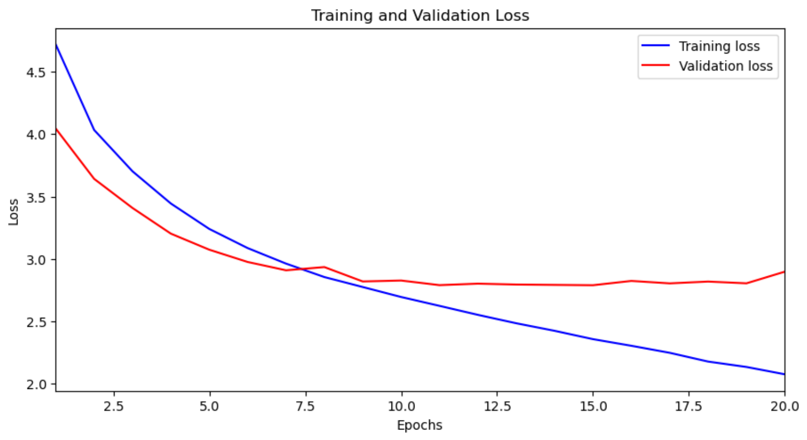


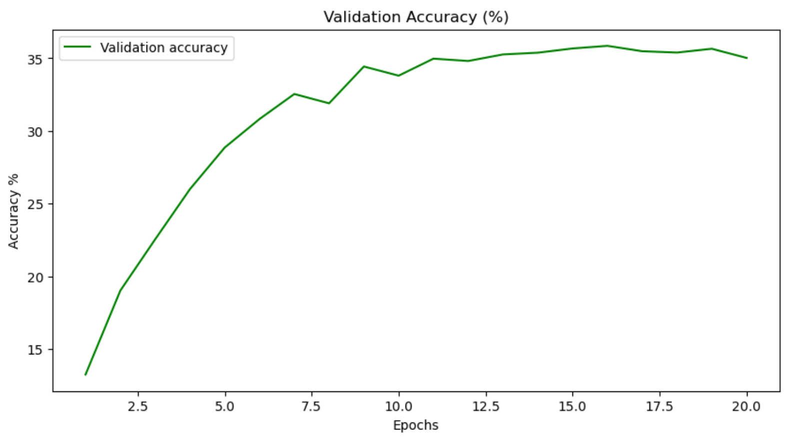


Accuracy: % 33.8

I observed persistent overfitting, so I'm introducing dropout layers between the fully connected layers with a dropout probability of 0.2. This adjustment should help regularize the model and reduce overfitting by randomly dropping out connections during training, encouraging the network to learn more robust features.

The third iteration of training for 50 epochs with early stopping:





I observed a decrease in overfitting and an increase in accuracy after introducing dropout layers.

Accuracy: %35.9

The fourth iteration of training:

For the fourth iteration of training, I have introduced a residual connection between the layers. The main motivation behind incorporating this residual connection is to transport the initial layer information to deeper layers of the neural network and mitigate the problem of diminishing gradients. Essentially, I have integrated the residual network between layer 3 and layer 5. The entire model architecture is depicted in the following figure:

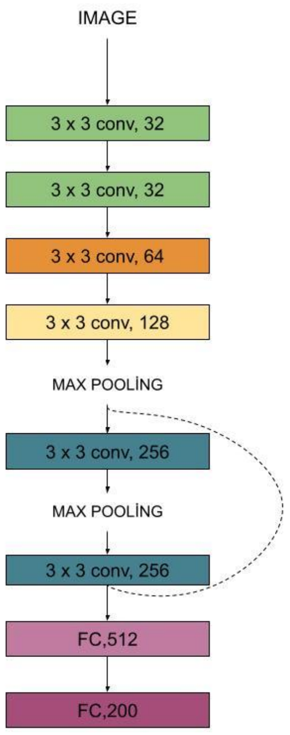
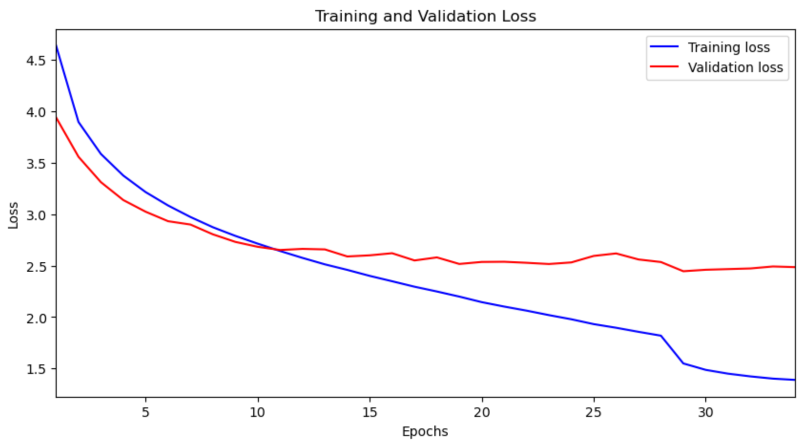
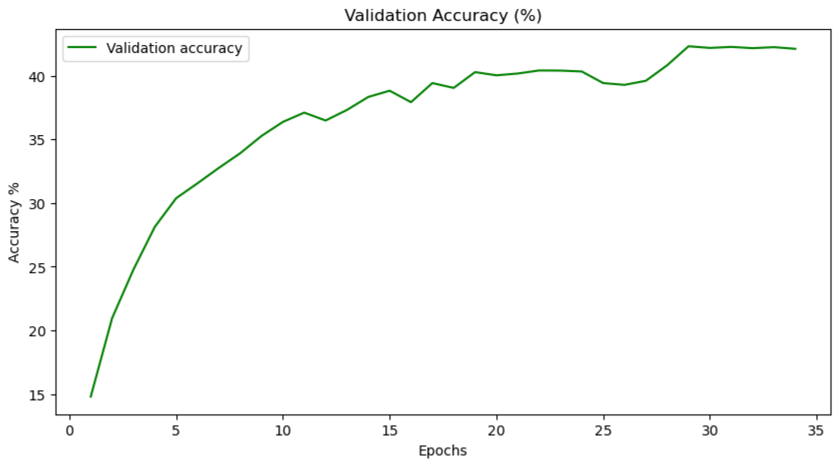


Figure 3 Model architecture.

For 50 epochs with early stopping





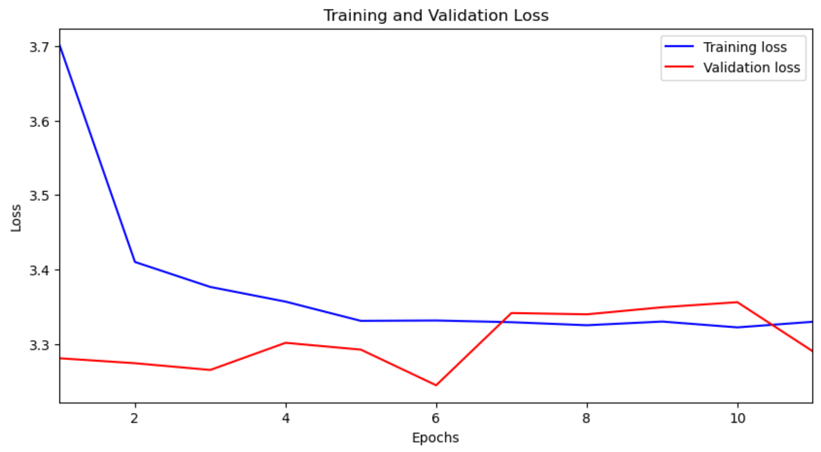
Accuracy: **% 42.3**

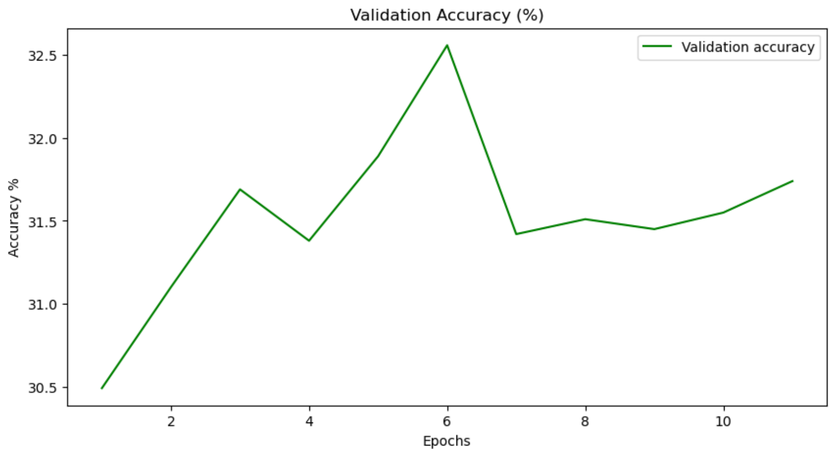
For the task, I have achieved increased accuracy results by carefully monitoring the loss and validation curves. Utilizing these plots, I iteratively adjusted the model architecture and hyperparameters. Finally, I achieve **%42.3 accuracy** over validation set. As a last step, I will compare against the state-of-the-art model, RESNET18.

Finetune Resnet18

ResNet18 is a pre-trained model on the ImageNet dataset, so I anticipate that its pre-trained weights will serve as a good starting point for Tiny ImageNet. I froze all layers except the last fully connected layer and modified it to output 200 classes.

50 epochs with early stopping:





We only trained the last layer, so it is highly likely that the limited training of just the last layer is insufficient for capturing the features specific to the Tiny ImageNet dataset. The frozen layers, which retain the features learned from the original ImageNet dataset, may not adequately capture the nuanced characteristics present in the Tiny ImageNet dataset. Therefore, the model's performance is hindered by the lack of adaptation of these frozen layers to the new dataset, resulting in suboptimal accuracy.

Accuracy: %32.1

Error Analysis by using my most accurate model:

Below are the top 5 accurately classified classes and inaccurately classified classes presented.

Top 5 Most Accurately Classified Classes	Validation accuracy	Top 5 Inaccurately Classified Classes	Validation accuracy
goldfish, Carassius auratus	%84	**Labrador retriever**	%8
school bus	%82	**syringe**	%8
bullet train, bullet	%78	**umbrella**	%8
trolleybus, trolley coach, trackless trolley	%76	**bucket, pail**	%6
dugong, Dugong dugon	%74	**chain**	%6

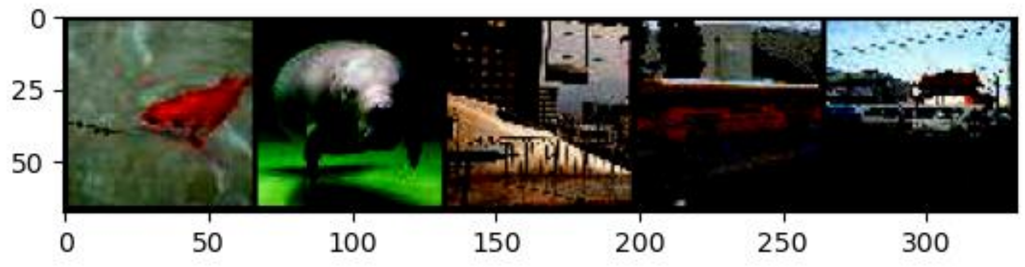


Figure 4 Top_5 most accurate class samples

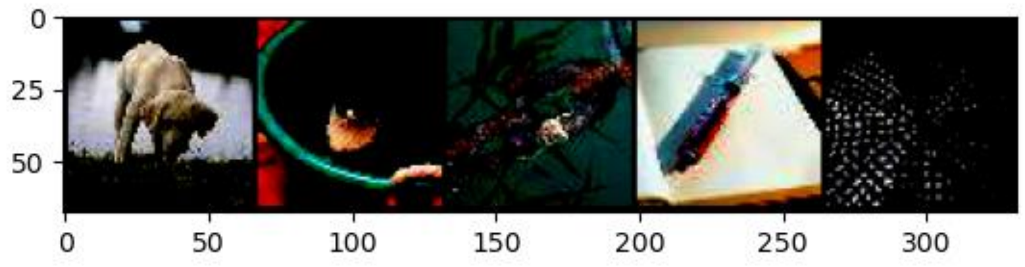


Figure 5 Top_5 Inaccurate class samples.

In order to comprehend the performance weaknesses and strengths of the model, I calculated the most accurately classified and misclassified classes. The results yielded interesting insights. Upon examining the misclassified classes in the training dataset, it became apparent that they contained diverse features within the class, making it challenging for the model to learn from these images. These inaccurately classified classes exhibited variations in shape, color, and background, posing difficulties for the model. Conversely, the most accurately classified class demonstrated more specific and shared features such as color, texture, and shape, thus facilitating the model's learning process. Moreover, the training set effectively represents the validation set, resulting in high accuracy in both datasets.

REFERENCES:

- ✓ The Official PYTORCH documentation.