INFO-6147 Deep Learning With Pytorch

Fall 2024

Project Final Report

Proposal: Tiny ImageNet Classification

Name: Mohamed Hossni Ali

ID: 1306850

Contents

Table of Figures

Introduction

Image classification tasks are important part of computer vision which is used in various applications. This project aims to create an image classification model using convolutional neural network (CNN) and then compare its results with state of the art models available on the web. The dataset that is used in this project is Tiny ImageNet.

Objective

The objective of this project is to create different classifiers with reasonable accuracy on Tiny ImageNet dataset. Additionally, transfer learning will be used to compare the model which is made from scratch to the industry state of the art models.

Dataset Description

Tiny ImageNet dataset is a group of colored images with 200 classes. The dataset contains 100,000 samples with balanced distribution of 500 images per class for training (50 for validation and 50 for testing). The images in this dataset are of size 64X64X3. The list of classes are presented in Appendix A [1].

The data comes in zip file which when extracted explodes multiple folders. One folder for training contains subfolders for each class which has images folder for images. The validation set is contained in one val folder that has the 10k validation images and text file that contain the class reference of these images.

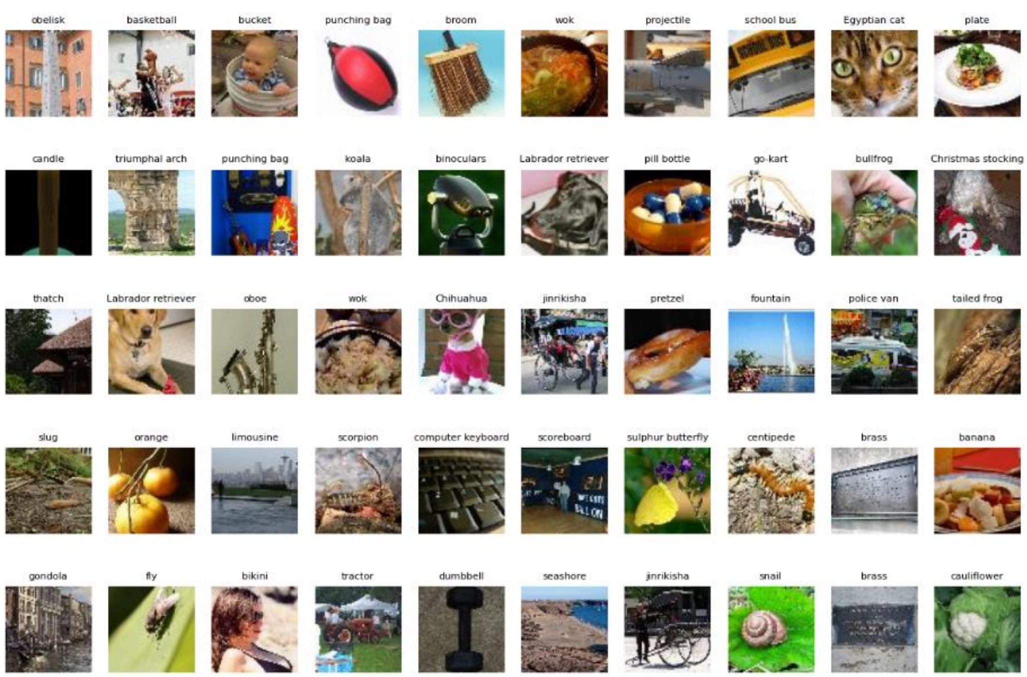

Figure 1: Tiny ImageNet sample

Data Preprocessing

Before training the model with the dataset the mean and standard deviation (std) values were calculated on the trainset for normalization. The mean was [0.4802, 0.4481, 0.3975] and the standard deviation [0.2296, 0.2263, 0.2255] for the three channels.

The ImageFolder function was used to create the training set data loader while a custom function was used to create the validation dataset loader as the labels had to be linked differently.

The final transformation included normalization and conversion to tensor.

Methodology

The methodology for classification in this project is conducted using CNN. CNNs can capture spatial information with filters and they would be the appropriate candidates for this task.

Multiple network designs were constructed and trained with the dataset. The following section describes each of these designs in details.

Classifier 1 Architecture

The first model constitutes of 8 convolutional layers and 3 fully connected layers at the end. ReLu activation function is applied after every convolution and max pooling is done after every two consecutive convolutions to reduce the size. A regulation was applied throughout by using dropout layers.

```
class Classifier(nn.Module):
  def __init__(self):
    super(Classifier, self).__init__()
    self.conv1 = nn.Conv2d(in_channels=3,out_channels=32,kernel_size=3, padding=1)
    self.BN1 = nn.BatchNorm2d(num_features=32)
    self.conv2 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)
    self.BN2 = nn.BatchNorm2d(num_features=32)

    self.conv2_drop = nn.Dropout2d()

    self.conv3 = nn.Conv2d(in_channels=32,out_channels=64,kernel_size=3, padding=1)
    self.BN3 = nn.BatchNorm2d(num_features=64)
    self.conv4 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)
    self.BN4 = nn.BatchNorm2d(num_features=64)

    self.conv5 = nn.Conv2d(in_channels=64,out_channels=128,kernel_size=3, padding=1)
    self.BN5 = nn.BatchNorm2d(num_features=128)
    self.conv6 = nn.Conv2d(in_channels=128,out_channels=128,kernel_size=3, padding=1)
    self.BN6 = nn.BatchNorm2d(num_features=128)

    self.conv7 = nn.Conv2d(in_channels=128,out_channels=256,kernel_size=3, padding=1)
    self.BN7 = nn.BatchNorm2d(num_features=256)
    self.conv8 = nn.Conv2d(in_channels=256,out_channels=256,kernel_size=3, padding=1)
    self.BN8 = nn.BatchNorm2d(num_features=256)

    self.fc1 = nn.Linear(4096, 1024)
    self.fc2 = nn.Linear(1024,512)
    self.fc3 = nn.Linear(512,200)
```

```
def forward(self, x):
  x = F.relu(self.BN1(self.conv1(x)))
  x = self.conv2_drop(F.max_pool2d(F.relu(self.conv2(x)), 2))
  x = self.conv2_drop(F.relu(self.BN3(self.conv3(x))))
  x = self.conv2_drop(F.max_pool2d(F.relu(self.BN4(self.conv4(x))), 2))
  x = F.relu(self.BN5(self.conv5(x)))
  x = self.conv2_drop(F.max_pool2d(F.relu(self.BN6(self.conv6(x))), 2))
  x = F.relu(self.BN7(self.conv7(x)))
  x = self.conv2_drop(F.max_pool2d(F.relu(self.BN8(self.conv8(x))), 2))
  x = x.view(x.size(0),-1)
  x = F.relu(self.fc1(x))
  x = F.relu(self.fc2(x))
  x = self.fc3(x)
  return x
```

Figure 2: Classifier 1 architecture

Classifier 2 Architecture

The second classification model was similar to the first except there are middle fully connected layers that get reshaped back into the correct dimensions. The philosophy behind this was to change the output of the convolutions with weights.

```python
class Classifier_middle_FC(nn.Module):
  def __init__(self):
    super(Classifier_middle_FC, self).__init__()
    self.conv1 = nn.Conv2d(in_channels=3,out_channels=32,kernel_size=3, padding=1)
    self.BN1 = nn.BatchNorm2d(num_features=32)
    self.conv2 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)
    self.BN2 = nn.BatchNorm2d(num_features=32)
    self.conv2_drop = nn.Dropout2d()

    self.conv3 = nn.Conv2d(in_channels=32,out_channels=64,kernel_size=3, padding=1)
    self.BN3 = nn.BatchNorm2d(num_features=64)
    self.conv4 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)
    self.BN4 = nn.BatchNorm2d(num_features=64)

    self.fc_inner_2 = nn.Linear(16384, 16384)

    self.conv5 = nn.Conv2d(in_channels=64,out_channels=128,kernel_size=3, padding=1)
    self.BN5 = nn.BatchNorm2d(num_features=128)
    self.conv6 = nn.Conv2d(in_channels=128,out_channels=128,kernel_size=3, padding=1)
    self.BN6 = nn.BatchNorm2d(num_features=128)

    self.fc_inner_3 = nn.Linear(8192, 8192)

    self.conv7 = nn.Conv2d(in_channels=128,out_channels=256,kernel_size=3, padding=1)
    self.BN7 = nn.BatchNorm2d(num_features=256)
    self.conv8 = nn.Conv2d(in_channels=256,out_channels=256,kernel_size=3, padding=1)
    self.BN8 = nn.BatchNorm2d(num_features=256)

    self.fc1 = nn.Linear(4096, 1024)
    self.fc2 = nn.Linear(1024,512)
    self.fc3 = nn.Linear(512,200)
```

```python
def forward(self, x):
  x = F.relu(self.BN1(self.conv1(x)))
  x = self.conv2_drop(F.max_pool2d(F.relu(self.conv2(x)), 2))

  x = self.conv2_drop(F.relu(self.BN3(self.conv3(x))))
  x = self.conv2_drop(F.max_pool2d(F.relu(self.BN4(self.conv4(x))), 2))
  x = x.view(x.size(0),-1)
  x = F.relu(self.fc_inner_2(x))
  x = x.view(x.size(0),64,16,16)

  x = F.relu(self.BN5(self.conv5(x)))
  x = self.conv2_drop(F.max_pool2d(F.relu(self.BN6(self.conv6(x))), 2))
  x = x.view(x.size(0),-1)
  x = F.relu(self.fc_inner_3(x))
  x = x.view(x.size(0),128,8,8)

  x = F.relu(self.BN7(self.conv7(x)))
  x = self.conv2_drop(F.max_pool2d(F.relu(self.BN8(self.conv8(x))), 2))
  x = x.view(x.size(0),-1)
  x = F.relu(self.fc1(x))
  x = F.relu(self.fc2(x))
  x = self.fc3(x)
  return x
```

Figure 3: Classifier 2 architecture

Classifier 3 Architecture

This model was designed with an intermediate linear layer that reduces the dimensions of the image before going into the last convolutional block. This was used as a replacement for max-pooling after the sixth convolutional block. The concept assumed that having weights that will reduce the dimension might work better than taking the max value using the max-pool method.

```python
class Classifier_FC_asPool(nn.Module):
  def __init__(self):
    super(Classifier_FC_asPool, self).__init__()
    self.conv1 = nn.Conv2d(in_channels=3,out_channels=32,kernel_size=3, padding=1)
    self.BN1 = nn.BatchNorm2d(num_features=32)
    self.conv2 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)
    self.BN2 = nn.BatchNorm2d(num_features=32)

    self.conv2_drop = nn.Dropout2d()

    self.conv3 = nn.Conv2d(in_channels=32,out_channels=64,kernel_size=3, padding=1)
    self.BN3 = nn.BatchNorm2d(num_features=64)
    self.conv4 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)
    self.BN4 = nn.BatchNorm2d(num_features=64)

    self.conv5 = nn.Conv2d(in_channels=64,out_channels=128,kernel_size=3, padding=1)
    self.BN5 = nn.BatchNorm2d(num_features=128)
    self.conv6 = nn.Conv2d(in_channels=128,out_channels=128,kernel_size=3, padding=1)
    self.BN6 = nn.BatchNorm2d(num_features=128)

    self.fc_inner_3 = nn.Linear(32768, 8192)

    #reshape here again
    self.conv7 = nn.Conv2d(in_channels=128,out_channels=256,kernel_size=3, padding=1)
    self.BN7 = nn.BatchNorm2d(num_features=256)
    self.conv8 = nn.Conv2d(in_channels=256,out_channels=256,kernel_size=3, padding=1)
    self.BN8 = nn.BatchNorm2d(num_features=256)

    self.fc1 = nn.Linear(4096, 1024)
    self.fc2 = nn.Linear(1024,512)
    self.fc3 = nn.Linear(512,200)
```

```python
  def forward(self, x):
    x = F.relu(self.BN1(self.conv1(x)))
    x = self.conv2_drop(F.max_pool2d(F.relu(self.BN1(self.conv2(x))),2))

    x = self.conv2_drop(F.relu(self.BN3(self.conv3(x))))
    x = self.conv2_drop(F.max_pool2d(F.relu(self.BN4(self.conv4(x))),2))

    x = F.relu(self.BN5(self.conv5(x)))
    x = self.conv2_drop(F.relu(self.BN6(self.conv6(x))))
    #reshaping to feed to FC_inner_3
    x = x.view(x.size(0),-1)
    x = F.relu(self.fc_inner_3(x))
    x = x.view(x.size(0),128,8,8)

    x = F.relu(self.BN7(self.conv7(x)))
    x = self.conv2_drop(F.max_pool2d(F.relu(self.BN8(self.conv8(x))), 2))
    x = x.view(x.size(0),-1)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

Figure 4: Classifier 3 architecture

Classifier 4 Architecture

This model has deeper convolutions (i.e each convolutional layer has more channels than the previous models). The first convolution starts with 128 channels and goes up to 1024 in contrast to the earlier model which had maximum of 256 channels in the last conv layer.

```
class Classifier_deep_convs(nn.Module):
    def __init__(self):
        super(Classifier_deep_convs, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3,out_channels=128,kernel_size=3, padding=1)
        self.BN1 = nn.BatchNorm2d(num_features=128)
        self.conv2 = nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1)
        self.BN2 = nn.BatchNorm2d(num_features=128)

        self.conv2_drop = nn.Dropout2d()


        self.conv3 = nn.Conv2d(in_channels=128,out_channels=256,kernel_size=3, padding=1)
        self.BN3 = nn.BatchNorm2d(num_features=256)
        self.conv4 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1)
        self.BN4 = nn.BatchNorm2d(num_features=256)



        self.conv5 = nn.Conv2d(in_channels=256,out_channels=512,kernel_size=3, padding=1)
        self.BN5 = nn.BatchNorm2d(num_features=512)
        self.conv6 = nn.Conv2d(in_channels=512,out_channels=512,kernel_size=3, padding=1)
        self.BN6 = nn.BatchNorm2d(num_features=512)


        self.conv7 = nn.Conv2d(in_channels=512,out_channels=1024,kernel_size=3, padding=1)
        self.BN7 = nn.BatchNorm2d(num_features=1024)
        self.conv8 = nn.Conv2d(in_channels=1024,out_channels=1024,kernel_size=3, padding=1)
        self.BN8 = nn.BatchNorm2d(num_features=1024)

        self.fc1 = nn.Linear(16384,8192)
        self.fc2 = nn.Linear(8192,4096)
        self.fc3 = nn.Linear(4096, 2048)
        self.fc4 = nn.Linear(2048, 1024)
        self.fc5 = nn.Linear(1024,512)
```

Figure 5: Classifier 4 architecture

Classifier 4.2 Architecture

Taking the previous model a step further and being inspired by resent, this model has an inner linear layers that act as middle classifier. The value output from the middle classifier (y) is then summed with the final x value to boost the classification.

```
class Classifier_deep_convs(nn.Module):
    def __init__(self):
        super(Classifier_deep_convs, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3,out_channels=128,kernel_size=3, padding=1)
        self.BN1 = nn.BatchNorm2d(num_features=128)
        self.conv2 = nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1)
        self.BN2 = nn.BatchNorm2d(num_features=128)

        self.conv2_drop = nn.Dropout2d()

        self.conv3 = nn.Conv2d(in_channels=128,out_channels=256,kernel_size=3, padding=1)
        self.BN3 = nn.BatchNorm2d(num_features=256)
        self.conv4 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1)
        self.BN4 = nn.BatchNorm2d(num_features=256)

        self.conv5 = nn.Conv2d(in_channels=256,out_channels=512,kernel_size=3, padding=1)
        self.BN5 = nn.BatchNorm2d(num_features=512)
        self.conv6 = nn.Conv2d(in_channels=512,out_channels=512,kernel_size=3, padding=1)
        self.BN6 = nn.BatchNorm2d(num_features=512)

        self.fc_mid1 = nn.Linear(32768,8192)
        self.fc_mid2 = nn.Linear(8192,4096)
        self.fc_mid3 = nn.Linear(4096, 2048)
        self.fc_mid4 = nn.Linear(2048, 1024)
        self.fc_mid5 = nn.Linear(1024,512)
        self.fc_mid6 = nn.Linear(512,200)

        self.conv7 = nn.Conv2d(in_channels=512,out_channels=1024,kernel_size=3, padding=1)
        self.BN7 = nn.BatchNorm2d(num_features=1024)
        self.conv8 = nn.Conv2d(in_channels=1024,out_channels=1024,kernel_size=3, padding=1)
        self.BN8 = nn.BatchNorm2d(num_features=1024)
```

```
inear(16384,8192)
        self.fc2 = nn.Linear(8192,4096)
        self.fc3 = nn.Linear(4096, 2048)
        self.fc4 = nn.Linear(2048, 1024)
        self.fc5 = nn.Linear(1024,512)
        self.fc6 = nn.Linear(512,200)

    def forward(self, x):
        x = F.relu(self.BN1(self.conv1(x)))
        x = self.conv2_drop(F.max_pool2d(F.relu(self.conv2(x)),2))

        x = self.conv2_drop(F.relu(self.BN3(self.conv3(x))))
        x = self.conv2_drop(F.max_pool2d(F.relu(self.BN4(self.conv4(x))),2))


        x = F.relu(self.BN5(self.conv5(x)))
        x = self.conv2_drop(F.max_pool2d(F.relu(self.BN6(self.conv6(x))),2))
        y = x.view(x.size(0),-1)
        y = F.relu(self.fc_mid1(y))
        y = F.relu(self.fc_mid2(y))
        y = F.relu(self.fc_mid3(y))
        y = F.relu(self.fc_mid4(y))
        y = F.relu(self.fc_mid5(y))
        y = self.fc_mid6(y)
        x = F.relu(self.BN7(self.conv7(x)))
        x = self.conv2_drop(F.max_pool2d(F.relu(self.BN8(self.conv8(x))), 2))
        x = x.view(x.size(0),-1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = F.relu(self.fc4(x))
        x = F.relu(self.fc5(x))
        x = self.fc6(x)
        return x+y
```

Figure 6: Classifier 4.2 architecture with return x+y

Classifier 5 Architecture

This classifier adds more convolutions to the previous classifiers and has 16 convolutional layers with large number of channels like the classifier 4.

```python
class Classifier_multi_convs(nn.Module):
  def __init__(self):
    super(Classifier_multi_convs, self).__init__()
    self.conv1 = nn.Conv2d(in_channels=3,out_channels=32,kernel_size=3, padding=1)
    self.BN1 = nn.BatchNorm2d(num_features=32)
    self.conv2 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)
    self.BN2 = nn.BatchNorm2d(num_features=32)
    self.conv3 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)
    self.BN3 = nn.BatchNorm2d(num_features=32)
    self.conv4 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)
    self.BN4 = nn.BatchNorm2d(num_features=32)

    self.conv2_drop = nn.Dropout2d()


    self.conv5 = nn.Conv2d(in_channels=32,out_channels=64,kernel_size=3, padding=1)
    self.BN5 = nn.BatchNorm2d(num_features=64)
    self.conv6 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)
    self.BN6 = nn.BatchNorm2d(num_features=64)
    self.conv7 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)
    self.BN7 = nn.BatchNorm2d(num_features=64)
    self.conv8 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1)
    self.BN8 = nn.BatchNorm2d(num_features=64)



    self.conv9 = nn.Conv2d(in_channels=64,out_channels=128,kernel_size=3, padding=1)
    self.BN9 = nn.BatchNorm2d(num_features=128)
    self.conv10 = nn.Conv2d(in_channels=128,out_channels=128,kernel_size=3, padding=1)
    self.BN10 = nn.BatchNorm2d(num_features=128)
    self.conv11 = nn.Conv2d(in_channels=128,out_channels=128,kernel_size=3, padding=1)
    self.BN11 = nn.BatchNorm2d(num_features=128)
    self.conv12 = nn.Conv2d(in_channels=128,out_channels=128,kernel_size=3, padding=1)
    self.BN12 = nn.BatchNorm2d(num_features=128)
```

```python
    self.conv13 = nn.Conv2d(in_channels=128,out_channels=256,kernel_size=3, padding=1)
    self.BN13 = nn.BatchNorm2d(num_features=256)
    self.conv14 = nn.Conv2d(in_channels=256,out_channels=256,kernel_size=3, padding=1)
    self.BN14 = nn.BatchNorm2d(num_features=256)
    self.conv15 = nn.Conv2d(in_channels=256,out_channels=256,kernel_size=3, padding=1)
    self.BN15 = nn.BatchNorm2d(num_features=256)
    self.conv16 = nn.Conv2d(in_channels=256,out_channels=256,kernel_size=3, padding=1)
    self.BN16 = nn.BatchNorm2d(num_features=256)

    #self.fc1 = nn.Linear(16384,8192)
    #self.fc2 = nn.Linear(8192,4096)
    self.fc1 = nn.Linear(4096, 2048)
    self.fc2 = nn.Linear(2048, 1024)
    self.fc3 = nn.Linear(1024,512)
    self.fc4 = nn.Linear(512,200)


  def forward(self, x):
    x = self.conv2_drop(F.relu(self.BN1(self.conv1(x))))
    x = self.conv2_drop(F.relu(self.BN2(self.conv2(x))))
    x = self.conv2_drop(F.relu(self.BN3(self.conv3(x))))
    x = self.conv2_drop(F.max_pool2d(F.relu(self.BN4(self.conv4(x))),2))

    x = self.conv2_drop(F.relu(self.BN5(self.conv5(x))))
    x = self.conv2_drop(F.relu(self.BN6(self.conv6(x))))
    x = self.conv2_drop(F.relu(self.BN7(self.conv7(x))))
    x = self.conv2_drop(F.max_pool2d(F.relu(self.BN8(self.conv8(x))),2))

    x = self.conv2_drop(F.relu(self.BN9(self.conv9(x))))
    x = self.conv2_drop(F.relu(self.BN10(self.conv10(x))))
    x = self.conv2_drop(F.relu(self.BN11(self.conv11(x))))
    x = self.conv2_drop(F.max_pool2d(F.relu(self.BN12(self.conv12(x))),2))
```

```python
    x = self.conv2_drop(F.relu(self.BN13(self.conv13(x))))
    x = self.conv2_drop(F.relu(self.BN14(self.conv14(x))))
    x = self.conv2_drop(F.relu(self.BN15(self.conv15(x))))
    x = self.conv2_drop(F.max_pool2d(F.relu(self.BN16(self.conv16(x))),2))

    x = x.view(x.size(0),-1)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = F.relu(self.fc3(x))
    x = self.fc4(x)
    return x
```

Figure 7: Classifier 5 architecture

Classifier 7 Architecture

Number 6 was skipped as there were two classifiers for number 4 (4 and 4.2). this classifier Adaptive Average pooling in between the convolutions that are based on Classifier 4 design and sums up these intermediary value when it returns the logit as x+x1+x2. This provides a much needed advantage of classifier 4.2 as it has a much smaller size after removing the FC layers that were previously put in 4.2.

```
class Classifier_deep_convs(nn.Module):
  def __init__(self):
    super(Classifier_deep_convs, self).__init__()
    self.conv1 = nn.Conv2d(in_channels=3,out_channels=128,kernel_size=3, padding=1)
    self.BN1 = nn.BatchNorm2d(num_features=128)
    self.conv2 = nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1)
    self.BN2 = nn.BatchNorm2d(num_features=128)

    self.conv2_drop = nn.Dropout2d()

    self.conv3 = nn.Conv2d(in_channels=128,out_channels=256,kernel_size=3, padding=1)
    self.BN3 = nn.BatchNorm2d(num_features=256)
    self.conv4 = nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1)
    self.BN4 = nn.BatchNorm2d(num_features=256)

    self.fcmid1 = nn.Linear(256,200)
    self.conv5 = nn.Conv2d(in_channels=256,out_channels=512,kernel_size=3, padding=1)
    self.BN5 = nn.BatchNorm2d(num_features=512)
    self.conv6 = nn.Conv2d(in_channels=512,out_channels=512,kernel_size=3, padding=1)
    self.BN6 = nn.BatchNorm2d(num_features=512)

    self.fcmid2 = nn.Linear(512,200)
    self.conv7 = nn.Conv2d(in_channels=512,out_channels=1024,kernel_size=3, padding=1)
    self.BN7 = nn.BatchNorm2d(num_features=1024)
    self.conv8 = nn.Conv2d(in_channels=1024,out_channels=1024,kernel_size=3, padding=1)
    self.BN8 = nn.BatchNorm2d(num_features=1024)


    self.fc1 = nn.Linear(1024,512)
    self.fc2 = nn.Linear(512,200)

  def forward(self, x):
    x = F.relu(self.BN1(self.conv1(x)))
    x = self.conv2_drop(F.max_pool2d(F.relu(self.conv2(x)),2))

    x = self.conv2_drop(F.relu(self.BN3(self.conv3(x))))
    x = self.conv2_drop(F.max_pool2d(F.relu(self.BN4(self.conv4(x))),2))

    x1 = F.adaptive_avg_pool2d(x, (1,1))
    x1 = x1.view(x1.size(0),-1)
    x1 = self.fcmid1(x1) #1st clf
```

```
x = F.relu(self.BN5(self.conv5(x)))
x = self.conv2_drop(F.max_pool2d(F.relu(self.BN6(self.conv6(x))),2))

x2 = F.adaptive_avg_pool2d(x, (1,1))
x2 = x2.view(x2.size(0),-1)
x2 = self.fcmid2(x2) #2nd clf


x = F.relu(self.BN7(self.conv7(x)))
x = self.conv2_drop(F.max_pool2d(F.relu(self.BN8(self.conv8(x))), 2))
x = F.adaptive_avg_pool2d(x,(1,1))
x = x.view(x.size(0),-1)
x = F.relu(self.fc1(x))
x = self.fc2(x)
x = x+x1+x2
return x
```

Figure 8: Classifier 7 architecture

Data Augmentation Methods:

Three types of Data Augmentation methods were used. The first one was switching off some pixels (5%) in the images at random every epoch. Second method was to produce new image samples by rotating at 90,45,25 and 65 degrees angles and flipping horizontally. The following sections have more details.

Switching off pixels

In this method it is like adding noise to the image. The 3-D tensor of 5% zeros was created at random every single epoch and then multiplied by the input images during the training. This method was not effective.

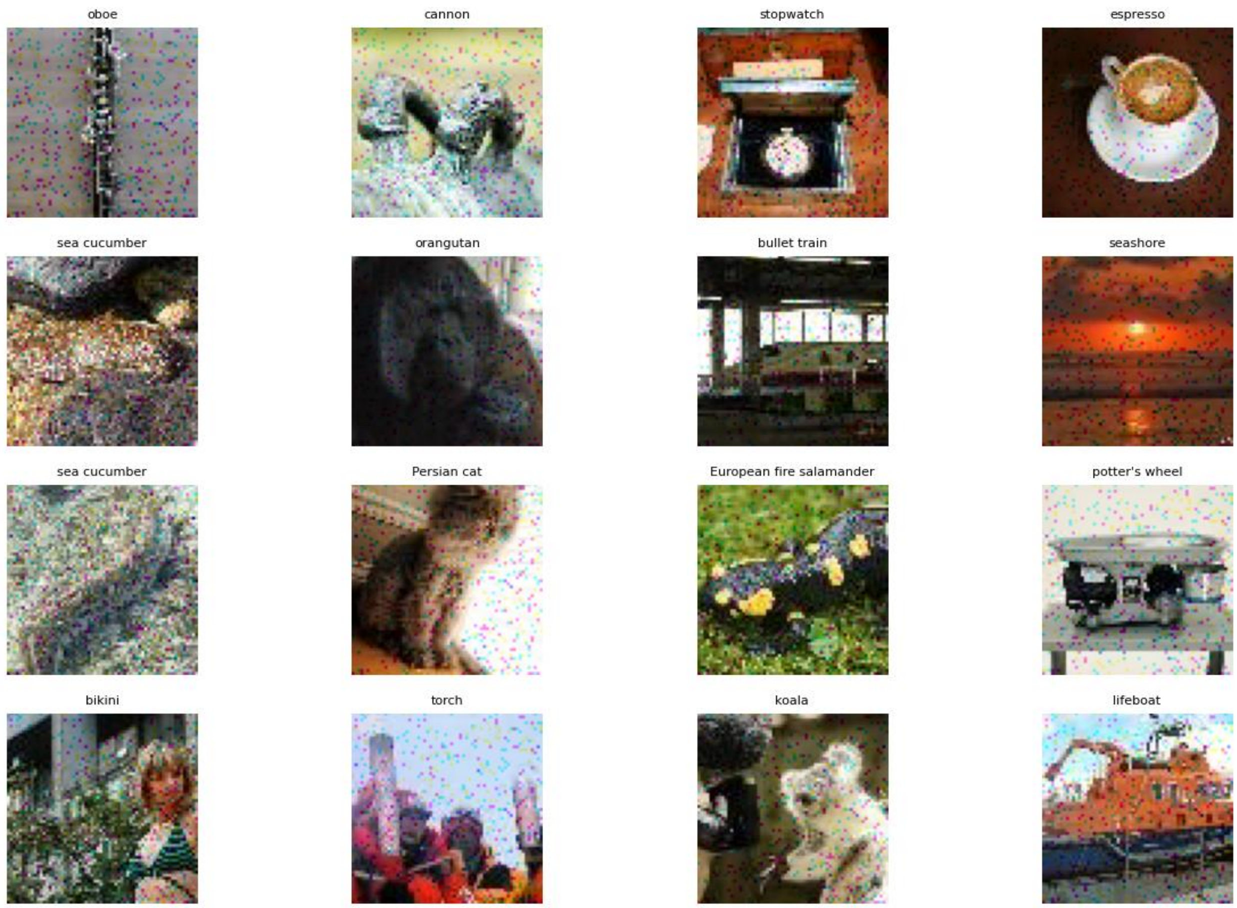

Figure 9: Data Augmentation, random switching off of pixel (noise)

Creating new images

This method involved rotating and flipping the images. With the angles used and the flip the result was 4000 images per class. This sums up to 800k images for the training set. This made it extremely difficult to run for long epochs due to resources limitations (colab allows for limited hours). When tested with classifier 4.2 the accuracy was 10% at 4 epochs which is a good jump. However, the resources did not allow to train further.

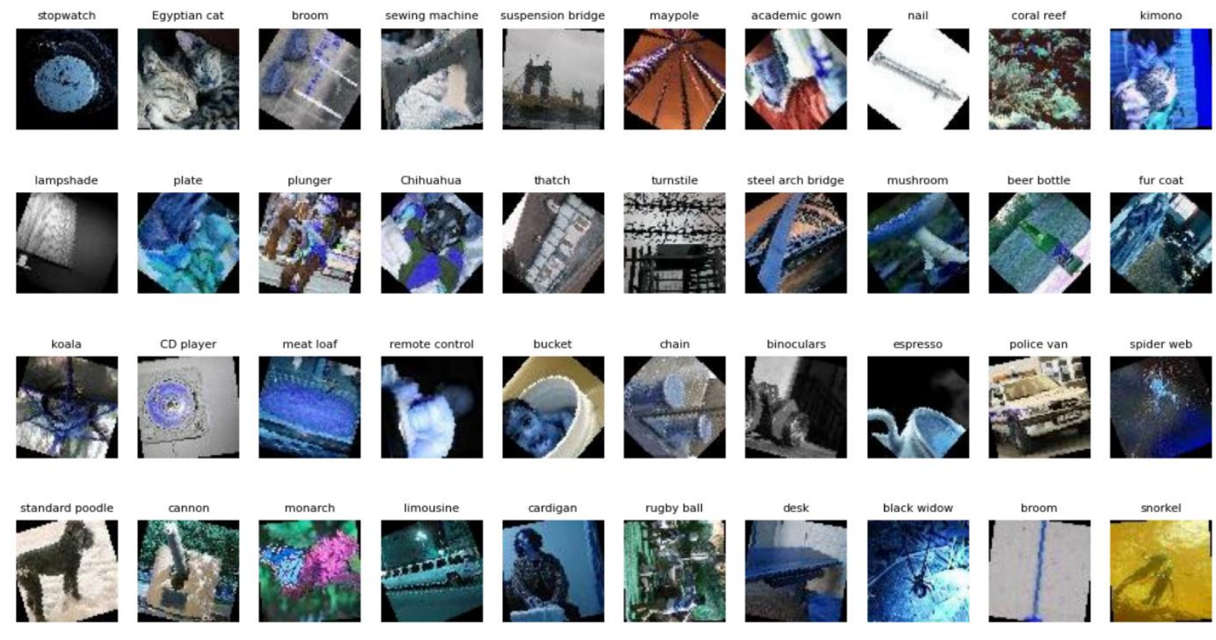

Figure 10: Data augmentation, rotation and flipping

Results

The results for training loss, test loss and accuracy were saved for all the models (except clf2 due to runtime restart). Below are the curves for the training loss for the classifier recorded every 100 iterations. Some classifiers were trained longer and others lesser than 30 epochs depending on the seen results during the training. It can be seen that resent classifier had the quickest drop in training loss.

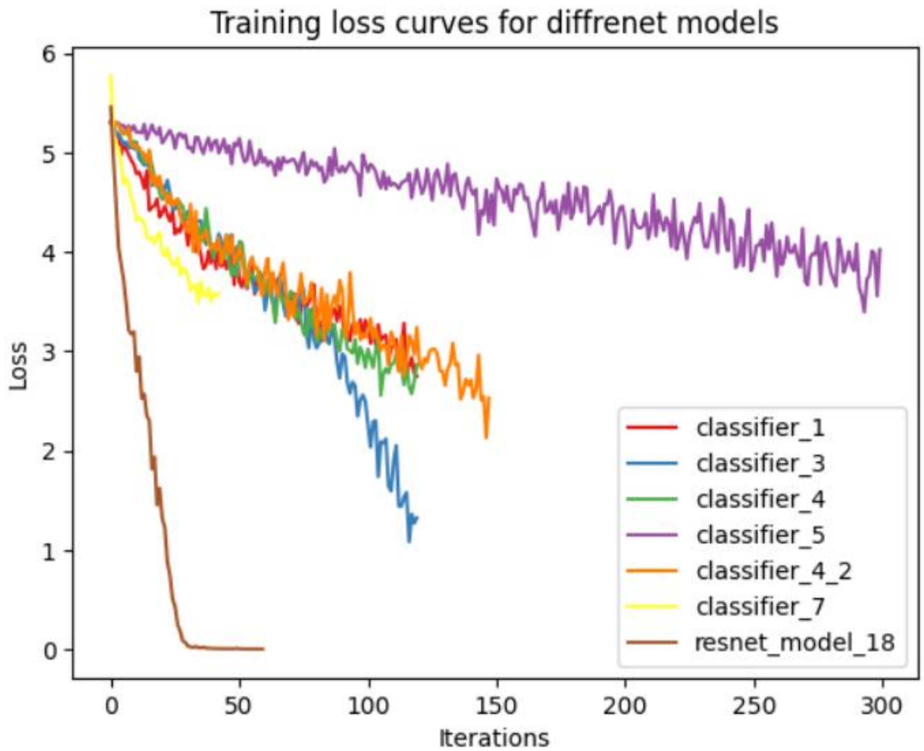

Figure 11: Training loss chart for all the models

The curves below show the test loss for the models. When the curve starts increasing like in the case of resent, it can be confirmed that the model is over fitting.

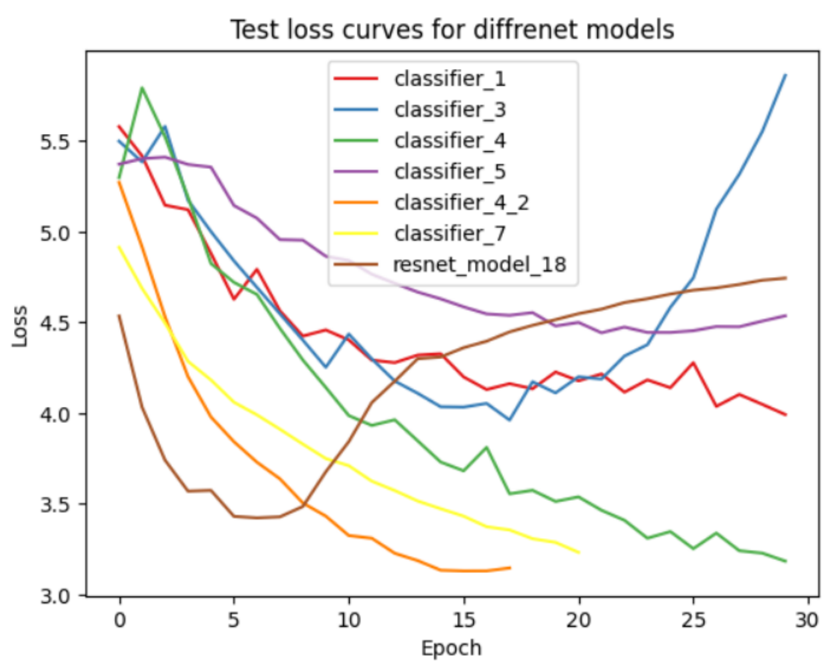

Figure 12:test loss for all models

The chart below shows the accuracy of the different models. Since model 4 showed some promising test accuracy results it was further developed with 4.2 and 7. These three classifiers (4, 4.2 and 7) performed better than resnet18 with delayed reach to results. However, from the test loss curves above they do not over fit like resnet18 did.

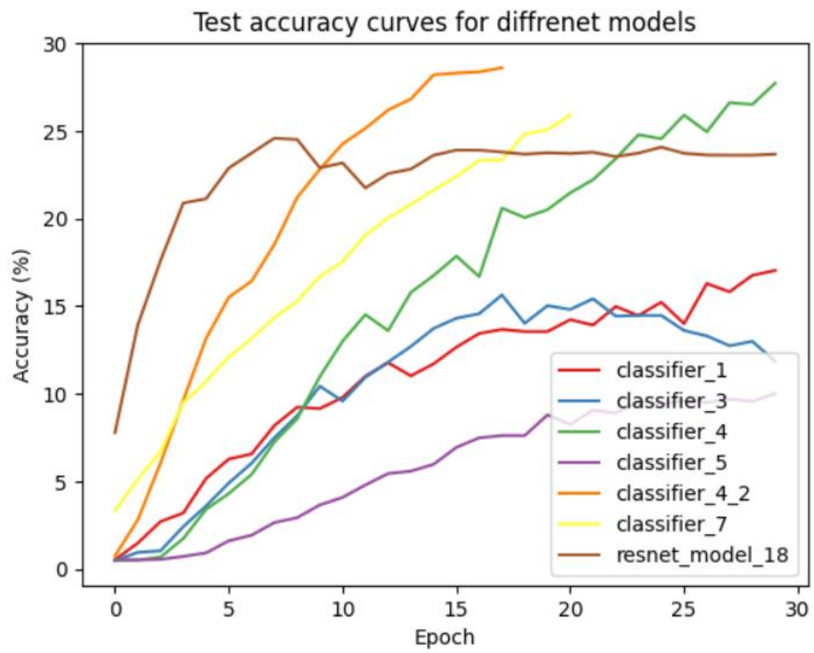

Figure 13: Models accuracy over epochs

Using a pretrained resnet18 model produce 53.5% accuracy over 20 epochs. This shows that the models designed in this project while they're less than 18 convolutions have achieved acceptable classification results over 200 classes.

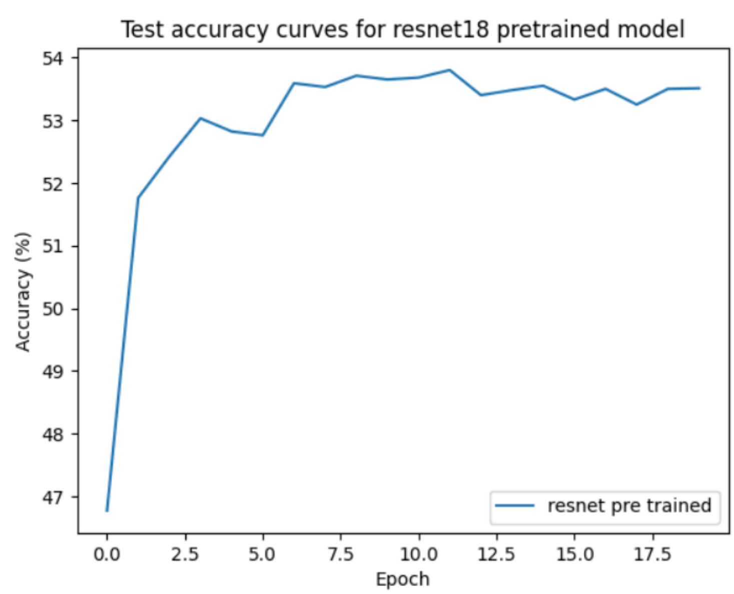

Figure 14: resnet18 pretrained model accuracy

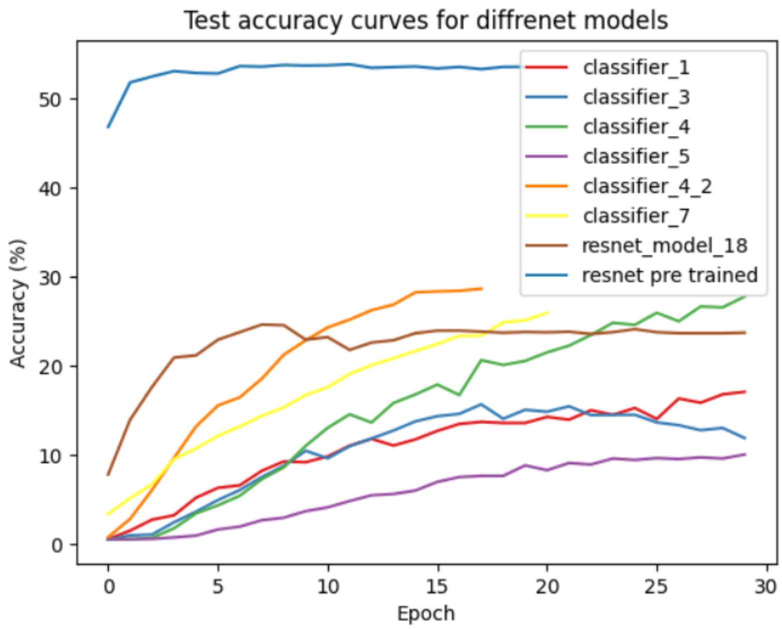

Figure 15: accuracy comparison for all the models

Conclusions and recommendations

Classifying 200 classes can be a challenging task. To achieve high accuracy many models were constructed and tested. The models designed in this project are comparable with resnet18 model which shows acceptable results.

There were resources limitations to be able to train more models with different architectures which lead to not being able to use the data augmentation (800k training images) to improve the analysis.

Further work can be done with model design and the use of other neural networks (other than CNN) to analyze how different neural networks architectures can change improve the accuracy of classification.

Project Hosting:

The project will be hosted in the following github repository

https://github.com/m-hossni/INFO-6147-Deep-Learning.git

Final Notes

There are multiple notebooks in the github repo above. This is because different models were used at different stages in different notebooks in google colab.

You may find several sections of the code repeated like image display, this was doen to ensre that the loader is assigned the correct labels.

You may also find some part of the training loop interrupted with error. This was done to avoid google colab erasing the variables and to be able to save the model parameters and logs.

Additionally, there are json logs for different models available in the github inside model_data folder. The models parameter were also saved but due to their size they are not pushed to github.

References:

Data source: Tiny ImageNet

[1] Data classes: Tiny-Imagenet-200/sets/words200.txt at master · rmccorm4/Tiny-Imagenet-200

Appendix A: Tiny ImageNet Classes

n01443537 goldfish, Carassius auratus

n01629819 European fire salamander, Salamandra salamandra

n01641577 bullfrog, Rana catesbeiana

n01644900 tailed frog, bell toad, ribbed toad, tailed toad, Ascaphus trui

n01698640 American alligator, Alligator mississipiensis

n01742172 boa constrictor, Constrictor constrictor

n01768244 trilobite

n01770393 scorpion

n01774384 black widow, Latrodectus mactans

n01774750 tarantula

n01784675 centipede

n01855672 goose

n01882714 koala, koala bear, kangaroo bear, native bear, Phascolarctos cinereus

n01910747 jellyfish

n01917289 brain coral

n01944390 snail

n01945685 slug

n01950731 sea slug, nudibranch

n01983481 American lobster, Northern lobster, Maine lobster, Homarus americanus

n01984695 spiny lobster, langouste, rock lobster, crawfish, crayfish, sea crawfish

n02002724 black stork, Ciconia nigra

n02056570 king penguin, Aptenodytes patagonica

n02058221 albatross, mollymawk

n02074367 dugong, Dugong dugon

n02085620 Chihuahua

n02094433 Yorkshire terrier

n02099601 golden retriever

n02099712 Labrador retriever

n02106662 German shepherd, German shepherd dog, German police dog, alsatian

n02113799 standard poodle

n02123045 tabby, tabby cat

n02123394 Persian cat

n02124075 Egyptian cat

n02125311 cougar, puma, catamount, mountain lion, painter, panther, Felis concolor

n02129165 lion, king of beasts, Panthera leo

n02132136 brown bear, bruin, Ursus arctos

n02165456 ladybug, ladybeetle, lady beetle, ladybird, ladybird beetle

n02190166 fly

n02206856 bee

n02226429 grasshopper, hopper

n02231487 walking stick, walkingstick, stick insect

n02233338 cockroach, roach

n02236044 mantis, mantid

n02268443 dragonfly, darning needle, devil's darning needle, sewing needle, snake feeder, snake doctor, mosquito hawk, skeeter hawk

n02279972 monarch, monarch butterfly, milkweed butterfly, Danaus plexippus

n02281406 sulphur butterfly, sulfur butterfly

n02321529 sea cucumber, holothurian

n02364673 guinea pig, Cavia cobaya

n02395406 hog, pig, grunter, squealer, Sus scrofa

n02403003 ox

n02410509 bison

n02415577 bighorn, bighorn sheep, cimarron, Rocky Mountain bighorn, Rocky Mountain sheep, Ovis canadensis

n02423022 gazelle

n02437312 Arabian camel, dromedary, Camelus dromedarius

n02480495 orangutan, orang, orangutang, Pongo pygmaeus

n02481823 chimpanzee, chimp, Pan troglodytes

n02486410 baboon

n02504458 African elephant, Loxodonta africana

n02509815 lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens

n02666196 abacus

n02669723 academic gown, academic robe, judge's robe

n02699494 altar

n02730930 apron

n02769748 backpack, back pack, knapsack, packsack, rucksack, haversack

n02788148 bannister, banister, balustrade, balusters, handrail

n02791270 barbershop

n02793495 barn

n02795169 barrel, cask

n02802426 basketball

n02808440 bathtub, bathing tub, bath, tub

n02814533 beach wagon, station wagon, wagon, estate car, beach waggon, station waggon, waggon

n02814860 beacon, lighthouse, beacon light, pharos

n02815834 beaker

n02823428 beer bottle

n02837789 bikini, two-piece

n02841315 binoculars, field glasses, opera glasses

n02843684 birdhouse

n02883205 bow tie, bow-tie, bowtie

n02892201 brass, memorial tablet, plaque

n02906734 broom

n02909870 bucket, pail

n02917067 bullet train, bullet

n02927161 butcher shop, meat market

n02948072 candle, taper, wax light

n02950826 cannon

n02963159 cardigan

n02977058 cash machine, cash dispenser, automated teller machine, automatic teller machine, automated teller, automatic teller, ATM

n02988304 CD player

n02999410 chain

n03014705 chest

n03026506 Christmas stocking

n03042490 cliff dwelling

n03085013 computer keyboard, keypad

n03089624 confectionery, confectionary, candy store

n03100240 convertible

n03126707 crane

n03160309 dam, dike, dyke

n03179701 desk

n03201208 dining table, board

n03250847 drumstick

n03255030 dumbbell

n03355925 flagpole, flagstaff

n03388043 fountain

n03393912 freight car

n03400231 frying pan, frypan, skillet

n03404251 fur coat

n03424325 gasmask, respirator, gas helmet

n03444034 go-kart

n03447447 gondola

n03544143	hourglass
n03584254	iPod
n03599486	jinrikisha, ricksha, rickshaw
n03617480	kimono
n03637318	lampshade, lamp shade
n03649909	lawn mower, mower
n03662601	lifeboat
n03670208	limousine, limo
n03706229	magnetic compass
n03733131	maypole
n03763968	military uniform
n03770439	miniskirt, mini
n03796401	moving van
n03804744	nail
n03814639	neck brace
n03837869	obelisk
n03838899	oboe, hautboy, hautbois
n03854065	organ, pipe organ
n03891332	parking meter
n03902125	pay-phone, pay-station
n03930313	picket fence, paling
n03937543	pill bottle
n03970156	plunger, plumber's helper
n03976657	pole
n03977966	police van, police wagon, paddy wagon, patrol wagon, wagon, black Maria
n03980874	poncho
n03983396	pop bottle, soda bottle
n03992509	potter's wheel
n04008634	projectile, missile
n04023962	punching bag, punch bag, punching ball, punchball
n04067472	reel
n04070727	refrigerator, icebox
n04074963	remote control, remote
n04099969	rocking chair, rocker
n04118538	rugby ball
n04133789	sandal
n04146614	school bus
n04149813	scoreboard
n04179913	sewing machine
n04251144	snorkel
n04254777	sock
n04259630	sombrero
n04265275	space heater
n04275548	spider web, spider's web
n04285008	sports car, sport car
n04311004	steel arch bridge
n04328186	stopwatch, stop watch
n04356056	sunglasses, dark glasses, shades
n04366367	suspension bridge
n04371430	swimming trunks, bathing trunks
n04376876	syringe
n04398044	teapot
n04399382	teddy, teddy bear
n04417672	thatch, thatched roof
n04456115	torch
n04465501	tractor
n04486054	triumphal arch
n04487081	trolleybus, trolley coach, trackless trolley
n04501370	turnstile
n04507155	umbrella
n04532106	vestment
n04532670	viaduct
n04540053	volleyball

n04560804	water jug
n04562935	water tower
n04596742	wok
n04597913	wooden spoon
n06596364	comic book
n07579787	plate
n07583066	guacamole
n07614500	ice cream, icecream
n07615774	ice lolly, lolly, lollipop, popsicle
n07695742	pretzel
n07711569	mashed potato
n07715103	cauliflower
n07720875	bell pepper
n07734744	mushroom
n07747607	orange
n07749582	lemon
n07753592	banana
n07768694	pomegranate
n07871810	meat loaf, meatloaf
n07873807	pizza, pizza pie
n07875152	potpie
n07920052	espresso
n09193705	alp
n09246464	cliff, drop, drop-off
n09256479	coral reef
n09332890	lakeside, lakeshore
n09428293	seashore, coast, seacoast, sea-coast
n12267677	acorn