Chapter 3

DESIGN OF THE PROPOSED SYSTEM

The software design used to develop this project is object-oriented design is based on function and procedures. It can be said as the development of software by building self-contained modules as objects that can be easily replaced, modified and reused. In this environment software is collection of discrete objects that encapsulate their data as well as the functionally to model real world "objects". Each object has attribute and methods. Objects are grouped into classes. Here each object is responsible for itself. Development concept of OOD are as follows,

- Defining objects, creating class diagram from conceptual diagram: Usually map entity to class.
- Identifying attributes.
- Use design patterns (if applicable): A design pattern is not a finished design, it is a description of a solution to a common problem, in a context. The main advantage of using a design pattern is that it can be reused in multiple applications. It can also be thought of as a template for how to solve a problem that can be used in many different situations and/or applications. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.
- Define application framework (if applicable): Application framework is usually a set of libraries or classes that are used to implement the standard structure of an application for a specific operating system. By bundling a large amount of reusable code into a framework, much time is saved for the developer, since he/she is saved the task of rewriting large amounts of standard code for each new application that is developed.
- Identify persistent objects/data (if applicable): Identify objects that have to last longer than a single runtime of the application. If a relational database is used, design the object relation mapping.
- Identify and define remote objects (if applicable).

3.1 Block diagram for our model

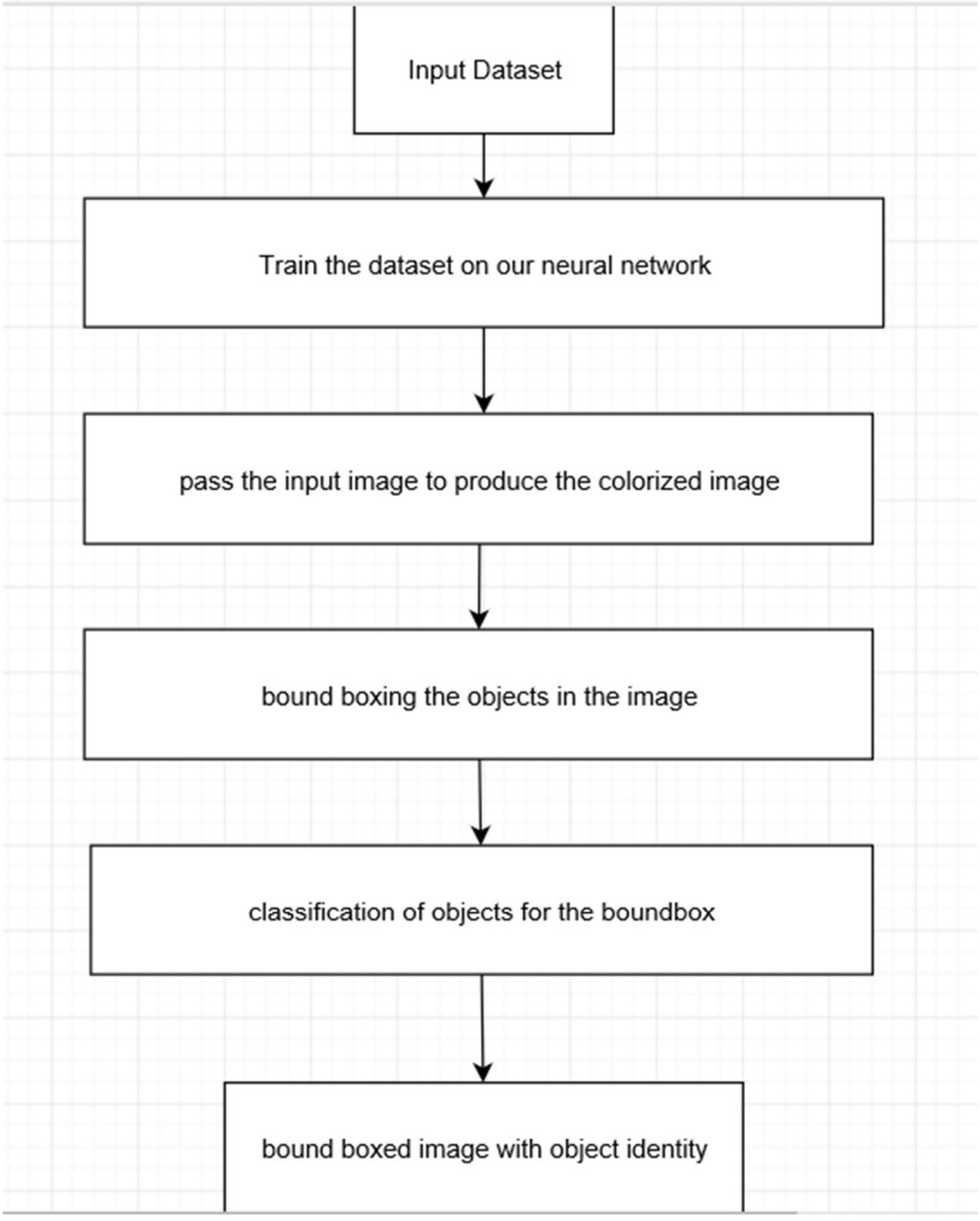

Figure 3.1: Flowchart Depicting Design Steps for Proposed system

As shown in Fig. 3.1, the input dataset is loaded and and converted to t7 format our model is trained on this dataset to generate the required weights. We can then test the grayscales to convert

them to colourized images the coloured images are passed through the AlexNet with the bound box to generate the bound boxed classification on the image.

3.2 UML Diagrams

UML is the international standard notation for object-oriented analysis and design. The heart of object-oriented problem solving is the construction of a model. The model abstracts the essential details of the underlying problem from its usually complicated real world. Several modelling tools are wrapped under the heading of unified modelling language. It is a language for visualizing, specifying, constructing, documenting.

3.2.2 Use Case Diagram for Our model

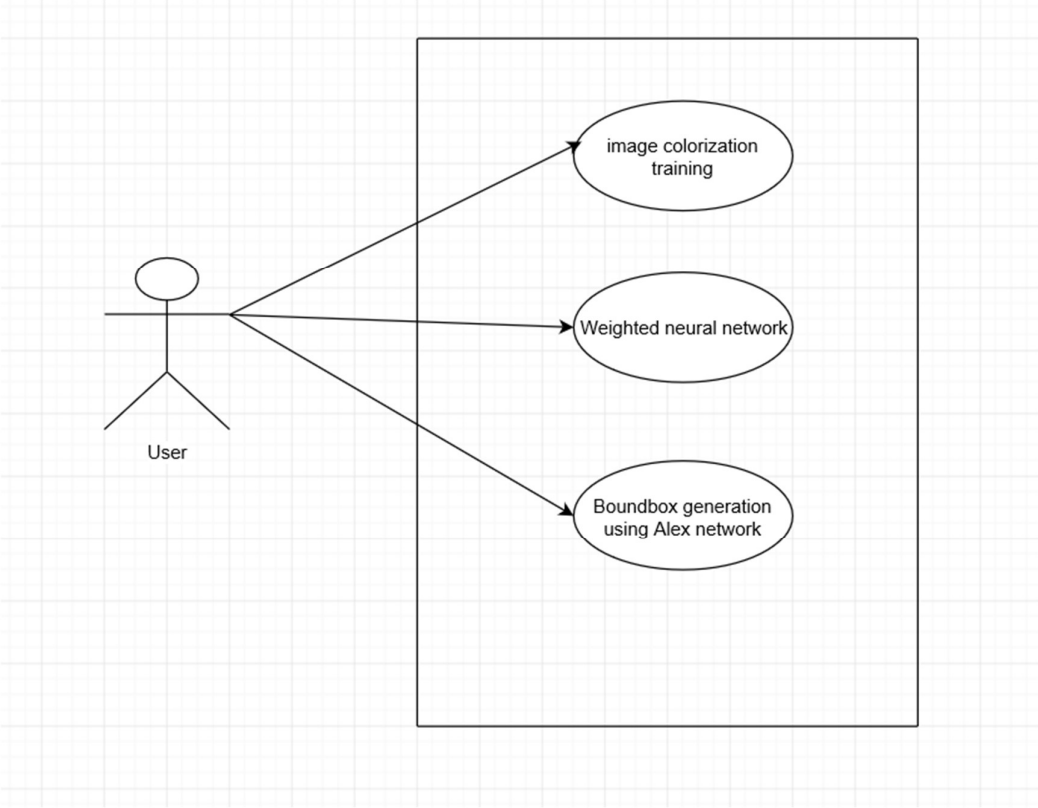

Figure 3.2: Use Case Diagram for Proposed model

Figure 3.2 shows the use case diagram for the proposed model. This system primarily has three functionalities – image colorization training, weighted network generation bound box generation. The same have been represented as use cases.

3.2.2 Activity Diagram for Our model

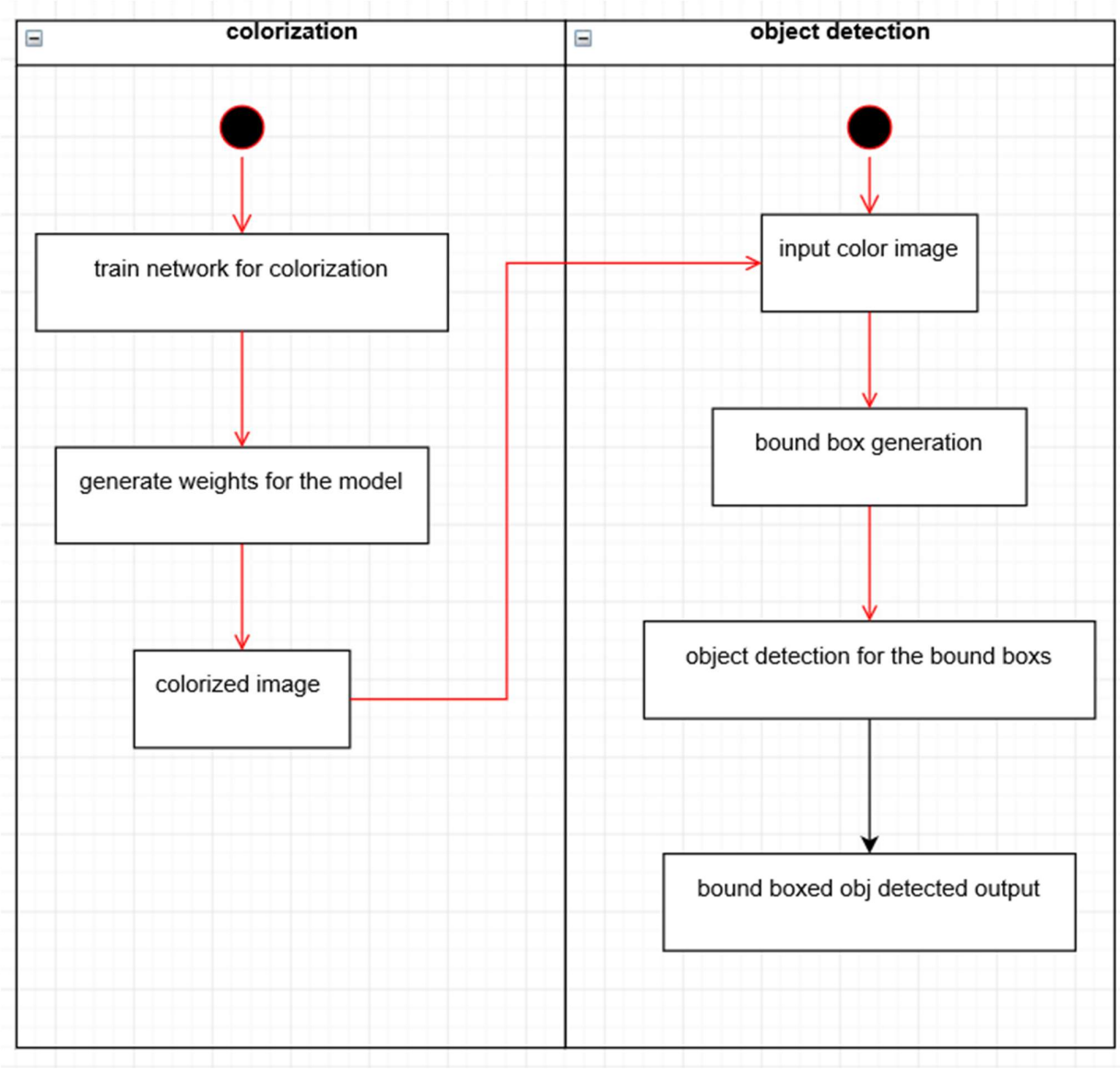

Fig 3.3 Activity diagram for the proposed system

Figure 3.3 shows the proper dataflow of this project that is the generation of the bound boxes and object classification for the bound box from a given grayscale image.

3.2.3 Sequence diagram for our model

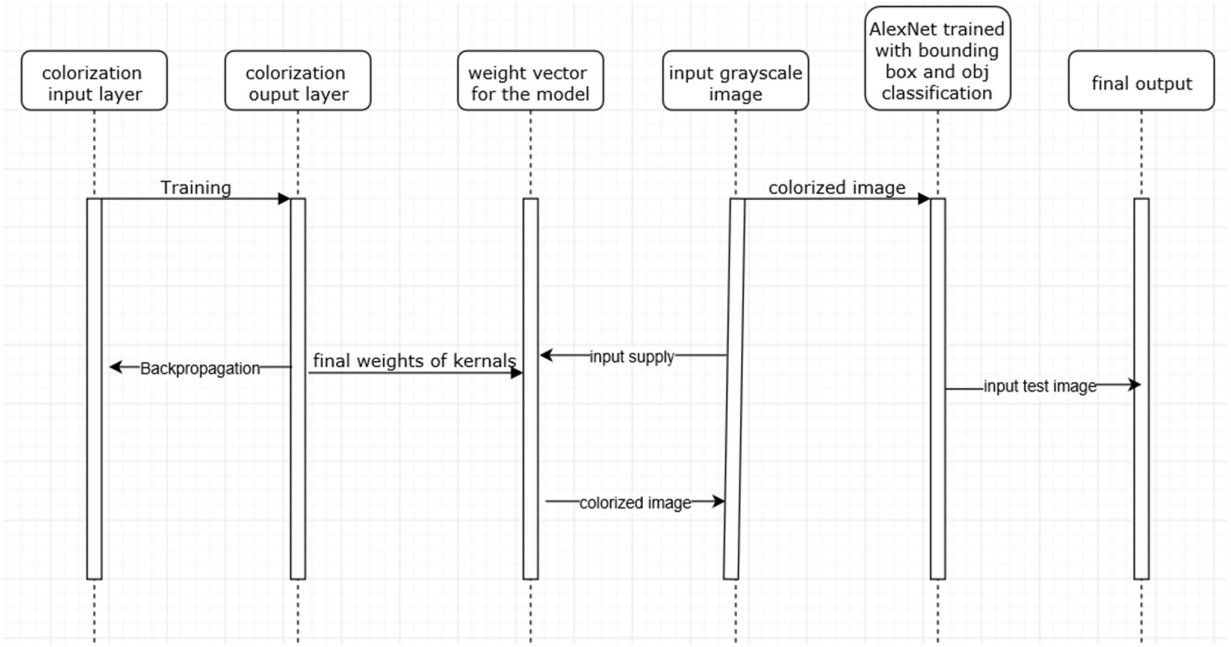

Fig 3.4 Sequence diagram for proposed system

Figure 3.4 shows the sequence diagram for this project a timestamp depiction of data flow is illustrated.

3.2.4 DFD-0 for our model

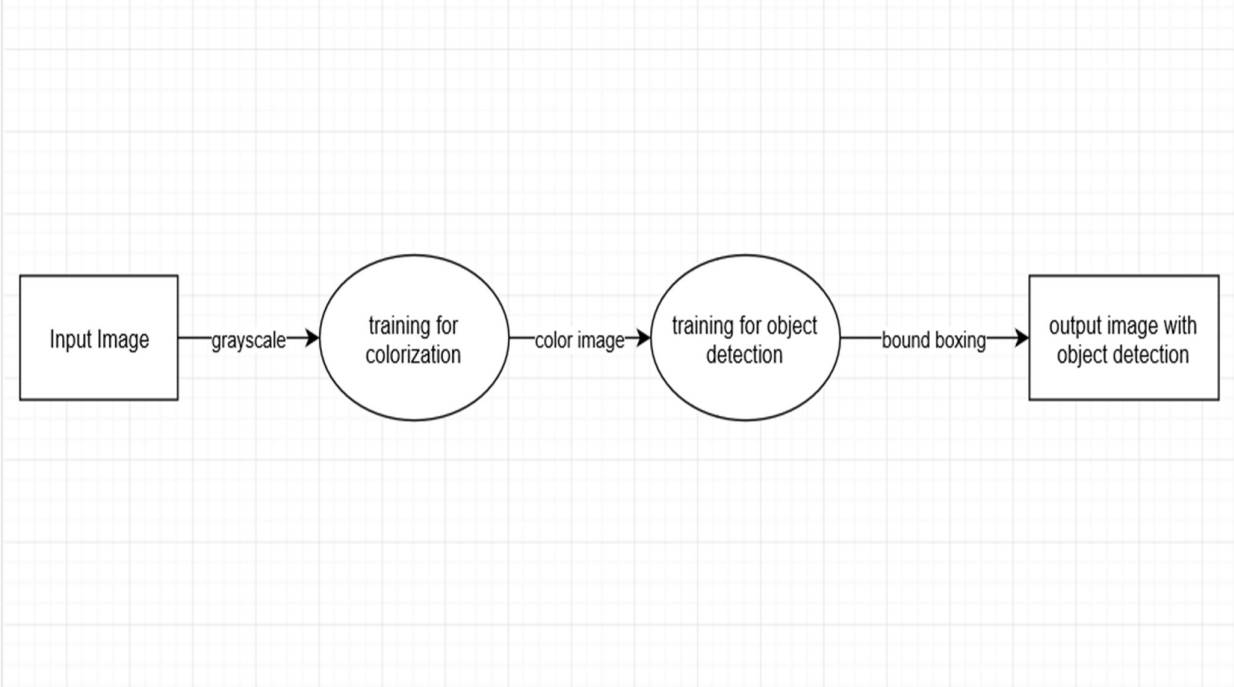

Fig 3.5 DFD-0 for our proposed model

Figure 3.5 shows the basic dfd-0 model for this proposed system with basic flow depiction of the system

3.2.5 DFD-1 for our model

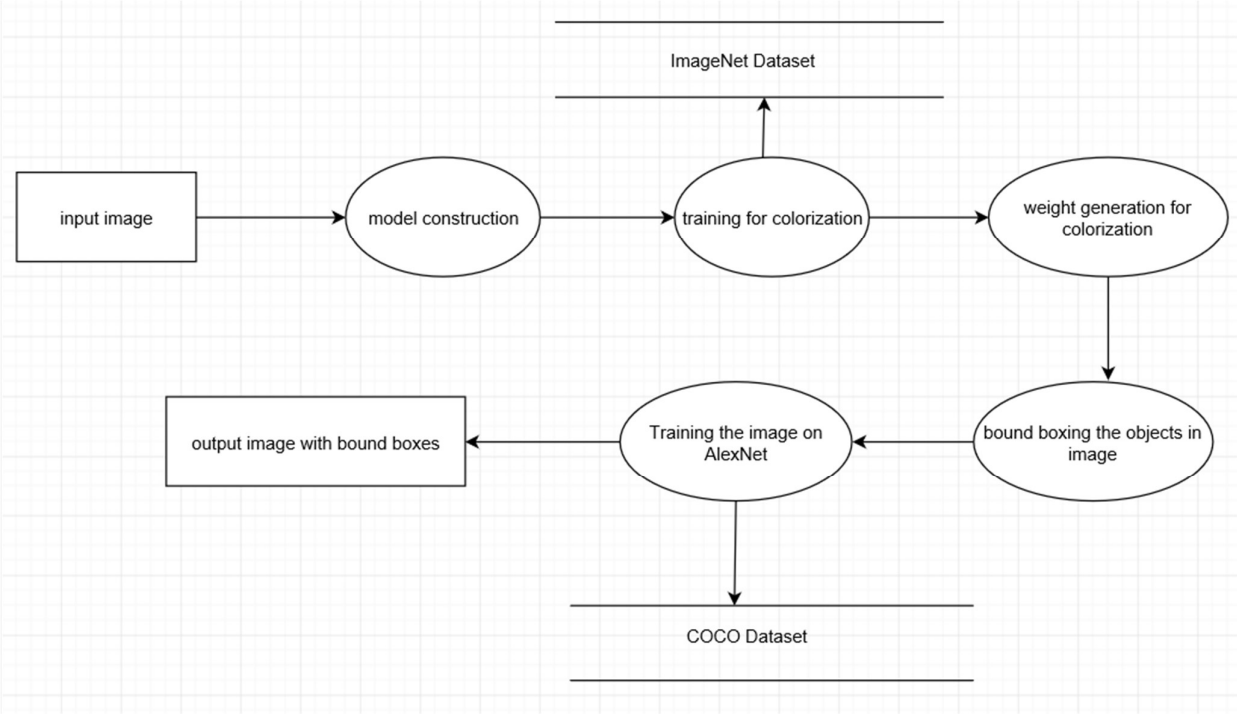

Fig 3.6 DFD-1 for our proposed model

Figure 3.6 describes the dfd-1 for our proposed model it gives a detailed description of the flow of this project.

Chapter 4
IMPLEMENTATION

4.1 Grayscale Image Colorization

4.1.1 Layers of Our Model

- **Fully-connected layer:** All neurons are connected between layers

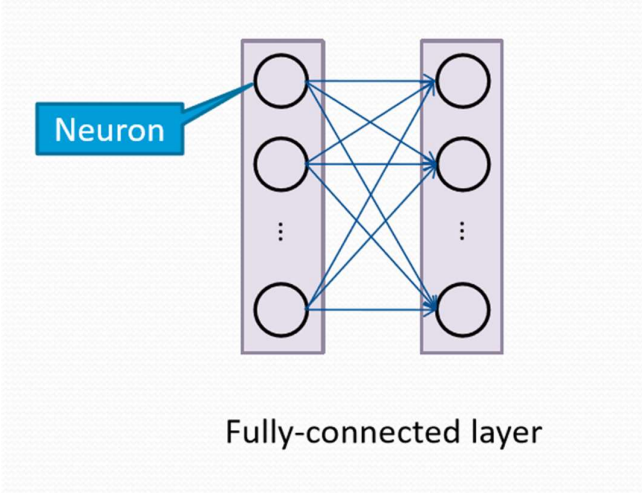

Fig 4.1 Fully connected network

- **Convolutional layer :** Takes into account underlying spatial structure

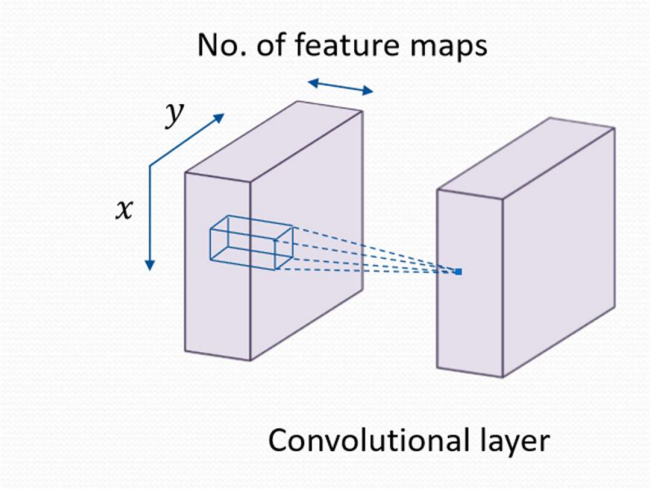

Fig 4.2 Convolution Network

4.1.2 Architecture

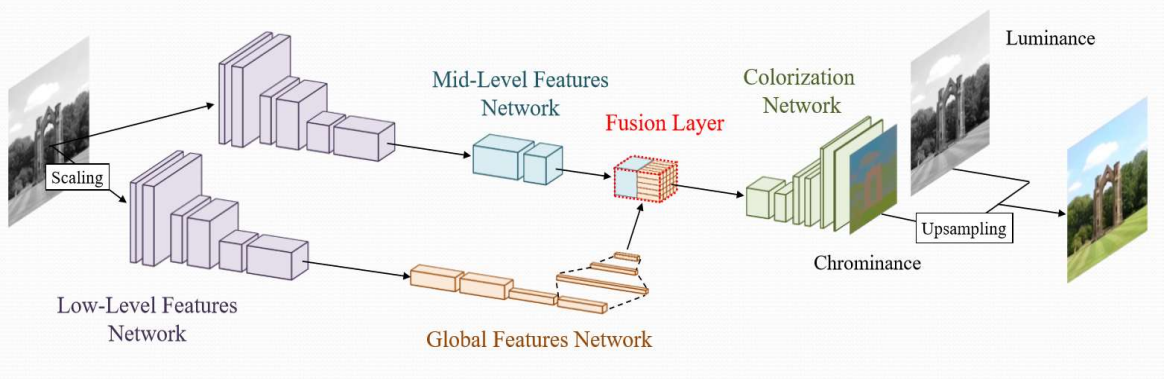

Fig 4.3 Architecture of our model

Figure 4.3 shows the end to end connectivity of architecture of this project, where in the low level features, mid level features and global level features are included and fused accordingly to form colorization network.

- Two branches: local features and global features
- Composed of four networks

4.1.3 Low-Level Features Network

A 6-layer Convolutional Neural Network obtains low-level features directly from the input image. The convolution filter bank the network represents are shared to feed both the global features network and the mid-level features network. Instead of using max-pooling layers to reduce the size of the feature maps, convolution layers with increased strides are used. This is also important for increasing the spatial support of each layer

- Extract low-level features such as edges and corners
- Lower resolution for efficient processing

4.1.4 Global Features Network

The global image features are obtained by further processing the low-level features with four convolutional layers followed by three fully-connected layers. This results in a 256-dimensional

vector representation of the image. The full details of the global image features network can be seen in Table 1-(b). Note that due to the nature of the linear layers in this network, it requires the input of the low-level features network to be of fixed size of 224 × 224 pixels.

4.1.5 Mid-Level Features Network

The mid-level features are obtained by processing the low-level features further with two convolutional layers. The output is bottle

necked from the original 512-channel low-level features to 256channel mid-level features. Note that unlike the global image features, the low-level and mid-level features networks are fully convolutional networks, such that the output is a scaled version of the input. In particular the output of the mid-level features network is a volume of size H/8 × W/8 ×256, where H and W are the original image height and width respectively.

- Extract mid-level features such as texture

4.1.6 Fusion Layer

- Combine the global features with the mid-level features
- The resulting features are independent of any resolution

4.1.7 Colorization Network

- Compute chrominance from the fused features
- Restore the image to the input resolution

In order to train the network, we use the Mean Square Error(MSE) criterion. Given a color image for training, convert the image to grayscale and CIE L*a*b* colorspace. The input of the model is the grayscale image while the target output is the a*b* components of the CIE L*a*b* colorspace. The a*b* components are globally normalized so they lie in the [0,1] range of the Sigmoid transfer function. Scale the target output to the size of the output of the colorization network and compute the MSE between the output and target output as the loss. This loss is then back-propagated through all the networks (global features, mid-level features and low-level features) to update all the parameters of the model.

4.2 Object Detection

4.2.1 Training

Pretrain the convolutional layers on the ImageNet 1000-class competition dataset. For pretraining the first 20 convolutional layers from followed by a average-pooling layer and a fully connected layer.Train this network for approximately a week and achieve a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to the GoogLeNet models in Caffe's Model Zoo. The Darknet framework is used for all training and inference. Convert the model to perform detection. Ren et al. show that adding both convolutional and connected layers to pretrained networks can improve performance . Following their example, add four convolutional layers and two fully connected layers with randomly initialized weights. Detection often requires fine-grained visual information so we increase the input resolution of the network from 224×224 to 448×448. Our final layer predicts both class probabilities and bounding box coordinates. Normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. Parametrize the bounding box x and y coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1. Use a linear activation function for the final layer and all other layers use the following leaky rectified linear activation: $\varphi(x) =(x, \text{if } x > ; 0\ 0.1x, \text{otherwise})$

4.2.2 Unified Detection

Unify the separate components of object detection into a single neural network. Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image. This design enables end-to-end training and realtime speeds while maintaining high average precision. Our system divides the input image into an S ×S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $Pr(Object)*IOU^{truth}_{pred}$. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth. Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x,y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, $Pr(Classi|Object)$. These probabilities are conditioned on the grid cell

containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B. At test time we multiply the conditional class probabilities and the individual box confidence predictions.

Chapter 5
RESULTS AND DISCUSSION

5.1 Data set Preparation/Collection

Our Project uses a set of Datasets:

1. **Nature dataset**

Set 1 (5.5GB) [Exif metadata]

Stereo Image and Range Data Collection

Content	Stereo images with registered 3D position data of natural and man-made scenes
Location	The University of Texas at Austin
Total images	98
Image sets	1
Total uncompressed disk space requirements	8.1GB (or 3.6GB for 720p)
Technical Details	ReadMe

Fig 5.1- A screenshot describing the properties of nature dataset

1. **ImageNet dataset**
2. **Coco-Net dataset**

5.2 Screenshots of Results Obtained

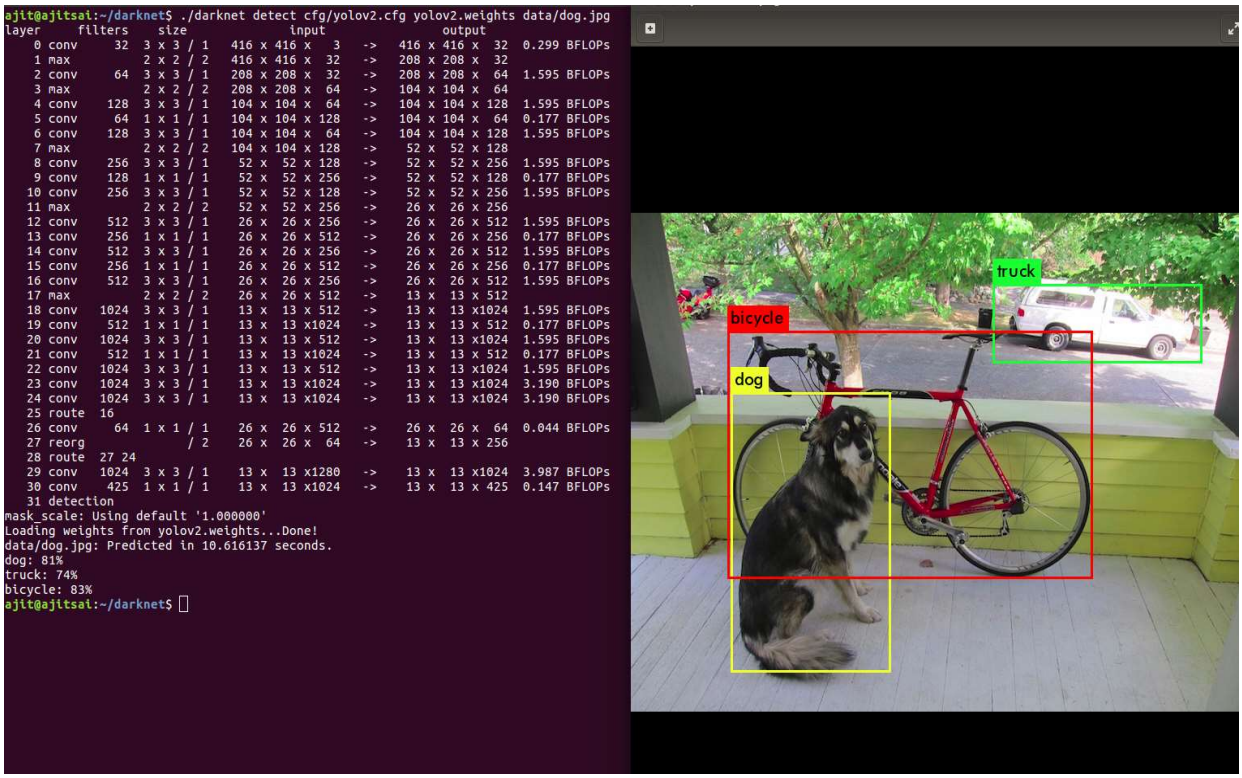

Figure 5.2: Screenshot of final output

Figure 5.2 depicts the final output of my project. Here we finally have the coloured version of our grayscale image followed by a bound box on the major objects in the image. We could also successfully classify the objects present in the bound boxes.

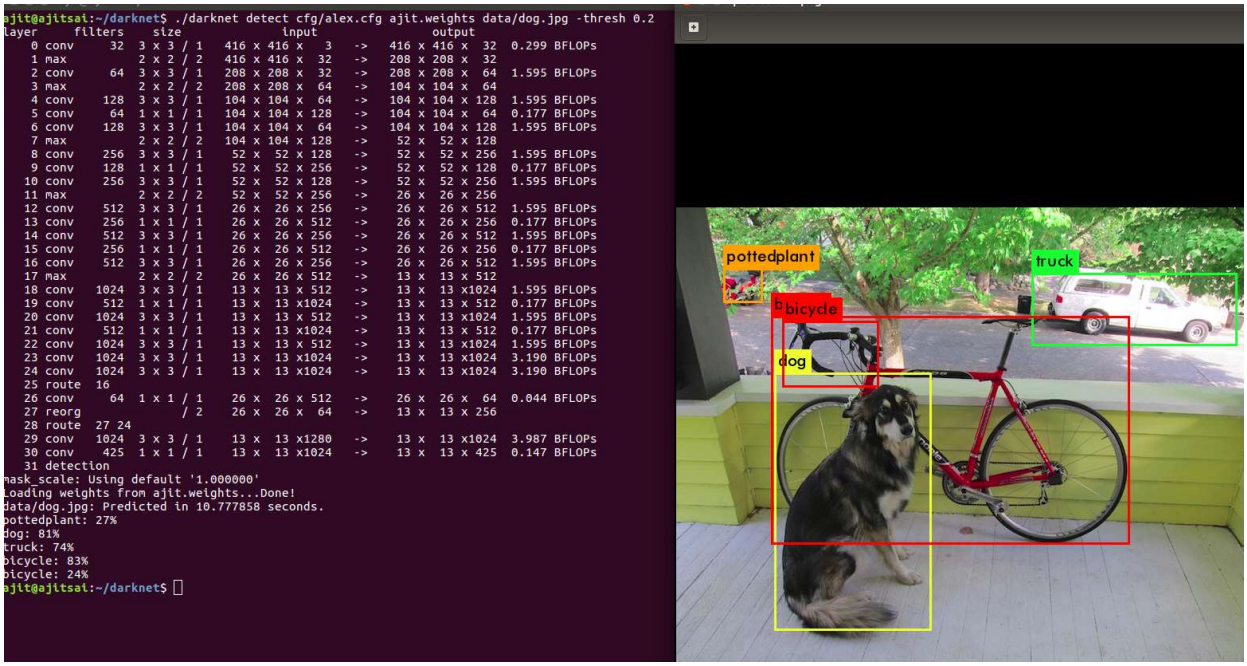

Fig 5.3 Screen shot of final output

Figure 5.3 is the final output of this project where in all the objects are described by the bound boxes, as we can see the threshold for the bounding box can be varied so that we detect more objects from the image. The lower the threshold, greater the number of bounding boxes in the image.

5.3 Screenshot of bound box

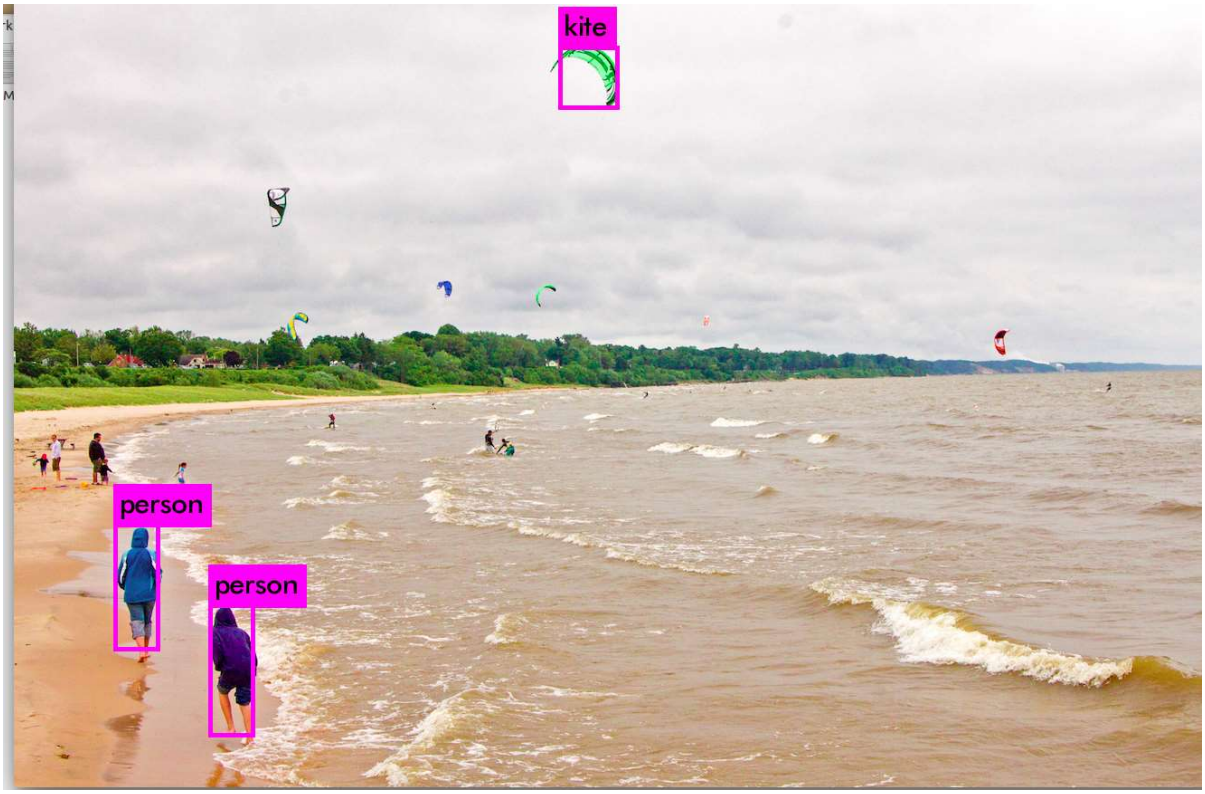

Figure 5.4: A screenshot depicting the bound boxes

The bounding boxes are formed based the highest probability of presesnce of object. As we can see in the image though many persons are present only people who are clearly found are desribed by the bound box. Our project can further describe all the objects in the image if we decrease the min threshold of the bounding boxes.

5.4 Object classification outputs

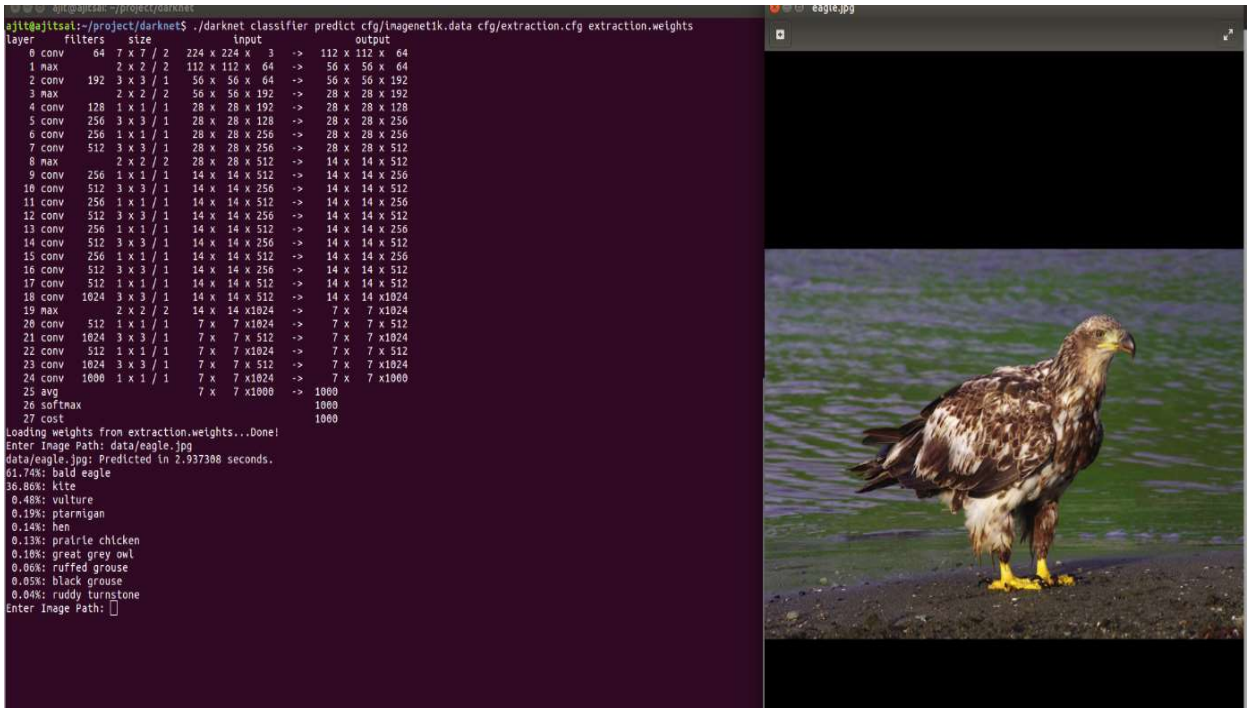

Figure 5.5: Screenshot to show the image classification

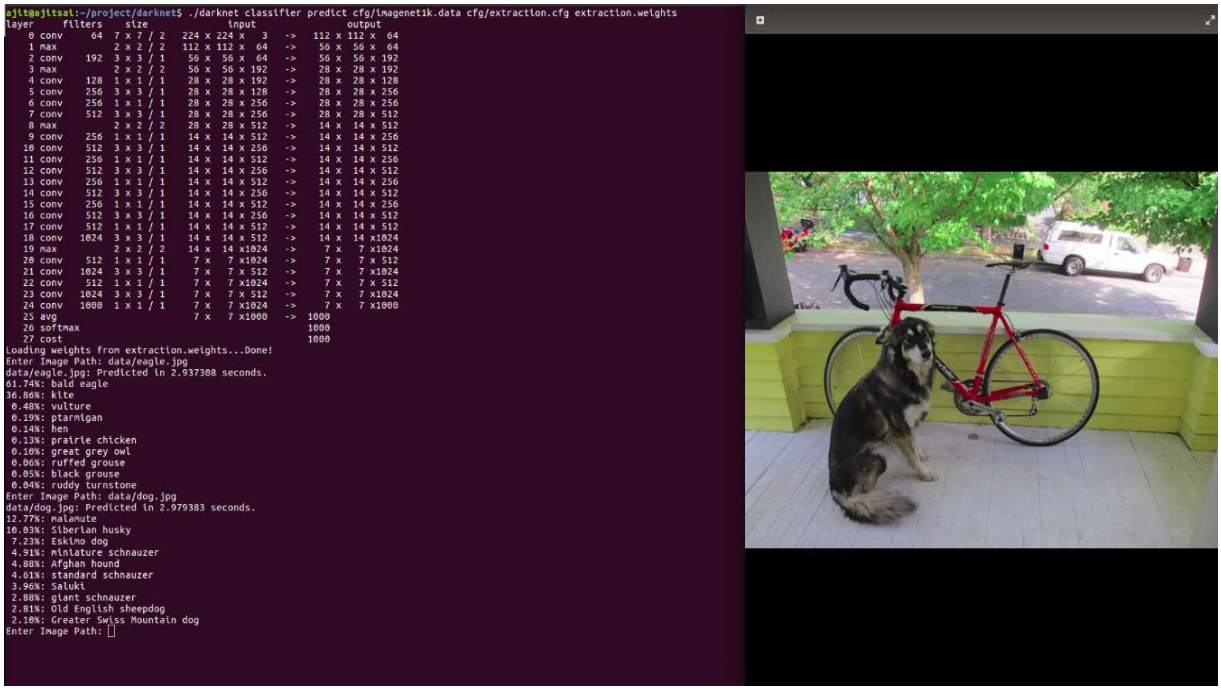

Figure 5.6 : Screenshot to show onject classification in the image

Figure 5.4 and figure 5.5 shows the object classification for the image based on imageNet 1000-class dataset

5.5 Colorization output part-1

Figure 5.7 : screenshot of colorized image

Figure 5.8 Screenshot of colorized image

Figure 5.6 and figure 5.7 show the colorization output when oout neural net is trained on nature images dataset

We can see that the colorization for the boy is poor because the training was done on images which depict images of nature.

5.6 Colorization output part-2

Figure 5.9: screenshot of colorized image

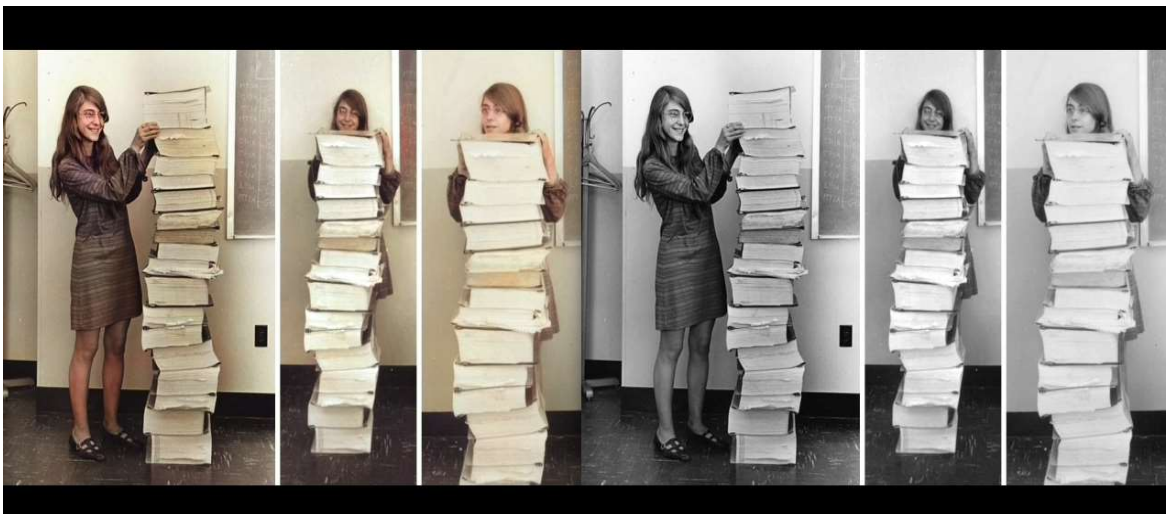

Figure 5.10 : screenshot of colored image

Here figures 5.8 and 5.9 are trained on 1000 images from ImageNet dataset set to get more clarity on human images i.e to colorize people with more accuracy.

Appendix

Model.lua

```
    local function vgg()
    model = nn.Sequential()
    model:add(Convolution(  1,  64, 3,3, 2,2, 1,1))
    model:add(ReLU(true))
    model:add(Convolution( 64, 128, 3,3, 1,1, 1,1))
    model:add(ReLU(true))
    model:add(Convolution(128, 128, 3,3, 2,2, 1,1))
    model:add(ReLU(true))
    model:add(Convolution(128, 256, 3,3, 1,1, 1,1))
    model:add(ReLU(true))
    model:add(Convolution(256, 256, 3,3, 2,2, 1,1))
    model:add(ReLU(true))
    model:add(Convolution(256, 512, 3,3, 1,1, 1,1))
    model:add(ReLU(true))
    return model
end

local function global_feature()
    model = nn.Sequential()
    model:add(Convolution(512, 512, 3,3, 2,2, 1,1))
    model:add(ReLU(true))
    model:add(Convolution(512, 512, 3,3, 1,1, 1,1))
    model:add(ReLU(true))
```

```
    model:add(Convolution(512, 512, 3,3, 2,2, 1,1))
    model:add(ReLU(true))
    model:add(Convolution(512, 512, 3,3, 1,1, 1,1))
    model:add(ReLU(true))
    model:add(nn.View(-1, 25088))
    model:add(nn.Linear(25088, 1024))
    model:add(ReLU(true))
    model:add(nn.Linear(1024, 512))
    model:add(ReLU(true))
    return model
end

local function classification(nLabels)
    model = nn.Sequential()
    model:add(nn.Linear(512, 512))
    model:add(ReLU(true))
    model:add(nn.Dropout(0.5))
    model:add(nn.Linear(512, nLabels))
    model:add(nn.LogSoftMax())
    return model
end

local function mid_level_feature()
    model = nn.Sequential()
    model:add(Convolution(512, 512, 3, 3, 1, 1, 1, 1))
    model:add(ReLU(true))
    model:add(Convolution(512, 256, 3, 3, 1, 1, 1, 1))
```

```
    model:add(ReLU(true))
    return model
end


local function img2feat(h, w)
    h_feat = h / 8
    w_feat = w / 8
    model = nn.Sequential()
    model:add(nn.Linear(512, 256))
    model:add(ReLU(true))
    model:add(nn.Replicate(h_feat, 2, 1))
    model:add(nn.Replicate(w_feat, 2, 2))
    return model

local function upsample_and_color()
    model = nn.Sequential()
    model:add(Convolution(512, 256, 3,3, 1,1, 1,1))
    model:add(ReLU(true))
    model:add(Convolution(256, 128, 3,3, 1,1, 1,1))
    model:add(ReLU(true))
    model:add(nn.SpatialUpSamplingNearest(2))
    model:add(Convolution(128, 64, 3,3, 1,1, 1,1))
    model:add(ReLU(true))
    model:add(Convolution( 64, 64, 3,3, 1,1, 1,1))
    model:add(ReLU(true))
    model:add(nn.SpatialUpSamplingNearest(2))
```

```
   model:add(Convolution( 64, 32, 3,3, 1,1, 1,1))
   model:add(ReLU(true))
   model:add(Convolution( 32,  16, 3,3, 1,1, 1,1))
   model:add(ReLU(true))
   model:add(nn.SpatialUpSamplingNearest(2))
   model:add(Convolution( 16, 8, 3,3, 1,1, 1,1))
   model:add(ReLU(true))
   model:add(Convolution( 8,  2, 3,3, 1,1, 1,1))
   model:add(nn.Tanh())
   return model
end
```

Train.lua

```
require ‘torch’
require 'optim'
require 'image'

require 'cutorch'
require 'cudnn'
require 'cunn'

require 'DataLoader'
local utils = require 'utils'
-- local models = require 'models'
local new_models = require 'model'
```

```
local cmd = torch.CmdLine()

-- Generic options
-- cmd:option('-arch', 'c9s1-32,d64,d128,R128,R128,R128,R128,R128,R128,u64,u32,c9s1-2')
cmd:option('-h5_file', './colorization_sigmoid/data/places.h5')
cmd:option('-resume_from_checkpoint', '')
cmd:option('-fine_tune', 'colornet.t7')

-- Upsampling options
-- cmd:option('-upsample_factor', 4)

-- Optimization
cmd:option('-num_iterations', 2000000)
cmd:option('-max_train', -1)
cmd:option('-batch_size', 32)
cmd:option('-learning_rate', 1e-6)
cmd:option('-lr_decay_every', -1)
cmd:option('-lr_decay_factor', 0.5)
cmd:option('-weight_decay', 0)
-- Checkpointing
cmd:option('-checkpoint_name', 'checkpoint')
cmd:option('-checkpoint_every', 1000)
cmd:option('-num_val_batches', 10)
-- Backend options
cmd:option('-gpu', 0)
cmd:option('-use_cudnn', 1)
cmd:option('-backend', 'cuda', 'cuda|opencl')
```

```
function main()
local opt = cmd:parse(arg)
-- Figure out the backend
local dtype, use_cudnn = utils.setup_gpu(opt.gpu, opt.backend, opt.use_cudnn == 1)

 -- Build the model
local model = nil
if opt.resume_from_checkpoint ~= '' then
print('Loading checkpoint from ' .. opt.resume_from_checkpoint)
model = torch.load(opt.resume_from_checkpoint).model:type(dtype)
else
print('Initializing model from scratch')
-- model = models.build_model(opt):type(dtype)
model = new_models.build_model(opt):type(dtype)
end

-- if use_cudnn then cudnn.convert(model, cudnn) end
model:training()
print(model)
local loader = DataLoader(opt)
local params, grad_params = model:getParameters()
local criterion = nn.MSECriterion():type(dtype)
local function f(x)
assert(x == params)
grad_params:zero()
```

```
-- x is y value, y is uv value
local x1, x2, y = loader:getBatch('train')
x1, x2, y = x1:type(dtype), x2:type(dtype), y:type(dtype)
x = {x1, x2}
-- Run model forward
local out = model:forward(x)
local grad_out = nil

-- This is a bit of a hack: if we are using reflect-start padding and the
-- output is not the same size as the input, lazily add reflection padding
-- to the start of the model so the input and output have the same size.

--[[
if opt.padding_type == 'reflect-start' and x:size(3) ~= out:size(3) then
local ph = (x:size(3) - out:size(3)) / 2
local pw = (x:size(4) - out:size(4)) / 2
local pad_mod = nn.SpatialReflectionPadding(pw, pw, ph, ph):type(dtype)
model:insert(pad_mod, 1)
out = model:forward(x)
end
]]--

local loss = criterion:forward(out,y)
grad_out = criterion:backward(out, y)
-- Run model backward
model:backward(x, grad_out)
```

```
-- Add regularization
-- grad_params:add(opt.weight_decay, params)

return loss, grad_params
end

local optim_state = {learningRate=opt.learning_rate}
local train_loss_history = {}
local val_loss_history = {}
local val_loss_history_ts = {}

for t = 1, opt.num_iterations do
-- collectgarbage()
local epoch = t / loader.num_minibatches['train']

local _, loss = optim.adam(f, params, optim_state)

table.insert(train_loss_history, loss[1])

print(string.format('Epoch %f, Iteration %d / %d, loss = %f',
epoch, t, opt.num_iterations, loss[1]), optim_state.learningRate)

if t % opt.checkpoint_every == 0 then
-- Check loss on the validation set
loader:reset('val')
model:evaluate()
local val_loss = 0
```

```
print 'Running on validation set ... '
local val_batches = opt.num_val_batches
for j = 1, val_batches do
-- local x, y = loader:getBatch('val')
-- x, y = x:type(dtype), y:type(dtype)

local x1, x2, y = loader:getBatch('val')
x1, x2, y = x1:type(dtype), x2:type(dtype), y:type(dtype)
x = {x1, x2}
local out = model:forward(x)
val_loss = val_loss + criterion:forward(out,y)

end

val_loss = val_loss / val_batches
print(string.format('val loss = %f', val_loss))
table.insert(val_loss_history, val_loss)
table.insert(val_loss_history_ts, t)
model:training()

-- Save a JSON checkpoint
local checkpoint = {
opt=opt,
train_loss_history=train_loss_history,
val_loss_history=val_loss_history,
val_loss_history_ts=val_loss_history_ts
}
```

```
local filename = string.format('%s.json', opt.checkpoint_name)
paths.mkdir(paths.dirname(filename))
utils.write_json(filename, checkpoint)

-- Save a torch checkpoint; convert the model to float first
model:clearState()
if use_cudnn then
cudnn.convert(model, nn)
end
model:float()
checkpoint.model = model
filename = string.format('%s.t7', opt.checkpoint_name)
torch.save(filename, checkpoint)

-- Convert the model back
model:type(dtype)
--[[
if use_cudnn then
cudnn.convert(model, cudnn)
end
]]--
params, grad_params = model:getParameters()
end

if opt.lr_decay_every > 0 and t % opt.lr_decay_every == 0 then
local new_lr = opt.lr_decay_factor * optim_state.learningRate
optim_state = {learningRate = new_lr}
```

end

object detection.cfg

```
[net]
# Testing
batch=1
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000 subdivisions=1
# Training
# batch=64
# subdivisions=8
width=416
height=416

max_batches = 500200
policy=steps
steps=400000,450000
scales=.1,.1
```

```
[convolutional]
batch_normalize=1
filters=32
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=64
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
filters=128
```

```
size=3
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=64
size=1
stride=1
pad=1
activation=leaky

[convolutional]
batch_normalize=1
filters=128
size=3
stride=1
pad=1
activation=leaky

[maxpool]
size=2
stride=2

[convolutional]
batch_normalize=1
```

filters=256

size=3

stride=1

pad=1

activation=leaky

[convolutional]

batch_normalize=1

filters=128

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch_normalize=1

filters=256

size=3

stride=1

pad=1

activation=leaky

[maxpool]

size=2

stride=2

[convolutional]

batch_normalize=1

filters=512

size=3

stride=1

pad=1

activation=leaky

[convolutional]

batch_normalize=1

filters=256

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch_normalize=1

filters=512

size=3

stride=1

pad=1

activation=leaky

[convolutional]

batch_normalize=1

filters=256

size=1

```
stride=1

pad=1

activation=leaky

[convolutional]

batch_normalize=1

filters=512

size=3

stride=1

pad=1

activation=leaky

[maxpool]

size=2

stride=2

[convolutional]

batch_normalize=1

filters=1024

size=3

stride=1

pad=1

activation=leaky

[convolutional]

batch_normalize=1

filters=512
```

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch_normalize=1

filters=1024

size=3

stride=1

pad=1

activation=leaky

[convolutional]

batch_normalize=1

filters=512

size=1

stride=1

pad=1

activation=leaky

[convolutional]

batch_normalize=1

filters=1024

size=3

stride=1

pad=1

activation=leaky

#######

[convolutional]

batch_normalize=1

size=3

stride=1

pad=1

filters=1024

activation=leaky

[convolutional]

batch_normalize=1

size=3

stride=1

pad=1

filters=1024

activation=leaky

[route]

layers=-9

[convolutional]

batch_normalize=1

size=1

```
stride=1

pad=1

filters=64

activation=leaky

[reorg]

stride=2

[route]

layers=-1,-4

[convolutional]

batch_normalize=1

size=3

stride=1

pad=1

filters=1024

activation=leaky

[convolutional]

size=1

stride=1

pad=1

filters=425

activation=linear
```

[region]

anchors = 0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828

bias_match=1

classes=80

coords=4

num=5

softmax=1

jitter=.3

rescore=1

object_scale=5

noobject_scale=1

class_scale=1

coord_scale=1

absolute=1

thresh = .6

random=1