

**EFFICIENT\_NET:**

*##%%writefile efficientnet\_model.py*

from efficientnet\_pytorch import EfficientNet

import torch.nn as nn

from torch.nn import functional as F

class **EfficientNetB1**(nn.Module):

def \_\_init\_\_(self, pretrained):

super(EfficientNetB1, self).\_\_init\_\_()

if pretrained **is** True:

self.model = EfficientNet.from\_pretrained("efficientnet-b1")

self.l0 = nn.Linear(1280, 1)

def forward(self, x):

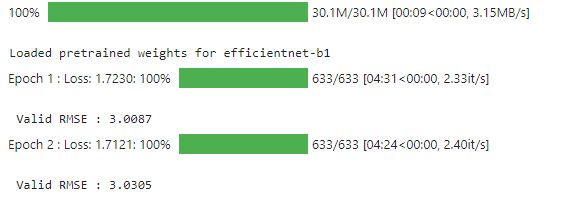
bs, \_, \_, \_ = x.shape

x = self.model.extract\_features(x)

x = F.adaptive\_avg\_pool2d(x, 1).reshape(bs, -1)

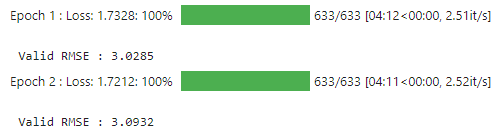
out = self.l0(x)

return out



**MOBILE\_NET\_V2:**





##%%writefile mobilenet\_v2\_model.py

from torchvision.models import mobilenet\_v2

import torch.nn as nn

from torch.nn import functional as F

class MobileNetV2(nn.Module):

def \_\_init\_\_(self, pretrained):

super(MobileNetV2, self).\_\_init\_\_()

if pretrained is True:

self.model = mobilenet\_v2(pretrained=True)

self.l0 = nn.Linear(1280, 1)

def forward(self, x):

bs, \_, \_, \_ = x.shape

x = self.model.features(x)

x = F.adaptive\_avg\_pool2d(x, 1).reshape(bs, -1)

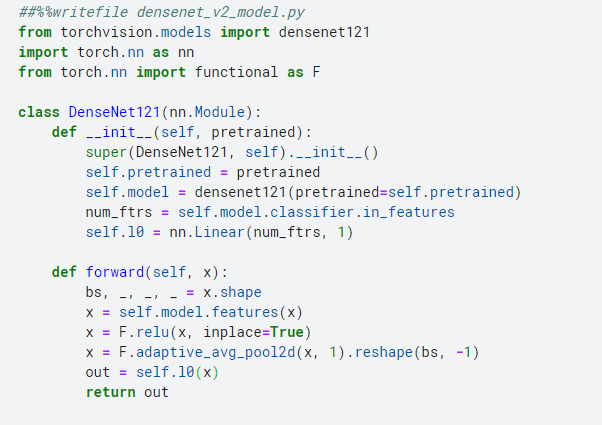
out = self.l0(x)

return out

# Create an instance of MobileNetV2

model = MobileNetV2(pretrained=True)

**DENSE\_NET:**

****

****

##%%writefile densenet\_121\_model.py

from torchvision.models import densenet121

import torch.nn as nn

from torch.nn import functional as F

class DenseNet121(nn.Module):

def \_\_init\_\_(self, pretrained):

super(DenseNet121, self).\_\_init\_\_()

self.pretrained = pretrained

self.model = densenet121(pretrained=self.pretrained)

num\_ftrs = self.model.classifier.in\_features

self.l0 = nn.Linear(num\_ftrs, 1)

def forward(self, x):

bs, \_, \_, \_ = x.shape

x = self.model.features(x)

x = F.relu(x, inplace=True)

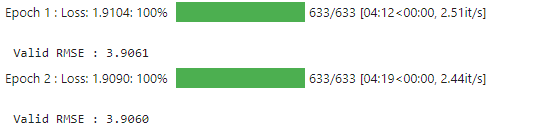
x = F.adaptive\_avg\_pool2d(x, 1).reshape(bs, -1)

out = self.l0(x)

return out

**Res\_NET:**

****

****

##%%writefile Resnet\_model.py

import torch.nn as nn

import torch.nn.functional as F

from torchvision.models import resnet18

class ResNet18(nn.Module):

def \_\_init\_\_(self, pretrained):

super(ResNet18, self).\_\_init\_\_()

if pretrained is True:

self.model = resnet18(pretrained=True)

else:

self.model = resnet18(pretrained=False, num\_classes=1000)

num\_ftrs = self.model.fc.in\_features

self.model.fc = nn.Linear(num\_ftrs, 1)

def forward(self, x):

bs, \_, \_, \_ = x.shape

x = self.model(x)

out = F.sigmoid(x)

return out