The standard old net:

```
class old_nn(nn.Module):
    def __init__(self):
        super(old_nn, self).__init__()
        self.fc1 = nn.Linear(32*32*3, 4096)
        self.fc2 = nn.Linear(4096, 4096)
        self.fc3 = nn.Linear(4096, n_classes) #last FC for classification

    def forward(self, x):
        x = x.view(x.shape[0], -1)
        x = F.sigmoid(self.fc1(x))
        x = F.sigmoid(self.fc2(x))
        x = self.fc3(x)
        return x


#function to define the convolutional network
class CNN(nn.Module):
```

The better CNN found from home work points:

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        #conv2d first parameter is the number of kernels at input (you get it from the output value of the previous layer)
        #conv2d second parameter is the number of kernels you wanna have in your convolution, so it will be the n. of kernels at output.
        #conv2d third, fourth and fifth parameters are, as you can read, kernel_size, stride and zero padding :)
        self.conv1 = nn.Conv2d(3, 128, kernel_size=5, stride=2, padding=0)
        self.conv1_batchn =nn.BatchNorm2d(128)
        self.conv2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv2_batchn =nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv3_batchn =nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=0)
        self.convf_batchn =nn.BatchNorm2d(256)
        self.fc1 = nn.Linear(256 * 4 * 4, 4096) #64 kernels e 4*4  immagine
        self.fc2 = nn.Linear(4096, n_classes) #last FC for classification


    def forward(self, x):
        x = self.conv1_batchn(F.relu(self.conv1(x)))
        x = self.conv2_batchn(F.relu(self.conv2(x)))
        x = self.conv3_batchn(F.relu(self.conv3(x)))
        x = self.convf_batchn (F.relu(self.pool(self.conv_final(x))))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))

        x = self.fc2(x)

        return x
```

The better CNN found freely:

```
class myCNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 128, kernel_size=5, stride=2, padding=0)
        self.conv1_batchn =nn.BatchNorm2d(128)
        self.conv2 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv2_batchn =nn.BatchNorm2d(128)
        self.conv3 = nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=0)
        self.conv3_batchn =nn.BatchNorm2d(128)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.conv_final = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=0)
        self.convf_batchn =nn.BatchNorm2d(256)
        self.fc1 = nn.Linear(256 * 4 * 4, 4096) #64 kernels e 4*4  immagine
        self.dropfc = nn.Dropout(0.8)
        self.fc2 = nn.Linear(4096, n_classes) #last FC for classification


    def forward(self, x):
        x = self.conv1_batchn(F.relu(self.conv1(x)))
        x = self.conv2_batchn(F.relu(self.conv2(x)))
        x = self.conv3_batchn(F.relu(self.conv3(x)))
        x = self.convf_batchn (F.relu(self.pool(self.conv_final(x))))
        x = x.view(x.shape[0], -1)
        x = F.relu(self.fc1(x))
        x = self.dropfc(x)
        x = self.fc2(x)
        return x
```

It’s required to uncomment the following lines and uncomment the similar lines if necessary.

```
  ##Uncomment the following row  for myCNN
  #transforms.RandomHorizontalFlip(),
```

```
##Uncomment the following rows for myCNN
#trainset = torchvision.datasets.CIFAR100(root='./data', train=True,
#                                        download=True, transform=transform_train)
#trainloader = torch.utils.data.DataLoader(trainset, batch_size=256,
#                                          shuffle=True, num_workers=4,drop_last=True)
#testset = torchvision.datasets.CIFAR100(root='./data', train=False,
#                                       download=True, transform=transform_test)
#testloader = torch.utils.data.DataLoader(testset, batch_size=256,
#                                         shuffle=False, num_workers=4,drop_last=True)
```

```
#for myCNN
#net = myCNN()
###
```

```
##Uncomment the following row  for myCNN
#optimizer = optim.Adam(net.parameters(), lr=0.0001) #better convergency w.r.t simple SGD :)
```