An Investigation of Pretraining-based ResNet Model for Face Recognition

Keyu Wang[1]

[1] School of Artificial Intelligence, Southeast University, Nanjing, China

ky-wang@seu.edu.cn

Abstract. Face recognition is an important issue in computer vision. The appearance of convolutional neural network has greatly promoted the development of face recognition. However, problems arise when only small amount training data is available. In this research, I propose a pretraining-based ResNet model for the task of Class Face Recognition Project. I show the training will be converged though only several hundred images are provided. Finally, I conduct experiments to evaluate the proposed pretraining-based ResNet. The results show that my proposed method achieves a better performance on the Class Face Recognition Project, comparing with traditional machine learning algorithm and simple conventional neural network architecture.

1.Introduction

Convolutional Neural Network (CNN) are critical for the success of the extensive use of face recognition since it can capture the local features of the image and is easier to train. The development of research for face recognition using CNN is further accelerated with the proposal of ResNet, which effectively solves the problem of gradient vanishing of deep neural network. However, problems arise, i.e., hard to converge in training, when only small amount training data is available. In this research, I will propose a pretrained-based Resnet model for the task of Class Face Recognition Project. I use the pre-training model trained in some corpus, and use designed ResNet model to train with images of classmates. Finally, I conduct experiments to evaluate the performance of my proposed method. The experimental results show that the model achieve good results in the class face recognition task.

The rest of this proposal is organized as follows: Section 2 introduces the preliminary concepts. In Section 3, I discuss about related work. Section 4 gives my model using pretrained-based ResNet. Section 5 reports evaluation results. Section 6 provides conclusion and future work.

2.Preliminaries

In this section, I introduce some preliminary concepts. I assume that the reader is familiar with basic concepts of the task of face recognition and conventional neural network. But some concepts are important enough to be emphasized here. Face recognition in a broad sense actually includes a series of related technologies for building a face recognition system, including face image acquisition, face location or detection, face recognition preprocessing, identity confirmation and identity search, etc. In the narrow sense, face recognition refers to the technology or system of identity confirmation or identity search through faces. Convolutional Neural Network,CNN) is a kind of feedforward neural network. Its artificial neurons can respond to a part of the surrounding units in the coverage area, and it has excellent performance for large-scale image processing. It includes a convolutional layer and a pooling layer. Below I introduce some background about Resnet and Pre-training model, which is used in my proposed method.

Resnet. A residual neural network (Resnet) is an artificial neural network (ANN). It is also used for Control Neural Network. It is a gateless or open-gated variant of the HighwayNets, the first working very deep feedforward neural network with hundreds of layers, much deeper than previous neural networks. Skip connections or shortcuts are used to jump over some layers. Typical ResNet models are implemented with double- or triple- layer skips that contain nonlinearities (ReLU) and batch normalization in between. Models with several parallel skips are referred to as DenseNets. In the context of residual neural networks, a non-residual network may be described as a plain network.

Pre-training model. Pre-training model is a deep learning architecture that has been trained to perform specific tasks on a large amount of data (for example, identifying classification problems in pictures). This kind of training is not easy to perform, and it usually requires a lot of resources. Beyond the resources available to many people who can be used for deep learning models, I don't have a large batch of GPUs. When talking about pre-training model, it usually refers to CNN (architecture for vision-related tasks) trained on Imagenet.

3.Literature Review

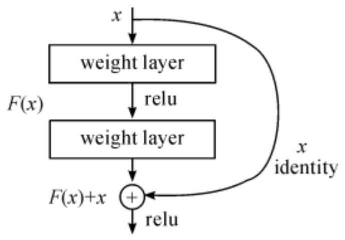
Research in face recognition can be dated back to 1960s [4]. From 1964 to 1966 Woodrow W. Bledsoe, along with Helen Chan and Charles Bisson of Panoramic Research, researched programming computers to recognize human faces. Teir program asks the administrator to locate the eyes, ears, nose and mouth in the photo. Ten, the reference data can be use comparison with the distance and measures. However, because of inconvenience, this work has not received much recognition. Peter Hart at the Stanford Research Institute continued this research, and found optimistic results when using a set of images instead of a set of feature points. Since then, there have been many researchers following on this subject and a substantial amount of efforts have been made to find the optimal face recognition method. In the 1970s, Goldstein, Harmon, and Lesk used 21 specific subjective markers such as hair color and lip thickness to automatically identify human faces. The attempt obtained good recognition accuracy. However, the feature measurement and locationing are manually calculated. It is impractical to apply this method to many faces. In 1991, Turk and Pentland proposed a method of using principal component analysis (PCA) to handle face data [5]. This is called the eigenface algorithm which is already become a golden standard for face recognition, inspired by eigenface, a large number of such algorithms were proposed [6–8].

4.Method

4.1 Model

In the framework of pre-training ResNet, I define the following modules A convolution layer MBConv2d. input parameters: in_planes, out_planes, kernel_size, stride, padding, convolution operation, batch nomalize and activation function relu for the input in turn, and obtain the output result.

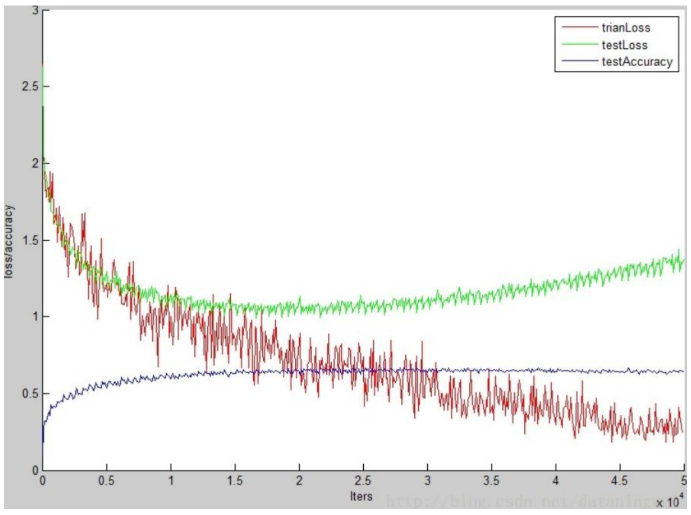
The residual network unit structure is shown in the following figure. Block 35, Block 17 and Block 8 respectively define different residual network module architectures. These modules are composed of the basic units of residual network. Detailed codes are attached in Appendix.



Pool+mixed layer, splicing convolution modules of different convolution, and then defining two kinds of modules, Mixed_6a and Mixed_7, through MaxPool2d pool layer. See in Appendix for detailed codes and for the overall Resnet framework.

4.2 Training

Download the pre-training model vggface2, and use the class photos for training. The training curves are shown in the following figure. It can be seen that the training loss can basically converge.



5.Experiment Evaluation and Results

5.1 Data

The photos of 76 students of prof. Wang's ML class. There are 5 photos for each students. The data are divided into training data and valid data with respect to the ratio of 4:1.

5.2 Baselines and Metrics

To evaluate my proposed method, some simple methods are selected as baselines.

- **Logistic Regression.**
- **LeNet.** It is a simple model of CNN.

These methods are evaluated with respect to the accuracy rate and the efficiency. the accuracy rate is defined:

$$\text{Accuracy}=\frac{\text{TP}}{\text{TP+FP}}\times 100\%$$

5.3 Experiment Results

Accuracy rate is showed in the table:

model	**Logistic Regression**	**LeNet**	**Pretrained-based ResNet**
Accuracy	3.3%	8.4%	63.4%

Experiment shows the method proposed can outperform the baselines.

6.Conclusion and future work

In this proposal, I introduce an approach to recognizing face images using pretrained-ResNet when small amount data provided. The experiment results can outperform traditional ML model and CNN model. Good results have also been achieved in classroom acceptance.

In future work, more detailed parameter tuning is needed to achieve a better accuracy.

References

1. Xu C, Wang Y, Tan T, Quan L (2004) Depth vs. intensity: which is more important for face recognition? Int Conf Pattern Recognit

2:342–345 4. Bowyer KW, Chang PFK (2006) A survey of approaches and challenges in 3d and multi-modal 3d + 2d face recognition. Comput Vis Image Underst 101:1–15

3. Zhao W, Chellappa PJPARR (2003) Face recognition: a literature survey. ACM Comput Surv 35:399–485

4. Bledsoe WW (1966) The model method in facial recognition. Panoramic Research

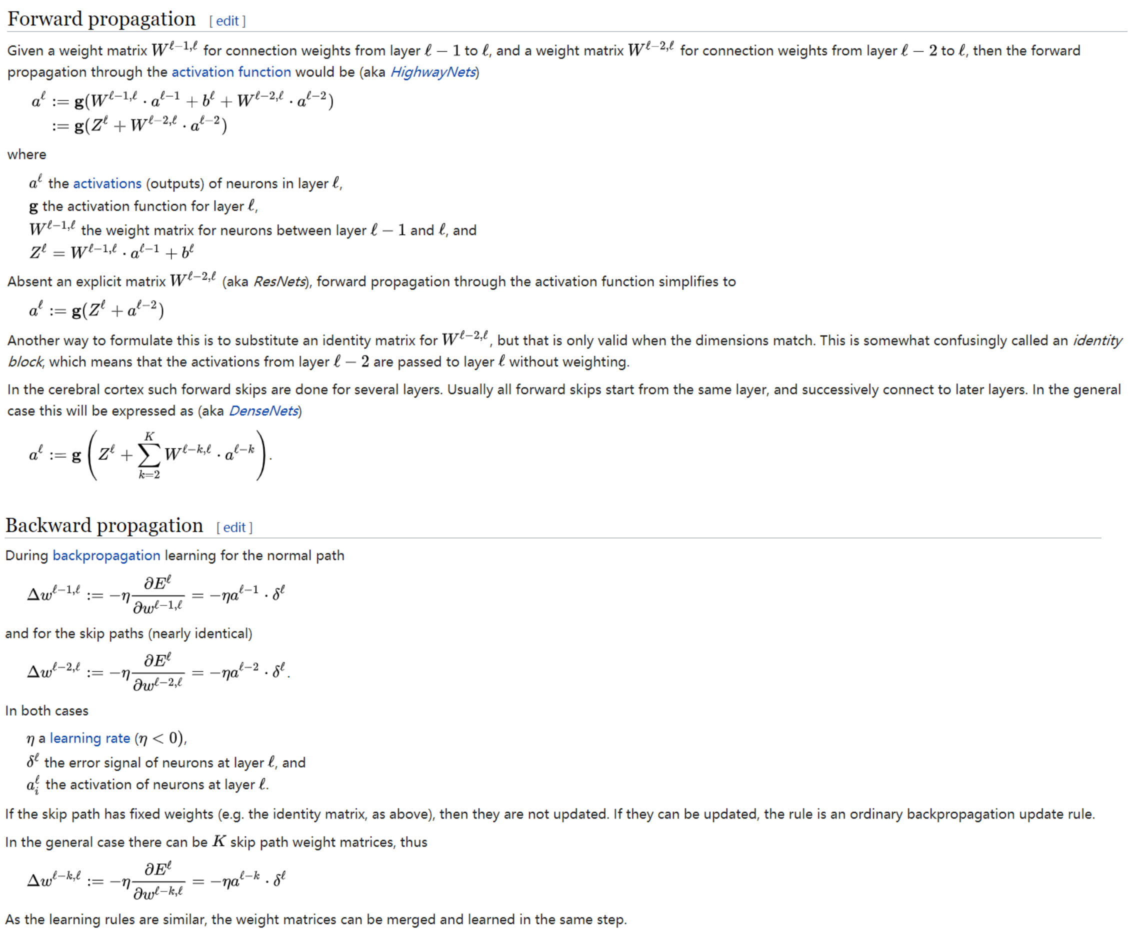
5. Pentland MT (1991) Face recognition using eigenfaces. Computer vision and pattern recognition. pp 586–591

6. Belhumeur PN, Hespanha DJKJP (1997) Eigenfaces vs. fsherfaces: recognition using class specifc linear projection. Trans Pattern Anal Mach Intell 19:711–720 18. Frey BJ, Colmenarez TSH A (1998) Mixtures of local linear subspaces for face recognition. In: Computer vision and pattern recognition 7. Moghaddam B, Jebara APT (2000) Bayesian face recognition. Pattern Recognit 33:1771–1782 20. Wiskott L, Fellous JM, Kruger N, Von Der Malsburg C (1997) Face recognition by elasric bunch graph matching. PAMI 17:775–779

8. Mpiperis I, Malassiotis MGSS (2007) 3-d face recognition with the geodesic polar representation. Inform Forensics Secur 5:537–547

Appendix

1. **mathematical principle explanation** (from Wikipedia)



Forward propagation [edit]

Given a weight matrix $W^{\ell-1,\ell}$ for connection weights from layer $\ell-1$ to ℓ, and a weight matrix $W^{\ell-2,\ell}$ for connection weights from layer $\ell-2$ to ℓ, then the forward propagation through the activation function would be (aka *HighwayNets*)

$$\begin{aligned} a^\ell &:= \mathbf{g}(W^{\ell-1,\ell}\cdot a^{\ell-1} + b^\ell + W^{\ell-2,\ell}\cdot a^{\ell-2}) \\ &:= \mathbf{g}(Z^\ell + W^{\ell-2,\ell}\cdot a^{\ell-2}) \end{aligned}$$

where

a^ℓ the activations (outputs) of neurons in layer ℓ,
$\mathbf{g}$ the activation function for layer ℓ,
$W^{\ell-1,\ell}$ the weight matrix for neurons between layer $\ell-1$ and ℓ, and
$Z^\ell = W^{\ell-1,\ell}\cdot a^{\ell-1} + b^\ell$

Absent an explicit matrix $W^{\ell-2,\ell}$ (aka *ResNets*), forward propagation through the activation function simplifies to

$$a^\ell := \mathbf{g}(Z^\ell + a^{\ell-2})$$

Another way to formulate this is to substitute an identity matrix for $W^{\ell-2,\ell}$, but that is only valid when the dimensions match. This is somewhat confusingly called an *identity block*, which means that the activations from layer $\ell-2$ are passed to layer ℓ without weighting.

In the cerebral cortex such forward skips are done for several layers. Usually all forward skips start from the same layer, and successively connect to later layers. In the general case this will be expressed as (aka *DenseNets*)

$$a^\ell := \mathbf{g}\left(Z^\ell + \sum_{k=2}^{K} W^{\ell-k,\ell}\cdot a^{\ell-k}\right).$$

Backward propagation [edit]

During backpropagation learning for the normal path

$$\Delta w^{\ell-1,\ell} := -\eta\frac{\partial E^\ell}{\partial w^{\ell-1,\ell}} = -\eta a^{\ell-1}\cdot\delta^\ell$$

and for the skip paths (nearly identical)

$$\Delta w^{\ell-2,\ell} := -\eta\frac{\partial E^\ell}{\partial w^{\ell-2,\ell}} = -\eta a^{\ell-2}\cdot\delta^\ell.$$

In both cases

η a learning rate ($\eta < 0$),
δ^ℓ the error signal of neurons at layer ℓ, and
a_i^ℓ the activation of neurons at layer ℓ.

If the skip path has fixed weights (e.g. the identity matrix, as above), then they are not updated. If they can be updated, the rule is an ordinary backpropagation update rule.

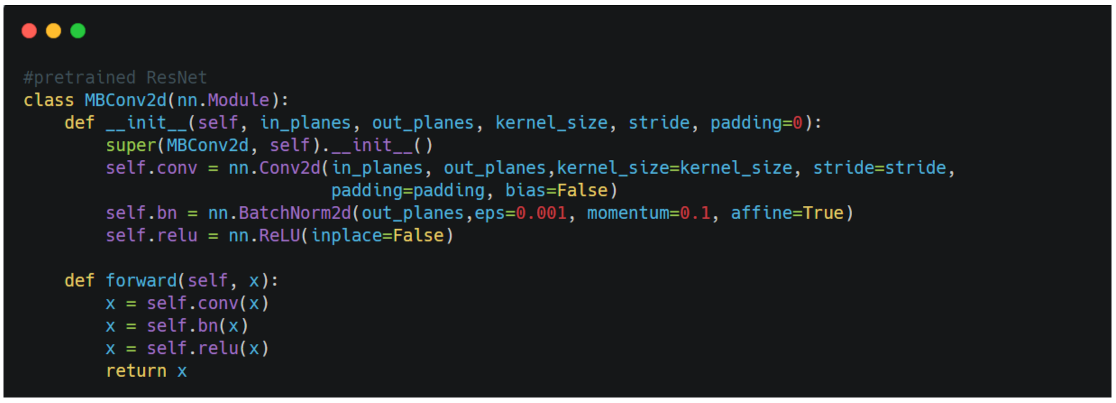
In the general case there can be K skip path weight matrices, thus

$$\Delta w^{\ell-k,\ell} := -\eta\frac{\partial E^\ell}{\partial w^{\ell-k,\ell}} = -\eta a^{\ell-k}\cdot\delta^\ell$$

As the learning rules are similar, the weight matrices can be merged and learned in the same step.

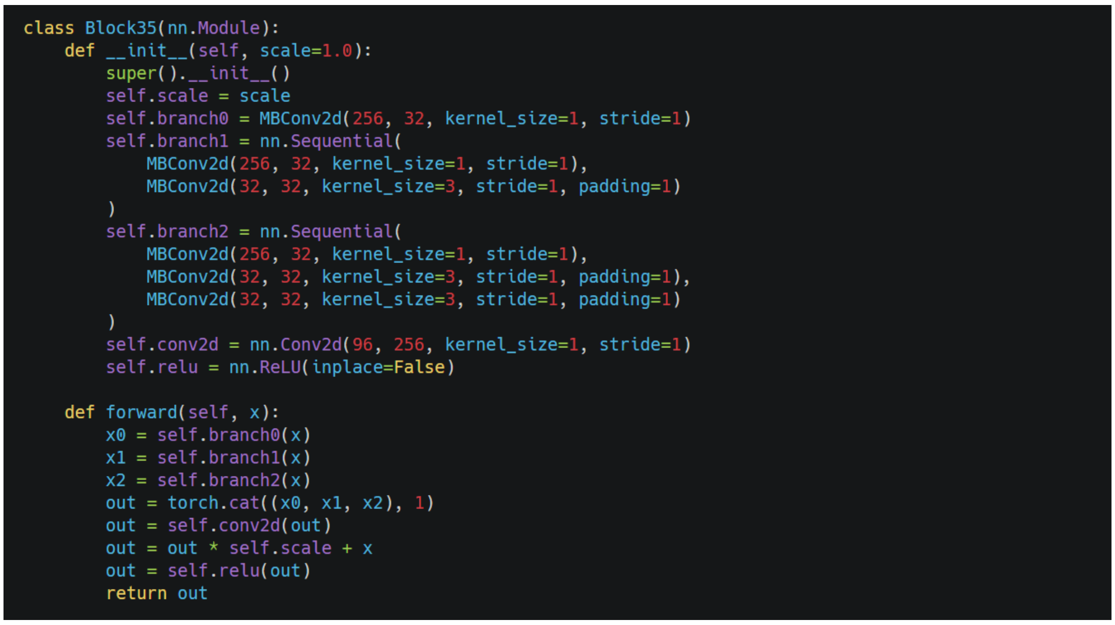
2. **source code of the model**

MBConv2d



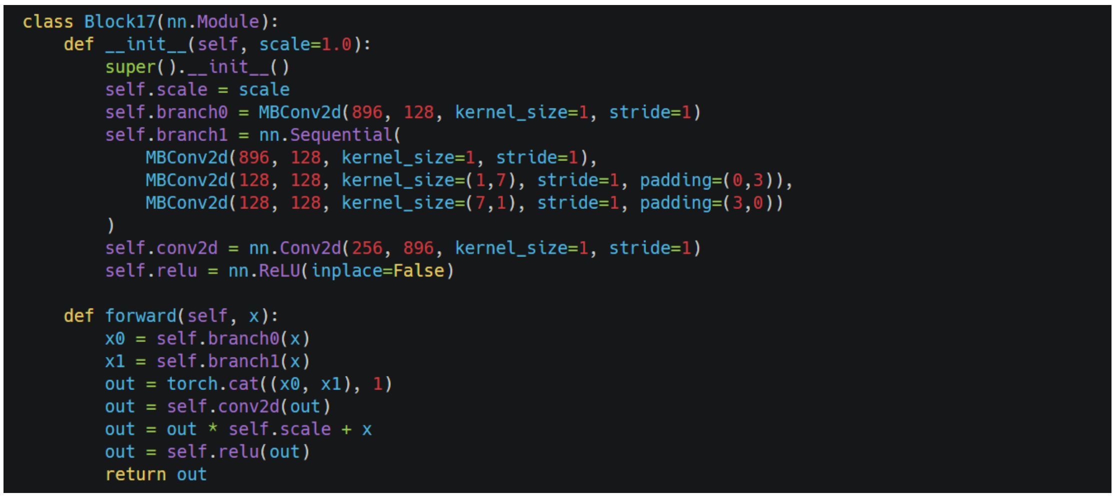
```
#pretrained ResNet
class MBConv2d(nn.Module):
    def __init__(self, in_planes, out_planes, kernel_size, stride, padding=0):
        super(MBConv2d, self).__init__()
        self.conv = nn.Conv2d(in_planes, out_planes,kernel_size=kernel_size, stride=stride,
                              padding=padding, bias=False)
        self.bn = nn.BatchNorm2d(out_planes,eps=0.001, momentum=0.1, affine=True)
        self.relu = nn.ReLU(inplace=False)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        x = self.relu(x)
        return x
```

Block35



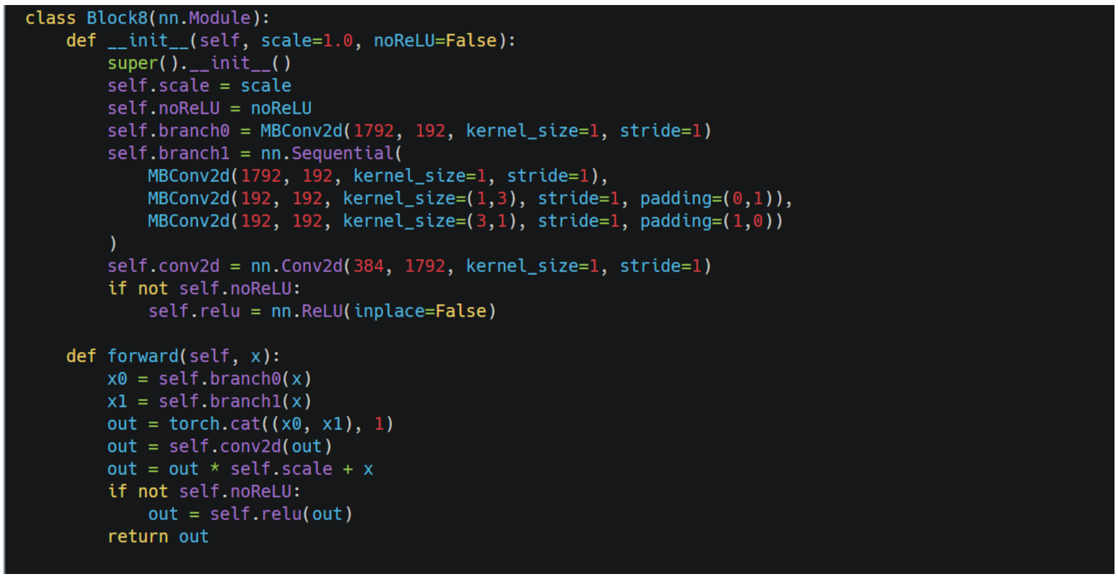
```python
class Block35(nn.Module):
    def __init__(self, scale=1.0):
        super().__init__()
        self.scale = scale
        self.branch0 = MBConv2d(256, 32, kernel_size=1, stride=1)
        self.branch1 = nn.Sequential(
            MBConv2d(256, 32, kernel_size=1, stride=1),
            MBConv2d(32, 32, kernel_size=3, stride=1, padding=1)
        )
        self.branch2 = nn.Sequential(
            MBConv2d(256, 32, kernel_size=1, stride=1),
            MBConv2d(32, 32, kernel_size=3, stride=1, padding=1),
            MBConv2d(32, 32, kernel_size=3, stride=1, padding=1)
        )
        self.conv2d = nn.Conv2d(96, 256, kernel_size=1, stride=1)
        self.relu = nn.ReLU(inplace=False)

    def forward(self, x):
        x0 = self.branch0(x)
        x1 = self.branch1(x)
        x2 = self.branch2(x)
        out = torch.cat((x0, x1, x2), 1)
        out = self.conv2d(out)
        out = out * self.scale + x
        out = self.relu(out)
        return out
```

Block17



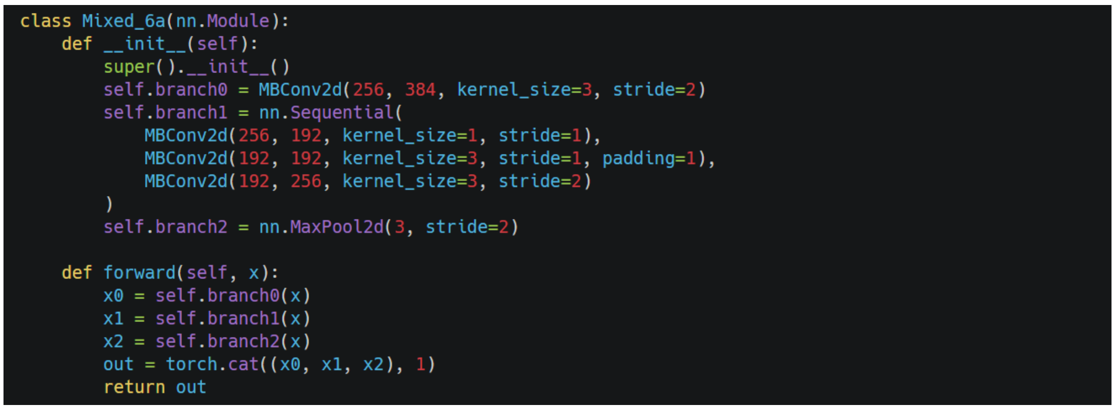
```python
class Block17(nn.Module):
    def __init__(self, scale=1.0):
        super().__init__()
        self.scale = scale
        self.branch0 = MBConv2d(896, 128, kernel_size=1, stride=1)
        self.branch1 = nn.Sequential(
            MBConv2d(896, 128, kernel_size=1, stride=1),
            MBConv2d(128, 128, kernel_size=(1,7), stride=1, padding=(0,3)),
            MBConv2d(128, 128, kernel_size=(7,1), stride=1, padding=(3,0))
        )
        self.conv2d = nn.Conv2d(256, 896, kernel_size=1, stride=1)
        self.relu = nn.ReLU(inplace=False)

    def forward(self, x):
        x0 = self.branch0(x)
        x1 = self.branch1(x)
        out = torch.cat((x0, x1), 1)
        out = self.conv2d(out)
        out = out * self.scale + x
        out = self.relu(out)
        return out
```

Block8



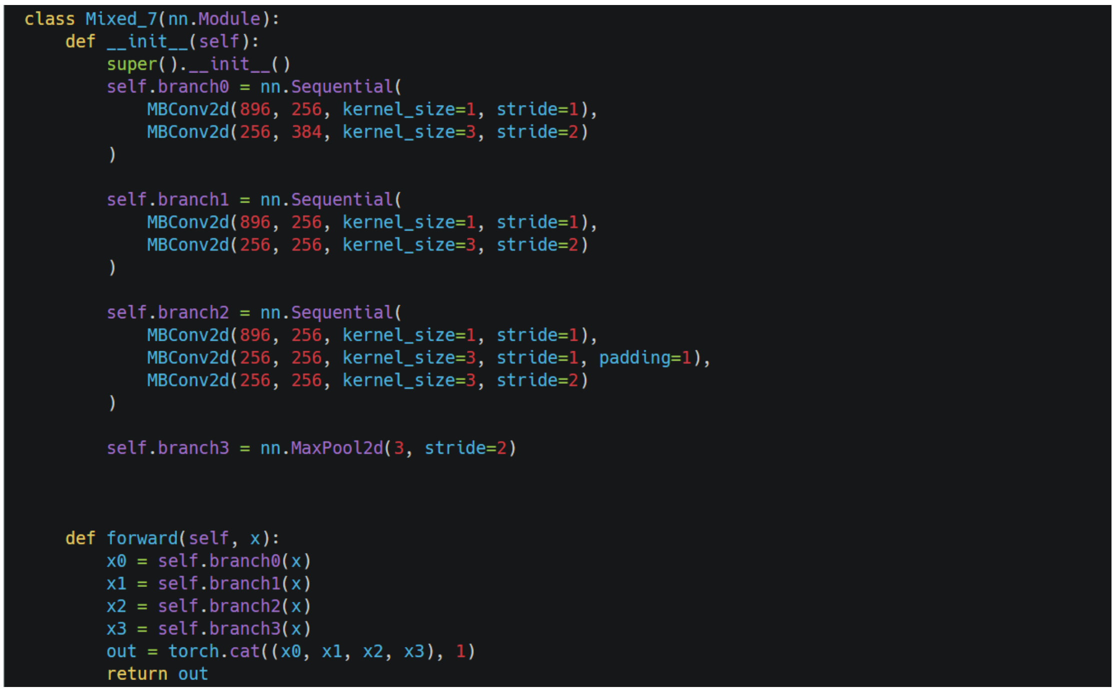
```python
class Block8(nn.Module):
    def __init__(self, scale=1.0, noReLU=False):
        super().__init__()
        self.scale = scale
        self.noReLU = noReLU
        self.branch0 = MBConv2d(1792, 192, kernel_size=1, stride=1)
        self.branch1 = nn.Sequential(
            MBConv2d(1792, 192, kernel_size=1, stride=1),
            MBConv2d(192, 192, kernel_size=(1,3), stride=1, padding=(0,1)),
            MBConv2d(192, 192, kernel_size=(3,1), stride=1, padding=(1,0))
        )
        self.conv2d = nn.Conv2d(384, 1792, kernel_size=1, stride=1)
        if not self.noReLU:
            self.relu = nn.ReLU(inplace=False)

    def forward(self, x):
        x0 = self.branch0(x)
        x1 = self.branch1(x)
        out = torch.cat((x0, x1), 1)
        out = self.conv2d(out)
        out = out * self.scale + x
        if not self.noReLU:
            out = self.relu(out)
        return out
```

Mixed_6a



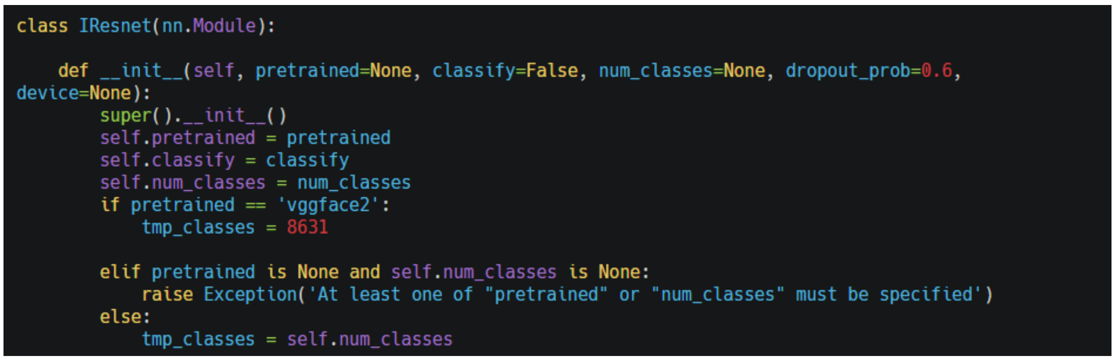
```python
class Mixed_6a(nn.Module):
    def __init__(self):
        super().__init__()
        self.branch0 = MBConv2d(256, 384, kernel_size=3, stride=2)
        self.branch1 = nn.Sequential(
            MBConv2d(256, 192, kernel_size=1, stride=1),
            MBConv2d(192, 192, kernel_size=3, stride=1, padding=1),
            MBConv2d(192, 256, kernel_size=3, stride=2)
        )
        self.branch2 = nn.MaxPool2d(3, stride=2)

    def forward(self, x):
        x0 = self.branch0(x)
        x1 = self.branch1(x)
        x2 = self.branch2(x)
        out = torch.cat((x0, x1, x2), 1)
        return out
```

Mixed_7

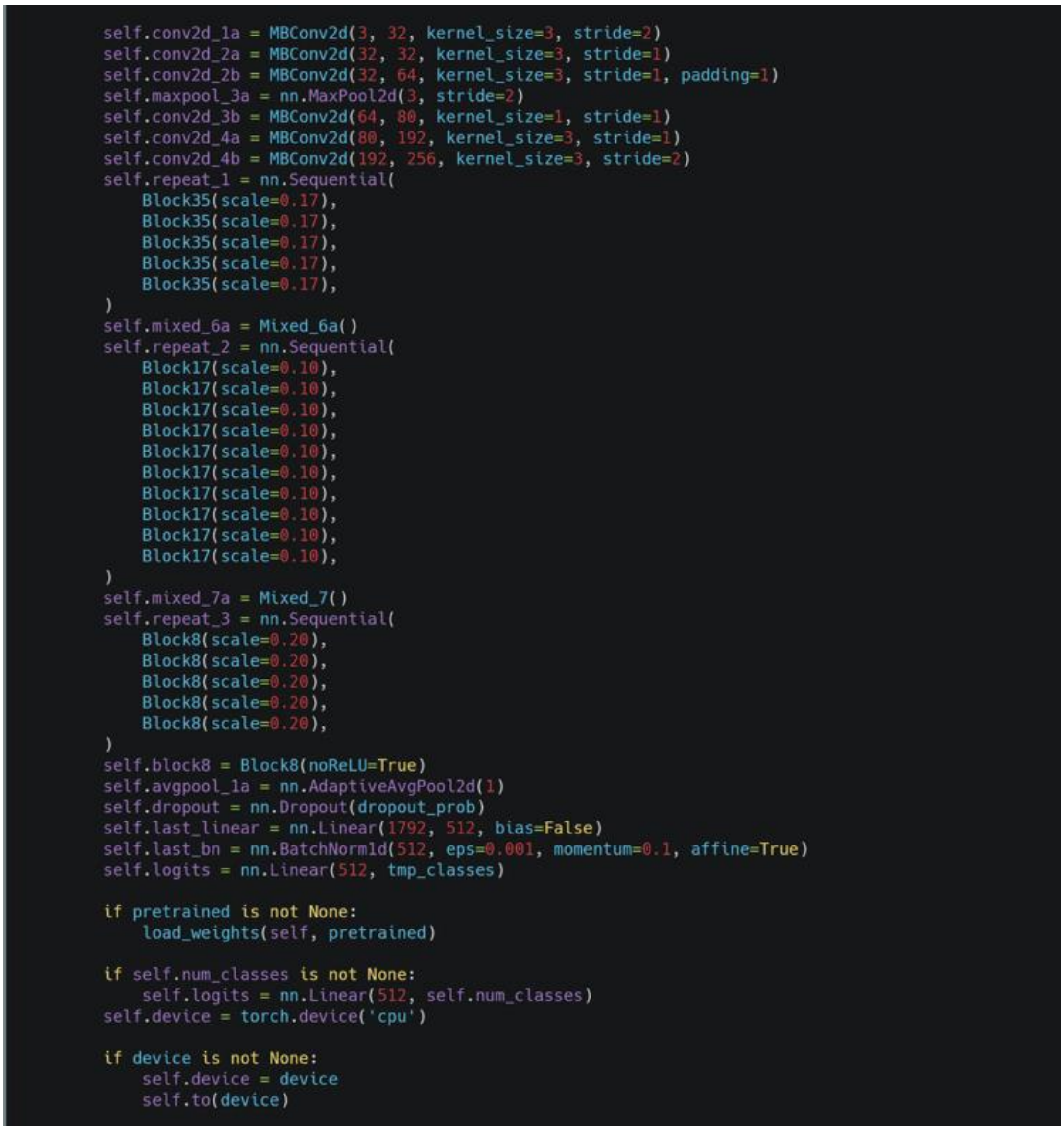


```python
class Mixed_7(nn.Module):
    def __init__(self):
        super().__init__()
        self.branch0 = nn.Sequential(
            MBConv2d(896, 256, kernel_size=1, stride=1),
            MBConv2d(256, 384, kernel_size=3, stride=2)
        )

        self.branch1 = nn.Sequential(
            MBConv2d(896, 256, kernel_size=1, stride=1),
            MBConv2d(256, 256, kernel_size=3, stride=2)
        )

        self.branch2 = nn.Sequential(
            MBConv2d(896, 256, kernel_size=1, stride=1),
            MBConv2d(256, 256, kernel_size=3, stride=1, padding=1),
            MBConv2d(256, 256, kernel_size=3, stride=2)
        )

        self.branch3 = nn.MaxPool2d(3, stride=2)



    def forward(self, x):
        x0 = self.branch0(x)
        x1 = self.branch1(x)
        x2 = self.branch2(x)
        x3 = self.branch3(x)
        out = torch.cat((x0, x1, x2, x3), 1)
        return out
```

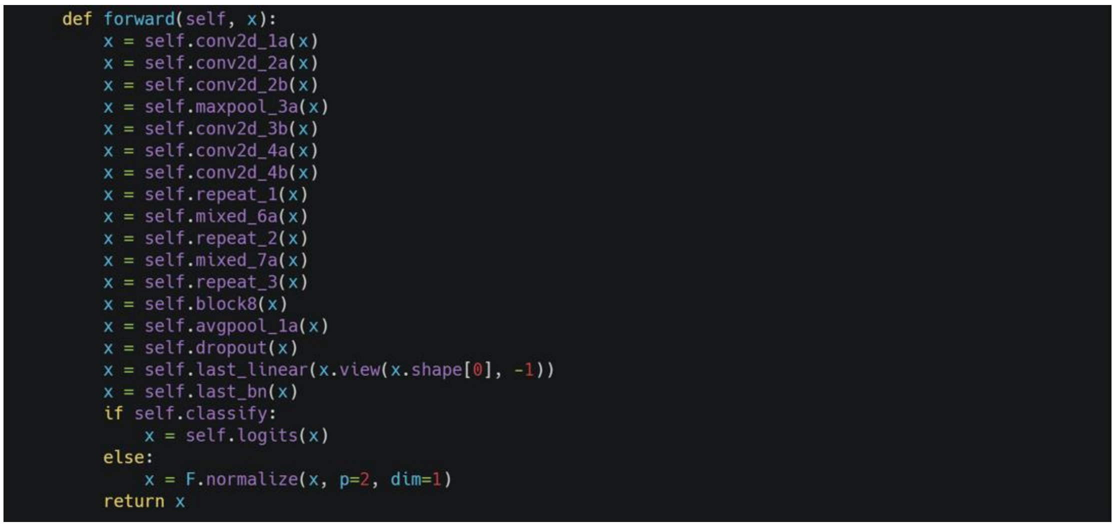
Overall architecture



```python
class IResnet(nn.Module):

    def __init__(self, pretrained=None, classify=False, num_classes=None, dropout_prob=0.6,
device=None):
        super().__init__()
        self.pretrained = pretrained
        self.classify = classify
        self.num_classes = num_classes
        if pretrained == 'vggface2':
            tmp_classes = 8631

        elif pretrained is None and self.num_classes is None:
            raise Exception('At least one of "pretrained" or "num_classes" must be specified')
        else:
            tmp_classes = self.num_classes
```



```
        self.conv2d_1a = MBConv2d(3, 32, kernel_size=3, stride=2)
        self.conv2d_2a = MBConv2d(32, 32, kernel_size=3, stride=1)
        self.conv2d_2b = MBConv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.maxpool_3a = nn.MaxPool2d(3, stride=2)
        self.conv2d_3b = MBConv2d(64, 80, kernel_size=1, stride=1)
        self.conv2d_4a = MBConv2d(80, 192, kernel_size=3, stride=1)
        self.conv2d_4b = MBConv2d(192, 256, kernel_size=3, stride=2)
        self.repeat_1 = nn.Sequential(
            Block35(scale=0.17),
            Block35(scale=0.17),
            Block35(scale=0.17),
            Block35(scale=0.17),
            Block35(scale=0.17),
        )
        self.mixed_6a = Mixed_6a()
        self.repeat_2 = nn.Sequential(
            Block17(scale=0.10),
            Block17(scale=0.10),
            Block17(scale=0.10),
            Block17(scale=0.10),
            Block17(scale=0.10),
            Block17(scale=0.10),
            Block17(scale=0.10),
            Block17(scale=0.10),
            Block17(scale=0.10),
            Block17(scale=0.10),
        )
        self.mixed_7a = Mixed_7()
        self.repeat_3 = nn.Sequential(
            Block8(scale=0.20),
            Block8(scale=0.20),
            Block8(scale=0.20),
            Block8(scale=0.20),
            Block8(scale=0.20),
        )
        self.block8 = Block8(noReLU=True)
        self.avgpool_1a = nn.AdaptiveAvgPool2d(1)
        self.dropout = nn.Dropout(dropout_prob)
        self.last_linear = nn.Linear(1792, 512, bias=False)
        self.last_bn = nn.BatchNorm1d(512, eps=0.001, momentum=0.1, affine=True)
        self.logits = nn.Linear(512, tmp_classes)

        if pretrained is not None:
            load_weights(self, pretrained)

        if self.num_classes is not None:
            self.logits = nn.Linear(512, self.num_classes)
        self.device = torch.device('cpu')

        if device is not None:
            self.device = device
            self.to(device)
```



```
    def forward(self, x):
        x = self.conv2d_1a(x)
        x = self.conv2d_2a(x)
        x = self.conv2d_2b(x)
        x = self.maxpool_3a(x)
        x = self.conv2d_3b(x)
        x = self.conv2d_4a(x)
        x = self.conv2d_4b(x)
        x = self.repeat_1(x)
        x = self.mixed_6a(x)
        x = self.repeat_2(x)
        x = self.mixed_7a(x)
        x = self.repeat_3(x)
        x = self.block8(x)
        x = self.avgpool_1a(x)
        x = self.dropout(x)
        x = self.last_linear(x.view(x.shape[0], -1))
        x = self.last_bn(x)
        if self.classify:
            x = self.logits(x)
        else:
            x = F.normalize(x, p=2, dim=1)
        return x
```