

<> Code

Issues 99

Pull requests 20

Discussions

Actions

Projects

Wiki

Security

New issue

Jump to bottom

model config explain #6142

Closed

1 task done

iumyx2612 opened this issue on Dec 31, 2021 · 28 comments

Labels

question

iumyx2612 commented on Dec 31, 2021

Contributor

Search before asking

☒ I have searched the YOLOv5 [issues](#) and [discussions](#) and found no similar questions.

Question

Can you clearly explain the config file, for example yolov5s.yaml

```
# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]], # 0-P1/2
  [-1, 1, Conv, [128, 3, 2]], # 1-P2/4
  [-1, 3, C3, [128]],
  [-1, 1, Conv, [256, 3, 2]], # 3-P3/8
  [-1, 9, C3, [256]],
  [-1, 1, Conv, [512, 3, 2]], # 5-P4/16
  [-1, 9, C3, [512]],
  [-1, 1, Conv, [1024, 3, 2]], # 7-P5/32
  [-1, 1, SPP, [1024, [5, 9, 13]]],
  [-1, 3, C3, [1024, False]], # 9
  ]
```

I understand that `module` is the module class from `models/common.py`
But what is `from`, `number` and `args` ?
And what is the meaning of the comments like `# 0-P1/2`, `# 1-P2/4` etc.
And how did a string from *.yaml file can be cast to a module class in `yolo.py` [line 251](#)

Additional

No response

iumyx2612 added the question label on Dec 31, 2021

glenn-jocher commented on Dec 31, 2021

Member

@iumyx2612



iumyx2612 commented on Jan 1, 2022

Contributor

Author

[@iumyx2612](#)

from : from which layer the module input comes from. Uses python syntax so -1 indicates prior layer. number : indicates the number of times a module repeats or how many repeats repeatable modules like C3 use args : module arguments (input channels inherited automatically)

For example:

```
[-1, 1, Conv, [128, 3, 2]], # 1-P2/4
```

should be:


```
Conv(c1=what_ever_channel_from_prior_layer, c2=128, k=3, s=2)
```

Am I right?



glenn-jocher commented on Jan 1, 2022

Member

[@iumyx2612](#) yes exactly, that's right!
 iumyx2612 closed this as completed on Jan 1, 2022

alkhalisy commented on Oct 13, 2023 • edited

Dear Sir

Can you clearly explain the the word 'nearest' , "None" and the value "2" in config file, for example yolov5s.yaml

```
[-1, 1, nn.Upsample, [None, 2, 'nearest']],
```

and the word "False" in

```
[-1, 3, C3, [512, False]],
```

Also the last '2' [-1, 1, Conv, [64, 6, 2, 2]] is denoted to Padding or Stride?



glenn-jocher commented on Oct 13, 2023

Member

[@alkhalisy](#) dear Sir,

In the YOLOv5 config file, the term 'nearest' in the line [-1, 1, nn.Upsample, [None, 2, 'nearest']] refers to the upsampling method used for resizing the input. Here, 'nearest' indicates that the nearest-neighbor upsampling method will be employed.

The value 'None' in the same line [None, 2, 'nearest'] refers to the size of the output after upsampling. When 'None' is used, the size of the output will be determined automatically.

Lastly, the '2' in `[-1, 1, Conv, [64, 6, 2, 2]]` represents the stride value of the convolutional layer. It determines the step size of the kernel as it moves across the input. In this case, a stride of '2' implies that the kernel will move by two units at each step.

I hope this clarifies your questions. Please let me know if you have any further inquiries.

Kind regards,
Glenn Jocher



alkhalisy commented on Oct 13, 2023

[@glenn-jocher](#) Dear Sir

Thank you very much for your clarifying , but please just another question , why the size of input and output in C3 module are same?. `[-1, 3, C3, [512, False]]`, can I ask for explanation of how C3 working? is the the attention mechanism you referred in the c3 module are the first two asymmetric convolutions used for compressed information ?



glenn-jocher commented on Oct 14, 2023

Member

[@alkhalisy](#)

Regarding your question about the input and output size in the C3 module, it may appear that they are the same, but in fact, the C3 module performs additional operations within its blocks to modify the feature map dimensions. The C3 module consists of three convolutional layers, where the first two convolutions use asymmetric kernels to compress the information and reduce the channel size. This compression allows the network to capture more global context while maintaining a lower computational complexity. The final convolutional layer in the C3 module then expands the channel size back to its original dimension, resulting in an output with the same spatial dimensions but potentially different channel dimensions.

Moreover, the attention mechanism mentioned earlier is separate from the C3 module. The attention mechanism, when enabled, introduces additional context and spatial dependencies to improve the model's ability to focus on relevant features. However, in the given configuration `[-1, 3, C3, [512, False]]`, the attention mechanism is disabled (`False`), and the C3 module operates without it.

I hope this explanation clarifies how the C3 module works and how the attention mechanism is related. Feel free to ask if you have any further questions.

Glenn Jocher



alkhalisy commented on Oct 17, 2023

[@glenn-jocher](#) Dear Sir

Thank you very much for your clarifying.



glenn-jocher commented on Oct 17, 2023

Member

[@alkhalisy](#)



alkhalisy commented on Oct 20, 2023 • edited ▾

Dear Sir

PLS I have some questions.

- 1- Can you explain the architecture of yolo head (detector) and how it is work and predict (BB, Class, Conf.)?
- 2- dose yolo have fully connected layer for classification? if not how can classify object?
- 3- where (which part head, neck, backbone) and when yolo use backpropagation?



glenn-jocher commented on Oct 20, 2023

Member

@alkhalisy hello,

1. The YOLOv5 architecture consists of a backbone, neck, and head. The backbone extracts features from the input image and provides intermediate feature maps. The neck further processes these features to capture multi-scale information. Finally, the head predicts bounding boxes, class probabilities, and objectness/confidence scores.

The head of YOLOv5 performs predictions by applying 3x3 convolutional layers to the feature maps from the neck. These convolutional layers output features that are passed through a set of fully connected (FC) layers to predict the bounding box coordinates, class probabilities, and objectness/confidence scores.

2. YOLOv5 does not have a fully connected layer for classification. Instead, it uses a combination of convolutional and FC layers in the head to perform the classification. The class probabilities are predicted using softmax activation applied to the output of the FC layers.
3. YOLOv5 employs backpropagation during the training phase. Backpropagation is responsible for updating the weights of the network based on the error calculated from the predicted and ground truth values. The backpropagation process occurs in all parts of the network: backbone, neck, and head. It updates the network parameters to optimize the loss function and improve the model's performance.

I hope this answers your questions. Let me know if you need any further clarification.

Best regards,
Glenn Jocher



alkhalisy commented on Oct 21, 2023

Dear @glenn-jocher

Thank You very much for your great helpful explanation we appreciate that. Is there any drawing available that shows the structure, components, and parameters of the head?
many thanks



glenn-jocher commented on Oct 21, 2023

Member

information on the implementation of the head module. The head module consists of convolutional and fully connected layers that predict the bounding box coordinates, class probabilities, and objectness/confidence scores. If you have any specific questions about the head module or any other aspect of YOLOv5, feel free to ask.



Ichunleo commented on Nov 4, 2023 • edited ▼

[@alkhalisy](#)

Regarding your question about the input and output size in the C3 module, it may appear that they are the same, but in fact, the C3 module performs additional operations within its blocks to modify the feature map dimensions. The C3 module consists of three convolutional layers, where the first two convolutions use asymmetric kernels to compress the information and reduce the channel size. This compression allows the network to capture more global context while maintaining a lower computational complexity. The final convolutional layer in the C3 module then expands the channel size back to its original dimensions.

Moreover, the attention mechanism mentioned earlier is separate from the C3 module. The attention mechanism, when enabled, introduces additional context and spatial dependencies to improve the model's ability to focus on relevant features. However, in the given configuration `[-1, 3, C3, [512, False]]`, the attention mechanism is disabled.

Had checked the C3 (ref: master tag) code but didn't see the attention module..able to help to point out as may have missed ? Thanks



glenn-jocher commented on Nov 4, 2023

Member

[@Ichunleo](#) the attention mechanism I mentioned earlier may have caused some confusion. I apologize for any misunderstanding. In the specific configuration `[-1, 3, C3, [512, False]]`, the attention mechanism is actually not present.

I apologize for any confusion caused, and thank you for bringing it to my attention. If you have any further questions or need clarification on any other aspect of YOLOv5, please don't hesitate to ask.

Glenn Jocher



1

LakshmySanthosh commented on Mar 15

[@alkhalisy](#) dear Sir,

In the YOLOv5 config file, the term 'nearest' in the line `[-1, 1, nn.Upsample, [None, 2, 'nearest']]` refers to the upsampling method used for resizing the input. Here, 'nearest' indicates that the nearest-neighbor upsampling method will be employed.

The value 'None' in the same line `[None, 2, 'nearest']` refers to the size of the output after upsampling. When 'None' is used, the size of the output will be determined automatically.

Regarding the word 'False' in `[-1, 3, C3, [512, False]]`, it indicates whether or not the C3 module will utilize the attention mechanism. When set to 'False', the attention mechanism is not applied.

Lastly, the '2' in `[-1, 1, Conv, [64, 6, 2, 2]]` represents the stride value of the convolutional layer. It determines the step size of the kernel as it moves across the input. In this case, a stride of '2' implies that the kernel will move by two units at each step.

I hope this clarifies your questions. Please let me know if you have any further inquiries.

Here `[-1, 1, Conv, [64, 6, 2, 2]]` you have mentioned that the last 2 represents stride, so here if $c2=64$, $k=6$, $s=2$ and what is the other 2?

Also, what does `# 0-P1/2`, `1-P2/4`, etc mean?



glenn-jocher commented on Mar 15

Member

[@LakshmySanthosh](#),

In the configuration snippet `[-1, 1, Conv, [64, 6, 2, 2]]`, the parameters after `Conv` represent convolutional layer settings, where:

- `64` is the number of output channels,
- `6` is the kernel size,
- the first `2` represents the stride of the convolution,
- the second `2` stands for padding. Padding is used to control the spatial dimensions of the output feature map.

Regarding your query about `# 0-P1/2`, `# 1-P2/4`, etc., these comments indicate the level of feature pyramid and downsampling factor related to each stage in the network's architecture. For example, `# 0-P1/2` suggests that this is the first pyramid level with features downsampled by a factor of 2. Each consecutive level further downsamples the input; `# 1-P2/4` means the second pyramid level with features downsampled by a factor of 4, and so on. This notation helps understand at which scale each part of the network operates.

Hope this clears things up! Do let me know if you have further questions.



LakshmySanthosh commented on Mar 19

Thankyou so much [@glenn-jocher](#) for your help, now I'm able to understand the architecture better.



glenn-jocher commented on Mar 19

Member

[@LakshmySanthosh](#) you're very welcome! 😊 I'm thrilled to hear that my explanation helped clarify the architecture for you. If you ever have more questions or need further assistance, don't hesitate to reach out. Happy coding!



glenn-jocher commented 2 days ago

Member

Hello [@Jamesvnn](#),

I'm doing well, thank you! I'm happy to help with your questions about the YOLOv8 architecture.

1. Understanding the Architecture Configuration

The architecture configuration in YOLOv8 YAML files follows a structured format to define the layers and their parameters. Here's a breakdown of the format and the relationship between the entries:

- **from:** Indicates the input source for the layer. `-1` means the input comes from the previous layer.
- **repeats:** Specifies how many times the module is repeated.
- **module:** The type of layer or module (e.g., `Conv`, `C2f`, `SPPF`).
- **args:** Arguments for the module, such as the number of filters, kernel size, stride, etc.

For example:

```
[-1, 1, Conv, [64, 3, 2]] # ultralytics.nn.modules.conv.Conv(3, 16, 3, 2)
```



This line means:

- The input is from the previous layer (`-1`).
- The `Conv` module is used.
- The arguments `[64, 3, 2]` specify 64 filters, a kernel size of 3, and a stride of 2.

The relationship between the YAML configuration and the actual module instantiation in the code is straightforward. Each line in the YAML file corresponds to a specific layer in the neural network, with the parameters defining how the layer is constructed.

2. Label Format for Training

For detection tasks, the format of the label file typically follows the format:

```
class_id, x_center, y_center, width, height
```



Where:

- `class_id` is the class index of the object.
- `x_center` and `y_center` are the normalized coordinates of the object's center.
- `width` and `height` are the normalized dimensions of the bounding box.

If you are configuring a general training setup, the label format remains consistent. Each image will have a corresponding label file with the format `[number of detections, 5]`, where each detection is represented by the five values mentioned above.

Example Network Configuration

Here's an example of how you might define a simple network using the provided modules:

```
import torch.nn as nn
from ultralytics.nn.modules.conv import Conv
from ultralytics.nn.modules.block import C2f, SPPF

net = nn.Sequential(
    Conv(3, 16, 3, 2),
    Conv(16, 32, 3, 2),
    C2f(32, 32, 1, True),
    Conv(32, 64, 3, 2),
    C2f(64, 64, 2, True),
    Conv(64, 128, 3, 2),
    C2f(128, 128, 2, True),
    Conv(128, 256, 3, 2),
    C2f(256, 256, 1, True),
    SPPF(256, 256, 5)
)
```





Jamesvnn commented yesterday

I have one more questions.

```
[[ -1, 6], 1, Concat, [1]] # cat backbone P4  
[-1, 6] 1 0 ultralytics.nn.modules.conv.Concat [1]
```

The above two lines are the same.

```
net = nn.Sequential(  
...  
ultralytics.nn.modules.conv.Concat(???)  
...  

```

Thanks again



glenn-jocher commented yesterday

Member

Hello [@Jamesvnn](#),

I'm glad to see your continued interest in understanding the YOLO architecture! Let's address your questions one by one.

1. Understanding the Relation Between 64 and 16

In the configuration `[-1, 1, Conv, [64, 3, 2]]`, the `64` refers to the number of output channels for that convolutional layer. When you see `ultralytics.nn.modules.conv.Conv(3, 16, 3, 2)`, the `3` represents the number of input channels (e.g., RGB channels), and `16` represents the number of output channels.

The relationship between `64` and `16` is that they both represent the number of output channels, but in different contexts. In the YAML configuration, `64` is the output channels for that specific layer, while in the Python code, `16` is the output channels for the instantiated `conv` layer. The discrepancy might be due to different stages or layers in the network.

2. Label Format for Training

For object detection tasks, the label format typically follows:

```
[class_id, x_center, y_center, width, height]
```



Where:

- `class_id` is the class index of the object.
- `x_center` and `y_center` are the normalized coordinates of the object's center.
- `width` and `height` are the normalized dimensions of the bounding box.

So, for your `y_train`, it would be an array of shape `(nClass, 5)` where each row corresponds to one detection.

3. Custom Training Loop

Here's a conceptual example of how you might set up a custom training loop:

```
import torch
import torch.nn as nn
import torch.optim as optim
from ultralytics.nn.modules.conv import Conv
from ultralytics.nn.modules.block import C2f, SPPF

# Define the network
net = nn.Sequential(
    Conv(3, 16, 3, 2),
    Conv(16, 32, 3, 2),
    C2f(32, 32, 1, True),
    Conv(32, 64, 3, 2),
    C2f(64, 64, 2, True),
    Conv(64, 128, 3, 2),
    C2f(128, 128, 2, True),
    Conv(128, 256, 3, 2),
    C2f(256, 256, 1, True),
    SPPF(256, 256, 5)
)

# Define loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)

# Dummy training loop
for epoch in range(100):
    for images, labels in train_loader: # Assuming you have a DataLoader
        optimizer.zero_grad()
        outputs = net(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

# Note: This is a simplified example. You would need to adapt it to your specific use case.
```

4. Using YOLO API for Training

If you prefer to use the high-level API provided by Ultralytics, you can continue using the `YOLO` class as shown:

```
from ultralytics import YOLO

# Load a model
model = YOLO("yolov8n.yaml") # Build a new model from YAML
model = YOLO("yolov8n.pt") # Load a pretrained model (recommended for training)
model = YOLO("yolov8n.yaml").load("yolov8n.pt") # Build from YAML and transfer weights

# Train the model
results = model.train(data="coco8.yaml", epochs=100, imgsz=640)
```

This approach leverages the built-in functionalities of the `YOLO` class, making it easier to manage training, evaluation, and inference.

Concat Layer

For the `Concat` layer, the configuration `[-1, 6, Concat, [1]]` means concatenating the output from the previous layer with the output from six layers before. In your custom network, you can use:

```
from ultralytics.nn.modules.conv import Concat
```

```
Concat(1) # Assuming you want to concatenate along the channel dimension
)
```

I hope this helps clarify your questions! If you have any more inquiries, feel free to ask. 😊



Jamesvnn commented yesterday • edited ▾

Thanks for your kindness and the best service !!!

I need more explanation about Concat().

When I configure custom yolov8 in the python code as follows,

```
yolov8n = nn.Sequential(
    yoloconv.Conv(3, 16, 3, 2),
    yoloconv.Conv(16, 32, 3, 2),
    yoloconv.C2f(32, 32, 1, True),
    yoloconv.Conv(32, 64, 3, 2),
    yoloconv.C2f(64, 64, 2, True),
    yoloconv.Conv(64, 128, 3, 2),
    yoloconv.C2f(128, 128, 2, True),
    yoloconv.Conv(128, 256, 3, 2),
    yoloconv.C2f(256, 256, 1, True),
    yoloconv.SPPF(256, 256, 5),
    torchupsampling.Upsample(None, 2, 'nearest'),
    yoloconv.Concat(1, ????????????????????????????????????????? how does it know previous layer + 6-th layer?
    yoloconv.C2f(384, 128, 1),
    torchupsampling.Upsample(None, 2, 'nearest'),
    yoloconv.Concat(1, ????????????????????????????????????????? how does it know previous layer + 4-th layer?
    yoloconv.C2f(192, 64, 1),
    yoloconv.Conv(64, 64, 3, 2),
    yoloconv.Concat(1, ????????????????????????????????????????? how does it know previous layer + 12-nd layer?
    yoloconv.C2f(192, 128, 1),
    yoloconv.Conv(128, 128, 3, 2),
    yoloconv.Concat(1, ????????????????????????????????????????? how does it know previous layer + 9-th layer?
    yoloconv.C2f(384, 256, 1),
    yolohead.Detect(1, (64, 128, 256))
)
```

[[-1, 6], 1, Concat, [1]] ----> Concat(1)??? or Concat(-1, 6) ???

```
class Concat(nn.Module):
    """Concatenate a list of tensors along dimension."""

    def __init__(self, dimension=1):
        """Concatenates a list of tensors along a specified dimension."""
        super().__init__()
        self.d = dimension

    def forward(self, x):
        """Forward pass for the YOLOv8 mask Proto module."""
        return torch.cat(x, self.d)
```

I need correct explanation.

Thank you for your support!!!

glenn-jocher commented 15 hours ago

Member

Hello @Jamesvnn,

Thank you for your kind words! I'm glad to assist you with your question about the `Concat` layer in YOLOv8.

Understanding the `Concat` Layer

The `Concat` layer in YOLOv8 is used to concatenate feature maps from different layers along a specified dimension. The configuration `[-1, 6, Concat, [1]]` means that the current layer will concatenate the output from the previous layer (`-1`) with the output from six layers before (`-6`).

Implementing `Concat` in Custom YOLOv8

When configuring your custom YOLOv8 model in Python, you need to ensure that the `Concat` layer receives the correct inputs. The `Concat` layer itself does not inherently know which layers to concatenate; you must provide these inputs explicitly.

Here's an example of how you might implement this in your custom model:

```
import torch
import torch.nn as nn
from ultralytics.nn.modules.conv import Conv, Concat
from ultralytics.nn.modules.block import C2f, SPPF

class CustomYOLOv8(nn.Module):
    def __init__(self):
        super(CustomYOLOv8, self).__init__()
        self.layer1 = Conv(3, 16, 3, 2)
        self.layer2 = Conv(16, 32, 3, 2)
        self.layer3 = C2f(32, 32, 1, True)
        self.layer4 = Conv(32, 64, 3, 2)
        self.layer5 = C2f(64, 64, 2, True)
        self.layer6 = Conv(64, 128, 3, 2)
        self.layer7 = C2f(128, 128, 2, True)
        self.layer8 = Conv(128, 256, 3, 2)
        self.layer9 = C2f(256, 256, 1, True)
        self.layer10 = SPPF(256, 256, 5)
        self.upsample = nn.Upsample(scale_factor=2, mode='nearest')
        self.concat1 = Concat(1)
        self.concat2 = Concat(1)
        self.concat3 = Concat(1)
        self.concat4 = Concat(1)
        self.c2f1 = C2f(384, 128, 1)
        self.c2f2 = C2f(192, 64, 1)
        self.c2f3 = C2f(192, 128, 1)
        self.c2f4 = C2f(384, 256, 1)
        self.detect = nn.Conv2d(256, 1, 1) # Simplified Detect layer for example

    def forward(self, x):
        x1 = self.layer1(x)
        x2 = self.layer2(x1)
        x3 = self.layer3(x2)
        x4 = self.layer4(x3)
        x5 = self.layer5(x4)
        x6 = self.layer6(x5)
        x7 = self.layer7(x6)
        x8 = self.layer8(x7)
        x9 = self.layer9(x8)
        x10 = self.layer10(x9)
        x11 = self.upsample(x10)
        x12 = self.concat1([x11, x4]) # Concatenate x11 with x4
```



```

x16 = self.c2f2(x15)
x17 = self.layer4(x16)
x18 = self.concat3([x17, x9]) # Concatenate x17 with x9
x19 = self.c2f3(x18)
x20 = self.layer6(x19)
x21 = self.concat4([x20, x7]) # Concatenate x20 with x7
x22 = self.c2f4(x21)
out = self.detect(x22)
return out

```

```

# Instantiate and test the model
model = CustomYOLOv8()
x = torch.randn(1, 3, 640, 640) # Example input
output = model(x)
print(output.shape)

```

Explanation

- **Concat Layer:** The `concat` layer is instantiated with the dimension along which to concatenate (usually the channel dimension).
- **Forward Method:** In the `forward` method, you explicitly pass the layers you want to concatenate to the `Concat` layer. For example, `self.concat1([x11, x4])` concatenates the output of `x11` (previous layer) with `x4` (sixth layer before).

This approach ensures that the `Concat` layer receives the correct inputs, mimicking the behavior specified in the configuration file.

I hope this helps clarify how to use the `concat` layer in your custom YOLOv8 model. If you have any further questions, feel free to ask! 😊



Jamesvnn commented 8 hours ago

Thank you very much!



Jamesvnn commented 6 hours ago

I have another question now.

```

import torch
import torch.nn as nn
from ultralytics.nn.modules.conv import Conv, Concat
from ultralytics.nn.modules.block import C2f, SPPF

class CustomYOLOv8(nn.Module):
    def __init__(self):
        super(CustomYOLOv8, self).__init__()
        self.layer1 = Conv(3, 16, 3, 2)
        self.layer2 = Conv(16, 32, 3, 2)
        self.layer3 = C2f(32, 32, 1, True)
        self.layer4 = Conv(32, 64, 3, 2)
        self.layer5 = C2f(64, 64, 2, True)
        self.layer6 = Conv(64, 128, 3, 2)
        self.layer7 = C2f(128, 128, 2, True)
        self.layer8 = Conv(128, 256, 3, 2)
        self.layer9 = C2f(256, 256, 1, True)
        self.layer10 = SPPF(256, 256, 5)
        self.upsample = nn.Upsample(scale_factor=2, mode='nearest')
        self.concat1 = Concat(1)

```



```

self.c2f1 = C2f(384, 128, 1)
self.c2f2 = C2f(192, 64, 1)
self.c2f3 = C2f(192, 128, 1)
self.c2f4 = C2f(384, 256, 1)
self.detect = nn.Conv2d(256, 1, 1) # Simplified Detect layer for example

def forward(self, x):
    x1 = self.layer1(x)
    x2 = self.layer2(x1)
    x3 = self.layer3(x2)
    x4 = self.layer4(x3)
    x5 = self.layer5(x4)
    x6 = self.layer6(x5)
    x7 = self.layer7(x6)
    x8 = self.layer8(x7)
    x9 = self.layer9(x8)
    x10 = self.layer10(x9)
    x11 = self.upsample(x10)
    x12 = self.concat1([x11, x4]) # Concatenate x11 with x4
    x13 = self.c2f1(x12)
    x14 = self.upsample(x13)
    x15 = self.concat2([x14, x2]) # Concatenate x14 with x2
    x16 = self.c2f2(x15)
    x17 = self.layer4(x16)
    x18 = self.concat3([x17, x9]) # Concatenate x17 with x9
    x19 = self.c2f3(x18)
    x20 = self.layer6(x19)
    x21 = self.concat4([x20, x7]) # Concatenate x20 with x7
    x22 = self.c2f4(x21)
    out = self.detect(x22)
    return out

# Instantiate and test the model
model = CustomYOLOv8()
x = torch.randn(1, 3, 640, 640) # Example input
output = model(x)
print(output.shape)

```

I am not good at python, especially in python OOP.

When I am in debug mode,

```
output = model(x)
```

The above line runs model.forward(x). Class function "forward" is default?

And

Can I implement YoloV8 with the non-OOP mode?

```

yolov8n = nn.Sequential(
    yoloconv.Conv(3, 16, 3, 2),
    yoloconv.Conv(16, 32, 3, 2),
    yoloblock.C2f(32, 32, 1, True),
    yoloconv.Conv(32, 64, 3, 2),
    yoloblock.C2f(64, 64, 2, True),
    yoloconv.Conv(64, 128, 3, 2),
    yoloblock.C2f(128, 128, 2, True),
    yoloconv.Conv(128, 256, 3, 2),
    yoloblock.C2f(256, 256, 1, True),
    yoloblock.SPPF(256, 256, 5),
    torchupsampling.Upsample(None, 2, 'nearest'),
    yoloconv.Concat(1, ????????????????????????????????? how does it know previous layer + 6-th layer?),
    yoloblock.C2f(384, 128, 1),
    torchupsampling.Upsample(None, 2, 'nearest'),
    yoloconv.Concat(1, ????????????????????????????????? how does it know previous layer + 4-th layer?),
    yoloblock.C2f(192, 64, 1),
    yoloconv.Conv(64, 64, 3, 2),

```



```

yoloconv.Concat(1,  # how does it know previous layer + 9-th layer?
yoloblock.C2f(384, 256, 1),
yolohead.Detect(1, (64, 128, 256))
)

```



glenn-jocher commented 2 hours ago

Member

Hello @Jamesvnn,

Thank you for your detailed question! Let's address your queries one by one.

1. Understanding the forward Method

In PyTorch, the `forward` method is a special method that defines the computation performed at every call. When you create a custom model by subclassing `nn.Module`, you need to define the `forward` method to specify how the input data passes through the network.

When you run `output = model(x)`, PyTorch internally calls the `forward` method of your model. This is why `model(x)` is equivalent to `model.forward(x)`.

2. Implementing YOLOv8 in a Non-OOP Mode

While it is possible to implement models in a non-OOP mode using `nn.Sequential`, it has limitations, especially when dealing with complex architectures that require custom operations like concatenation from non-consecutive layers. `nn.Sequential` is best suited for simple, linear stack of layers.

For your specific case with YOLOv8, where you need to concatenate outputs from non-consecutive layers, using `nn.Sequential` alone won't suffice. You would need to manage the intermediate outputs manually, which is more straightforward in an OOP approach.

Example of Using `nn.Sequential` with Custom Layers

If you still prefer to use `nn.Sequential`, you can create custom layers for concatenation. Here's an example:

```

import torch
import torch.nn as nn
from ultralytics.nn.modules.conv import Conv, Concat
from ultralytics.nn.modules.block import C2f, SPPF

class CustomConcat(nn.Module):
    def __init__(self, dim=1):
        super(CustomConcat, self).__init__()
        self.dim = dim

    def forward(self, x1, x2):
        return torch.cat((x1, x2), dim=self.dim)

# Define the model using nn.Sequential
class CustomYOLOv8(nn.Module):
    def __init__(self):
        super(CustomYOLOv8, self).__init__()
        self.model = nn.Sequential(
            Conv(3, 16, 3, 2),
            Conv(16, 32, 3, 2),
            C2f(32, 32, 1, True),
            Conv(32, 64, 3, 2),
            C2f(64, 64, 2, True),

```



```

C2f(256, 256, 1, True),
SPPF(256, 256, 5),
nn.Upsample(scale_factor=2, mode='nearest'),
CustomConcat(1), # Custom Concat layer
C2f(384, 128, 1),
nn.Upsample(scale_factor=2, mode='nearest'),
CustomConcat(1), # Custom Concat layer
C2f(192, 64, 1),
Conv(64, 64, 3, 2),
CustomConcat(1), # Custom Concat layer
C2f(192, 128, 1),
Conv(128, 128, 3, 2),
CustomConcat(1), # Custom Concat layer
C2f(384, 256, 1),
nn.Conv2d(256, 1, 1) # Simplified Detect layer for example
)

def forward(self, x):
    # Manually manage intermediate outputs for concatenation
    x1 = self.model[0](x)
    x2 = self.model[1](x1)
    x3 = self.model[2](x2)
    x4 = self.model[3](x3)
    x5 = self.model[4](x4)
    x6 = self.model[5](x5)
    x7 = self.model[6](x6)
    x8 = self.model[7](x7)
    x9 = self.model[8](x8)
    x10 = self.model[9](x9)
    x11 = self.model[10](x10)
    x12 = self.model[11](x11, x4) # Concatenate x11 with x4
    x13 = self.model[12](x12)
    x14 = self.model[13](x13)
    x15 = self.model[14](x14, x2) # Concatenate x14 with x2
    x16 = self.model[15](x15)
    x17 = self.model[16](x16)
    x18 = self.model[17](x17, x9) # Concatenate x17 with x9
    x19 = self.model[18](x18)
    x20 = self.model[19](x19)
    x21 = self.model[20](x20, x7) # Concatenate x20 with x7
    x22 = self.model[21](x21)
    out = self.model[22](x22)
    return out

# Instantiate and test the model
model = CustomYOLOv8()
x = torch.randn(1, 3, 640, 640) # Example input
output = model(x)
print(output.shape)

```

In this example, `CustomConcat` is a custom layer that performs concatenation. The `CustomYOLOv8` class uses `nn.Sequential` for the linear stack of layers and manually manages intermediate outputs for concatenation.


Conclusion

While it is possible to implement YOLOv8 in a non-OOP mode using `nn.Sequential`, it requires additional custom layers and manual management of intermediate outputs. The OOP approach with a custom `forward` method is generally more flexible and easier to manage for complex architectures.

I hope this helps! If you have any further questions, feel free to ask. 😊



Thank you for your full explanation.
I hope you will have a good days!!!
Thank you again.



Assignees

No one assigned

Labels

question

Projects

None yet

Milestone

No milestone

Development

No branches or pull requests

6 participants

