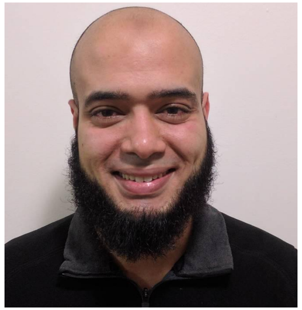
Machine Learning
Fine-tuning

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

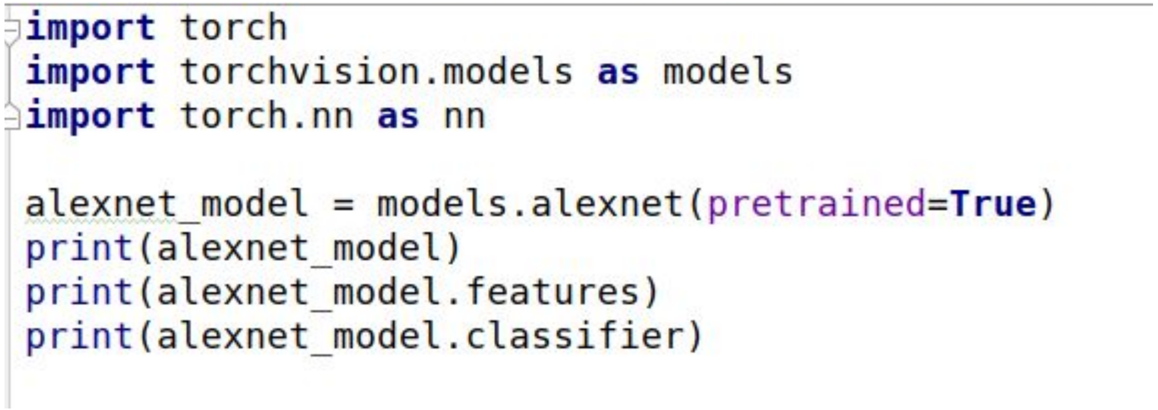


Exploring pretrained models

- There are many models that are trained on popular tasks
- Some of them, like image classifiers are already in your framework
- Let’s explore PyTorch Alexnet

AlexNet

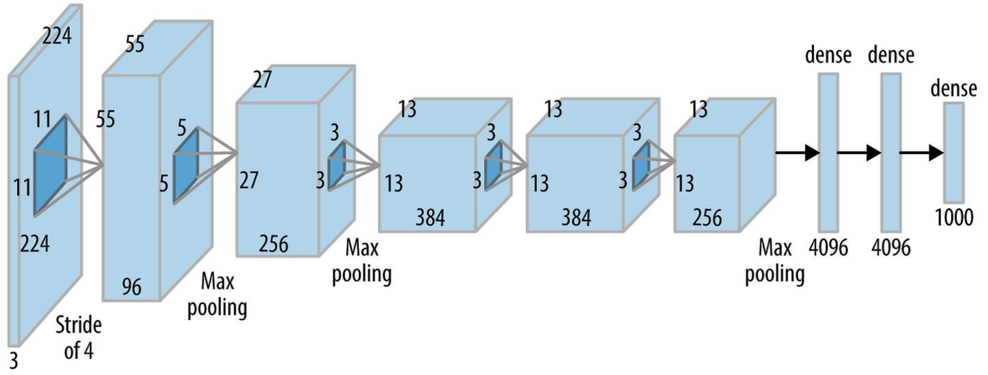
- You can load the pretrained model from pytorch
- The code is arranged nicely as
  - Features attribute: it has all the CNN layers and ends with last heatmap
  - Classifier attribute: the last fully connected layers
  - Note: different networks might be arranged differently



```python
import torch
import torchvision.models as models
import torch.nn as nn

alexnet_model = models.alexnet(pretrained=True)
print(alexnet_model)
print(alexnet_model.features)
print(alexnet_model.classifier)
```

Fine-tuning

- Assume a pre-trained exists for a CNN train with 1 million images of wide diversity of scenes to classify 1000 classes
- Except the last 2 layers, all these CNN layers has a lot of generic features
- What If I want to classify 5 classes for images about a person action? A specific animal species?
- We can simply drop the last 2 layers only and add 2 layers ending with classes
- Now, fine-tune this network with way less images!

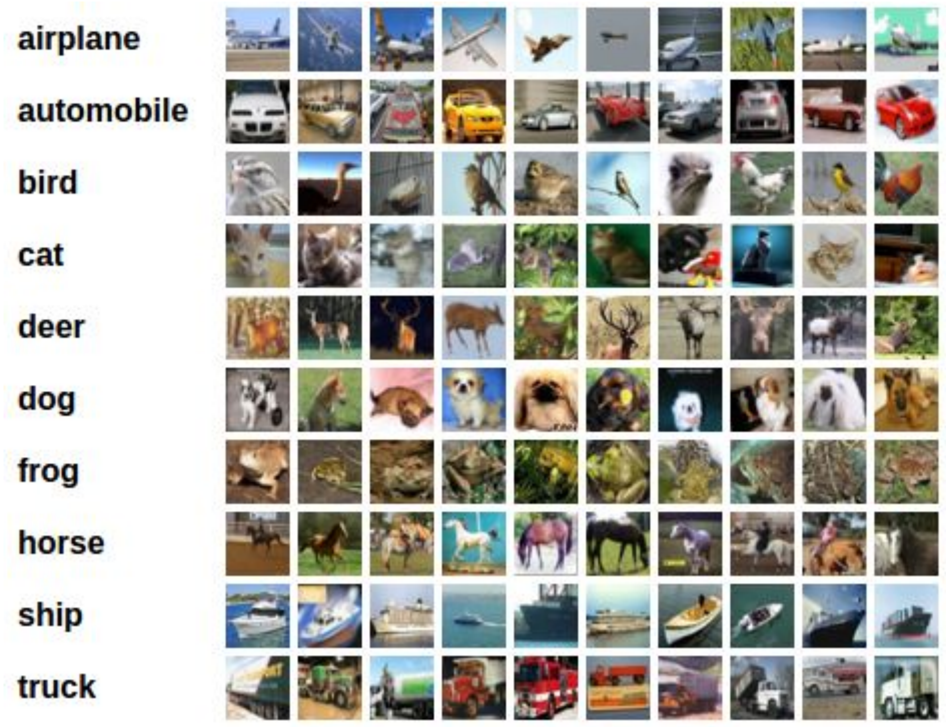


Fine-tuning

- Fine-tuning a Convolutional Neural Network (CNN) involves taking a pre-trained network and adapting its parameters to a new, but **similar** problem
    - Choose a Pre-trained Model (e.g. imagenet, resnet50, etc)
    - Prepare Your Dataset (which can be a small dataset)
    - Modify the Network Architecture: Replace the final fully connected layer(s) with new layers suited to your specific problem.
        - Initialize the weights of new layers
    - Where to freeze?
        - You may want to freeze the early layers and never update as they are already good low-level to medium level features. Only fine-tune the high-level features
    - Hyperparameter Tuning: Explore relevant hyperparameters. You may start from a **lower** LR and train for a **few** epochs (already weights went in a long journey)

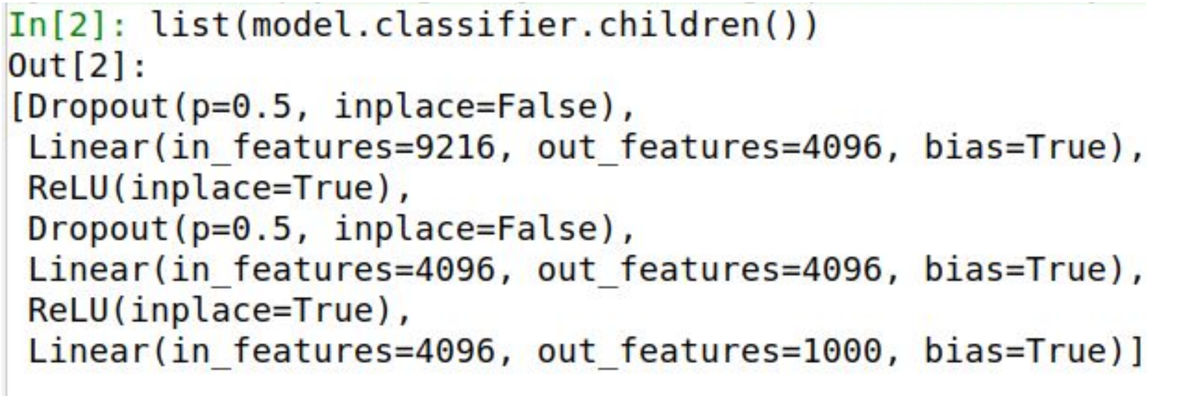
Fine-tuning AlexNet - Simple

- Assume we wanted to fine-tune alexnet for CIFAR10 dataset
    - This dataset is small dimension images with 10 different classes



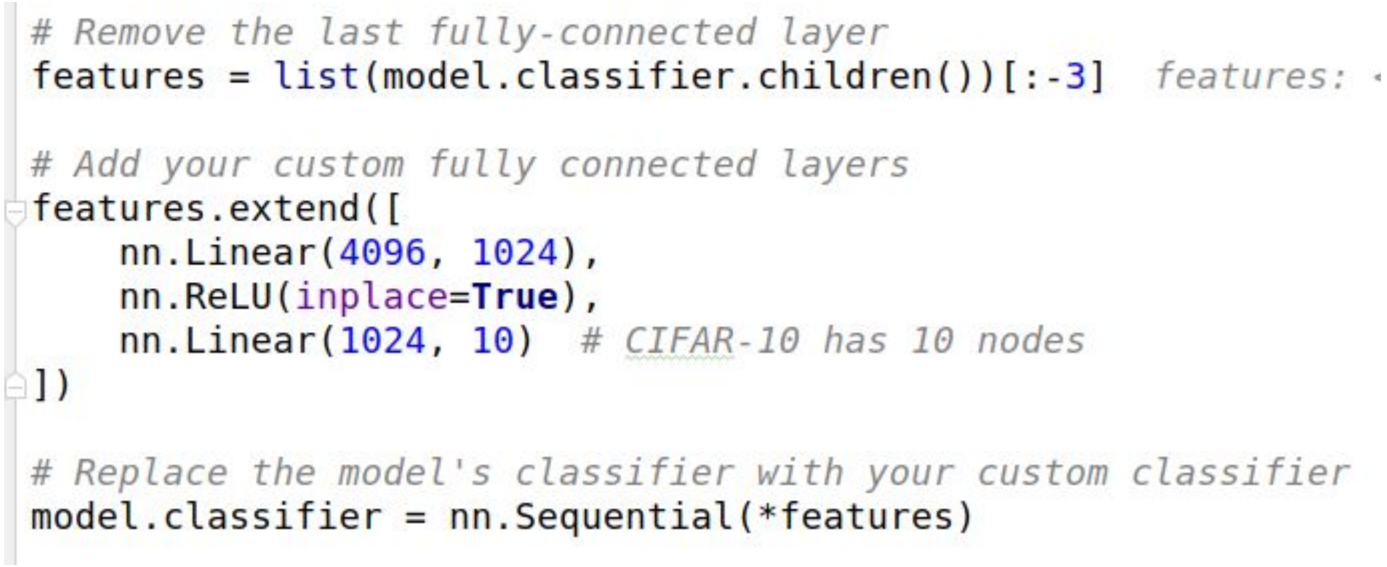
Fine-tuning AlexNet - Simple

- Assume we just want to replace one of the fully connected layers
- Let’s explore what we have
- It seems from model.classifier.children(), the classifier actually has 3 FC
- Let’s say we want to make last layers as
  - 9216 ⇒ 1096
  - 4096 ⇒ 1024
  - 1024 ⇒ 10
- Then we need to remove last 2 FCs
  - We remove 3 layers
    - L/R/L



```
In[2]: list(model.classifier.children())
Out[2]:
[Dropout(p=0.5, inplace=False),
 Linear(in_features=9216, out_features=4096, bias=True),
 ReLU(inplace=True),
 Dropout(p=0.5, inplace=False),
 Linear(in_features=4096, out_features=4096, bias=True),
 ReLU(inplace=True),
 Linear(in_features=4096, out_features=1000, bias=True)]
```

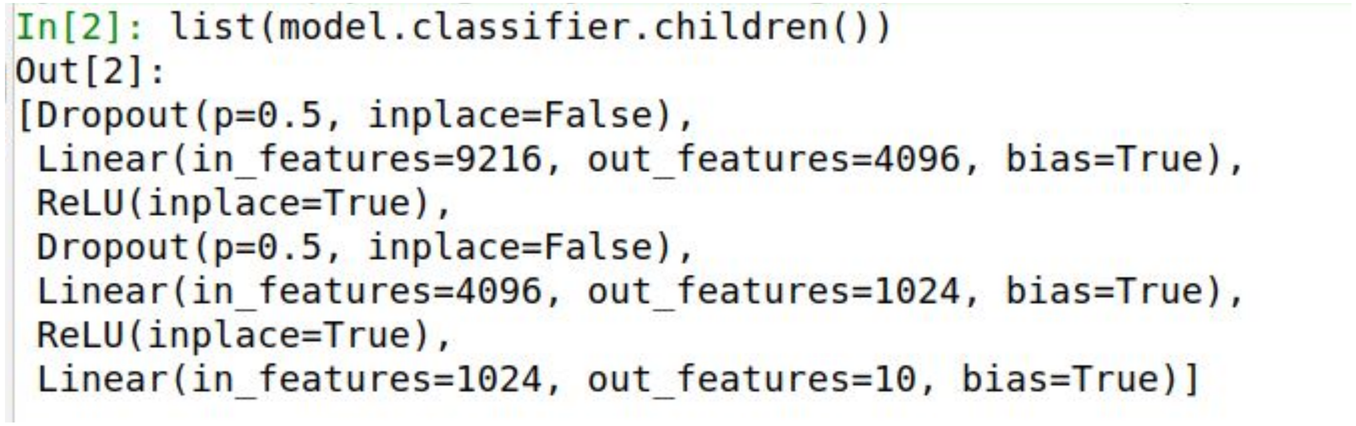
Fine-tuning AlexNet - Simple



```python
# Remove the last fully-connected layer
features = list(model.classifier.children())[:-3]  features:

# Add your custom fully connected layers
features.extend([
    nn.Linear(4096, 1024),
    nn.ReLU(inplace=True),
    nn.Linear(1024, 10)  # CIFAR-10 has 10 nodes
])

# Replace the model's classifier with your custom classifier
model.classifier = nn.Sequential(*features)
```

Fine-tuning AlexNet - Simple

- Now, your model is ready for normal train/test cycle
- Refer to the code for full normal details
- Tip: we refer to such critical pretrained models as **backbones**



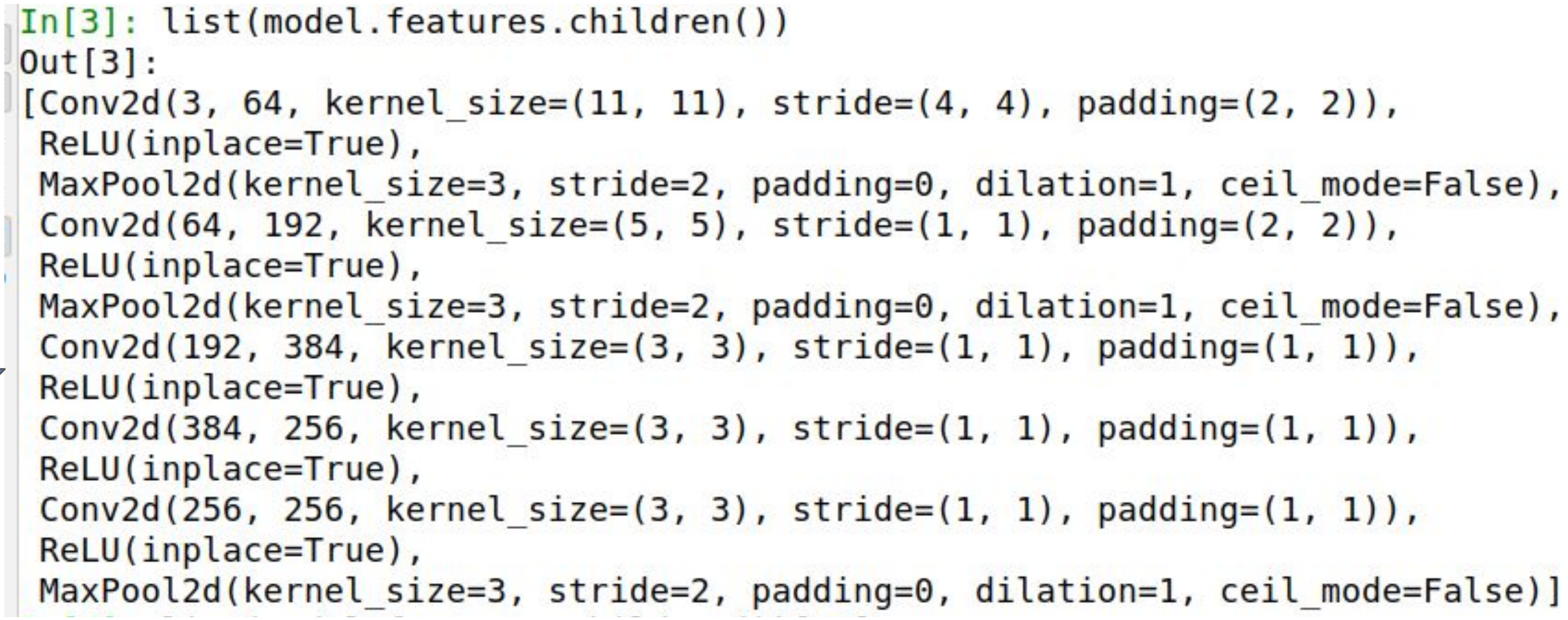
```
In[2]: list(model.classifier.children())
Out[2]:
[Dropout(p=0.5, inplace=False),
 Linear(in_features=9216, out_features=4096, bias=True),
 ReLU(inplace=True),
 Dropout(p=0.5, inplace=False),
 Linear(in_features=4096, out_features=1024, bias=True),
 ReLU(inplace=True),
 Linear(in_features=1024, out_features=10, bias=True)]
```

Fine-tuning AlexNet - Complex

- What if we want
    - Use only a part of the pretrained CNN
        - Also freeze these layers
    - Extend the CNN with our new CNN layers
    - Then build a relevant FC layers

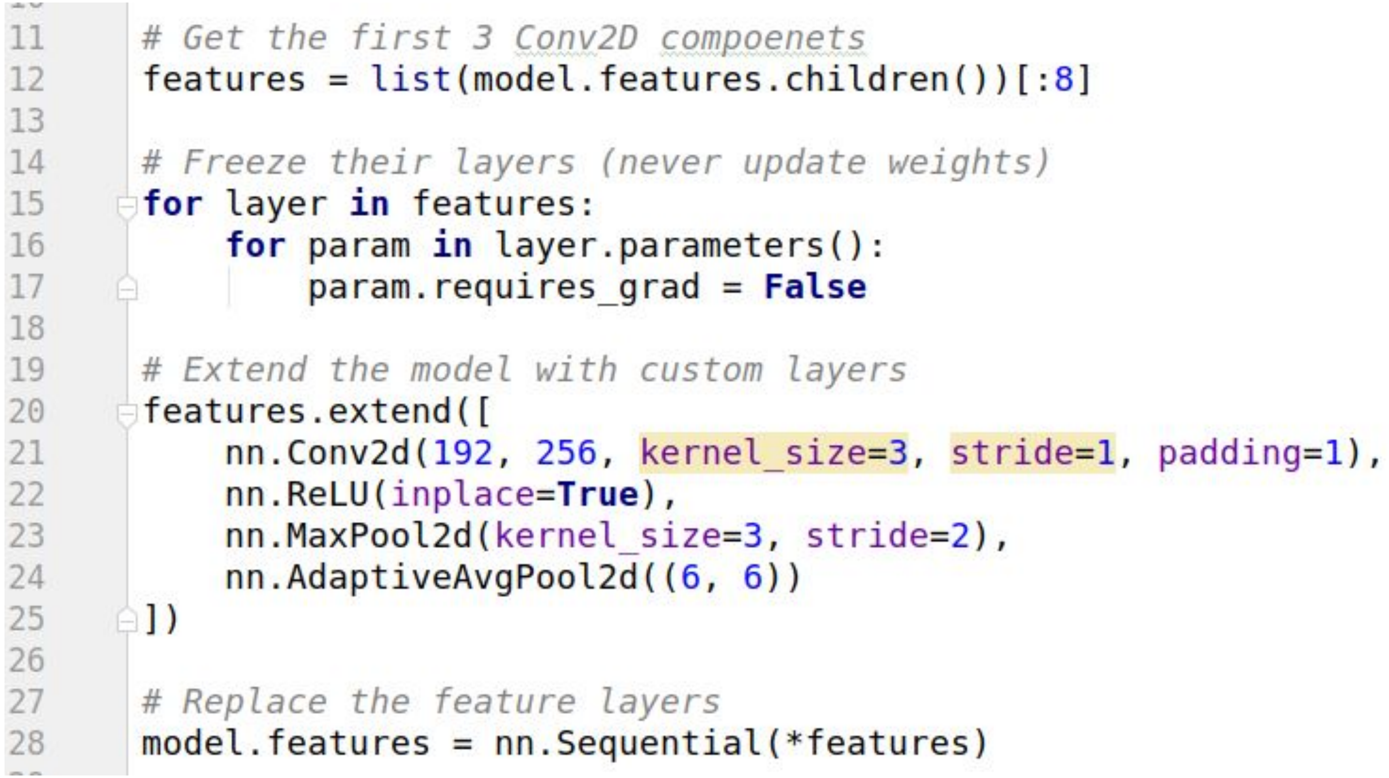
Fine-tuning AlexNet - Complex

- What if we want to keep only the first 3 Conv2D filters?
  - Then we need to keep the first 8 layers



```
In[3]: list(model.features.children())
Out[3]:
[Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2)),
 ReLU(inplace=True),
 MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False),
 Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2)),
 ReLU(inplace=True),
 MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False),
 Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
 ReLU(inplace=True),
 Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
 ReLU(inplace=True),
 Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
 ReLU(inplace=True),
 MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)]
```

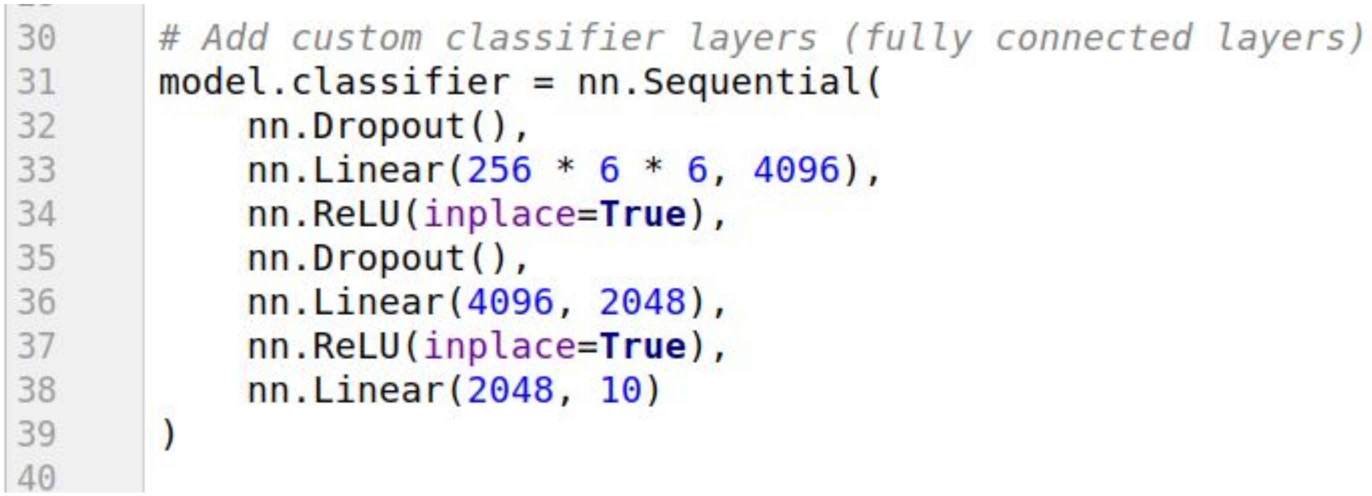
Fine-tuning AlexNet - Complex



```python
# Get the first 3 Conv2D compoenets
features = list(model.features.children())[:8]

# Freeze their layers (never update weights)
for layer in features:
    for param in layer.parameters():
        param.requires_grad = False

# Extend the model with custom layers
features.extend([
    nn.Conv2d(192, 256, kernel_size=3, stride=1, padding=1),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.AdaptiveAvgPool2d((6, 6))
])

# Replace the feature layers
model.features = nn.Sequential(*features)
```

Fine-tuning AlexNet - Complex

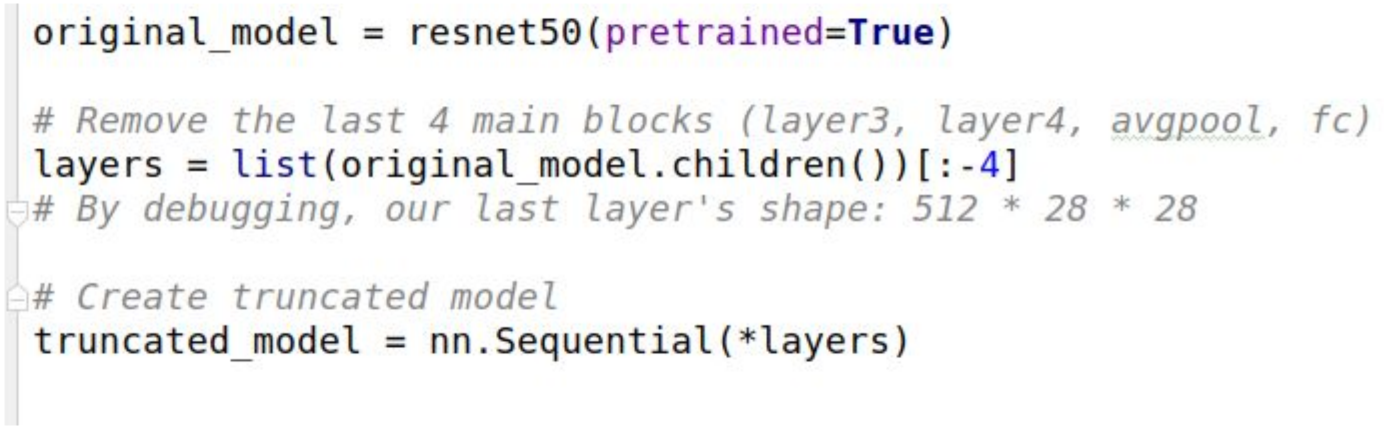
- Last layer ended with 256 filters and 6x6 grid
- Let's update our FC layers
- Now, train normally



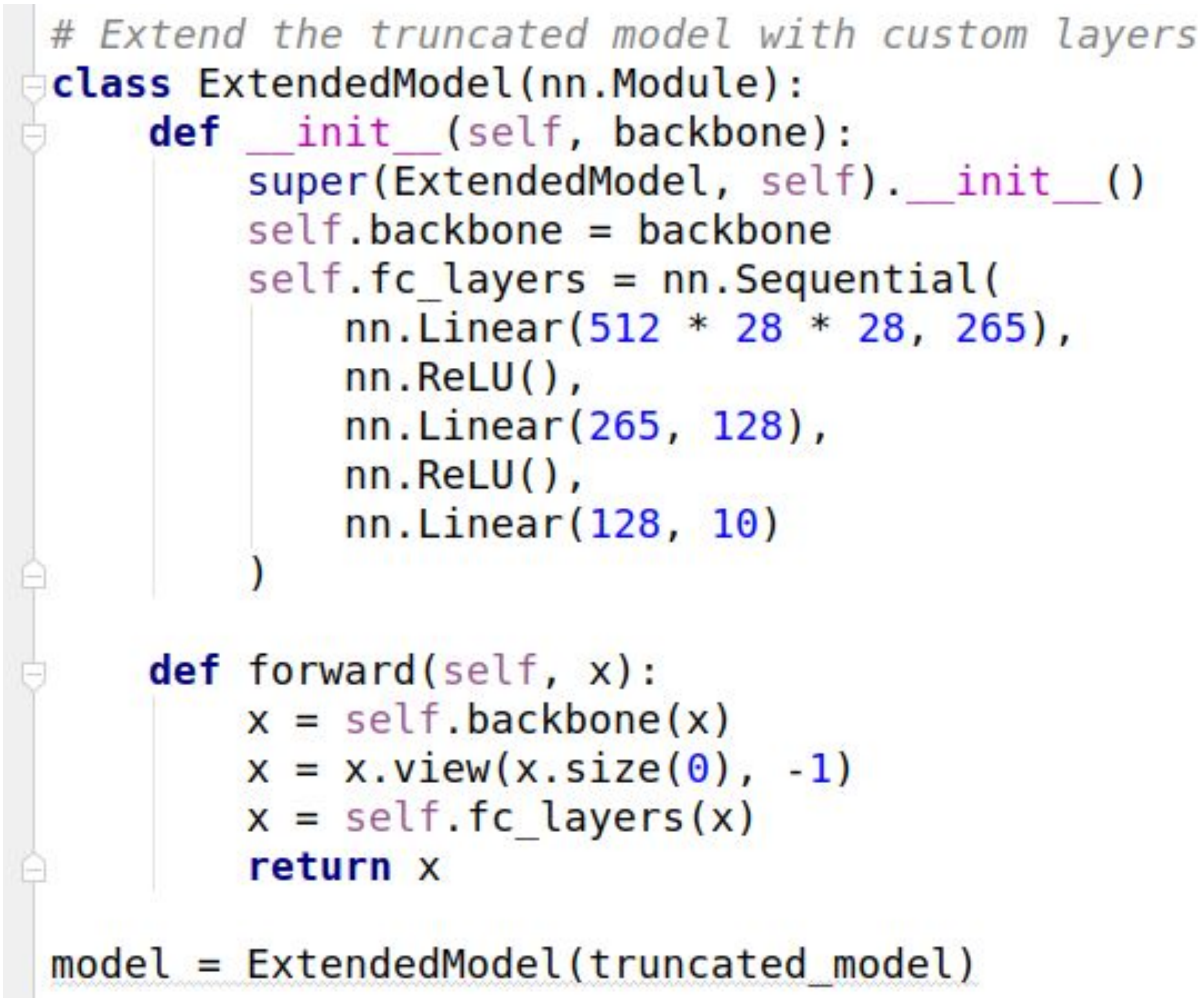
```python
# Add custom classifier layers (fully connected layers)
model.classifier = nn.Sequential(
    nn.Dropout(),
    nn.Linear(256 * 6 * 6, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(),
    nn.Linear(4096, 2048),
    nn.ReLU(inplace=True),
    nn.Linear(2048, 10)
)
```

Fine-tuning **ResNet** - Complex

- Sometimes, you want your backbone part to be the early stage of another model
  - This is just matter of coding
- Prepare the pretrained model and pass it to your new model
- Let’s say we want to remove the last 4 main blocks of resnet



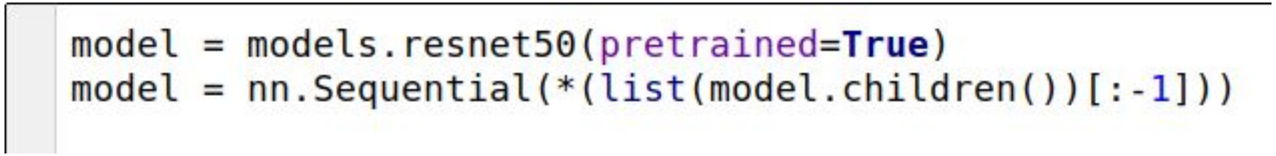
```
original_model = resnet50(pretrained=True)

# Remove the last 4 main blocks (layer3, layer4, avgpool, fc)
layers = list(original_model.children())[:-4]
# By debugging, our last layer's shape: 512 * 28 * 28

# Create truncated model
truncated_model = nn.Sequential(*layers)
```



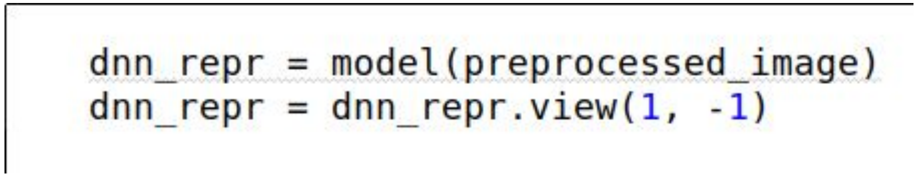
```python
# Extend the truncated model with custom layers
class ExtendedModel(nn.Module):
    def __init__(self, backbone):
        super(ExtendedModel, self).__init__()
        self.backbone = backbone
        self.fc_layers = nn.Sequential(
            nn.Linear(512 * 28 * 28, 265),
            nn.ReLU(),
            nn.Linear(265, 128),
            nn.ReLU(),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        x = self.backbone(x)
        x = x.view(x.size(0), -1)
        x = self.fc_layers(x)
        return x

model = ExtendedModel(truncated_model)
```

Features Extraction

- Another possible scenario (pre trained / fine tuned networks) is features extraction
- This typically happens from the layer **before the last layer**
- For example, the last layer of resnet 50 has 1000 features for categories
  - The one before it has 2048 features
  - Then we can pass an image to the model and extract 2048 equal representation



```
model = models.resnet50(pretrained=True)
model = nn.Sequential(*(list(model.children())[:-1]))
```



```
dnn_repr = model(preprocessed_image)
dnn_repr = dnn_repr.view(1, -1)
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."