Overall Approach

- PyTorch framework was adopted as I am most familiar with it.

- Transfer learning by applying pre-trained convolutional neural networks (CNNs).

- Model fitting is done on Google Colab, train and test data are loaded to zip files, the zip files are uploaded to Google drive which is mounted via the python code and then unzipped into the Google's Linux server before model training is done.

- Webscraping (read the last section) was also done on the Google Colab's Linux server as well via an additional python notebook to add to the train folders that are already unzipped.

- The submitted codes are all changed such that the code can be run on the computer itself for CAX to test out (no more drive mounting etc.)

Trying Different Networks

The following models from PyTorch (table below) were tried due to my familiarity with them in previous projects and their top-1 and top-5 error rates in ImageNet.

Network	Top-1 error	Top-5 error
VGG-11 with batch normalization	29.62	10.19
VGG-13 with batch normalization	28.45	9.63
VGG-16 with batch normalization	26.63	8.50
VGG-19 with batch normalization	25.76	8.15
ResNet-18	30.24	10.92
ResNet-34	26.70	8.58
ResNet-50	23.85	7.13
ResNet-101	22.63	6.44
ResNet-152	21.69	5.94
Densenet-121	25.35	7.83
Densenet-169	24.00	7.00
Densenet-201	22.80	6.43
Densenet-161	22.35	6.20
Inception v3	22.55	6.44
GoogleNet	30.22	10.47
ResNeXt-50-32x4d	22.38	6.30
ResNeXt-101-32x8d	20.69	5.47

In this preliminary run, only fully-connected layers and last 1 to 3 layers of the CNN itself were trained, the rest were frozen.

10% of training data was used as validation set to see which models performed the best.

Not surprisingly, ResNeXt-101-32x8d, ResNeXt-50-32x4d, ResNet-101, ResNet-152, Densenet-161 and Inception v3 had good results with about 70% accuracy.

Learning rate and optimizer

'Adam' optimizer was used as 'Stochastic Gradient Descent' was training way too slowly without significant increase in accuracy.

A good learning rate (LR) was chosen by trial and error, LR of 0.0001 proved to be a good starting rate.

When validation cross-entropy loss starts to increase, LR is lowered by a factor of 10. Training is stopped when this does not increase cross-entropy loss further.

Freezing layers in networks

ResNeXt-101-32x8d, ResNeXt-50-32x4d, ResNet-101, ResNet-152, Densenet-161 and Inception v3

The top layers of the network are frozen as they should be picking up very generic features such as edges that apply to all kinds of image data.

There's no way to tell how many bottom layers should be frozen other than trial and error. For all the networks listed above, the bottom layers were unfrozen layer by layer to see which one performed the best.

Layers that are trained in this dataset have weights initialized from the pre-trained networks and not at random.

Notable models found (above 82% accuracy):

ResNeXt-101-32x8d (**Best**) (18^{th} 'sub-layer' in layer 3 onwards were unfreezed, the rest frozen, meaning all 'sub-layers' in layer 4 are trained also, and of course the fully-connected layer) (refer to Appendix 1 to see what this means) (about 86% accuracy)

ResNeXt-101-32x8d (17^{th} sub-layer in layer 3 onwards were unfreezed)

ResNeXt-101-32x8d (20^{th} sub-layer in layer 3 onwards were unfreezed)

ResNet-152 (24^{th} sub-layer in layer 3 onwards were unfreezed)

ResNet-152 (27^{th} sub-layer in layer 3 onwards were unfreezed)

Other notes (What didn't work in improving accuracy):

- It is found that a fully connected network with only one layer performs the best. Adding more layers or Dropout, ReLU did not help, the convolutional layers are still the most important.

- Tried training the fully connected network first while freezing all convolutional layers before unfreezing bottom convolutional networks to train, however it did not help in increasing accuracy.

- Tried different learning rates using different optimizers (in the code they are optimizer 1, 2 and 3). For example, a higher learning rate was set for the fully connected layer compared to convolutional layers, but this did not help as well.

Augmentation

Using the *torchvision.transforms* library, image augmentation was performed to increase more train data to try to boost the model accuracy.

The usual augmentation techniques (brightness, saturation, contrast, hue, flip-horizontal, rotation, random crop) were all considered.

Hue: Generating random images of different hue to augment the train data should negatively affect the accuracy and this was confirmed. This is not surprising and confirms that colour is an important feature (e.g. there are no blue charmanders etc.) so this method of augmentation fails.

Flip Horizontal: Randomly flipping images horizontally to augment the train data also negatively affected the accuracy. This is probably because words like “Pokemon”, “Han Solo” etc. in the train images themselves become flipped when they of course shouldn’t be.

Random Rotation: This method of augmentation also negatively affected the accuracy no matter how many degrees of rotation was set. Looking at test set, we can see that almost all the images are upright. Rotating the train images was hence not necessary and even decreased the accuracy.

Random Crop: This method of augmentation also was undesirable as the main feature in the image may not be cropped during preprocessing (e.g. for some image the Charmander in the Charmander t-shirt may not be cropped, making this training image useless).

Saturation and Contrast: This was harder to tell intuitively by observing the test images, but this method of augmentation did not increase the accuracy.

Brightness: As seen in test images, the products are taken under different lighting conditions and have different shades of brightness (although colour is the same, some images are clearly darker then others). Randomly generating images of different brightness (capped at a factor of 0.05) increased the accuracy to above 88%. However, the accuracy of the model fluctuates due to the randomness in generating images with different brightness and perhaps to a smaller extend, the randomness in choosing validation images.

The final method to augment train data is the following in the code (only brightness).
transforms.ColorJitter(brightness=.05, saturation=0, contrast=0)

Adding more training data via webscraping

Using the label names themselves as google search term

Included in the code is a '*scrap image from google.ipynb*' python notebook. All the different labels of the images were searched on google and 100 images were downloaded for each label and placed into the appropriate labelled folder.

Problems encountered and solution:

- There were a lot of corrupted ones which had to be identified via code and deleted (e.g. do not end in .jpg, cannot be opened etc.).
- When I googled 'ben myself, it was a disaster as many people with the name 'ben' came up. As such, the train data for 'ben' was not augmented at all.
- The label 'Goku_1' was not a good search term and code has to be manually modified to search for 'Goku' before manually renaming the folder to 'Goku_1'.
- As my computer is slow, the whole model fitting was done on Google Colab and there was no manually opening each downloaded image to verify that they are exactly suitable for fitting into model.

> *Disclaimer:*
> *As the training images that were obtained via webscraping were saved directly onto Google Colab's Linux runtime session to save time, the* ***original additional images to train the data are automatically deleted*** *after 12 hours. Great care has been taken to reproduce these additional images for training by running the webscraping code to download the images again and zipping them up for CAX to use.*
>
> ***These final training images are in the zipped folder named 'train_expanded_character' within the uploaded code folder.***

The validation set are still taken from the original train data to ensure that the additional train data helps in identify the product images from this challenge and not in identifying the new train data itself.

This adding of training images greatly improved the validation accuracy.

This, together with the previous architecture and augmentation, i.e. ResNeXt-101-32x8d (18^{th} 'sub-layer' in layer 3 onwards unfreezed), random adjust brightness capped at 0.05, and also carefully lowering the learning rate when cross-entropy loss of validation set increases achieved an accuracy of about 90% on the validation set, the validation accuracy still fluctuates due to the random adjust brightness augmentation. Upon uploading the submission file, it was found to have accuracy of over 91% on the public test set itself.)

This is the selected model.

Using the label names + ' product' as google search term to try to include only product

Also tried to use label names + ' product' as google search term for webscraping. (e.g. charmander product, angrybirds product) to add to the training set.

This method fails badly! This is because the variety of images that came up was way too much and was unrelated to the test set.

Final Model

Taking into account everything, and also noting what was mentioned in the previous section. The final model was determined.

ResNeXt-101-32x8d (unfreeze 18th sublayer from 3rd layer onwards)

Random adjust brightness capped at 0.05 for image augmentation

Carefully lowering the learning rate when cross-entropy loss of validation set

Webscraping for more images while removing flawed images

Before the final submission, only 1 dummy image is placed in the validation directory (as this code requires validation directory with at least 1 image to run, feel free to randomly copy and paste any amount of images from the training set to this directory to use as validation images when running it). This is to ensure that the maximum number of images are used for training.

As we cannot observe the validation accuracy and the validation loss now to set the learning rate, the learning rate and how it decreases is set similar to how the model was trained at the final run with the validation set (LR of 0.00005 for 4 epoch, change to 0.00001 for 4 epoch, then 0.000001 for 4 epoch).

Other models and architectures that were worth noting were mentioned earlier and are worth exploring further if time permits.

Appendix 1

resnext101_32x8d architecture
The layers highlighted in yellow are the ones whose weights are trained.

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32,
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32,
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32,
bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (layer2): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), groups=32,
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32,
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32,
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (3): Bottleneck(
      (conv1): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), groups=32,
bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(512, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (layer3): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(512, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (3): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (4): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (5): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (6): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (7): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (8): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (9): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (10): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (11): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (12): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
```

```
    (13): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (14): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (15): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (16): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (17): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (18): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (19): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (20): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (21): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (22): Bottleneck(
      (conv1): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(1024, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (layer4): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(2048, 2048, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(2048, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
      (downsample): Sequential(
        (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): Bottleneck(
      (conv1): Conv2d(2048, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(2048, 2048, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(2048, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
    (2): Bottleneck(
      (conv1): Conv2d(2048, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(2048, 2048, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=32, bias=False)
      (bn2): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(2048, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=2048, out_features=1000, bias=True)
)
```