Deep Computer Vision with Convolutional Neural Networks

Chapter 14 – Deep Computer Vision Using Convolutional Neural Networks

❗ **This will be very slow, unless you are using a GPU for the later code**

❗ **If you do not, then you should run this notebook in Colab, using a GPU runtime**

File name convention: For group 42 and memebers Richard Stallman and Linus Torvalds it would be:
"07_Stallman_Torvalds.pdf".

Submission via blackboard (UA).

Feel free to answer free text questions in text cells using markdown and possibly $\LaTeX$ if you want to.

You don't have to understand every line of code here and it is not intended for you to try to understand every line of code.
Big blocks of code are usually meant to just be clicked through.

Setup

In [1]:

```python
# Python ≥3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn ≥0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset
import torchvision
from tensorflow import keras
```

```python
import numpy as np
import os

np.random.seed(42)
torch.manual_seed(42)
torch.cuda.manual_seed_all

%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

In [2]:

```python
def plot_image(image):
    plt.imshow(image, cmap="gray", interpolation="nearest")
    plt.axis("off")

def plot_color_image(image):
    plt.imshow(image, interpolation="nearest")
    plt.axis("off")
```

Let's **import some data** to see how convolutional filters work. One is a scenic image of china and the other is an image of a flower. The first thing we should do is **normalize the pixels**.

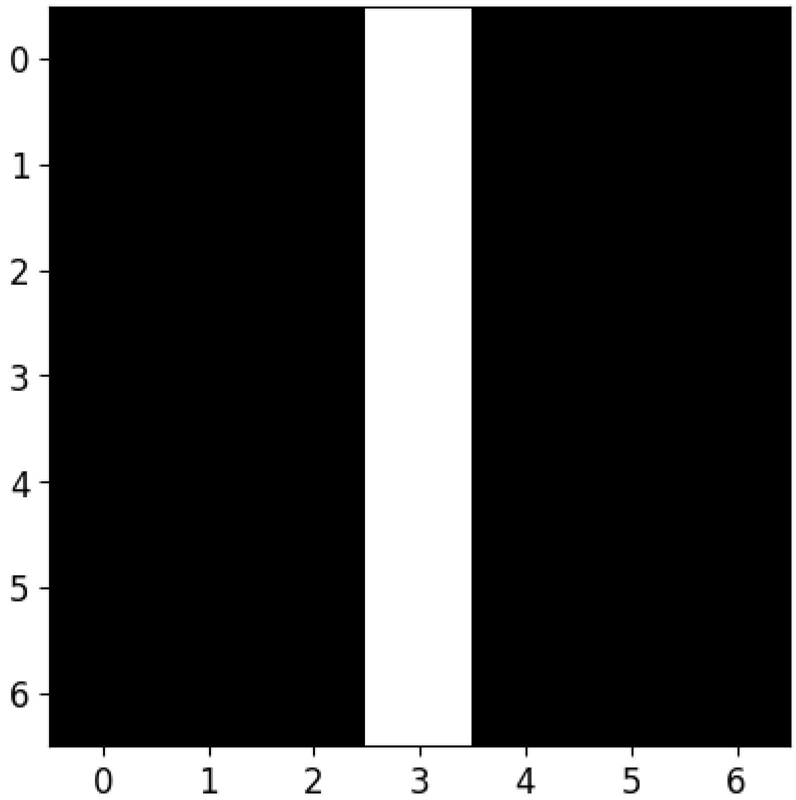
In [3]:

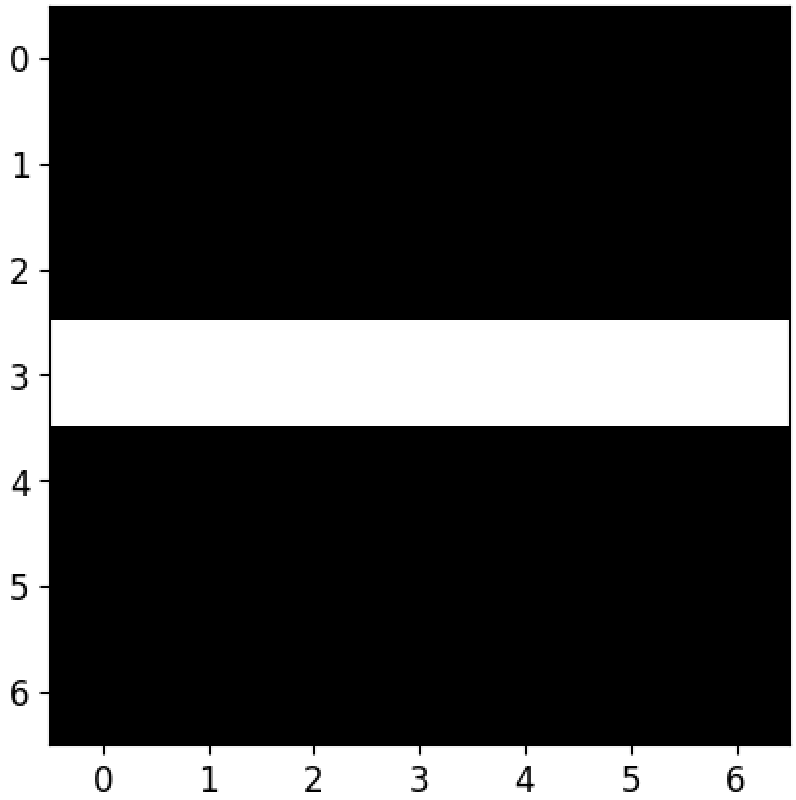
```python
import numpy as np
from sklearn.datasets import load_sample_image

# Load sample images
china = load_sample_image("china.jpg") / 255
flower = load_sample_image("flower.jpg") / 255
images = np.array([china, flower])
batch_size, height, width, channels = images.shape
print(batch_size, height, width, channels)

images = torch.from_numpy(images).permute(0, 3, 1, 2)

plt.imshow(china)
plt.axis("off") # Not shown in the book
plt.show()
plt.imshow(flower)
plt.axis("off") # Not shown in the book
plt.show()
```

```
2 427 640 3
```

Next let's create some filters. Here we're creating image filters that have the shape **7x7x3x2**. So that's a 7x7 grid which will pass over three color channels and we have two filters for each of those dimensions. Here we want two filters as the final dimension to demonstrate creating **vertical** and **horizontal** filters.

In [4]:
```
# Create 2 filters
filters = torch.zeros((2, channels, 7, 7), dtype=torch.float32)
filters[0, :, :, 3] = 1  # vertical line
filters[1, :, 3, :] = 1  # horizontal line
plt.imshow(torch.moveaxis(filters[0,:,:,:], 0, 2))
plt.show()
plt.imshow(torch.moveaxis(filters[1,:,:,:], 0, 2))
plt.show()
```





Notice that when we look at the shape of the outputs of our filters it now has **final dimension 2 instead of 3**. What we've done here is **reduced our 3** red, green, blue **(RGB) channels to two filter channels** that have picked out the vertical and horizontal lines in all three color channels then added them up.

In [5]:
```
print(images.shape)
print(filters.shape)
```

```
torch.Size([2, 3, 427, 640])
torch.Size([2, 3, 7, 7])
```

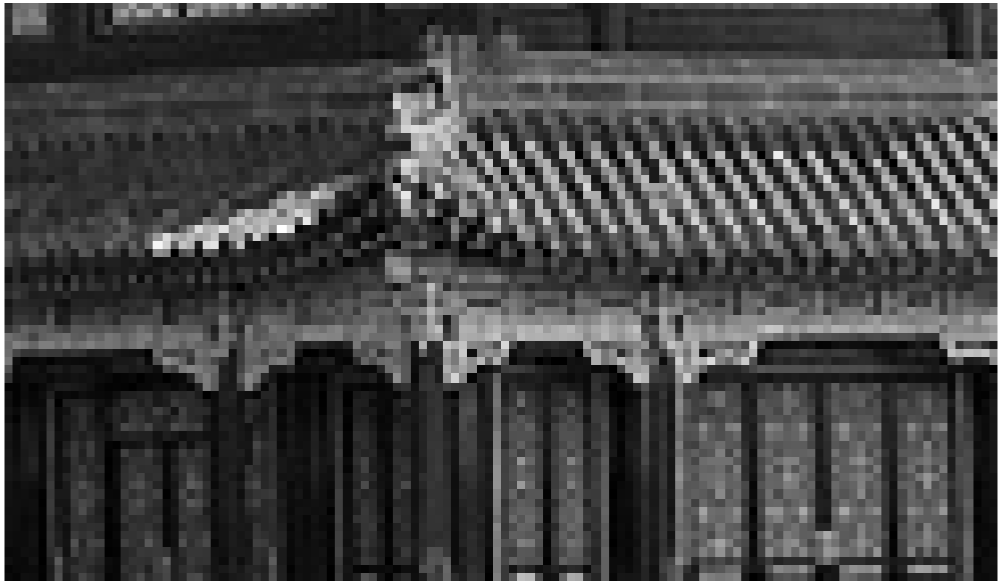
In [6]:
```
outputs = nn.functional.conv2d(images.to(torch.float32), weight=filters, bias=None, stride=1, padding='same')
print(outputs.shape)

plt.imshow(outputs[0, 1, :, :], cmap='gray') # plot 1st image's 2nd feature map
plt.axis("off") # Not shown in the book
plt.show()
```

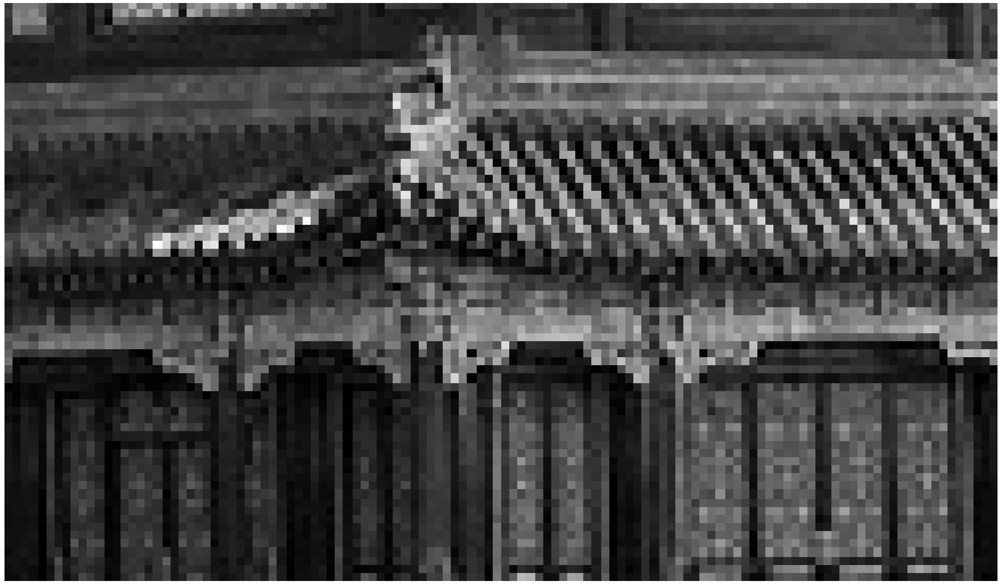
```
torch.Size([2, 2, 427, 640])
```

In [7]:
```
def crop(images):
    try:
      return images[:, 150:220, 130:250]
    except:
      return images[150:220, 130:250]
```

Let's look at our different color channels to see what an unfiltered image looks like.

In [8]:
```
plot_image(crop(images[0, 0, :, :]))
plt.show()
plot_image(crop(images[0, 1, :, :]))
plt.show()
plot_image(crop(images[0, 2, :, :]))
plt.show()
```

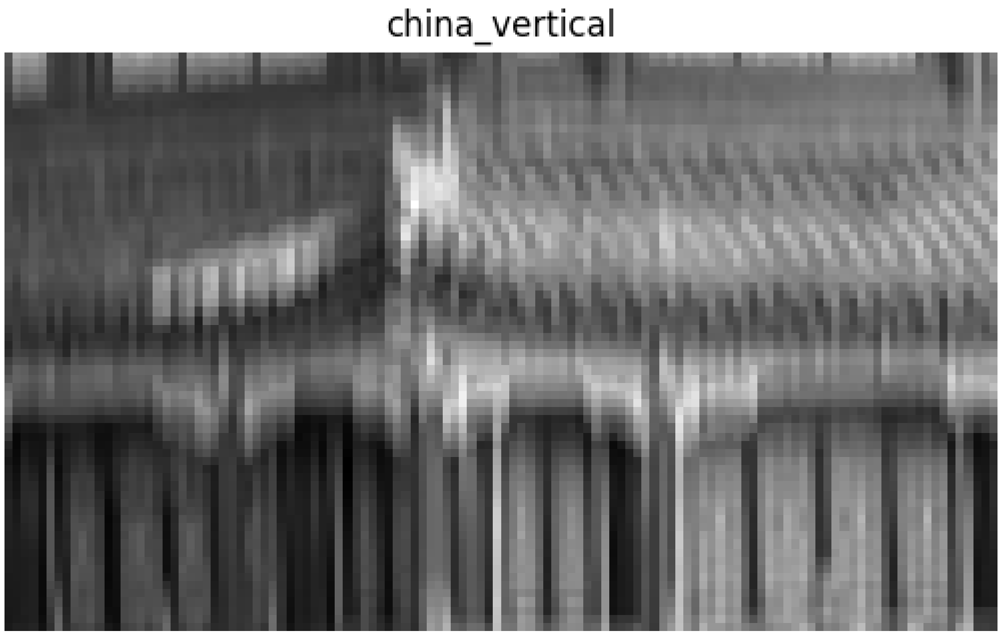


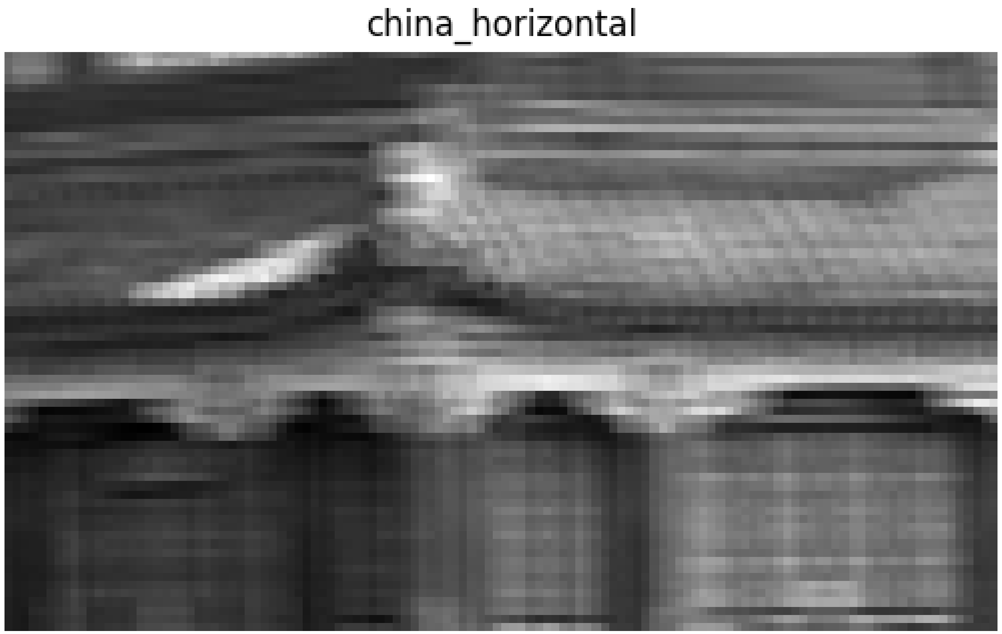


Basics: Filters and Pooling

Task 1: Filters

In [9]:
```
for feature_map_index, filename in enumerate(["china_vertical", "china_horizontal"]):
    plot_image(crop(outputs[0, feature_map_index, :, :]))
    plt.title(filename)
    plt.show()
```





Task 1 a): Describe how the filters work and what their purpose in a CNN is.

↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ your code goes below

Task 1 a) answer: The code iterates over two filters (corresponding to vertical and horizontal edge detection):

- china_vertical and china_horizontal suggest that these filters extract vertical and horizontal features from the image.
- outputs[0, feature_map_index, :, :] represents the feature maps extracted by these filters.
- plot_image(crop(...)) visualizes the results of the filters, showing how they highlight specific aspects of the input.

↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑ your code goes above

Convolutional Layer in Pytorch

To create a 2D convolutional layer use `nn.Conv2d` (https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html).

Task 1 b)
Create a convolutional layer with 32 filters and `kernel_size` `(3,3)`. Apply it to `images[0:1]` and explain the shape of the output. **Do not explicitly pass any filters** this time. Instead, use the default random initialization for pytorch convolutional layers. Run it a couple of times and notice that you get a different image each time.

You can plot the resulting images if you want (for example `plot_image(new_images[0,0,:,:])` for the first filter).

↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ your code goes below

In [10]:
```
conv2d_layer = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=(3,3))
im = images[0:1].to(torch.float32)
new_images = conv2d_layer(im).detach().numpy()
print(new_images.shape)
plot_image(new_images[0,0,:,:])
```

```
(1, 32, 425, 638)
```

Task 1b) shape explanation:

- 1: Batch size remains the same (we took only one image).
- 32: Number of output channels (filters applied).
- The original image size was (1, 3, 427, 640). After applying a (3,3) convolution with no padding, the height and width decrease by 2 pixels.

↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑ your code goes above

Cropping the Images

In [11]:
```
cropped_images = np.array([crop(image) for image in images], dtype=np.float32)
```

In [12]:
```
plot_image(cropped_images[0, 0, :, :])
plt.show()
```

Task 2: Max Pooling Layer in Pytorch

Pooling layers are used to **shrink the input image** in order to reduce the computational load, the memory usage, and the number of parameters.

Task 2 a)

- Create a max pool layer of kernel_size=(2,2) (https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html)
- apply the max pool layer to the `cropped_images` assigning the result to the variable `output`
- **Note:** Be sure to convert the input `cropped_images` to tensor and to the right datatype beforehand using `torch.from_numpy(cropped_images).to(torch.float32)` and use `.detach().numpy()` afterward to convert your model output to numpy for visualization.

↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ your code goes below

In [13]:
```
maxpool_layer = nn.MaxPool2d(kernel_size=(2,2))
im = torch.from_numpy(cropped_images).to(torch.float32)
```

```
output = maxpool_layer(im).detach().numpy()
```

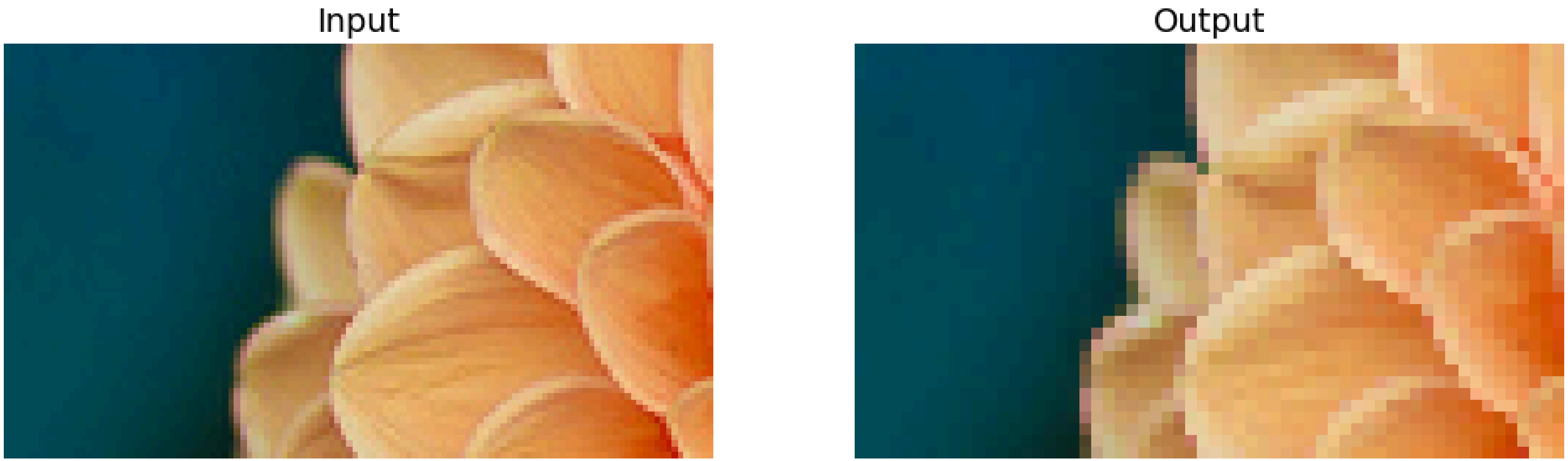
In [14]:
```
output = maxpool_layer(torch.from_numpy(cropped_images).to(torch.float32)).detach().numpy()
```

↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑ your code goes above

In [15]:
```
fig = plt.figure(figsize=(12, 8))
gs = mpl.gridspec.GridSpec(nrows=1, ncols=2, width_ratios=[1, 1])

ax1 = fig.add_subplot(gs[0, 0])
ax1.set_title("Input", fontsize=14)
ax1.imshow(np.moveaxis(cropped_images[0], 0, 2))  # plot the 1st image
ax1.axis("off")
ax2 = fig.add_subplot(gs[0, 1])
ax2.set_title("Output", fontsize=14)
ax2.imshow(np.moveaxis(output[0], 0, 2))  # plot the output for the 1st image
ax2.axis("off")
plt.show()
```

In [16]:
```
fig = plt.figure(figsize=(12, 8))
gs = mpl.gridspec.GridSpec(nrows=1, ncols=2, width_ratios=[1, 1])

ax1 = fig.add_subplot(gs[0, 0])
ax1.set_title("Input", fontsize=14)
ax1.imshow(np.moveaxis(cropped_images[1], 0, 2))
ax1.axis("off")
ax2 = fig.add_subplot(gs[0, 1])
ax2.set_title("Output", fontsize=14)
```

```python
ax2.imshow(np.moveaxis(output[1], 0, 2))
ax2.axis("off")
plt.show()
```



Task 2 b)

Describe the effect of the max pooling layer. What are its benefits for a Neural Network? What are the downsides?

↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ your code goes below

Task 2b) answer

- Max pooling is a downsampling operation commonly used in CNN. It reduces the spatial dimensions of feature maps by selecting the maximum value in a predefined window and moving the window with a stride. This process retains the most prominent features while reducing computational complexity.
- Downsides of Max Pooling
    1. Loss of spatial information: Since it retains only the maximum value in each pooling region, some finer details of the image are lost, which might be important for tasks like segmentation or precise localization.
    2. Aggressive downsampling can harm small features: If small but important details (e.g., fine textures or small objects in an image) get removed during pooling, the network may struggle to recognize them.
    3. Fixed pooling strategy: Unlike attention mechanisms that adaptively focus on important regions, max pooling blindly selects the strongest activation without considering context.

↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑ your code goes above

Tackling Fashion MNIST With a CNN

In [17]:
```
(X_train_full, y_train_full), (X_test, y_test) = keras.datasets.fashion_mnist.load_data()

X_train, X_valid = X_train_full[:-5000], X_train_full[-5000:]
y_train, y_valid = y_train_full[:-5000], y_train_full[-5000:]

# normalization
X_mean = X_train.mean(axis=0, keepdims=True)
X_std = X_train.std(axis=0, keepdims=True) + 1e-7
X_train = (X_train - X_mean) / X_std
X_valid = (X_valid - X_mean) / X_std
X_test = (X_test - X_mean) / X_std


#Notice that pytorch convolutional layers expect the 1-axis to be the channels
#dimension whereas generally linear layers will act on the last axis.

X_train = X_train[:, np.newaxis, ...]
X_valid = X_valid[:, np.newaxis, ...]
X_test = X_test[:, np.newaxis, ...]
```

In [18]:
```
class ClassificationDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.from_numpy(X.copy()).float()
        self.y = torch.from_numpy(y.copy()).long()
    def __len__(self):
        return len(self.X)
    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

train_data = ClassificationDataset(X_train, y_train)
valid_data = ClassificationDataset(X_valid, y_valid)
test_data = ClassificationDataset(X_test, y_test)

batch_size = 256

train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
valid_loader = DataLoader(valid_data, batch_size=batch_size, shuffle=False)
```

In [19]:
```
from functools import partial

DefaultConv2d = partial(nn.Conv2d,
                        kernel_size=3, padding='same')

model = nn.Sequential(
    DefaultConv2d(in_channels=1, out_channels=64, kernel_size=7),
    nn.MaxPool2d(kernel_size=(2,2)),
    DefaultConv2d(in_channels=64, out_channels=128),
    DefaultConv2d(in_channels=128, out_channels=128),
    nn.MaxPool2d(kernel_size=(2,2)),
    DefaultConv2d(in_channels=128, out_channels=256),
    DefaultConv2d(in_channels=256, out_channels=256),
    nn.MaxPool2d(kernel_size=(2,2)),
    nn.Flatten(),
    nn.Linear(in_features=64*36, out_features=128),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(in_features=128, out_features=64),
    nn.ReLU(),
    nn.Dropout(0.5),
    nn.Linear(in_features=64, out_features=10),
)
```

Visualization of Model Structure

This is not necessary, but maybe interesting.

In [ ]:
```
!pip install torchviz
```

In [20]:
```
from torchviz import make_dot
x = torch.randn(1,1,28,28)
y = model(x)

#make_dot generates an image of your model and .render() outputs it to a file.
#Click the folder icon on the left side of colab and you should see a file
#call model_image.png that shows the model
make_dot(y.mean(), params=dict(model.named_parameters())).render("model_image", format="png")
```

Out[20]:
```
'model_image.png'
```

Training and Testing Loops

Note that compared to previous training loops this one has now introduced the concept of a **"device"**. Here that is included so that you can use **GPU** for the larger models in this notebook like ResNet. The correct way to use a device is to **pass the model and data to the same device *before* doing operations**. GPU's are a type of processor that are especially good at matrix-based operations such as those used in graphics as well as machine learning.

In [21]:
```
def train_and_validate(train_loader, val_loader, model, optimizer, criterion, num_epochs, metric=None, scheduler=None, device='
    history = {
        'epoch': [],
        'train_loss': [],
        'train_metric': [],
        'val_loss': [],
        'val_metric': [],
        'learning_rate': []
    }  # Initialize a dictionary to store epoch-wise results

    model.to(device)  # Move the model to the specified device

    with torch.no_grad():
        proper_dtype = torch.int64
        X,y = next(iter(train_loader))
        X = X.to(device)
        y = y.to(device)
        try:
            loss = criterion(model(X), y.to(proper_dtype))
        except:
            try:
                proper_dtype = torch.float32
                loss = criterion(model(X), y.to(proper_dtype))
            except:
                print("No valid data-type could be found")

    for epoch in range(num_epochs):
        model.train()  # Set the model to training mode
        epoch_loss = 0.0  # Initialize the epoch loss and metric values
        epoch_metric = 0.0

        # Training loop
        for X, y in train_loader:
            X = X.to(device)
            y = y.to(device)
            y = y.to(proper_dtype)
            optimizer.zero_grad()  # Clear existing gradients
```

```
        outputs = model(X)  # Make predictions
        loss = criterion(outputs, y)  # Compute the loss
        loss.backward()  # Compute gradients
        optimizer.step()  # Update model parameters

        epoch_loss += loss.item()

        # THESE LINES HAVE BEEN UPDATED TO ACCOUNT FOR DEFAULT ARGUMENTS
        if metric is not None:
            epoch_metric += metric(outputs, y)
        else:
            epoch_metric += 0.0

    # Average training loss and metric
    epoch_loss /= len(train_loader)
    epoch_metric /= len(train_loader)

    # Validation loop
    model.eval()  # Set the model to evaluation mode
    with torch.no_grad():  # Disable gradient calculation
        val_loss = 0.0
        val_metric = 0.0
        for X_val, y_val in val_loader:
            X_val = X_val.to(device)
            y_val = y_val.to(device)
            y_val = y_val.to(proper_dtype)
            outputs_val = model(X_val)  # Make predictions
            val_loss += criterion(outputs_val, y_val).item()  # Compute loss
            if metric is not None:
                val_metric += metric(outputs_val, y_val)
            else:
                val_metric += 0.0

        val_loss /= len(val_loader)
        val_metric /= len(val_loader)

    # Append epoch results to history
    history['epoch'].append(epoch)
    history['train_loss'].append(epoch_loss)
    history['train_metric'].append(epoch_metric)
    history['val_loss'].append(val_loss)
    history['val_metric'].append(val_metric)
    history['learning_rate'].append(optimizer.param_groups[0]['lr'])

    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {epoch_loss:.4f}, '
          f'Train Metric: {epoch_metric:.4f}, Val Loss: {val_loss:.4f}, '
```

```
            f'Val Metric: {val_metric:.4f}')

        if scheduler is not None:
            scheduler.step()

    return history, model
```

In [22]:
```
def test_model(model, data_loader, criterion, metric=None, device='cpu'):
    model.to(device)  # Move the model to the specified device

    model.eval()  # Set the model to evaluation mode

    total_loss = 0.0  # Initialize the total loss and metric values
    total_metric = 0.0

    with torch.no_grad():
        proper_dtype = torch.int64
        X,y = next(iter(data_loader))
        X = X.to(device)
        y = y.to(device)
        try:
            loss = criterion(model(X), y.to(proper_dtype))
        except:
            try:
                proper_dtype = torch.float32
                loss = criterion(model(X), y.to(proper_dtype))
            except:
                print("No valid data-type could be found")


    with torch.no_grad():  # Disable gradient tracking
        for batch in data_loader:
            X, y = batch
            X = X.to(device)
            y = y.to(device)
            y = y.to(proper_dtype)
            # Pass the data to the model and make predictions
            outputs = model(X)

            # Compute the loss
            loss = criterion(outputs, y)

            # Add the loss and metric for the batch to the total values
            total_loss += loss.item()

            if metric is not None:
```

```
            total_metric += metric(outputs, y)
        else:
            total_metric += 0.0

    # Average loss and metric for the entire dataset
    avg_loss = total_loss / len(data_loader)
    avg_metric = total_metric / len(data_loader)

    print(f'Test Loss: {avg_loss:.4f}, Test Metric: {avg_metric:.4f}')

    return avg_loss, avg_metric
```

In [23]:
```
def accuracy_metric(pred, target):
    if len(pred.shape) == 1:
        accuracy = torch.sum(torch.eq(pred > 0.5, target)).item() / len(pred)
    else:
        pred = pred.argmax(dim=1)
        accuracy = torch.sum(pred == target).item() / len(pred)
    return accuracy
```

GPU Time:

If you haven't enabled GPU in your colab notebook, now is the time to do so.
Only one group member should be working with GPU at a time as you will each have a limit on how often and for how long colab will allow you to use gpu.

In [24]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [25]:
```
print(device) # should be cuda if Colab is connected to gpu
```

```
cuda
```

Task 3:

- Train the model using `nn.CrossEntropyLoss` as `loss`, `torch.optim.NAdam` as optimizer with `lr=2e-4`, and `"accuracy_metric"` for `metric`
- fit the model for 20 epochs using `train_loader` and 'valid_loader'
- `evaluate` the model on `test_loader`
- predict the first 20 instances of `X_test` and compare them to `y_test`
    - **Note:** Remember to convert `X_test` to tensor first using `torch.from_numpy()`

↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ your code goes below

In [ ]:

```
import torch
from torch import nn
from torch.utils.data import DataLoader

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.NAdam(model.parameters(), lr=2e-4)

history, model = train_and_validate(train_loader, valid_loader, model, optimizer, criterion, num_epochs=20, metric=accuracy_met

test_loss, test_metric = test_model(model, test_loader, criterion, metric=accuracy_metric, device=device)

# Predictions
X_test_tensor = torch.from_numpy(X_test[:20]).float().to(device)
with torch.no_grad():
    model.eval()
    predictions = model(X_test_tensor)
    predicted_labels = predictions.argmax(dim=1)

print("Predicted labels:", predicted_labels)
print("True labels:", y_test[:20])
```

```
Epoch [1/20], Train Loss: 1.0721, Train Metric: 0.6284, Val Loss: 0.5088, Val Metric: 0.8193
Epoch [2/20], Train Loss: 0.6345, Train Metric: 0.7824, Val Loss: 0.4124, Val Metric: 0.8469
Epoch [3/20], Train Loss: 0.5208, Train Metric: 0.8246, Val Loss: 0.3633, Val Metric: 0.8691
Epoch [4/20], Train Loss: 0.4641, Train Metric: 0.8455, Val Loss: 0.3523, Val Metric: 0.8734
Epoch [5/20], Train Loss: 0.4232, Train Metric: 0.8569, Val Loss: 0.3207, Val Metric: 0.8826
Epoch [6/20], Train Loss: 0.3939, Train Metric: 0.8683, Val Loss: 0.3123, Val Metric: 0.8847
Epoch [7/20], Train Loss: 0.3661, Train Metric: 0.8767, Val Loss: 0.2959, Val Metric: 0.8949
Epoch [8/20], Train Loss: 0.3487, Train Metric: 0.8826, Val Loss: 0.2870, Val Metric: 0.8981
Epoch [9/20], Train Loss: 0.3288, Train Metric: 0.8912, Val Loss: 0.2781, Val Metric: 0.8962
Epoch [10/20], Train Loss: 0.3105, Train Metric: 0.8941, Val Loss: 0.2766, Val Metric: 0.9032
Epoch [11/20], Train Loss: 0.2926, Train Metric: 0.9005, Val Loss: 0.2855, Val Metric: 0.8998
Epoch [12/20], Train Loss: 0.2818, Train Metric: 0.9040, Val Loss: 0.2603, Val Metric: 0.9049
Epoch [13/20], Train Loss: 0.2655, Train Metric: 0.9105, Val Loss: 0.2679, Val Metric: 0.9014
Epoch [14/20], Train Loss: 0.2573, Train Metric: 0.9125, Val Loss: 0.2529, Val Metric: 0.9104
Epoch [15/20], Train Loss: 0.2444, Train Metric: 0.9168, Val Loss: 0.2489, Val Metric: 0.9132
Epoch [16/20], Train Loss: 0.2364, Train Metric: 0.9197, Val Loss: 0.2622, Val Metric: 0.9082
Epoch [17/20], Train Loss: 0.2254, Train Metric: 0.9230, Val Loss: 0.2594, Val Metric: 0.9100
Epoch [18/20], Train Loss: 0.2155, Train Metric: 0.9251, Val Loss: 0.2561, Val Metric: 0.9113
Epoch [19/20], Train Loss: 0.2052, Train Metric: 0.9286, Val Loss: 0.2520, Val Metric: 0.9145
Epoch [20/20], Train Loss: 0.1980, Train Metric: 0.9315, Val Loss: 0.2619, Val Metric: 0.9095
Test Loss: 0.2796, Test Metric: 0.9064
Predicted labels: tensor([9, 2, 1, 1, 6, 1, 4, 6, 5, 7, 4, 5, 7, 3, 4, 1, 2, 2, 8, 0],
       device='cuda:0')
True labels: [9 2 1 1 6 1 4 6 5 7 4 5 7 3 4 1 2 4 8 0]
```

↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑ your code goes above

Task 4: ResNet-34

ResNet is built on the idea of using **residual connections** between early layers and later layers that the early layers are **not directly attached** to. In effect, this results in **more total connections** in the neural network without actually having to add additional weights to the model.

Imagine you had a function that required **100 coefficients with 10 operations**. Instead, you realize that many of the coefficients are **related** to one another so you decide to **recycle terms** and instead **include more recursive operations**. Now your function has **20 coefficients** but you're doing **30 operations**.

This is the idea of ResNet. We **reuse the same weights** multiple times but connecting them to **different layers** each time. This can lead to models that are on the order of 5+ times smaller without meaningfully reducing performance.

Pytorch Implementation of Resnet

The following is pytorch's highly-optimized implementation of resnet. However, we'll need to modify it slightly to get the channel and target dimensions to match our problem.

In [ ]:
```
from functools import partial
from typing import Any, Callable, List, Optional, Type, Union

from torch import Tensor

from torchvision.transforms._presets import ImageClassification
from torchvision.utils import _log_api_usage_once
from torchvision.models._api import register_model, Weights, WeightsEnum
from torchvision.models._meta import _IMAGENET_CATEGORIES
from torchvision.models._utils import _ovewrite_named_param, handle_legacy_interface
```

In [ ]:
```
def conv3x3(in_planes: int, out_planes: int, stride: int = 1, groups: int = 1, dilation: int = 1) -> nn.Conv2d:
    """3x3 convolution with padding"""
    return nn.Conv2d(
        in_planes,
        out_planes,
        kernel_size=3,
        stride=stride,
        padding=dilation,
        groups=groups,
        bias=False,
        dilation=dilation,
    )


def conv1x1(in_planes: int, out_planes: int, stride: int = 1) -> nn.Conv2d:
    """1x1 convolution"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride, bias=False)


class BasicBlock(nn.Module):
    expansion: int = 1

    def __init__(
        self,
        inplanes: int,
        planes: int,
```

```python
        stride: int = 1,
        downsample: Optional[nn.Module] = None,
        groups: int = 1,
        base_width: int = 64,
        dilation: int = 1,
        norm_layer: Optional[Callable[..., nn.Module]] = None,
    ) -> None:
        super().__init__()
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        if groups != 1 or base_width != 64:
            raise ValueError("BasicBlock only supports groups=1 and base_width=64")
        if dilation > 1:
            raise NotImplementedError("Dilation > 1 not supported in BasicBlock")
        # Both self.conv1 and self.downsample layers downsample the input when stride != 1
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = norm_layer(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = norm_layer(planes)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x: Tensor) -> Tensor:
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        if self.downsample is not None:
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)

        return out


class Bottleneck(nn.Module):
    # Bottleneck in torchvision places the stride for downsampling at 3x3 convolution(self.conv2)
    # while original implementation places the stride at the first 1x1 convolution(self.conv1)
    # according to "Deep residual learning for image recognition" https://arxiv.org/abs/1512.03385.
```

```
    # This variant is also known as ResNet V1.5 and improves accuracy according to
    # https://ngc.nvidia.com/catalog/model-scripts/nvidia:resnet_50_v1_5_for_pytorch.

    expansion: int = 4

    def __init__(
        self,
        inplanes: int,
        planes: int,
        stride: int = 1,
        downsample: Optional[nn.Module] = None,
        groups: int = 1,
        base_width: int = 64,
        dilation: int = 1,
        norm_layer: Optional[Callable[..., nn.Module]] = None,
    ) -> None:
        super().__init__()
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        width = int(planes * (base_width / 64.0)) * groups
        # Both self.conv2 and self.downsample layers downsample the input when stride != 1
        self.conv1 = conv1x1(inplanes, width)
        self.bn1 = norm_layer(width)
        self.conv2 = conv3x3(width, width, stride, groups, dilation)
        self.bn2 = norm_layer(width)
        self.conv3 = conv1x1(width, planes * self.expansion)
        self.bn3 = norm_layer(planes * self.expansion)
        self.relu = nn.ReLU(inplace=True)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x: Tensor) -> Tensor:
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)
        out = self.relu(out)

        out = self.conv3(out)
        out = self.bn3(out)

        if self.downsample is not None:
```

```python
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)

        return out
```

In [ ]:
```python
class ResNet(nn.Module):
    def __init__(
        self,
        block: Type[Union[BasicBlock, Bottleneck]],
        layers: List[int],
        input_channels: int = 3,
        num_classes: int = 1000,
        zero_init_residual: bool = False,
        groups: int = 1,
        width_per_group: int = 64,
        replace_stride_with_dilation: Optional[List[bool]] = None,
        norm_layer: Optional[Callable[..., nn.Module]] = None,
    ) -> None:
        super().__init__()
        _log_api_usage_once(self)
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        self._norm_layer = norm_layer

        self.inplanes = 64
        self.dilation = 1
        if replace_stride_with_dilation is None:
            # each element in the tuple indicates if we should replace
            # the 2x2 stride with a dilated convolution instead
            replace_stride_with_dilation = [False, False, False]
        if len(replace_stride_with_dilation) != 3:
            raise ValueError(
                "replace_stride_with_dilation should be None "
                f"or a 3-element tuple, got {replace_stride_with_dilation}"
            )
        self.groups = groups
        self.base_width = width_per_group
        self.conv1 = nn.Conv2d(input_channels, self.inplanes, kernel_size=7, stride=2, padding=3, bias=False)
        self.bn1 = norm_layer(self.inplanes)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block, 64, layers[0])
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2, dilate=replace_stride_with_dilation[0])
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2, dilate=replace_stride_with_dilation[1])
```

```python
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2, dilate=replace_stride_with_dilation[2])
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * block.expansion, num_classes)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode="fan_out", nonlinearity="relu")
            elif isinstance(m, (nn.BatchNorm2d, nn.GroupNorm)):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)

        # Zero-initialize the last BN in each residual branch,
        # so that the residual branch starts with zeros, and each residual block behaves like an identity.
        # This improves the model by 0.2~0.3% according to https://arxiv.org/abs/1706.02677
        if zero_init_residual:
            for m in self.modules():
                if isinstance(m, Bottleneck) and m.bn3.weight is not None:
                    nn.init.constant_(m.bn3.weight, 0)  # type: ignore[arg-type]
                elif isinstance(m, BasicBlock) and m.bn2.weight is not None:
                    nn.init.constant_(m.bn2.weight, 0)  # type: ignore[arg-type]

    def _make_layer(
        self,
        block: Type[Union[BasicBlock, Bottleneck]],
        planes: int,
        blocks: int,
        stride: int = 1,
        dilate: bool = False,
    ) -> nn.Sequential:
        norm_layer = self._norm_layer
        downsample = None
        previous_dilation = self.dilation
        if dilate:
            self.dilation *= stride
            stride = 1
        if stride != 1 or self.inplanes != planes * block.expansion:
            downsample = nn.Sequential(
                conv1x1(self.inplanes, planes * block.expansion, stride),
                norm_layer(planes * block.expansion),
            )

        layers = []
        layers.append(
            block(
                self.inplanes, planes, stride, downsample, self.groups, self.base_width, previous_dilation, norm_layer
            )
```

```
        )
        self.inplanes = planes * block.expansion
        for _ in range(1, blocks):
            layers.append(
                block(
                    self.inplanes,
                    planes,
                    groups=self.groups,
                    base_width=self.base_width,
                    dilation=self.dilation,
                    norm_layer=norm_layer,
                )
            )

        return nn.Sequential(*layers)

    def _forward_impl(self, x: Tensor) -> Tensor:
        # See note [TorchScript super()]
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)

        return x

    def forward(self, x: Tensor) -> Tensor:
        return self._forward_impl(x)


def _resnet(
    block: Type[Union[BasicBlock, Bottleneck]],
    layers: List[int],
    weights: Optional[WeightsEnum],
    progress: bool,
    **kwargs: Any,
) -> ResNet:
    if weights is not None:
```

```python
        _ovewrite_named_param(kwargs, "num_classes", len(weights.meta["categories"]))

    model = ResNet(block, layers, **kwargs)

    if weights is not None:
        model.load_state_dict(weights.get_state_dict(progress=progress, check_hash=True))

    return model
```

In [ ]:
```python
__all__ = [
    "ResNet",
    "ResNet34_Weights",
    "resnet34_modified",
    ]

_COMMON_META = {
    "min_size": (1, 1),
    "categories": _IMAGENET_CATEGORIES,
}

class ResNet34_Weights(WeightsEnum):
    IMAGENET1K_V1 = Weights(
        url="https://download.pytorch.org/models/resnet34-b627a593.pth",
        transforms=partial(ImageClassification, crop_size=224),
        meta={
            **_COMMON_META,
            "num_params": 21797672,
            "recipe": "https://github.com/pytorch/vision/tree/main/references/classification#resnet",
            "_metrics": {
                "ImageNet-1K": {
                    "acc@1": 73.314,
                    "acc@5": 91.420,
                }
            },
            "_ops": 3.664,
            "_file_size": 83.275,
            "_docs": """These weights reproduce closely the results of the paper using a simple training recipe.""",
        },
    )
    DEFAULT = IMAGENET1K_V1

def _resnet34_modified(input_channels: int, num_classes: int, block: Type[Union[BasicBlock, Bottleneck]], layers: List[int], we
    if weights is not None:
        _ovewrite_named_param(kwargs, "num_classes", len(weights.meta["categories"]))

    model = ResNet(block, layers, input_channels=input_channels, num_classes=num_classes, **kwargs)
```

```
    if weights is not None:
        # Load state dict but ignore first conv layer if number of input channels is not 3
        state_dict = weights.get_state_dict(progress=progress, check_hash=True)
        if input_channels != 3:
            state_dict.pop('conv1.weight', None)
        model.load_state_dict(state_dict, strict=False)

    return model


def resnet34_modified(input_channels: int, num_classes: int, *, weights: Optional[ResNet34_Weights] = None, progress: bool = Tr
    return _resnet34_modified(input_channels, num_classes, BasicBlock, [3, 4, 6, 3], weights, progress, **kwargs)
```

Modified version of resnet

This is our modified version of resnet which has had the input channels and output target classes modified so as to be manually adjustable for our needs.

In [ ]:
```
def _resnet34_modified(input_channels: int, num_classes: int, block: Type[Union[BasicBlock, Bottleneck]], layers: List[int], we
    if weights is not None:
        _ovewrite_named_param(kwargs, "num_classes", len(weights.meta["categories"]))

    model = ResNet(block, layers, input_channels=input_channels, num_classes=num_classes, **kwargs)

    if weights is not None:
        # Load state dict but ignore first conv layer if number of input channels is not 3
        state_dict = weights.get_state_dict(progress=progress, check_hash=True)
        if input_channels != 3:
            state_dict.pop('conv1.weight', None)
        model.load_state_dict(state_dict, strict=False)

    return model


def resnet34_modified(input_channels: int, num_classes: int, *, weights: Optional[ResNet34_Weights] = None, progress: bool = Tr
    return _resnet34_modified(input_channels, num_classes, BasicBlock, [3, 4, 6, 3], weights, progress, **kwargs)
```

In [ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [ ]:
```
resnet = resnet34_modified(input_channels=1, num_classes=10)
```

In [ ]:
```
print(resnet)
```

```
ResNet(
  (conv1): Conv2d(1, 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=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
    )
    (2): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
```

```
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (4): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (5): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (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): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
```

```
  (fc): Linear(in_features=512, out_features=10, bias=True)
)
```

Task 4:

a) Train the ResNet-34 model with Adam optimizer with a learning rate of 1e-3 and train 10 for epochs

b) Compare the performance the results with the ones from Task 3.

↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ your code goes below

In [ ]:
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

history, model = train_and_validate(train_loader, valid_loader, resnet, optimizer, criterion, num_epochs=10, metric=accuracy_me

test_loss, test_metric = test_model(model, test_loader, criterion, metric=accuracy_metric, device=device)

# Predictions
X_test_tensor = torch.from_numpy(X_test[:20]).float().to(device)
with torch.no_grad():
    model.eval()
    predictions = model(X_test_tensor)
    predicted_labels = predictions.argmax(dim=1)

print("Predicted labels:", predicted_labels)
print("True labels:", y_test[:20])
```

```
Epoch [1/10], Train Loss: 0.4748, Train Metric: 0.8289, Val Loss: 0.3606, Val Metric: 0.8621
Epoch [2/10], Train Loss: 0.3060, Train Metric: 0.8891, Val Loss: 0.3718, Val Metric: 0.8703
Epoch [3/10], Train Loss: 0.2586, Train Metric: 0.9053, Val Loss: 0.2726, Val Metric: 0.8985
Epoch [4/10], Train Loss: 0.2302, Train Metric: 0.9145, Val Loss: 0.2720, Val Metric: 0.9022
Epoch [5/10], Train Loss: 0.2045, Train Metric: 0.9226, Val Loss: 0.3287, Val Metric: 0.8816
Epoch [6/10], Train Loss: 0.1849, Train Metric: 0.9307, Val Loss: 0.2915, Val Metric: 0.8929
Epoch [7/10], Train Loss: 0.1636, Train Metric: 0.9398, Val Loss: 0.2703, Val Metric: 0.9032
Epoch [8/10], Train Loss: 0.1488, Train Metric: 0.9448, Val Loss: 0.2706, Val Metric: 0.9038
Epoch [9/10], Train Loss: 0.1373, Train Metric: 0.9486, Val Loss: 0.2686, Val Metric: 0.9122
Epoch [10/10], Train Loss: 0.1187, Train Metric: 0.9560, Val Loss: 0.2693, Val Metric: 0.9075
Test Loss: 0.2925, Test Metric: 0.9041
Predicted labels: tensor([9, 2, 1, 1, 0, 1, 4, 6, 5, 7, 4, 5, 5, 3, 4, 1, 2, 4, 8, 0],
       device='cuda:0')
True labels: [9 2 1 1 6 1 4 6 5 7 4 5 7 3 4 1 2 4 8 0]
```

Task 4b) answer:

- Task 4 achieved a lower training loss (0.1187 vs. 0.1980), indicating that the model fit better to the training data in fewer epochs.

- Validation loss was slightly lower in Task 3 (0.2619 vs. 0.2693), suggesting that the 20-epoch model generalizes slightly better than the 10-epoch model.
- Validation accuracy is almost the same (~90.9% for Task 3 vs. ~90.7% for Task 4)
- Task 3 achieved a slightly lower test loss (0.2796 vs. 0.2925), which suggests a marginally better generalization.
- Test accuracy is nearly identical (90.64% vs. 90.41%), so both models perform similarly on unseen data.

↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑ your code goes above

Task 5: Pretrained Models for Transfer Learning

In this section we follow loosely the pytorch
Transfer Learning example written by Sasank Chilamkurthy.
We'll be using a bee/ant classification dataset.

Here we'll show the effects on model performance when using
a model which has weights **pretrained on a general dataset**
as compared with a model which is **trained from scratch**. In
this case we'll be looking at the same resnset34 from above
but with pretrained model weights.

These models may take over an hour to train if not on GPU.

In [ ]:
```
from torchvision.models import resnet34
from torchvision import datasets, models, transforms
```

In [ ]:
```
!wget https://download.pytorch.org/tutorial/hymenoptera_data.zip
```

```
--2025-03-13 20:43:36--  https://download.pytorch.org/tutorial/hymenoptera_data.zip
Resolving download.pytorch.org (download.pytorch.org)... 18.238.238.23, 18.238.238.114, 18.238.238.104, ...
Connecting to download.pytorch.org (download.pytorch.org)|18.238.238.23|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 47286322 (45M) [application/zip]
Saving to: 'hymenoptera_data.zip'

hymenoptera_data.zi 100%[===================>]  45.10M   235MB/s    in 0.2s

2025-03-13 20:43:36 (235 MB/s) - 'hymenoptera_data.zip' saved [47286322/47286322]
```

In [ ]:
```
!unzip hymenoptera_data.zip
```

```
Archive:  hymenoptera_data.zip
   creating: hymenoptera_data/
   creating: hymenoptera_data/train/
   creating: hymenoptera_data/train/ants/
  inflating: hymenoptera_data/train/ants/0013035.jpg
  inflating: hymenoptera_data/train/ants/1030023514_aad5c608f9.jpg
  inflating: hymenoptera_data/train/ants/1095476100_3906d8afde.jpg
  inflating: hymenoptera_data/train/ants/1099452230_d1949d3250.jpg
  inflating: hymenoptera_data/train/ants/116570827_e9c126745d.jpg
  inflating: hymenoptera_data/train/ants/1225872729_6f0856588f.jpg
  inflating: hymenoptera_data/train/ants/1262877379_64fcada201.jpg
  inflating: hymenoptera_data/train/ants/1269756697_0bce92cdab.jpg
  inflating: hymenoptera_data/train/ants/1286984635_5119e80de1.jpg
  inflating: hymenoptera_data/train/ants/132478121_2a430adea2.jpg
  inflating: hymenoptera_data/train/ants/1360291657_dc248c5eea.jpg
  inflating: hymenoptera_data/train/ants/1368913450_e146e2fb6d.jpg
  inflating: hymenoptera_data/train/ants/1473187633_63ccaacea6.jpg
  inflating: hymenoptera_data/train/ants/148715752_302c84f5a4.jpg
  inflating: hymenoptera_data/train/ants/1489674356_09d48dde0a.jpg
  inflating: hymenoptera_data/train/ants/149244013_c529578289.jpg
  inflating: hymenoptera_data/train/ants/150801003_3390b73135.jpg
  inflating: hymenoptera_data/train/ants/150801171_cd86f17ed8.jpg
  inflating: hymenoptera_data/train/ants/154124431_65460430f2.jpg
  inflating: hymenoptera_data/train/ants/162603798_40b51f1654.jpg
  inflating: hymenoptera_data/train/ants/1660097129_384bf54490.jpg
  inflating: hymenoptera_data/train/ants/167890289_dd5ba923f3.jpg
  inflating: hymenoptera_data/train/ants/1693954099_46d4c20605.jpg
  inflating: hymenoptera_data/train/ants/175998972.jpg
  inflating: hymenoptera_data/train/ants/178538489_bec7649292.jpg
  inflating: hymenoptera_data/train/ants/1804095607_0341701e1c.jpg
  inflating: hymenoptera_data/train/ants/1808777855_2a895621d7.jpg
  inflating: hymenoptera_data/train/ants/188552436_605cc9b36b.jpg
  inflating: hymenoptera_data/train/ants/1917341202_d00a7f9af5.jpg
  inflating: hymenoptera_data/train/ants/1924473702_daa9aacdbe.jpg
  inflating: hymenoptera_data/train/ants/196057951_63bf063b92.jpg
  inflating: hymenoptera_data/train/ants/196757565_326437f5fe.jpg
  inflating: hymenoptera_data/train/ants/201558278_fe4caecc76.jpg
  inflating: hymenoptera_data/train/ants/201790779_527f4c0168.jpg
  inflating: hymenoptera_data/train/ants/2019439677_2db655d361.jpg
  inflating: hymenoptera_data/train/ants/207947948_3ab29d7207.jpg
  inflating: hymenoptera_data/train/ants/20935278_9190345f6b.jpg
  inflating: hymenoptera_data/train/ants/224655713_3956f7d39a.jpg
  inflating: hymenoptera_data/train/ants/2265824718_2c96f485da.jpg
  inflating: hymenoptera_data/train/ants/2265825502_fff99cfd2d.jpg
  inflating: hymenoptera_data/train/ants/226951206_d6bf946504.jpg
  inflating: hymenoptera_data/train/ants/2278278459_6b99605e50.jpg
```

```
inflating: hymenoptera_data/train/ants/2288450226_a6e96e8fdf.jpg
inflating: hymenoptera_data/train/ants/2288481644_83ff7e4572.jpg
inflating: hymenoptera_data/train/ants/2292213964_ca51ce4bef.jpg
inflating: hymenoptera_data/train/ants/24335309_c5ea483bb8.jpg
inflating: hymenoptera_data/train/ants/245647475_9523dfd13e.jpg
inflating: hymenoptera_data/train/ants/255434217_1b2b3fe0a4.jpg
inflating: hymenoptera_data/train/ants/258217966_d9d90d18d3.jpg
inflating: hymenoptera_data/train/ants/275429470_b2d7d9290b.jpg
inflating: hymenoptera_data/train/ants/28847243_e79fe052cd.jpg
inflating: hymenoptera_data/train/ants/318052216_84dff3f98a.jpg
inflating: hymenoptera_data/train/ants/334167043_cbd1adaeb9.jpg
inflating: hymenoptera_data/train/ants/339670531_94b75ae47a.jpg
inflating: hymenoptera_data/train/ants/342438950_a3da61deab.jpg
inflating: hymenoptera_data/train/ants/36439863_0bec9f554f.jpg
inflating: hymenoptera_data/train/ants/374435068_7eee412ec4.jpg
inflating: hymenoptera_data/train/ants/382971067_0bfd33afe0.jpg
inflating: hymenoptera_data/train/ants/384191229_5779cf591b.jpg
inflating: hymenoptera_data/train/ants/386190770_672743c9a7.jpg
inflating: hymenoptera_data/train/ants/392382602_1b7bed32fa.jpg
inflating: hymenoptera_data/train/ants/403746349_71384f5b58.jpg
inflating: hymenoptera_data/train/ants/408393566_b5b694119b.jpg
inflating: hymenoptera_data/train/ants/424119020_6d57481dab.jpg
inflating: hymenoptera_data/train/ants/424873399_47658a91fb.jpg
inflating: hymenoptera_data/train/ants/450057712_771b3bfc91.jpg
inflating: hymenoptera_data/train/ants/45472593_bfd624f8dc.jpg
inflating: hymenoptera_data/train/ants/459694881_ac657d3187.jpg
inflating: hymenoptera_data/train/ants/460372577_f2f6a8c9fc.jpg
inflating: hymenoptera_data/train/ants/460874319_0a45ab4d05.jpg
inflating: hymenoptera_data/train/ants/466430434_4000737de9.jpg
inflating: hymenoptera_data/train/ants/470127037_513711fd21.jpg
inflating: hymenoptera_data/train/ants/474806473_ca6caab245.jpg
inflating: hymenoptera_data/train/ants/475961153_b8c13fd405.jpg
inflating: hymenoptera_data/train/ants/484293231_e53cfc0c89.jpg
inflating: hymenoptera_data/train/ants/49375974_e28ba6f17e.jpg
inflating: hymenoptera_data/train/ants/506249802_207cd979b4.jpg
inflating: hymenoptera_data/train/ants/506249836_717b73f540.jpg
inflating: hymenoptera_data/train/ants/512164029_c0a66b8498.jpg
inflating: hymenoptera_data/train/ants/512863248_43c8ce579b.jpg
inflating: hymenoptera_data/train/ants/518773929_734dbc5ff4.jpg
inflating: hymenoptera_data/train/ants/522163566_fec115ca66.jpg
inflating: hymenoptera_data/train/ants/522415432_2218f34bf8.jpg
inflating: hymenoptera_data/train/ants/531979952_bde12b3bc0.jpg
inflating: hymenoptera_data/train/ants/533848102_70a85ad6dd.jpg
inflating: hymenoptera_data/train/ants/535522953_308353a07c.jpg
inflating: hymenoptera_data/train/ants/540889389_48bb588b21.jpg
inflating: hymenoptera_data/train/ants/541630764_dbd285d63c.jpg
```

```
  inflating: hymenoptera_data/train/ants/543417860_b14237f569.jpg
  inflating: hymenoptera_data/train/ants/560966032_988f4d7bc4.jpg
  inflating: hymenoptera_data/train/ants/5650366_e22b7e1065.jpg
  inflating: hymenoptera_data/train/ants/6240329_72c01e663e.jpg
  inflating: hymenoptera_data/train/ants/6240338_93729615ec.jpg
  inflating: hymenoptera_data/train/ants/649026570_e58656104b.jpg
  inflating: hymenoptera_data/train/ants/662541407_ff8db781e7.jpg
  inflating: hymenoptera_data/train/ants/67270775_e9fdf77e9d.jpg
  inflating: hymenoptera_data/train/ants/6743948_2b8c096dda.jpg
  inflating: hymenoptera_data/train/ants/684133190_35b62c0c1d.jpg
  inflating: hymenoptera_data/train/ants/69639610_95e0de17aa.jpg
  inflating: hymenoptera_data/train/ants/707895295_009cf23188.jpg
  inflating: hymenoptera_data/train/ants/7759525_1363d24e88.jpg
  inflating: hymenoptera_data/train/ants/795000156_a9900a4a71.jpg
  inflating: hymenoptera_data/train/ants/822537660_caf4ba5514.jpg
  inflating: hymenoptera_data/train/ants/82852639_52b7f7f5e3.jpg
  inflating: hymenoptera_data/train/ants/841049277_b28e58ad05.jpg
  inflating: hymenoptera_data/train/ants/886401651_f878e888cd.jpg
  inflating: hymenoptera_data/train/ants/892108839_f1aad4ca46.jpg
  inflating: hymenoptera_data/train/ants/938946700_ca1c669085.jpg
  inflating: hymenoptera_data/train/ants/957233405_25c1d1187b.jpg
  inflating: hymenoptera_data/train/ants/9715481_b3cb4114ff.jpg
  inflating: hymenoptera_data/train/ants/998118368_6ac1d91f81.jpg
  inflating: hymenoptera_data/train/ants/ant photos.jpg
  inflating: hymenoptera_data/train/ants/Ant_1.jpg
  inflating: hymenoptera_data/train/ants/army-ants-red-picture.jpg
  inflating: hymenoptera_data/train/ants/formica.jpeg
  inflating: hymenoptera_data/train/ants/hormiga_co_por.jpg
  inflating: hymenoptera_data/train/ants/imageNotFound.gif
  inflating: hymenoptera_data/train/ants/kurokusa.jpg
  inflating: hymenoptera_data/train/ants/MehdiabadiAnt2_600.jpg
  inflating: hymenoptera_data/train/ants/Nepenthes_rafflesiana_ant.jpg
  inflating: hymenoptera_data/train/ants/swiss-army-ant.jpg
  inflating: hymenoptera_data/train/ants/termite-vs-ant.jpg
  inflating: hymenoptera_data/train/ants/trap-jaw-ant-insect-bg.jpg
  inflating: hymenoptera_data/train/ants/VietnameseAntMimicSpider.jpg
   creating: hymenoptera_data/train/bees/
  inflating: hymenoptera_data/train/bees/1092977343_cb42b38d62.jpg
  inflating: hymenoptera_data/train/bees/1093831624_fb5fbe2308.jpg
  inflating: hymenoptera_data/train/bees/1097045929_1753d1c765.jpg
  inflating: hymenoptera_data/train/bees/1232245714_f862fbe385.jpg
  inflating: hymenoptera_data/train/bees/129236073_0985e91c7d.jpg
  inflating: hymenoptera_data/train/bees/1295655112_7813f37d21.jpg
  inflating: hymenoptera_data/train/bees/132511197_0b86ad0fff.jpg
  inflating: hymenoptera_data/train/bees/132826773_dbbcb117b9.jpg
  inflating: hymenoptera_data/train/bees/150013791_969d9a968b.jpg
```

```
  inflating: hymenoptera_data/train/bees/1508176360_2972117c9d.jpg
  inflating: hymenoptera_data/train/bees/154600396_53e1252e52.jpg
  inflating: hymenoptera_data/train/bees/16838648_415acd9e3f.jpg
  inflating: hymenoptera_data/train/bees/1691282715_0addfdf5e8.jpg
  inflating: hymenoptera_data/train/bees/17209602_fe5a5a746f.jpg
  inflating: hymenoptera_data/train/bees/174142798_e5ad6d76e0.jpg
  inflating: hymenoptera_data/train/bees/1799726602_8580867f71.jpg
  inflating: hymenoptera_data/train/bees/1807583459_4fe92b3133.jpg
  inflating: hymenoptera_data/train/bees/196430254_46bd129ae7.jpg
  inflating: hymenoptera_data/train/bees/196658222_3fffd79c67.jpg
  inflating: hymenoptera_data/train/bees/198508668_97d818b6c4.jpg
  inflating: hymenoptera_data/train/bees/2031225713_50ed499635.jpg
  inflating: hymenoptera_data/train/bees/2037437624_2d7bce461f.jpg
  inflating: hymenoptera_data/train/bees/2053200300_8911ef438a.jpg
  inflating: hymenoptera_data/train/bees/205835650_e6f2614bee.jpg
  inflating: hymenoptera_data/train/bees/208702903_42fb4d9748.jpg
  inflating: hymenoptera_data/train/bees/21399619_3e61e5bb6f.jpg
  inflating: hymenoptera_data/train/bees/2227611847_ec72d40403.jpg
  inflating: hymenoptera_data/train/bees/2321139806_d73d899e66.jpg
  inflating: hymenoptera_data/train/bees/2330918208_8074770c20.jpg
  inflating: hymenoptera_data/train/bees/2345177635_caf07159b3.jpg
  inflating: hymenoptera_data/train/bees/2358061370_9daabbd9ac.jpg
  inflating: hymenoptera_data/train/bees/2364597044_3c3e3fc391.jpg
  inflating: hymenoptera_data/train/bees/2384149906_2cd8b0b699.jpg
  inflating: hymenoptera_data/train/bees/2397446847_04ef3cd3e1.jpg
  inflating: hymenoptera_data/train/bees/2405441001_b06c36fa72.jpg
  inflating: hymenoptera_data/train/bees/2445215254_51698ff797.jpg
  inflating: hymenoptera_data/train/bees/2452236943_255bfd9e58.jpg
  inflating: hymenoptera_data/train/bees/2467959963_a7831e9ff0.jpg
  inflating: hymenoptera_data/train/bees/2470492904_837e97800d.jpg
  inflating: hymenoptera_data/train/bees/2477324698_3d4b1b1cab.jpg
  inflating: hymenoptera_data/train/bees/2477349551_e75c97cf4d.jpg
  inflating: hymenoptera_data/train/bees/2486729079_62df0920be.jpg
  inflating: hymenoptera_data/train/bees/2486746709_c43cec0e42.jpg
  inflating: hymenoptera_data/train/bees/2493379287_4100e1dacc.jpg
  inflating: hymenoptera_data/train/bees/2495722465_879acf9d85.jpg
  inflating: hymenoptera_data/train/bees/2528444139_fa728b0f5b.jpg
  inflating: hymenoptera_data/train/bees/2538361678_9da84b77e3.jpg
  inflating: hymenoptera_data/train/bees/2551813042_8a070aeb2b.jpg
  inflating: hymenoptera_data/train/bees/2580598377_a4caecdb54.jpg
  inflating: hymenoptera_data/train/bees/2601176055_8464e6aa71.jpg
  inflating: hymenoptera_data/train/bees/2610833167_79bf0bcae5.jpg
  inflating: hymenoptera_data/train/bees/2610838525_fe8e3cae47.jpg
  inflating: hymenoptera_data/train/bees/2617161745_fa3ebe85b4.jpg
  inflating: hymenoptera_data/train/bees/2625499656_e3415e374d.jpg
  inflating: hymenoptera_data/train/bees/2634617358_f32fd16bea.jpg
```

```
  inflating: hymenoptera_data/train/bees/2638074627_6b3ae746a0.jpg
  inflating: hymenoptera_data/train/bees/2645107662_b73a8595cc.jpg
  inflating: hymenoptera_data/train/bees/2651621464_a2fa8722eb.jpg
  inflating: hymenoptera_data/train/bees/2652877533_a564830cbf.jpg
  inflating: hymenoptera_data/train/bees/266644509_d30bb16a1b.jpg
  inflating: hymenoptera_data/train/bees/2683605182_9d2a0c66cf.jpg
  inflating: hymenoptera_data/train/bees/2704348794_eb5d5178c2.jpg
  inflating: hymenoptera_data/train/bees/2707440199_cd170bd512.jpg
  inflating: hymenoptera_data/train/bees/2710368626_cb42882dc8.jpg
  inflating: hymenoptera_data/train/bees/2722592222_258d473e17.jpg
  inflating: hymenoptera_data/train/bees/2728759455_ce9bb8cd7a.jpg
  inflating: hymenoptera_data/train/bees/2756397428_1d82a08807.jpg
  inflating: hymenoptera_data/train/bees/2765347790_da6cf6cb40.jpg
  inflating: hymenoptera_data/train/bees/2781170484_5d61835d63.jpg
  inflating: hymenoptera_data/train/bees/279113587_b4843db199.jpg
  inflating: hymenoptera_data/train/bees/2792000093_e8ae0718cf.jpg
  inflating: hymenoptera_data/train/bees/2801728106_833798c909.jpg
  inflating: hymenoptera_data/train/bees/2822388965_f6dca2a275.jpg
  inflating: hymenoptera_data/train/bees/2861002136_52c7c6f708.jpg
  inflating: hymenoptera_data/train/bees/2908916142_a7ac8b57a8.jpg
  inflating: hymenoptera_data/train/bees/29494643_e3410f0d37.jpg
  inflating: hymenoptera_data/train/bees/2959730355_416a18c63c.jpg
  inflating: hymenoptera_data/train/bees/2962405283_22718d9617.jpg
  inflating: hymenoptera_data/train/bees/3006264892_30e9cced70.jpg
  inflating: hymenoptera_data/train/bees/3030189811_01d095b793.jpg
  inflating: hymenoptera_data/train/bees/3030772428_8578335616.jpg
  inflating: hymenoptera_data/train/bees/3044402684_3853071a87.jpg
  inflating: hymenoptera_data/train/bees/3074585407_9854eb3153.jpg
  inflating: hymenoptera_data/train/bees/3079610310_ac2d0ae7bc.jpg
  inflating: hymenoptera_data/train/bees/3090975720_71f12e6de4.jpg
  inflating: hymenoptera_data/train/bees/3100226504_c0d4f1e3f1.jpg
  inflating: hymenoptera_data/train/bees/342758693_c56b89b6b6.jpg
  inflating: hymenoptera_data/train/bees/354167719_22dca13752.jpg
  inflating: hymenoptera_data/train/bees/359928878_b3b418c728.jpg
  inflating: hymenoptera_data/train/bees/365759866_b15700c59b.jpg
  inflating: hymenoptera_data/train/bees/36900412_92b81831ad.jpg
  inflating: hymenoptera_data/train/bees/39672681_1302d204d1.jpg
  inflating: hymenoptera_data/train/bees/39747887_42df2855ee.jpg
  inflating: hymenoptera_data/train/bees/421515404_e87569fd8b.jpg
  inflating: hymenoptera_data/train/bees/444532809_9e931e2279.jpg
  inflating: hymenoptera_data/train/bees/446296270_d9e8b93ecf.jpg
  inflating: hymenoptera_data/train/bees/452462677_7be43af8ff.jpg
  inflating: hymenoptera_data/train/bees/452462695_40a4e5b559.jpg
  inflating: hymenoptera_data/train/bees/457457145_5f86eb7e9c.jpg
  inflating: hymenoptera_data/train/bees/465133211_80e0c27f60.jpg
  inflating: hymenoptera_data/train/bees/469333327_358ba8fe8a.jpg
```

```
inflating: hymenoptera_data/train/bees/472288710_2abee16fa0.jpg
inflating: hymenoptera_data/train/bees/473618094_8ffdcab215.jpg
inflating: hymenoptera_data/train/bees/476347960_52edd72b06.jpg
inflating: hymenoptera_data/train/bees/478701318_bbd5e557b8.jpg
inflating: hymenoptera_data/train/bees/507288830_f46e8d4cb2.jpg
inflating: hymenoptera_data/train/bees/509247772_2db2d01374.jpg
inflating: hymenoptera_data/train/bees/513545352_fd3e7c7c5d.jpg
inflating: hymenoptera_data/train/bees/522104315_5d3cb2758e.jpg
inflating: hymenoptera_data/train/bees/537309131_532bfa59ea.jpg
inflating: hymenoptera_data/train/bees/586041248_3032e277a9.jpg
inflating: hymenoptera_data/train/bees/760526046_547e8b381f.jpg
inflating: hymenoptera_data/train/bees/760568592_45a52c847f.jpg
inflating: hymenoptera_data/train/bees/774440991_63a4aa0cbe.jpg
inflating: hymenoptera_data/train/bees/85112639_6e860b0469.jpg
inflating: hymenoptera_data/train/bees/873076652_eb098dab2d.jpg
inflating: hymenoptera_data/train/bees/90179376_abc234e5f4.jpg
inflating: hymenoptera_data/train/bees/92663402_37f379e57a.jpg
inflating: hymenoptera_data/train/bees/95238259_98470c5b10.jpg
inflating: hymenoptera_data/train/bees/969455125_58c797ef17.jpg
inflating: hymenoptera_data/train/bees/98391118_bdb1e80cce.jpg
 creating: hymenoptera_data/val/
 creating: hymenoptera_data/val/ants/
inflating: hymenoptera_data/val/ants/10308379_1b6c72e180.jpg
inflating: hymenoptera_data/val/ants/1053149811_f62a3410d3.jpg
inflating: hymenoptera_data/val/ants/1073564163_225a64f170.jpg
inflating: hymenoptera_data/val/ants/1119630822_cd325ea21a.jpg
inflating: hymenoptera_data/val/ants/1124525276_816a07c17f.jpg
inflating: hymenoptera_data/val/ants/11381045_b352a47d8c.jpg
inflating: hymenoptera_data/val/ants/119785936_dd428e40c3.jpg
inflating: hymenoptera_data/val/ants/1247887232_edcb61246c.jpg
inflating: hymenoptera_data/val/ants/1262751255_c56c042b7b.jpg
inflating: hymenoptera_data/val/ants/1337725712_2eb53cd742.jpg
inflating: hymenoptera_data/val/ants/1358854066_5ad8015f7f.jpg
inflating: hymenoptera_data/val/ants/1440002809_b268d9a66a.jpg
inflating: hymenoptera_data/val/ants/147542264_79506478c2.jpg
inflating: hymenoptera_data/val/ants/152286280_411648ec27.jpg
inflating: hymenoptera_data/val/ants/153320619_2aeb5fa0ee.jpg
inflating: hymenoptera_data/val/ants/153783656_85f9c3ac70.jpg
inflating: hymenoptera_data/val/ants/157401988_d0564a9d02.jpg
inflating: hymenoptera_data/val/ants/159515240_d5981e20d1.jpg
inflating: hymenoptera_data/val/ants/161076144_124db762d6.jpg
inflating: hymenoptera_data/val/ants/161292361_c16e0bf57a.jpg
inflating: hymenoptera_data/val/ants/170652283_ecdaff5d1a.jpg
inflating: hymenoptera_data/val/ants/17081114_79b9a27724.jpg
inflating: hymenoptera_data/val/ants/172772109_d0a8e15fb0.jpg
inflating: hymenoptera_data/val/ants/1743840368_b5ccda82b7.jpg
```

```
  inflating: hymenoptera_data/val/ants/181942028_961261ef48.jpg
  inflating: hymenoptera_data/val/ants/183260961_64ab754c97.jpg
  inflating: hymenoptera_data/val/ants/2039585088_c6f47c592e.jpg
  inflating: hymenoptera_data/val/ants/205398178_c395c5e460.jpg
  inflating: hymenoptera_data/val/ants/208072188_f293096296.jpg
  inflating: hymenoptera_data/val/ants/209615353_eeb38ba204.jpg
  inflating: hymenoptera_data/val/ants/2104709400_8831b4fc6f.jpg
  inflating: hymenoptera_data/val/ants/212100470_b485e7b7b9.jpg
  inflating: hymenoptera_data/val/ants/2127908701_d49dc83c97.jpg
  inflating: hymenoptera_data/val/ants/2191997003_379df31291.jpg
  inflating: hymenoptera_data/val/ants/2211974567_ee4606b493.jpg
  inflating: hymenoptera_data/val/ants/2219621907_47bc7cc6b0.jpg
  inflating: hymenoptera_data/val/ants/2238242353_52c82441df.jpg
  inflating: hymenoptera_data/val/ants/2255445811_dabcdf7258.jpg
  inflating: hymenoptera_data/val/ants/239161491_86ac23b0a3.jpg
  inflating: hymenoptera_data/val/ants/263615709_cfb28f6b8e.jpg
  inflating: hymenoptera_data/val/ants/308196310_1db5ffa01b.jpg
  inflating: hymenoptera_data/val/ants/319494379_648fb5a1c6.jpg
  inflating: hymenoptera_data/val/ants/35558229_1fa4608a7a.jpg
  inflating: hymenoptera_data/val/ants/412436937_4c2378efc2.jpg
  inflating: hymenoptera_data/val/ants/436944325_d4925a38c7.jpg
  inflating: hymenoptera_data/val/ants/445356866_6cb3289067.jpg
  inflating: hymenoptera_data/val/ants/459442412_412fecf3fe.jpg
  inflating: hymenoptera_data/val/ants/470127071_8b8ee2bd74.jpg
  inflating: hymenoptera_data/val/ants/477437164_bc3e6e594a.jpg
  inflating: hymenoptera_data/val/ants/488272201_c5aa281348.jpg
  inflating: hymenoptera_data/val/ants/502717153_3e4865621a.jpg
  inflating: hymenoptera_data/val/ants/518746016_bcc28f8b5b.jpg
  inflating: hymenoptera_data/val/ants/540543309_ddbb193ee5.jpg
  inflating: hymenoptera_data/val/ants/562589509_7e55469b97.jpg
  inflating: hymenoptera_data/val/ants/57264437_a19006872f.jpg
  inflating: hymenoptera_data/val/ants/573151833_ebbc274b77.jpg
  inflating: hymenoptera_data/val/ants/649407494_9b6bc4949f.jpg
  inflating: hymenoptera_data/val/ants/751649788_78dd7d16ce.jpg
  inflating: hymenoptera_data/val/ants/768870506_8f115d3d37.jpg
  inflating: hymenoptera_data/val/ants/800px-Meat_eater_ant_qeen_excavating_hole.jpg
  inflating: hymenoptera_data/val/ants/8124241_36b290d372.jpg
  inflating: hymenoptera_data/val/ants/8398478_50ef10c47a.jpg
  inflating: hymenoptera_data/val/ants/854534770_31f6156383.jpg
  inflating: hymenoptera_data/val/ants/892676922_4ab37dce07.jpg
  inflating: hymenoptera_data/val/ants/94999827_36895faade.jpg
  inflating: hymenoptera_data/val/ants/Ant-1818.jpg
  inflating: hymenoptera_data/val/ants/ants-devouring-remains-of-large-dead-insect-on-red-tile-in-Stellenbosch-South-Africa-clos
eup-1-DHD.jpg
  inflating: hymenoptera_data/val/ants/desert_ant.jpg
  inflating: hymenoptera_data/val/ants/F.pergan.28(f).jpg
```

```
  inflating: hymenoptera_data/val/ants/Hormiga.jpg
   creating: hymenoptera_data/val/bees/
  inflating: hymenoptera_data/val/bees/1032546534_06907fe3b3.jpg
  inflating: hymenoptera_data/val/bees/10870992_eebeeb3a12.jpg
  inflating: hymenoptera_data/val/bees/1181173278_23c36fac71.jpg
  inflating: hymenoptera_data/val/bees/1297972485_33266a18d9.jpg
  inflating: hymenoptera_data/val/bees/1328423762_f7a88a8451.jpg
  inflating: hymenoptera_data/val/bees/1355974687_1341c1face.jpg
  inflating: hymenoptera_data/val/bees/144098310_a4176fd54d.jpg
  inflating: hymenoptera_data/val/bees/1486120850_490388f84b.jpg
  inflating: hymenoptera_data/val/bees/149973093_da3c446268.jpg
  inflating: hymenoptera_data/val/bees/151594775_ee7dc17b60.jpg
  inflating: hymenoptera_data/val/bees/151603988_2c6f7d14c7.jpg
  inflating: hymenoptera_data/val/bees/1519368889_4270261ee3.jpg
  inflating: hymenoptera_data/val/bees/152789693_220b003452.jpg
  inflating: hymenoptera_data/val/bees/177677657_a38c97e572.jpg
  inflating: hymenoptera_data/val/bees/1799729694_0c40101071.jpg
  inflating: hymenoptera_data/val/bees/181171681_c5a1a82ded.jpg
  inflating: hymenoptera_data/val/bees/187130242_4593a4c610.jpg
  inflating: hymenoptera_data/val/bees/203868383_0fcbb48278.jpg
  inflating: hymenoptera_data/val/bees/2060668999_e11edb10d0.jpg
  inflating: hymenoptera_data/val/bees/2086294791_6f3789d8a6.jpg
  inflating: hymenoptera_data/val/bees/2103637821_8d26ee6b90.jpg
  inflating: hymenoptera_data/val/bees/2104135106_a65eede1de.jpg
  inflating: hymenoptera_data/val/bees/215512424_687e1e0821.jpg
  inflating: hymenoptera_data/val/bees/2173503984_9c6aaaa7e2.jpg
  inflating: hymenoptera_data/val/bees/220376539_20567395d8.jpg
  inflating: hymenoptera_data/val/bees/224841383_d050f5f510.jpg
  inflating: hymenoptera_data/val/bees/2321144482_f3785ba7b2.jpg
  inflating: hymenoptera_data/val/bees/238161922_55fa9a76ae.jpg
  inflating: hymenoptera_data/val/bees/2407809945_fb525ef54d.jpg
  inflating: hymenoptera_data/val/bees/2415414155_1916f03b42.jpg
  inflating: hymenoptera_data/val/bees/2438480600_40a1249879.jpg
  inflating: hymenoptera_data/val/bees/2444778727_4b781ac424.jpg
  inflating: hymenoptera_data/val/bees/2457841282_7867f16639.jpg
  inflating: hymenoptera_data/val/bees/2470492902_3572c90f75.jpg
  inflating: hymenoptera_data/val/bees/2478216347_535c8fe6d7.jpg
  inflating: hymenoptera_data/val/bees/2501530886_e20952b97d.jpg
  inflating: hymenoptera_data/val/bees/2506114833_90a41c5267.jpg
  inflating: hymenoptera_data/val/bees/2509402554_31821cb0b6.jpg
  inflating: hymenoptera_data/val/bees/2525379273_dcb26a516d.jpg
  inflating: hymenoptera_data/val/bees/26589803_5ba7000313.jpg
  inflating: hymenoptera_data/val/bees/2668391343_45e272cd07.jpg
  inflating: hymenoptera_data/val/bees/2670536155_c170f49cd0.jpg
  inflating: hymenoptera_data/val/bees/2685605303_9eed79d59d.jpg
  inflating: hymenoptera_data/val/bees/2702408468_d9ed795f4f.jpg
```

```
  inflating: hymenoptera_data/val/bees/2709775832_85b4b50a57.jpg
  inflating: hymenoptera_data/val/bees/2717418782_bd83307d9f.jpg
  inflating: hymenoptera_data/val/bees/272986700_d4d4bf8c4b.jpg
  inflating: hymenoptera_data/val/bees/2741763055_9a7bb00802.jpg
  inflating: hymenoptera_data/val/bees/2745389517_250a397f31.jpg
  inflating: hymenoptera_data/val/bees/2751836205_6f7b5eff30.jpg
  inflating: hymenoptera_data/val/bees/2782079948_8d4e94a826.jpg
  inflating: hymenoptera_data/val/bees/2809496124_5f25b5946a.jpg
  inflating: hymenoptera_data/val/bees/2815838190_0a9889d995.jpg
  inflating: hymenoptera_data/val/bees/2841437312_789699c740.jpg
  inflating: hymenoptera_data/val/bees/2883093452_7e3a1eb53f.jpg
  inflating: hymenoptera_data/val/bees/290082189_f66cb80bfc.jpg
  inflating: hymenoptera_data/val/bees/296565463_d07a7bed96.jpg
  inflating: hymenoptera_data/val/bees/3077452620_548c79fda0.jpg
  inflating: hymenoptera_data/val/bees/348291597_ee836fbb1a.jpg
  inflating: hymenoptera_data/val/bees/350436573_41f4ecb6c8.jpg
  inflating: hymenoptera_data/val/bees/353266603_d3eac7e9a0.jpg
  inflating: hymenoptera_data/val/bees/372228424_16da1f8884.jpg
  inflating: hymenoptera_data/val/bees/400262091_701c00031c.jpg
  inflating: hymenoptera_data/val/bees/416144384_961c326481.jpg
  inflating: hymenoptera_data/val/bees/44105569_16720a960c.jpg
  inflating: hymenoptera_data/val/bees/456097971_860949c4fc.jpg
  inflating: hymenoptera_data/val/bees/464594019_1b24a28bb1.jpg
  inflating: hymenoptera_data/val/bees/485743562_d8cc6b8f73.jpg
  inflating: hymenoptera_data/val/bees/540976476_844950623f.jpg
  inflating: hymenoptera_data/val/bees/54736755_c057723f64.jpg
  inflating: hymenoptera_data/val/bees/57459255_752774f1b2.jpg
  inflating: hymenoptera_data/val/bees/576452297_897023f002.jpg
  inflating: hymenoptera_data/val/bees/586474709_ae436da045.jpg
  inflating: hymenoptera_data/val/bees/590318879_68cf112861.jpg
  inflating: hymenoptera_data/val/bees/59798110_2b6a3c8031.jpg
  inflating: hymenoptera_data/val/bees/603709866_a97c7cfc72.jpg
  inflating: hymenoptera_data/val/bees/603711658_4c8cd2201e.jpg
  inflating: hymenoptera_data/val/bees/65038344_52a45d090d.jpg
  inflating: hymenoptera_data/val/bees/6a00d8341c630a53ef00e553d0beb18834-800wi.jpg
  inflating: hymenoptera_data/val/bees/72100438_73de9f17af.jpg
  inflating: hymenoptera_data/val/bees/759745145_e8bc776ec8.jpg
  inflating: hymenoptera_data/val/bees/936182217_c4caa5222d.jpg
  inflating: hymenoptera_data/val/bees/abeja.jpg
```

In [ ]:

```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
```

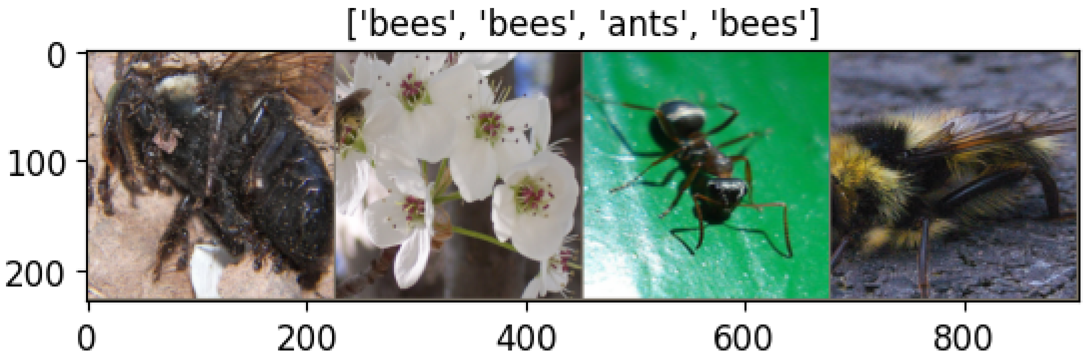
```
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
}
```

In [ ]:
```
data_dir = 'hymenoptera_data'
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                          data_transforms[x])
                  for x in ['train', 'val']}
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                             shuffle=True, num_workers=4)
              for x in ['train', 'val']}
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
class_names = image_datasets['train'].classes

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:617: UserWarning: This DataLoader will create 4 worker pr
ocesses in total. Our suggested max number of worker in current system is 2, which is smaller than what this DataLoader is going
to create. Please be aware that excessive worker creation might get DataLoader running slow or even freeze, lower the worker num
ber to avoid potential slowness/freeze if necessary.
  warnings.warn(
```

In [ ]:
```
def imshow(inp, title=None):
    """Display image for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001)  # pause a bit so that plots are updated


# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

# Make a grid from batch
out = torchvision.utils.make_grid(inputs)
```

```
imshow(out, title=[class_names[x] for x in classes])
```



In [ ]:
```
#Notice that resnet by default is designed for 1000 classes so we change that to 2
resnet_untrained = nn.Sequential(
    resnet34(pretrained=False),
    nn.Linear(1000, 2)
)
print(resnet_untrained)
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated
since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `No
ne` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `wei
ghts=None`.
  warnings.warn(msg)
```

```
Sequential(
  (0): 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=True)
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (1): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
      (2): BasicBlock(
        (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (layer2): Sequential(
      (0): BasicBlock(
        (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (downsample): Sequential(
          (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): BasicBlock(
        (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
```

```
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (4): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (5): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (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): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

```
    (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
    (fc): Linear(in_features=512, out_features=1000, bias=True)
  )
  (1): Linear(in_features=1000, out_features=2, bias=True)
)
```

In [ ]:
```python
optimizer = torch.optim.Adam(resnet_untrained.parameters(), lr=1e-5)
criterion = nn.CrossEntropyLoss()
num_epochs = 20
history, resnet_untrained = train_and_validate(dataloaders['train'], dataloaders['val'], resnet_untrained, optimizer, criterion
```

```
Epoch [1/20], Train Loss: 0.7041, Train Metric: 0.5164, Val Loss: 0.7197, Val Metric: 0.4679
Epoch [2/20], Train Loss: 0.6741, Train Metric: 0.5861, Val Loss: 0.6272, Val Metric: 0.6538
Epoch [3/20], Train Loss: 0.6900, Train Metric: 0.5697, Val Loss: 0.6886, Val Metric: 0.5256
Epoch [4/20], Train Loss: 0.6570, Train Metric: 0.5697, Val Loss: 0.6466, Val Metric: 0.5897
Epoch [5/20], Train Loss: 0.6551, Train Metric: 0.6230, Val Loss: 0.6336, Val Metric: 0.6282
Epoch [6/20], Train Loss: 0.6542, Train Metric: 0.5902, Val Loss: 0.6445, Val Metric: 0.6603
Epoch [7/20], Train Loss: 0.6383, Train Metric: 0.6434, Val Loss: 0.6384, Val Metric: 0.6667
Epoch [8/20], Train Loss: 0.6512, Train Metric: 0.6107, Val Loss: 0.6177, Val Metric: 0.6410
Epoch [9/20], Train Loss: 0.6493, Train Metric: 0.6270, Val Loss: 0.5816, Val Metric: 0.6923
Epoch [10/20], Train Loss: 0.6558, Train Metric: 0.6475, Val Loss: 0.6993, Val Metric: 0.5769
Epoch [11/20], Train Loss: 0.5974, Train Metric: 0.6721, Val Loss: 0.5764, Val Metric: 0.6667
Epoch [12/20], Train Loss: 0.6765, Train Metric: 0.5861, Val Loss: 0.6539, Val Metric: 0.6026
Epoch [13/20], Train Loss: 0.6422, Train Metric: 0.6107, Val Loss: 0.5935, Val Metric: 0.6859
Epoch [14/20], Train Loss: 0.5670, Train Metric: 0.7213, Val Loss: 0.7254, Val Metric: 0.5962
Epoch [15/20], Train Loss: 0.6404, Train Metric: 0.6352, Val Loss: 0.5589, Val Metric: 0.7051
Epoch [16/20], Train Loss: 0.6363, Train Metric: 0.6598, Val Loss: 0.6624, Val Metric: 0.6090
Epoch [17/20], Train Loss: 0.6195, Train Metric: 0.6639, Val Loss: 0.6330, Val Metric: 0.6410
Epoch [18/20], Train Loss: 0.5996, Train Metric: 0.6721, Val Loss: 0.6639, Val Metric: 0.7051
Epoch [19/20], Train Loss: 0.5986, Train Metric: 0.6885, Val Loss: 0.6765, Val Metric: 0.6538
Epoch [20/20], Train Loss: 0.5818, Train Metric: 0.7008, Val Loss: 0.8231, Val Metric: 0.5897
```

In [ ]:
```python
resnet_pretrained = nn.Sequential(
    resnet34(pretrained=True),
    nn.Linear(1000, 2)
)
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `No
ne` for 'weights' are deprecated since 0.13 and may be removed in the future. The current behavior is equivalent to passing `wei
ghts=ResNet34_Weights.IMAGENET1K_V1`. You can also use `weights=ResNet34_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to /root/.cache/torch/hub/checkpoints/resnet34-b627a59
3.pth
100%|██████████| 83.3M/83.3M [00:00<00:00, 190MB/s]
```

In [ ]:
```python
optimizer = torch.optim.Adam(resnet_pretrained.parameters(), lr=1e-5)
criterion = nn.CrossEntropyLoss()
```

```
num_epochs = 20
history, resnet_pretrained = train_and_validate(dataloaders['train'], dataloaders['val'], resnet_pretrained, optimizer, criteri
```

```
Epoch [1/20], Train Loss: 0.6592, Train Metric: 0.6557, Val Loss: 0.3305, Val Metric: 0.8782
Epoch [2/20], Train Loss: 0.4777, Train Metric: 0.7541, Val Loss: 0.2588, Val Metric: 0.9103
Epoch [3/20], Train Loss: 0.3953, Train Metric: 0.8115, Val Loss: 0.2444, Val Metric: 0.8974
Epoch [4/20], Train Loss: 0.3659, Train Metric: 0.8320, Val Loss: 0.2017, Val Metric: 0.9103
Epoch [5/20], Train Loss: 0.3257, Train Metric: 0.8402, Val Loss: 0.2080, Val Metric: 0.9167
Epoch [6/20], Train Loss: 0.3417, Train Metric: 0.8361, Val Loss: 0.2044, Val Metric: 0.9167
Epoch [7/20], Train Loss: 0.3752, Train Metric: 0.8156, Val Loss: 0.2071, Val Metric: 0.9167
Epoch [8/20], Train Loss: 0.2786, Train Metric: 0.8689, Val Loss: 0.2085, Val Metric: 0.9167
Epoch [9/20], Train Loss: 0.3093, Train Metric: 0.8689, Val Loss: 0.2104, Val Metric: 0.9167
Epoch [10/20], Train Loss: 0.2167, Train Metric: 0.9180, Val Loss: 0.1931, Val Metric: 0.9103
Epoch [11/20], Train Loss: 0.2614, Train Metric: 0.8811, Val Loss: 0.1815, Val Metric: 0.9167
Epoch [12/20], Train Loss: 0.3063, Train Metric: 0.8770, Val Loss: 0.2057, Val Metric: 0.9038
Epoch [13/20], Train Loss: 0.3303, Train Metric: 0.8689, Val Loss: 0.1888, Val Metric: 0.9103
Epoch [14/20], Train Loss: 0.2621, Train Metric: 0.8689, Val Loss: 0.1954, Val Metric: 0.9231
Epoch [15/20], Train Loss: 0.1922, Train Metric: 0.9221, Val Loss: 0.1742, Val Metric: 0.9423
Epoch [16/20], Train Loss: 0.2545, Train Metric: 0.8975, Val Loss: 0.1718, Val Metric: 0.9295
Epoch [17/20], Train Loss: 0.2399, Train Metric: 0.8770, Val Loss: 0.1948, Val Metric: 0.9038
Epoch [18/20], Train Loss: 0.2648, Train Metric: 0.8934, Val Loss: 0.2052, Val Metric: 0.9038
Epoch [19/20], Train Loss: 0.1863, Train Metric: 0.9385, Val Loss: 0.2168, Val Metric: 0.9231
Epoch [20/20], Train Loss: 0.2760, Train Metric: 0.8852, Val Loss: 0.2529, Val Metric: 0.8910
```

Task 5:

- Task 5a) Explain transfer learning and its benefits
- Task 5b) Compare the two trainings above (with/without pretraining).
  What is the difference and which one performs better here?

↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓ your code goes below

Task 5a) answer:Transfer learning is a machine learning technique where a model trained on one task is adapted for a different but related task. Instead of training a neural network from scratch, a pre-trained model (usually trained on a large dataset like ImageNet) is used as a starting point, and the final layers are fine-tuned for the new task. Benefits of Transfer Learning:

- Faster Training: Since the pre-trained model already has learned low-level features (e.g., edges, textures, and patterns), only the final layers need to be trained, reducing training time significantly.
- Higher Accuracy with Less Data: Pre-trained models have already learned general features from large datasets, which helps when working with smaller datasets.
- Better Generalization: Models trained from scratch often require massive amounts of labeled data to generalize well. Transfer learning enables models to achieve good performance even with limited data.

- Efficient Use of Computational Resources: Since most of the network weights are pre-trained, training a new model requires fewer computational resources.
- Avoids Overfitting: By leveraging learned features from large-scale datasets, transfer learning reduces overfitting, especially when the new dataset is small.

Task 5b) answer:

- With pretraining, the model reaches higher accuracy faster and achieves a much lower loss.
- Without pretraining, the model struggles to learn early on (train accuracy starts at 51.64%, compared to 65.57% in the pretraining case).
- The final validation accuracy is significantly higher in the pretrained model (89.10% vs. 58.97%), indicating better generalization.

↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑ your code goes above

Task 6: High Accuracy CNN for MNIST

Build your own CNN and try to achieve the highest possible accuracy on MNIST.
A basic structure is given below, play around with it.

Try a model which uses 2 convolutional layers, followed by 1 pooling layer, then dropout 25%, then a Linear layer, another dropout layer but with 50% dropout, and finally the output layer. It reaches about 99.2% accuracy on the test set. This places this model roughly in the top 20% in the MNIST Kaggle competition.

In order to reach an accuracy higher than 99.5% on the test set you might try:

a) batch normalization layers
b) set a learning scheduler (Check Chapter 11)
c) add image augmentation (Check Chapter 14)
d) create an ensemble (Check Chapter 14)
e) use hyperparameter tuning

As long as you implement at least **two** of the above you will get full points on this one.

In [ ]:
```
import numpy as np
import tensorflow as tf
```

```python
from tensorflow import keras
from tensorflow.keras import layers, models, regularizers
from keras.datasets import mnist
from tensorflow.keras.preprocessing.image import ImageDataGenerator

print("TensorFlow version:", tf.__version__)
print("Num GPUs Available:", len(tf.config.list_physical_devices('GPU')))
tf.keras.backend.clear_session()

# Load MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Normalize the images to [0, 1] range
train_images = train_images.astype("float32") / 255.0
test_images = test_images.astype("float32") / 255.0

# Reshape to add channel dimension
train_images = np.expand_dims(train_images, axis=-1)
test_images = np.expand_dims(test_images, axis=-1)

# Convert labels to categorical (one-hot encoding)
num_classes = 10
train_labels = keras.utils.to_categorical(train_labels, num_classes)
test_labels = keras.utils.to_categorical(test_labels, num_classes)

# Data Augmentation
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1
)
datagen.fit(train_images)

# Build CNN model with batch normalization and dropout
def build_model():
    model = models.Sequential()

    # add more Cov layer
    model.add(layers.Conv2D(32, (3, 3), activation='relu', padding="same", input_shape=(28, 28, 1)))
    model.add(layers.BatchNormalization())
    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding="same"))
    model.add(layers.BatchNormalization())

    # add more layer
    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding="same"))
```

```python
    model.add(layers.BatchNormalization())

    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Dropout(0.2))

    # Fully connected layer
    model.add(layers.Flatten())
    model.add(layers.Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.0005))) #  L2 reg
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.4))  #  Dropout

    # Output layer
    model.add(layers.Dense(num_classes, activation='softmax'))

    return model

# Create the model
model = build_model()

# use CosineAnnealing learning scheduler
initial_learning_rate = 0.001
lr_schedule = keras.optimizers.schedules.CosineDecay(initial_learning_rate, decay_steps=10000)

model.compile(optimizer=keras.optimizers.Adam(learning_rate=lr_schedule),
              loss="categorical_crossentropy",
              metrics=["accuracy"])

# Train the model with data augmentation
history = model.fit(datagen.flow(train_images, train_labels, batch_size=256),
                    epochs=35,
                    validation_data=(test_images, test_labels),
                    verbose=1)

# Evaluate on test data
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=0)

# Display final accuracy
print(f"Test Accuracy: {test_acc:.4f}")
```

```
TensorFlow version: 2.18.0
Num GPUs Available: 1
Epoch 1/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 31s 107ms/step - accuracy: 0.8632 - loss: 1.0901 - val_accuracy: 0.1525 - val_loss: 2.9341
Epoch 2/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 84ms/step - accuracy: 0.9706 - loss: 0.4048 - val_accuracy: 0.7135 - val_loss: 0.9560
Epoch 3/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 84ms/step - accuracy: 0.9762 - loss: 0.2871 - val_accuracy: 0.9781 - val_loss: 0.2618
Epoch 4/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 83ms/step - accuracy: 0.9770 - loss: 0.2646 - val_accuracy: 0.9901 - val_loss: 0.2349
Epoch 5/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 86ms/step - accuracy: 0.9793 - loss: 0.2639 - val_accuracy: 0.9892 - val_loss: 0.2526
Epoch 6/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 19s 83ms/step - accuracy: 0.9795 - loss: 0.2797 - val_accuracy: 0.9841 - val_loss: 0.2544
Epoch 7/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 84ms/step - accuracy: 0.9797 - loss: 0.2818 - val_accuracy: 0.9653 - val_loss: 0.3218
Epoch 8/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 85ms/step - accuracy: 0.9821 - loss: 0.2746 - val_accuracy: 0.9854 - val_loss: 0.2568
Epoch 9/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 83ms/step - accuracy: 0.9802 - loss: 0.2785 - val_accuracy: 0.9917 - val_loss: 0.2415
Epoch 10/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 85ms/step - accuracy: 0.9828 - loss: 0.2620 - val_accuracy: 0.9888 - val_loss: 0.2349
Epoch 11/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 84ms/step - accuracy: 0.9824 - loss: 0.2616 - val_accuracy: 0.9856 - val_loss: 0.2495
Epoch 12/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 19s 83ms/step - accuracy: 0.9832 - loss: 0.2604 - val_accuracy: 0.9911 - val_loss: 0.2061
Epoch 13/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 84ms/step - accuracy: 0.9844 - loss: 0.2344 - val_accuracy: 0.9858 - val_loss: 0.2343
Epoch 14/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 19s 83ms/step - accuracy: 0.9849 - loss: 0.2296 - val_accuracy: 0.9873 - val_loss: 0.2315
Epoch 15/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 82ms/step - accuracy: 0.9837 - loss: 0.2396 - val_accuracy: 0.9913 - val_loss: 0.2072
Epoch 16/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 85ms/step - accuracy: 0.9870 - loss: 0.2118 - val_accuracy: 0.9921 - val_loss: 0.2096
Epoch 17/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 19s 81ms/step - accuracy: 0.9860 - loss: 0.2191 - val_accuracy: 0.9896 - val_loss: 0.1987
Epoch 18/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 85ms/step - accuracy: 0.9869 - loss: 0.2020 - val_accuracy: 0.9902 - val_loss: 0.1906
Epoch 19/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 84ms/step - accuracy: 0.9874 - loss: 0.1936 - val_accuracy: 0.9910 - val_loss: 0.1768
Epoch 20/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 83ms/step - accuracy: 0.9884 - loss: 0.1852 - val_accuracy: 0.9901 - val_loss: 0.1793
Epoch 21/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 85ms/step - accuracy: 0.9884 - loss: 0.1803 - val_accuracy: 0.9891 - val_loss: 0.1720
Epoch 22/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 84ms/step - accuracy: 0.9883 - loss: 0.1769 - val_accuracy: 0.9932 - val_loss: 0.1501
```

```
Epoch 23/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 83ms/step - accuracy: 0.9902 - loss: 0.1565 - val_accuracy: 0.9931 - val_loss: 0.1471
Epoch 24/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 86ms/step - accuracy: 0.9906 - loss: 0.1478 - val_accuracy: 0.9919 - val_loss: 0.1342
Epoch 25/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 83ms/step - accuracy: 0.9899 - loss: 0.1412 - val_accuracy: 0.9933 - val_loss: 0.1274
Epoch 26/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 82ms/step - accuracy: 0.9913 - loss: 0.1302 - val_accuracy: 0.9947 - val_loss: 0.1160
Epoch 27/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 84ms/step - accuracy: 0.9916 - loss: 0.1217 - val_accuracy: 0.9928 - val_loss: 0.1120
Epoch 28/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 83ms/step - accuracy: 0.9924 - loss: 0.1115 - val_accuracy: 0.9939 - val_loss: 0.1020
Epoch 29/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 82ms/step - accuracy: 0.9929 - loss: 0.1038 - val_accuracy: 0.9916 - val_loss: 0.1014
Epoch 30/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 85ms/step - accuracy: 0.9919 - loss: 0.1003 - val_accuracy: 0.9942 - val_loss: 0.0874
Epoch 31/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 83ms/step - accuracy: 0.9932 - loss: 0.0888 - val_accuracy: 0.9953 - val_loss: 0.0795
Epoch 32/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 85ms/step - accuracy: 0.9933 - loss: 0.0818 - val_accuracy: 0.9948 - val_loss: 0.0752
Epoch 33/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 84ms/step - accuracy: 0.9954 - loss: 0.0734 - val_accuracy: 0.9952 - val_loss: 0.0676
Epoch 34/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 83ms/step - accuracy: 0.9952 - loss: 0.0665 - val_accuracy: 0.9954 - val_loss: 0.0615
Epoch 35/35
235/235 ━━━━━━━━━━━━━━━━━━━━ 20s 86ms/step - accuracy: 0.9958 - loss: 0.0600 - val_accuracy: 0.9953 - val_loss: 0.0585
Test Accuracy: 0.9953
```