Image Data Augmentation

This lab was adapted from https://www.manning.com/books/deep-learning-with-python, Chapter 5

Training a convnet from scratch on a small dataset

Having to train an image classification model using only very little data is a common situation, which you likely encounter yourself in practice if you ever do computer vision in a professional context.

Having "few" samples can mean anywhere from a few hundreds to a few tens of thousands of images. As a practical example, we will focus on classifying images as "dogs" or "cats", in a dataset containing 4000 pictures of cats and dogs (2000 cats, 2000 dogs). We will use 2000 pictures for training, 1000 for validation, and finally 1000 for testing.

In this section, we will review one basic strategy to tackle this problem: training a new model from scratch on what little data we have. We will start by naively training a small convnet on our 2000 training samples, without any regularization, to set a baseline for what can be achieved. This will get us to a classification accuracy of 71%. At that point, our main issue will be overfitting. Then we will introduce *data augmentation*, a powerful technique for mitigating overfitting in computer vision. By leveraging data augmentation, we will improve our network to reach an accuracy of 82%.

In the next section, we will review two more essential techniques for applying deep learning to small datasets: *doing feature extraction with a pre-trained network* (this will get us to an accuracy of 90% to 93%), and *fine-tuning a pre-trained network* (this will get us to our final accuracy of 95%). Together, these three strategies -- training a small model from scratch, doing feature extracting using a pre-trained model, and fine-tuning a pre-trained model -- will constitute your future toolbox for tackling the problem of doing computer vision with small datasets.

The relevance of deep learning for small-data problems

You will sometimes hear that deep learning only works when lots of data is available. This is in part a valid point: one fundamental characteristic of deep learning is that it is able to find interesting features in the training data on its own, without any need for manual feature engineering, and this can only be achieved when lots of training examples are available. This is especially true for problems where the input samples are very high-dimensional, like images.

However, what constitutes "lots" of samples is relative -- relative to the size and depth of the network you are trying to train, for starters. It isn't possible to train a convnet to solve a complex problem with just a few tens of samples, but a few hundreds can potentially suffice if the model is small and well-regularized and if the task is simple. Because convnets learn local, translation-invariant features, they are very data-efficient on perceptual problems. Training a convnet from scratch on a very small image dataset will still yield reasonable results despite a relative lack of data, without the need for any custom feature engineering. You will see this in action in this section.

But what's more, deep learning models are by nature highly repurposable: you can take, say, an image classification or speech-to-text model trained on a large-scale dataset then reuse it on a significantly different problem with only minor changes. Specifically, in the case of computer vision, many pre-trained models (usually trained on the ImageNet dataset) are now publicly available for download and can be used to bootstrap powerful vision models out of very little data. That's what we will do in the next section.

For now, let's get started by getting our hands on the data.

Downloading the data

The cats vs. dogs dataset that we will use isn't packaged with Keras. It was made available by Kaggle.com as part of a computer vision competition in late 2013, back when convnets weren't quite mainstream. You can download the original dataset at: `https://www.kaggle.com/c/dogs-vs-cats/data` (you will need to create a Kaggle account if you don't already have one -- don't worry, the process is painless).

The pictures are medium-resolution color JPEGs. They look like this:

Unsurprisingly, the cats vs. dogs Kaggle competition in 2013 was won by entrants who used convnets. The best entries could achieve up to 95% accuracy. In our own example, we will get fairly close to this accuracy (in the next section), even though we will be training our models on less than 10% of the data that was available to the competitors. This original dataset contains 25,000 images of dogs and cats (12,500 from each class) and is 543MB large (compressed). After downloading and uncompressing it, we will create a new dataset containing three subsets: a training set with 1000 samples of each class, a validation set with 500 samples of each class, and finally a test set with 500 samples of each class.

Here are a few lines of code to do this:

Dependencies

In [2]:
```
! pip install torchinfo
```

```
Collecting torchinfo
  Downloading torchinfo-1.6.5-py3-none-any.whl (21 kB)
Installing collected packages: torchinfo
Successfully installed torchinfo-1.6.5
```

In [3]:
```
import os
import shutil
import glob
import random
import concurrent
import zipfile
import torch
import PIL
import numpy as np
import torch.nn as nn
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader, random_split, Dataset
from IPython.display import Image
from torchinfo import summary
```

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

Download data & unzip:

Download the dataset from https://www.kaggle.com/c/dogs-vs-cats/data.

Zip it and move it to your Google drive.

In [4]:
```
import os
from google.colab import drive

drive.mount('/content/drive') # mount the drive
```

```
Mounted at /content/drive
```

Unzip the file and save to a folder in your google drive.

Note, change the path to where you leave the zip file in the lines below

In [ ]:
```
cwd = os.getcwd()
print(cwd)
os.chdir('/content/drive/MyDrive/BIA667_Lab')

cwd = os.getcwd()

print(cwd)
```

```
/content/drive/MyDrive/BIA667_Lab
/content/drive/MyDrive/BIA667_Lab
```

In [ ]:
```
# Unzip the files to your google drive

with zipfile.ZipFile(os.path.join(cwd, 'dog_cats.zip'), 'r') as zip_ref:
    zip_ref.extractall('./')
```

Re-organize data & generate small sample:

Dir structure:

```
In [ ]: # cwd
cwd = os.getcwd()

# dirs
dog_cat_dir = os.path.join(cwd, 'dog_cat')
if not os.path.exists(dog_cat_dir):
    os.mkdir(dog_cat_dir)

cat_dir = os.path.join(dog_cat_dir, 'cat')
if not os.path.exists(cat_dir):
    os.mkdir(cat_dir)

dog_dir = os.path.join(dog_cat_dir, 'dog')
if not os.path.exists(dog_dir):
    os.mkdir(dog_dir)

# sample random 1000 pics for cats & dogs
cat_pics = np.random.choice(glob.glob(os.path.join(cwd, 'train', 'cat.*.jpg')), size=1000, repla
dog_pics = np.random.choice(glob.glob(os.path.join(cwd, 'train', 'dog.*.jpg')), size=1000, repla

# move to dir
def move_to_folder(scr_file, des_folder):
    des_name = os.path.basename(scr_file)
    shutil.copyfile(scr_file, os.path.join(des_folder, des_name))
# cats
with concurrent.futures.ThreadPoolExecutor(max_workers=4) as pool:
    [pool.submit(move_to_folder, file_name, cat_dir) for file_name in cat_pics]
# dogs
with concurrent.futures.ThreadPoolExecutor(max_workers=4) as pool:
    [pool.submit(move_to_folder, file_name, dog_dir) for file_name in dog_pics]
```

```
In [ ]: print(os.listdir(cat_dir)[:10])
print(os.listdir(dog_dir)[:10])
```

```
['cat.9364.jpg', 'cat.6242.jpg', 'cat.5122.jpg', 'cat.1095.jpg', 'cat.4210.jpg', 'cat.9023.jpg',
'cat.9280.jpg', 'cat.2765.jpg', 'cat.7621.jpg', 'cat.1231.jpg']
['dog.11506.jpg', 'dog.4292.jpg', 'dog.9579.jpg', 'dog.2306.jpg', 'dog.9640.jpg', 'dog.8689.jp
g', 'dog.4635.jpg', 'dog.8913.jpg', 'dog.12194.jpg', 'dog.4416.jpg']
```

Show a cat and dog example

```
In [ ]: Image(os.path.join(cat_dir, os.listdir(cat_dir)[0]))
```

Out[ ]:

In [ ]:
```
Image(os.path.join(dog_dir, os.listdir(dog_dir)[0]))
```

Out[ ]:

Define Model

In [ ]:
```
class CNN_Classifier(nn.Module):
    def __init__(self):
        super(CNN_Classifier, self).__init__()
        # Conv net
        self.convnet = nn.Sequential(
            # input(3, 150, 150)
            nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3), # (32, 148, 148)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # (32, 74, 74)
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3),  # (64, 72, 72)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # (64, 36, 36)
            nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3),  # (128, 34, 34)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # (128, 17, 17)
            nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3),  # (128, 15, 15)
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),  # (128, 7, 7)
            nn.Flatten() # 6272 = 128 * 7 * 7
        )
        # classifier
        self.classifier = nn.Sequential(
            nn.Linear(in_features=6272, out_features=512),
            nn.ReLU(),
            nn.Linear(in_features=512, out_features=1)
        )

    def forward(self, x):
        x = self.convnet(x)
        x = self.classifier(x)

        return x
```

In [ ]:

```
sample_model = CNN_Classifier()
summary(sample_model, (10, 3, 150, 150))
```

Out[ ]:

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNN_Classifier                           --                        --
├─Sequential: 1-1                        [10, 6272]                --
│    └─Conv2d: 2-1                       [10, 32, 148, 148]        896
│    └─ReLU: 2-2                         [10, 32, 148, 148]        --
│    └─MaxPool2d: 2-3                    [10, 32, 74, 74]          --
│    └─Conv2d: 2-4                       [10, 64, 72, 72]          18,496
│    └─ReLU: 2-5                         [10, 64, 72, 72]          --
│    └─MaxPool2d: 2-6                    [10, 64, 36, 36]          --
│    └─Conv2d: 2-7                       [10, 128, 34, 34]         73,856
│    └─ReLU: 2-8                         [10, 128, 34, 34]         --
│    └─MaxPool2d: 2-9                    [10, 128, 17, 17]         --
│    └─Conv2d: 2-10                      [10, 128, 15, 15]         147,584
│    └─ReLU: 2-11                        [10, 128, 15, 15]         --
│    └─MaxPool2d: 2-12                   [10, 128, 7, 7]           --
│    └─Flatten: 2-13                     [10, 6272]                --
├─Sequential: 1-2                        [10, 1]                   --
│    └─Linear: 2-14                      [10, 512]                 3,211,776
│    └─ReLU: 2-15                        [10, 512]                 --
│    └─Linear: 2-16                      [10, 1]                   513
==========================================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
Total mult-adds (G): 2.37
==========================================================================================
Input size (MB): 2.70
Forward/backward pass size (MB): 96.80
Params size (MB): 13.81
Estimated Total Size (MB): 113.31
==========================================================================================
```

Dataset

Self-defined Dataset:

In [ ]:
```
class ImgDataset(Dataset):
    def __init__(self, img_path, img_labels, img_transforms=None):
        self.img_path = img_path
        self.img_labels = torch.Tensor(img_labels)
        if img_transforms is None:
            self.transforms = transforms.ToTensor()
        else:
            self.transforms = img_transforms

    def __getitem__(self, index):
        # Load image
        cur_path = self.img_path[index]
        cur_img = PIL.Image.open(cur_path)
        cur_img = self.transforms(cur_img)

        return cur_img, self.img_labels[index]

    def __len__(self):
        return len(self.img_path)
```

In [ ]:
```
images_list = glob.glob(os.path.join(dog_cat_dir, '*', '*.jpg'))
# label: 0 for cat, 1 for dog
def extract_class(img_path):
    base_path = os.path.basename(img_path)
    return base_path.split('.')[0]

labels = [0 if extract_class(cur_path) == 'cat' else 1 for cur_path in images_list]
```

In [ ]:
```
resize_transforms = transforms.Compose([transforms.Resize((150, 150)), transforms.ToTensor()])
```

In [ ]:
```
dog_cat_dataset = ImgDataset(img_path=images_list, img_labels=labels, img_transforms=resize_tran
```

In [ ]:
```
print(dog_cat_dataset[0])

dog_cat_dataset[0][0].size()
```

```
(tensor([[[0.3373, 0.3020, 0.3373,  ..., 0.2667, 0.2510, 0.2549],
         [0.3098, 0.2941, 0.2980,  ..., 0.2824, 0.2784, 0.2745],
         [0.3020, 0.3059, 0.2824,  ..., 0.2627, 0.2667, 0.2549],
         ...,
         [0.2941, 0.2667, 0.3059,  ..., 0.2039, 0.2000, 0.1961],
         [0.3137, 0.2941, 0.2627,  ..., 0.2000, 0.1961, 0.1961],
         [0.2824, 0.3059, 0.2667,  ..., 0.2000, 0.1961, 0.1961]],

        [[0.3098, 0.2745, 0.3098,  ..., 0.2549, 0.2471, 0.2510],
         [0.2824, 0.2667, 0.2706,  ..., 0.2667, 0.2745, 0.2706],
         [0.2745, 0.2784, 0.2549,  ..., 0.2510, 0.2627, 0.2510],
         ...,
         [0.2706, 0.2431, 0.2824,  ..., 0.1804, 0.1804, 0.1765],
         [0.2902, 0.2706, 0.2392,  ..., 0.1725, 0.1765, 0.1765],
         [0.2588, 0.2824, 0.2431,  ..., 0.1765, 0.1765, 0.1765]],

        [[0.3412, 0.3059, 0.3412,  ..., 0.2784, 0.2706, 0.2745],
         [0.3137, 0.2980, 0.3020,  ..., 0.2941, 0.2980, 0.2941],
         [0.3059, 0.3098, 0.2863,  ..., 0.2784, 0.2863, 0.2745],
         ...,
         [0.2784, 0.2510, 0.2902,  ..., 0.1686, 0.1686, 0.1647],
         [0.2980, 0.2784, 0.2471,  ..., 0.1647, 0.1647, 0.1647],
         [0.2667, 0.2902, 0.2510,  ..., 0.1647, 0.1647, 0.1647]]]), tensor(0.))
```

```
Out[ ]: torch.Size([3, 150, 150])
```

ImageFolder:

```
In [ ]: from torchvision.datasets import ImageFolder
        dog_cat_dataset_folder = ImageFolder(dog_cat_dir, transform=transforms.ToTensor())
```

```
In [ ]: dog_cat_dataset_folder[0]
```

```
Out[ ]: (tensor([[[0.6000, 0.6039, 0.6118,  ..., 0.2392, 0.2549, 0.2588],
          [0.6000, 0.6039, 0.6078,  ..., 0.2471, 0.2627, 0.2706],
          [0.6078, 0.6118, 0.6118,  ..., 0.2549, 0.2745, 0.2863],
          ...,
          [0.7961, 0.7961, 0.7961,  ..., 0.8078, 0.8078, 0.8078],
          [0.8000, 0.8039, 0.8039,  ..., 0.8118, 0.8118, 0.8118],
          [0.8078, 0.8078, 0.8078,  ..., 0.8118, 0.8118, 0.8118]],

         [[0.5843, 0.5882, 0.5961,  ..., 0.1882, 0.2039, 0.2078],
          [0.5843, 0.5882, 0.5922,  ..., 0.1961, 0.2118, 0.2196],
          [0.5922, 0.5961, 0.5961,  ..., 0.2039, 0.2235, 0.2353],
          ...,
          [0.7333, 0.7333, 0.7333,  ..., 0.7294, 0.7294, 0.7294],
          [0.7373, 0.7412, 0.7412,  ..., 0.7333, 0.7333, 0.7333],
          [0.7451, 0.7451, 0.7451,  ..., 0.7333, 0.7333, 0.7333]],

         [[0.5373, 0.5412, 0.5490,  ..., 0.1529, 0.1686, 0.1725],
          [0.5373, 0.5412, 0.5451,  ..., 0.1608, 0.1765, 0.1843],
          [0.5451, 0.5490, 0.5490,  ..., 0.1686, 0.1882, 0.2000],
          ...,
          [0.6039, 0.6039, 0.6039,  ..., 0.5922, 0.5922, 0.5922],
          [0.6078, 0.6118, 0.6118,  ..., 0.5961, 0.5961, 0.5961],
          [0.6157, 0.6157, 0.6157,  ..., 0.5961, 0.5961, 0.5961]]]), 0)
```

Split Dataset

```
In [ ]: split_size = (np.array([0.6, 0.2, 0.2]) * len(dog_cat_dataset)).round().astype(np.int)
```

```
train_data, valid_data, test_data = random_split(dog_cat_dataset, split_size)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing th
is will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `
np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check
the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.
0-notes.html#deprecations
  """Entry point for launching an IPython kernel.
```

Train Function

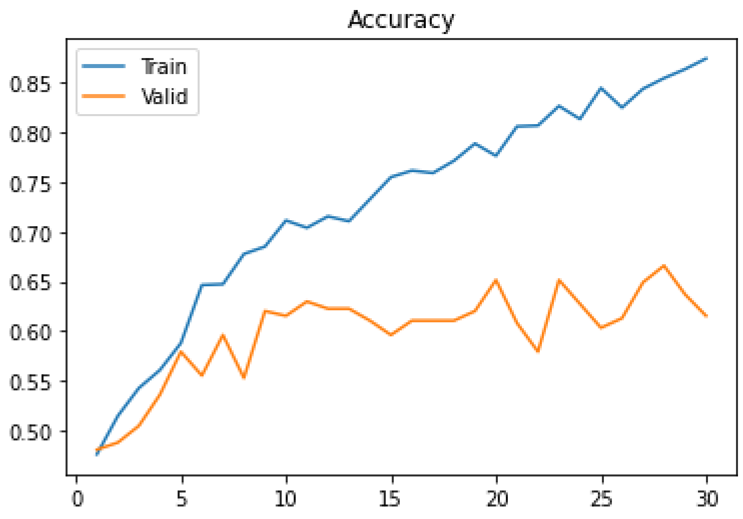
In [ ]:
```
def train_model(model, train_dataset, test_dataset, device,
                lr=0.0001, epochs=30, batch_size=256):

    # construct dataloader
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    # move model to device
    model = model.to(device)

    # history
    history = {'train_loss': [],
               'train_acc': [],
               'test_loss': [],
               'test_acc': []}
    # setup loss function and optimizer
    criterion = nn.BCEWithLogitsLoss()
    optimizer = torch.optim.RMSprop(model.parameters(), lr=lr)
    # training loop
    print('Training Start')
    for epoch in range(epochs):
        model.train()
        train_loss = 0
        train_acc = 0
        test_loss = 0
        test_acc = 0

        for x, y in train_loader:
            # move data to device
            x = x.to(device)
            y = y.to(device)
            # forward
            outputs = model(x).view(-1)  # (num_batch)
            cur_train_loss = criterion(outputs, y)
            pred = torch.sigmoid(outputs)
            pred = torch.round(pred)
            cur_train_acc = (pred == y).sum().item() / batch_size
            # backward
            cur_train_loss.backward()
            optimizer.step()
            optimizer.zero_grad()
            # loss and acc
            train_loss += cur_train_loss
            train_acc += cur_train_acc

        # test start
        model.eval()
        with torch.no_grad():
            for x, y in test_loader:
                # move
                x = x.to(device)
                y = y.to(device)
                # predict
                outputs = model(x).view(-1)
                pred = torch.round(torch.sigmoid(outputs))
                cur_test_loss = criterion(outputs, y)
                cur_test_acc = (pred == y).sum().item() / batch_size
                # loss and acc
                test_loss += cur_test_loss
                test_acc += cur_test_acc

        # epoch output
        train_loss = (train_loss/len(train_loader)).item()
        train_acc = train_acc/len(train_loader)
        val_loss = (test_loss/len(test_loader)).item()
        val_acc = test_acc/len(test_loader)
        history['train_loss'].append(train_loss)
        history['train_acc'].append(train_acc)
        history['test_loss'].append(val_loss)
        history['test_acc'].append(val_acc)
        print(f"Epoch:{epoch + 1} / {epochs}, train loss:{train_loss:.4f} train_acc:{train_acc:.

    return history
```

Train Model

In [ ]:
```
cnn_model = CNN_Classifier()
```

In [ ]:
```
history = train_model(cnn_model, train_data, valid_data, device, batch_size=32, epochs=30, lr=0.
Training Start
Epoch:1 / 30, train loss:0.6971 train_acc:0.4762, valid loss:0.6925 valid acc:0.4808
Epoch:2 / 30, train loss:0.6916 train_acc:0.5148, valid loss:0.6954 valid acc:0.4880
Epoch:3 / 30, train loss:0.6855 train_acc:0.5428, valid loss:0.6842 valid acc:0.5048
Epoch:4 / 30, train loss:0.6743 train_acc:0.5609, valid loss:0.6750 valid acc:0.5361
Epoch:5 / 30, train loss:0.6538 train_acc:0.5880, valid loss:0.6595 valid acc:0.5793
Epoch:6 / 30, train loss:0.6339 train_acc:0.6464, valid loss:0.6583 valid acc:0.5553
Epoch:7 / 30, train loss:0.6201 train_acc:0.6472, valid loss:0.6569 valid acc:0.5962
Epoch:8 / 30, train loss:0.6021 train_acc:0.6776, valid loss:0.6788 valid acc:0.5529
Epoch:9 / 30, train loss:0.5895 train_acc:0.6850, valid loss:0.6352 valid acc:0.6202
Epoch:10 / 30, train loss:0.5551 train_acc:0.7113, valid loss:0.6458 valid acc:0.6154
Epoch:11 / 30, train loss:0.5611 train_acc:0.7039, valid loss:0.6317 valid acc:0.6298
Epoch:12 / 30, train loss:0.5377 train_acc:0.7155, valid loss:0.6622 valid acc:0.6226
Epoch:13 / 30, train loss:0.5308 train_acc:0.7105, valid loss:0.6356 valid acc:0.6226
Epoch:14 / 30, train loss:0.5158 train_acc:0.7327, valid loss:0.6682 valid acc:0.6106
Epoch:15 / 30, train loss:0.4915 train_acc:0.7549, valid loss:0.7853 valid acc:0.5962
Epoch:16 / 30, train loss:0.4805 train_acc:0.7615, valid loss:0.6594 valid acc:0.6106
Epoch:17 / 30, train loss:0.4691 train_acc:0.7590, valid loss:0.7324 valid acc:0.6106
Epoch:18 / 30, train loss:0.4642 train_acc:0.7714, valid loss:0.6570 valid acc:0.6106
Epoch:19 / 30, train loss:0.4384 train_acc:0.7887, valid loss:0.7471 valid acc:0.6202
Epoch:20 / 30, train loss:0.4483 train_acc:0.7763, valid loss:0.6369 valid acc:0.6514
Epoch:21 / 30, train loss:0.4143 train_acc:0.8059, valid loss:0.6941 valid acc:0.6082
Epoch:22 / 30, train loss:0.4016 train_acc:0.8067, valid loss:0.8481 valid acc:0.5793
Epoch:23 / 30, train loss:0.3790 train_acc:0.8265, valid loss:0.6736 valid acc:0.6514
Epoch:24 / 30, train loss:0.3832 train_acc:0.8133, valid loss:0.7256 valid acc:0.6274
Epoch:25 / 30, train loss:0.3476 train_acc:0.8446, valid loss:0.8696 valid acc:0.6034
Epoch:26 / 30, train loss:0.3609 train_acc:0.8248, valid loss:0.7353 valid acc:0.6130
Epoch:27 / 30, train loss:0.3407 train_acc:0.8438, valid loss:0.6994 valid acc:0.6490
Epoch:28 / 30, train loss:0.3250 train_acc:0.8544, valid loss:0.6931 valid acc:0.6659
Epoch:29 / 30, train loss:0.2963 train_acc:0.8635, valid loss:0.7336 valid acc:0.6370
Epoch:30 / 30, train loss:0.2784 train_acc:0.8742, valid loss:0.8052 valid acc:0.6154
```

In [ ]:
```
plt.plot(range(1, 31), history['train_acc'], label='Train')
plt.plot(range(1, 31), history['test_acc'], label='Valid')
plt.title('Accuracy')
plt.legend()
plt.show()
```



In [ ]:
```
plt.plot(range(1, 31), history['train_loss'], label='Train')
plt.plot(range(1, 31), history['test_loss'], label='Valid')
plt.title('Loss')
plt.legend()
plt.show()
```

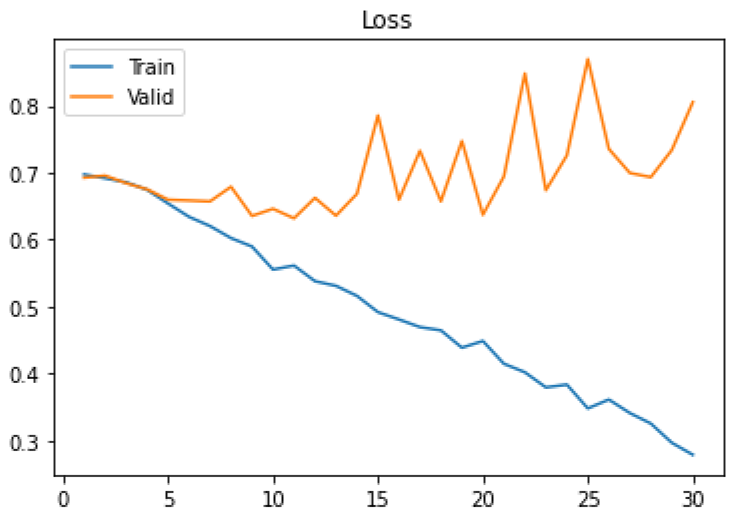


Image Data Augmentation

The image transformations are implemented in torchvision. Some useful data augmentations:

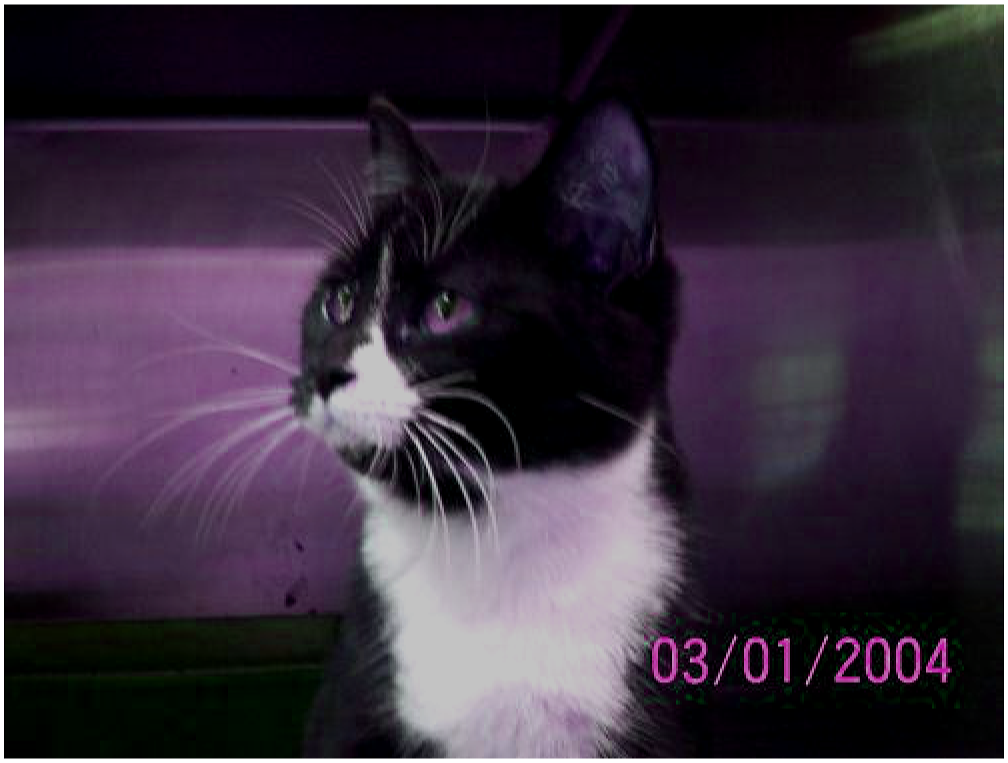
- torchvision.transforms.ColorJitter: Randomly change the brightness, contrast, saturation and hue of an image.
- torchvision.transforms.RandomHorizontalFlip & torchvision.transforms.RandomVerticalFlip: Horizontally or vertically flip the given image randomly with a given probability.
- torchvision.transforms.RandomGrayscale(p=0.1): Randomly convert image to grayscale with a a probability.
- RandomRotation: Randomly rotate images within certain degrees.

For the full list of availabe tranformations, please see this. The illustrations of transformations are available here.

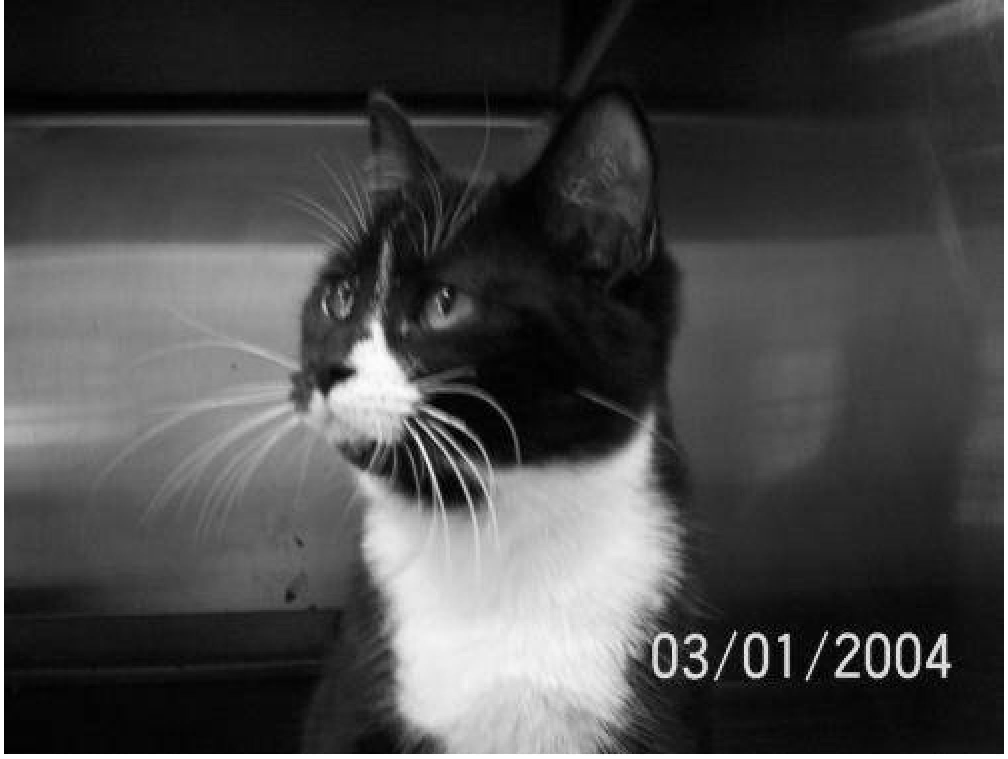
Image Transformation Example

```
In [ ]: # get a sample image
sample_img_path = os.path.join(cat_dir, os.listdir(cat_dir)[4])
sample_img = PIL.Image.open(sample_img_path)
display(sample_img)
```

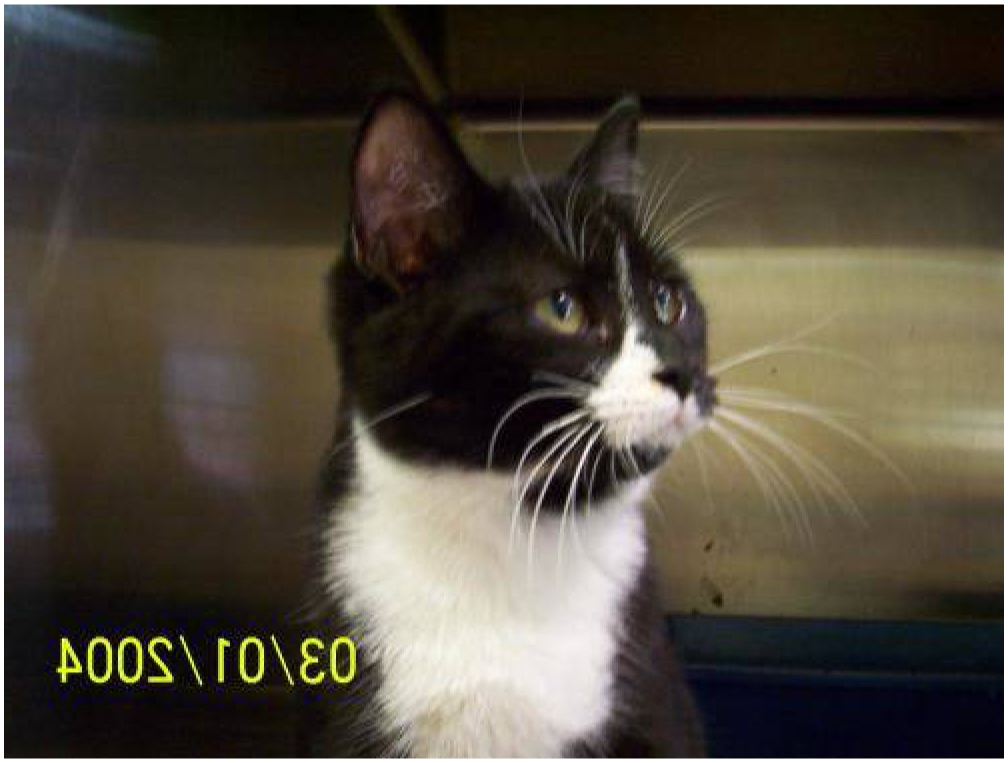
```
In [ ]: # apply random color jitter
        transformed_img = transforms.ColorJitter(brightness=0.5, contrast=0.5, saturation=0.5, hue=0.5)(s
        display(transformed_img)
```



```
In [ ]: transformed_img = transforms.RandomGrayscale(p=1.0)(sample_img)
        display(transformed_img)
```



```
In [ ]: transformed_img = transforms.RandomHorizontalFlip(p=1.0)(sample_img)
        display(transformed_img)
```



```
In [ ]: # Chain the transformations to a pipeline
        transform_pipeline = transforms.Compose(
            [transforms.RandomGrayscale(p=1.0),
             transforms.RandomHorizontalFlip(p=1.0)])

        transformed_img = transform_pipeline(sample_img)
        display(transformed_img)
```

Apply Transfoamation on Dataset

In [ ]:
```python
class ImgDataset(Dataset):
    def __init__(self, img_path, img_labels, img_transforms=None):
        self.img_path = img_path
        self.img_labels = torch.Tensor(img_labels)
        if img_transforms is None:
            self.transforms = transforms.ToTensor()
        else:
            self.transforms = img_transforms

    def __getitem__(self, index):
        # Load image
        cur_path = self.img_path[index]
        cur_img = PIL.Image.open(cur_path)
        cur_img = self.transforms(cur_img)

        return cur_img, self.img_labels[index]

    def __len__(self):
        return len(self.img_path)
```

Build a transformation pipeline:

In [ ]:
```python
transformations = transforms.Compose([transforms.Resize((150, 150)),  # resize to input shape of
                                      transforms.ColorJitter(brightness=0.3, contrast=0.3, satur
                                      transforms.RandomRotation(40),
                                      transforms.RandomAffine(degrees=0, scale=(0.8, 1.2), shear
                                      transforms.RandomHorizontalFlip(p=0.5),
                                      transforms.ToTensor()  # convert PIL to Tensor
                                      ])
```

New dataset with transformation pipline:

In [ ]:
```python
dog_cat_dataset_transformed = ImgDataset(img_path=images_list, img_labels=labels, img_transforms
split_size = (np.array([0.6, 0.2, 0.2]) * len(dog_cat_dataset_transformed)).round().astype(np.in
train_data, valid_data, test_data = random_split(dog_cat_dataset_transformed, split_size)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DeprecationWarning: `np.int` is
a deprecated alias for the builtin `int`. To silence this warning, use `int` by itself. Doing th
is will not modify any behavior and is safe. When replacing `np.int`, you may wish to use e.g. `
np.int64` or `np.int32` to specify the precision. If you wish to review your current use, check
the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.
0-notes.html#deprecations
  This is separate from the ipykernel package so we can avoid doing imports until
```

Re-Train CNN Model:

```
In [ ]: cnn_model = CNN_Classifier()
```

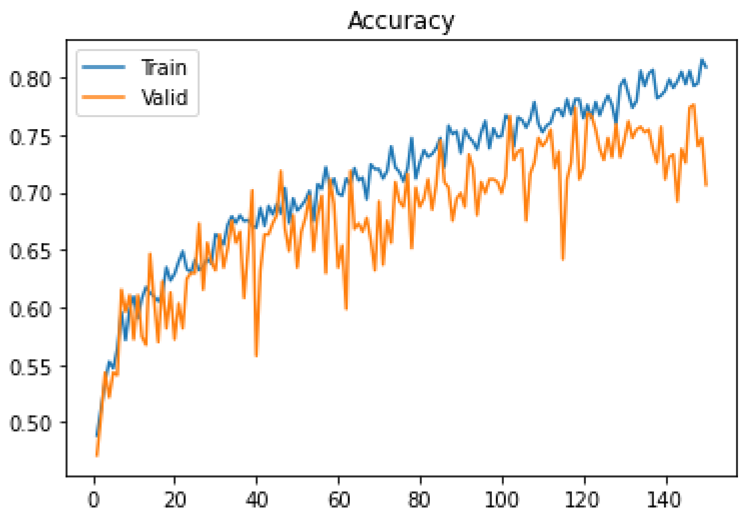
```
In [ ]: history = train_model(cnn_model, train_data, valid_data, device, batch_size=32, epochs=150, lr=0
Training Start
Epoch:1 / 150, train loss:0.6941 train_acc:0.4885, valid loss:0.6933 valid acc:0.4712
Epoch:2 / 150, train loss:0.6931 train_acc:0.5115, valid loss:0.6883 valid acc:0.5048
Epoch:3 / 150, train loss:0.6907 train_acc:0.5345, valid loss:0.6817 valid acc:0.5433
Epoch:4 / 150, train loss:0.6813 train_acc:0.5526, valid loss:0.6801 valid acc:0.5216
Epoch:5 / 150, train loss:0.6821 train_acc:0.5469, valid loss:0.6699 valid acc:0.5433
Epoch:6 / 150, train loss:0.6757 train_acc:0.5641, valid loss:0.6637 valid acc:0.5409
Epoch:7 / 150, train loss:0.6664 train_acc:0.5962, valid loss:0.6487 valid acc:0.6154
Epoch:8 / 150, train loss:0.6649 train_acc:0.5715, valid loss:0.6444 valid acc:0.5962
Epoch:9 / 150, train loss:0.6587 train_acc:0.5987, valid loss:0.6393 valid acc:0.6106
Epoch:10 / 150, train loss:0.6556 train_acc:0.6094, valid loss:0.6429 valid acc:0.5721
Epoch:11 / 150, train loss:0.6609 train_acc:0.5905, valid loss:0.6475 valid acc:0.6106
Epoch:12 / 150, train loss:0.6572 train_acc:0.6077, valid loss:0.6456 valid acc:0.5745
Epoch:13 / 150, train loss:0.6533 train_acc:0.6176, valid loss:0.6604 valid acc:0.5673
Epoch:14 / 150, train loss:0.6501 train_acc:0.6127, valid loss:0.6219 valid acc:0.6466
Epoch:15 / 150, train loss:0.6445 train_acc:0.6094, valid loss:0.6477 valid acc:0.6058
Epoch:16 / 150, train loss:0.6542 train_acc:0.6053, valid loss:0.6650 valid acc:0.5697
Epoch:17 / 150, train loss:0.6502 train_acc:0.6118, valid loss:0.6340 valid acc:0.6226
Epoch:18 / 150, train loss:0.6340 train_acc:0.6349, valid loss:0.6402 valid acc:0.5817
Epoch:19 / 150, train loss:0.6371 train_acc:0.6234, valid loss:0.6324 valid acc:0.6130
Epoch:20 / 150, train loss:0.6354 train_acc:0.6291, valid loss:0.6704 valid acc:0.5721
Epoch:21 / 150, train loss:0.6389 train_acc:0.6398, valid loss:0.6559 valid acc:0.6034
Epoch:22 / 150, train loss:0.6239 train_acc:0.6488, valid loss:0.6274 valid acc:0.5817
Epoch:23 / 150, train loss:0.6207 train_acc:0.6332, valid loss:0.6195 valid acc:0.6250
Epoch:24 / 150, train loss:0.6259 train_acc:0.6308, valid loss:0.6293 valid acc:0.6298
Epoch:25 / 150, train loss:0.6275 train_acc:0.6414, valid loss:0.6132 valid acc:0.6298
Epoch:26 / 150, train loss:0.6253 train_acc:0.6324, valid loss:0.6147 valid acc:0.6731
Epoch:27 / 150, train loss:0.6113 train_acc:0.6357, valid loss:0.6207 valid acc:0.6154
Epoch:28 / 150, train loss:0.6183 train_acc:0.6423, valid loss:0.5943 valid acc:0.6562
Epoch:29 / 150, train loss:0.6144 train_acc:0.6382, valid loss:0.6068 valid acc:0.6394
Epoch:30 / 150, train loss:0.6096 train_acc:0.6637, valid loss:0.6098 valid acc:0.6322
Epoch:31 / 150, train loss:0.6080 train_acc:0.6604, valid loss:0.6154 valid acc:0.6635
Epoch:32 / 150, train loss:0.6153 train_acc:0.6546, valid loss:0.6293 valid acc:0.6346
Epoch:33 / 150, train loss:0.6032 train_acc:0.6719, valid loss:0.6055 valid acc:0.6514
Epoch:34 / 150, train loss:0.5929 train_acc:0.6793, valid loss:0.5719 valid acc:0.6755
Epoch:35 / 150, train loss:0.6015 train_acc:0.6735, valid loss:0.5976 valid acc:0.6562
Epoch:36 / 150, train loss:0.5850 train_acc:0.6801, valid loss:0.5870 valid acc:0.6659
Epoch:37 / 150, train loss:0.5962 train_acc:0.6752, valid loss:0.6807 valid acc:0.6082
Epoch:38 / 150, train loss:0.5789 train_acc:0.6760, valid loss:0.5941 valid acc:0.6514
Epoch:39 / 150, train loss:0.5843 train_acc:0.6711, valid loss:0.5671 valid acc:0.7019
Epoch:40 / 150, train loss:0.5901 train_acc:0.6694, valid loss:0.7096 valid acc:0.5577
Epoch:41 / 150, train loss:0.5947 train_acc:0.6867, valid loss:0.5946 valid acc:0.6322
Epoch:42 / 150, train loss:0.5828 train_acc:0.6711, valid loss:0.5741 valid acc:0.6635
Epoch:43 / 150, train loss:0.5719 train_acc:0.6883, valid loss:0.5719 valid acc:0.6635
Epoch:44 / 150, train loss:0.5814 train_acc:0.6809, valid loss:0.5737 valid acc:0.6731
Epoch:45 / 150, train loss:0.5750 train_acc:0.6900, valid loss:0.5751 valid acc:0.6803
Epoch:46 / 150, train loss:0.5777 train_acc:0.6817, valid loss:0.5554 valid acc:0.7188
Epoch:47 / 150, train loss:0.5692 train_acc:0.7039, valid loss:0.5829 valid acc:0.6683
Epoch:48 / 150, train loss:0.5814 train_acc:0.6735, valid loss:0.5734 valid acc:0.6490
Epoch:49 / 150, train loss:0.5716 train_acc:0.6949, valid loss:0.5883 valid acc:0.6803
Epoch:50 / 150, train loss:0.5761 train_acc:0.6842, valid loss:0.6127 valid acc:0.6346
Epoch:51 / 150, train loss:0.5701 train_acc:0.6883, valid loss:0.5633 valid acc:0.6659
Epoch:52 / 150, train loss:0.5810 train_acc:0.6933, valid loss:0.5549 valid acc:0.6779
Epoch:53 / 150, train loss:0.5660 train_acc:0.7015, valid loss:0.5804 valid acc:0.6971
Epoch:54 / 150, train loss:0.5812 train_acc:0.6752, valid loss:0.6145 valid acc:0.6490
Epoch:55 / 150, train loss:0.5701 train_acc:0.7072, valid loss:0.5526 valid acc:0.6803
Epoch:56 / 150, train loss:0.5508 train_acc:0.7031, valid loss:0.5898 valid acc:0.6971
Epoch:57 / 150, train loss:0.5531 train_acc:0.7220, valid loss:0.6268 valid acc:0.6298
Epoch:58 / 150, train loss:0.5641 train_acc:0.7039, valid loss:0.5464 valid acc:0.7115
Epoch:59 / 150, train loss:0.5473 train_acc:0.7122, valid loss:0.5623 valid acc:0.6899
Epoch:60 / 150, train loss:0.5695 train_acc:0.6990, valid loss:0.5941 valid acc:0.6346
Epoch:61 / 150, train loss:0.5530 train_acc:0.6974, valid loss:0.5869 valid acc:0.6538
```

```
Epoch:62 / 150, train loss:0.5497 train_acc:0.7122, valid loss:0.6296 valid acc:0.5986
Epoch:63 / 150, train loss:0.5472 train_acc:0.7072, valid loss:0.5270 valid acc:0.7188
Epoch:64 / 150, train loss:0.5430 train_acc:0.7212, valid loss:0.5632 valid acc:0.6683
Epoch:65 / 150, train loss:0.5489 train_acc:0.7105, valid loss:0.5752 valid acc:0.6731
Epoch:66 / 150, train loss:0.5486 train_acc:0.7130, valid loss:0.6006 valid acc:0.6659
Epoch:67 / 150, train loss:0.5440 train_acc:0.6941, valid loss:0.5611 valid acc:0.6779
Epoch:68 / 150, train loss:0.5490 train_acc:0.7245, valid loss:0.5791 valid acc:0.6587
Epoch:69 / 150, train loss:0.5488 train_acc:0.7204, valid loss:0.6110 valid acc:0.6322
Epoch:70 / 150, train loss:0.5397 train_acc:0.7204, valid loss:0.5636 valid acc:0.6923
Epoch:71 / 150, train loss:0.5456 train_acc:0.7122, valid loss:0.6409 valid acc:0.6370
Epoch:72 / 150, train loss:0.5356 train_acc:0.7188, valid loss:0.5646 valid acc:0.6755
Epoch:73 / 150, train loss:0.5259 train_acc:0.7401, valid loss:0.6002 valid acc:0.6562
Epoch:74 / 150, train loss:0.5227 train_acc:0.7220, valid loss:0.5439 valid acc:0.7091
Epoch:75 / 150, train loss:0.5423 train_acc:0.7179, valid loss:0.5935 valid acc:0.6923
Epoch:76 / 150, train loss:0.5361 train_acc:0.7097, valid loss:0.5704 valid acc:0.6875
Epoch:77 / 150, train loss:0.5293 train_acc:0.7212, valid loss:0.5297 valid acc:0.7163
Epoch:78 / 150, train loss:0.5126 train_acc:0.7475, valid loss:0.5699 valid acc:0.6514
Epoch:79 / 150, train loss:0.5326 train_acc:0.7122, valid loss:0.5423 valid acc:0.7043
Epoch:80 / 150, train loss:0.5241 train_acc:0.7262, valid loss:0.5295 valid acc:0.6875
Epoch:81 / 150, train loss:0.5322 train_acc:0.7368, valid loss:0.5320 valid acc:0.6947
Epoch:82 / 150, train loss:0.5254 train_acc:0.7311, valid loss:0.5532 valid acc:0.7115
Epoch:83 / 150, train loss:0.5190 train_acc:0.7336, valid loss:0.5546 valid acc:0.6851
Epoch:84 / 150, train loss:0.5079 train_acc:0.7385, valid loss:0.5349 valid acc:0.7067
Epoch:85 / 150, train loss:0.5092 train_acc:0.7475, valid loss:0.5067 valid acc:0.7452
Epoch:86 / 150, train loss:0.5202 train_acc:0.7220, valid loss:0.5358 valid acc:0.7091
Epoch:87 / 150, train loss:0.4918 train_acc:0.7582, valid loss:0.5552 valid acc:0.7043
Epoch:88 / 150, train loss:0.4953 train_acc:0.7508, valid loss:0.6091 valid acc:0.6755
Epoch:89 / 150, train loss:0.5033 train_acc:0.7533, valid loss:0.5764 valid acc:0.6947
Epoch:90 / 150, train loss:0.5194 train_acc:0.7344, valid loss:0.5543 valid acc:0.6995
Epoch:91 / 150, train loss:0.4910 train_acc:0.7549, valid loss:0.5430 valid acc:0.6875
Epoch:92 / 150, train loss:0.4955 train_acc:0.7484, valid loss:0.5011 valid acc:0.7332
Epoch:93 / 150, train loss:0.5054 train_acc:0.7442, valid loss:0.5117 valid acc:0.7212
Epoch:94 / 150, train loss:0.4901 train_acc:0.7377, valid loss:0.5878 valid acc:0.6803
Epoch:95 / 150, train loss:0.4996 train_acc:0.7525, valid loss:0.5356 valid acc:0.7091
Epoch:96 / 150, train loss:0.4846 train_acc:0.7623, valid loss:0.5475 valid acc:0.6995
Epoch:97 / 150, train loss:0.4961 train_acc:0.7385, valid loss:0.5582 valid acc:0.7115
Epoch:98 / 150, train loss:0.4906 train_acc:0.7558, valid loss:0.5498 valid acc:0.7115
Epoch:99 / 150, train loss:0.4774 train_acc:0.7484, valid loss:0.5321 valid acc:0.7091
Epoch:100 / 150, train loss:0.4908 train_acc:0.7492, valid loss:0.5515 valid acc:0.6995
Epoch:101 / 150, train loss:0.4777 train_acc:0.7673, valid loss:0.5284 valid acc:0.7139
Epoch:102 / 150, train loss:0.4745 train_acc:0.7632, valid loss:0.5188 valid acc:0.7668
Epoch:103 / 150, train loss:0.4982 train_acc:0.7401, valid loss:0.5400 valid acc:0.7284
Epoch:104 / 150, train loss:0.4804 train_acc:0.7656, valid loss:0.5014 valid acc:0.7356
Epoch:105 / 150, train loss:0.4753 train_acc:0.7632, valid loss:0.4826 valid acc:0.7380
Epoch:106 / 150, train loss:0.4796 train_acc:0.7566, valid loss:0.5776 valid acc:0.6755
Epoch:107 / 150, train loss:0.4763 train_acc:0.7640, valid loss:0.5201 valid acc:0.7163
Epoch:108 / 150, train loss:0.4541 train_acc:0.7788, valid loss:0.5199 valid acc:0.7260
Epoch:109 / 150, train loss:0.4755 train_acc:0.7599, valid loss:0.5171 valid acc:0.7476
Epoch:110 / 150, train loss:0.4729 train_acc:0.7525, valid loss:0.5233 valid acc:0.7404
Epoch:111 / 150, train loss:0.4743 train_acc:0.7582, valid loss:0.5058 valid acc:0.7452
Epoch:112 / 150, train loss:0.4776 train_acc:0.7607, valid loss:0.5222 valid acc:0.7548
Epoch:113 / 150, train loss:0.4549 train_acc:0.7714, valid loss:0.4941 valid acc:0.7212
Epoch:114 / 150, train loss:0.4554 train_acc:0.7730, valid loss:0.5041 valid acc:0.7356
Epoch:115 / 150, train loss:0.4699 train_acc:0.7664, valid loss:0.6800 valid acc:0.6418
Epoch:116 / 150, train loss:0.4582 train_acc:0.7812, valid loss:0.5449 valid acc:0.7115
Epoch:117 / 150, train loss:0.4693 train_acc:0.7681, valid loss:0.5253 valid acc:0.7260
Epoch:118 / 150, train loss:0.4690 train_acc:0.7812, valid loss:0.4649 valid acc:0.7740
Epoch:119 / 150, train loss:0.4470 train_acc:0.7812, valid loss:0.5200 valid acc:0.7115
Epoch:120 / 150, train loss:0.4661 train_acc:0.7648, valid loss:0.5053 valid acc:0.7212
Epoch:121 / 150, train loss:0.4437 train_acc:0.7763, valid loss:0.4779 valid acc:0.7692
Epoch:122 / 150, train loss:0.4651 train_acc:0.7648, valid loss:0.4750 valid acc:0.7668
Epoch:123 / 150, train loss:0.4454 train_acc:0.7788, valid loss:0.4982 valid acc:0.7548
```

```
Epoch:124 / 150, train loss:0.4538 train_acc:0.7673, valid loss:0.5032 valid acc:0.7380
Epoch:125 / 150, train loss:0.4554 train_acc:0.7771, valid loss:0.4849 valid acc:0.7284
Epoch:126 / 150, train loss:0.4411 train_acc:0.7845, valid loss:0.5265 valid acc:0.7476
Epoch:127 / 150, train loss:0.4435 train_acc:0.7763, valid loss:0.5093 valid acc:0.7308
Epoch:128 / 150, train loss:0.4572 train_acc:0.7599, valid loss:0.4833 valid acc:0.7596
Epoch:129 / 150, train loss:0.4414 train_acc:0.7928, valid loss:0.5027 valid acc:0.7308
Epoch:130 / 150, train loss:0.4352 train_acc:0.7985, valid loss:0.4927 valid acc:0.7452
Epoch:131 / 150, train loss:0.4256 train_acc:0.7862, valid loss:0.4765 valid acc:0.7620
Epoch:132 / 150, train loss:0.4521 train_acc:0.7738, valid loss:0.5423 valid acc:0.7476
Epoch:133 / 150, train loss:0.4388 train_acc:0.7796, valid loss:0.4879 valid acc:0.7548
Epoch:134 / 150, train loss:0.4137 train_acc:0.8059, valid loss:0.4743 valid acc:0.7572
Epoch:135 / 150, train loss:0.4269 train_acc:0.7928, valid loss:0.4647 valid acc:0.7524
Epoch:136 / 150, train loss:0.4175 train_acc:0.8035, valid loss:0.4990 valid acc:0.7548
Epoch:137 / 150, train loss:0.4379 train_acc:0.8067, valid loss:0.5030 valid acc:0.7380
Epoch:138 / 150, train loss:0.4236 train_acc:0.7821, valid loss:0.4952 valid acc:0.7260
Epoch:139 / 150, train loss:0.4274 train_acc:0.7845, valid loss:0.4823 valid acc:0.7572
Epoch:140 / 150, train loss:0.4360 train_acc:0.7887, valid loss:0.5678 valid acc:0.7115
Epoch:141 / 150, train loss:0.4135 train_acc:0.7985, valid loss:0.5308 valid acc:0.7308
Epoch:142 / 150, train loss:0.4165 train_acc:0.7911, valid loss:0.4990 valid acc:0.7332
Epoch:143 / 150, train loss:0.4186 train_acc:0.7969, valid loss:0.5814 valid acc:0.6923
Epoch:144 / 150, train loss:0.4115 train_acc:0.8051, valid loss:0.5119 valid acc:0.7380
Epoch:145 / 150, train loss:0.4236 train_acc:0.7944, valid loss:0.4984 valid acc:0.7260
Epoch:146 / 150, train loss:0.4018 train_acc:0.8059, valid loss:0.4668 valid acc:0.7740
Epoch:147 / 150, train loss:0.4185 train_acc:0.7928, valid loss:0.4598 valid acc:0.7764
Epoch:148 / 150, train loss:0.4160 train_acc:0.7952, valid loss:0.5022 valid acc:0.7404
Epoch:149 / 150, train loss:0.3950 train_acc:0.8158, valid loss:0.4911 valid acc:0.7476
Epoch:150 / 150, train loss:0.4025 train_acc:0.8092, valid loss:0.5282 valid acc:0.7067
```

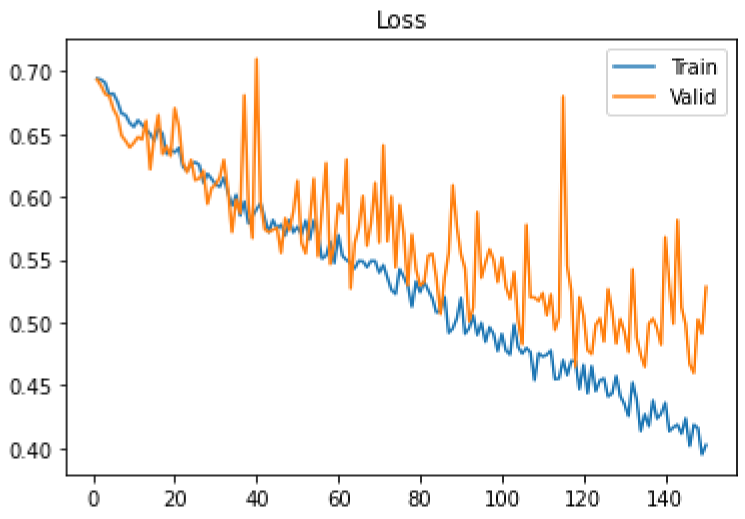
In [ ]:

```python
plt.plot(range(1, 151), history['train_acc'], label='Train')
plt.plot(range(1, 151), history['test_acc'], label='Valid')
plt.title('Accuracy')
plt.legend()
plt.show()
```



In [ ]:

```python
plt.plot(range(1, 151), history['train_loss'], label='Train')
plt.plot(range(1, 151), history['test_loss'], label='Valid')
plt.title('Loss')
plt.legend()
plt.show()
```


Loss
Train
Valid
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0
20
40
60
80
100
120
140

```
In [ ]:
```