Dog breed classifier

https://www.kaggle.com/datasets/gpiosenka/70-dog-breedsimage-data-set

Data

from 70 dog breeds 8 were chosen to be used in the classifier the training data consists of 80 - 120 pictures for each breed with a fixed size of 224 x 224

- Border Collie
- Borzoi
- Cocker
- German Sheperd
- Golden Retriever
- Greyhound
- Pomeranian
- Shiba Inu

the test data consists of 10 pictures for each breed with the same dimensions

Visualizing Data

In [ ]:
```
import matplotlib.pyplot as plt
from matplotlib.image import imread
import pandas as pd
import os
```

In [ ]:
```
data_dir = "train"
test_data_dir = "test"

breeds = os.listdir(data_dir)
dogs = []

for breed in breeds:
    files = os.listdir(f'{data_dir}/{breed}')
    dogs.append({"Folder":breed,"Count":len(files)})

df = pd.DataFrame.from_records(dogs)
print(df)
```

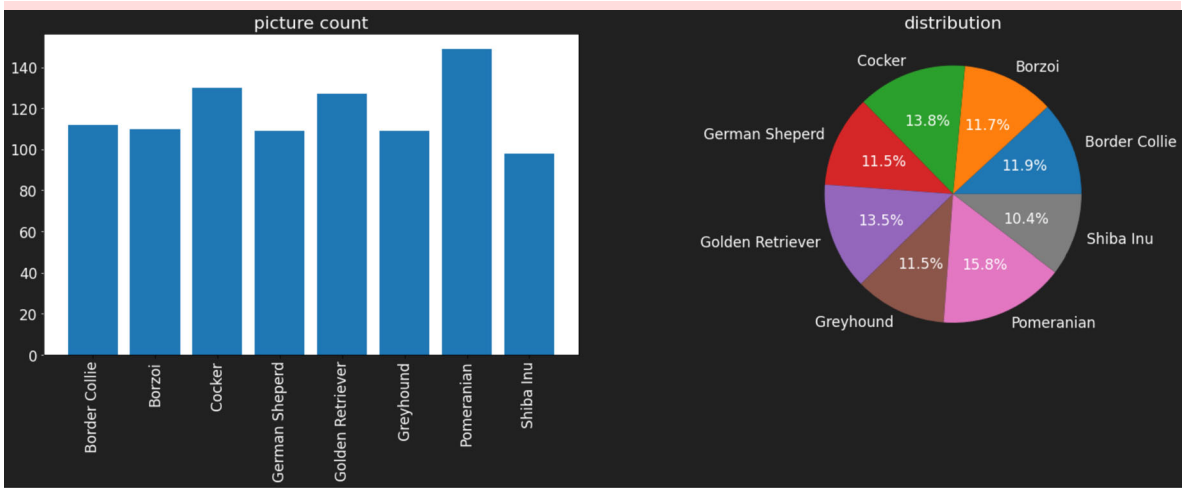
```
            Folder  Count
0     Border Collie    112
1           Borzoi    110
2           Cocker    130
3   German Sheperd    109
4  Golden Retriever    127
5         Greyhound    109
6        Pomeranian    149
7         Shiba Inu     98
```

In [ ]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(25, 7))


ax1.bar(df["Folder"], df["Count"])
ax1.set_xticklabels(df["Folder"], rotation=90, color='white')
ax1.set_title('picture count', color='white')
ax1.tick_params(axis='y', colors='white')

# Pie chart
ax2.pie(df["Count"], labels=df["Folder"], autopct='%1.1f%%')
ax2.set_title('distribution')

plt.show()
```



the data is evenly distributed among the 8 different dog breeds

In [ ]:
```
fig, pics = plt.subplots(2, 4, figsize=(10, 10))



for i, pic in enumerate(pics.flat):
    img = imread(f'{data_dir}/{breeds[i]}/001.jpg')
    pic.imshow(img)
    pic.set_title(breeds[i])
    pic.axis('off')

plt.tight_layout()
plt.show()
```

Augmenting Data

to further increase the amount of data, torchvision was used to augment more data from already existing pictures

In [ ]:
```
import torch
import torchvision
from torchvision import transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
import torchvision.transforms.functional as TF
```

In [ ]:
```
# Define data augmentation and normalization transforms
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(30),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1
    transforms.ToTensor(),
])

test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# Load the train and test data with transformations
dataset = ImageFolder(data_dir, transform=train_transform)
test_dataset = ImageFolder(test_data_dir, transform=test_transform)
```

In [ ]:
```
example_augmentation = DataLoader(dataset, 8, shuffle=True)
```

```
for batch_images, batch_labels in example_augmentation:
    # Display the batch of images in a subplot
    fig, axes = plt.subplots(2, 4, figsize=(12, 6))

    for i, ax in enumerate(axes.flat):
        # Convert the image tensor to a NumPy array
        img = TF.to_pil_image(batch_images[i])

        ax.imshow(img)
        ax.axis('off')
        ax.set_title(f"{breeds[batch_labels[i].item()]}")


    plt.tight_layout()
    plt.show()
    break
```

Splitting training data into training and validation data

In [ ]:
```
from torch.utils.data.dataloader import DataLoader
from torch.utils.data import random_split

batch_size = 32
val_size = 100
train_size = len(dataset) - val_size

train_data,val_data = random_split(dataset,[train_size,val_size])
print(f"Length of Train Data : {len(train_data)}")
print(f"Length of Validation Data : {len(val_data)}")
```

```
Length of Train Data : 844
Length of Validation Data : 100
```

Loading Data into batches

In [ ]:
```
#load the train and validation into batches.
train_dl = DataLoader(train_data, batch_size, shuffle = True, num_workers = 4, p
val_dl = DataLoader(val_data, batch_size*2, num_workers = 4, pin_memory = True)
```

Defining NN and functions

In [ ]:
```
import torch.nn as nn
import torch.nn.functional as F
```

In [ ]:
```
class ImageClassificationBase(nn.Module):

    def training_step(self, batch):
        images, labels = batch
        out = self(images)                    # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images)                      # Generate predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs] # collect losses for bat
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]    # collect accuracies for
        epoch_acc = torch.stack(batch_accs).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}
            epoch, result['train_loss'], result['val_loss'], result['val_acc']))
```

In [ ]:
```
class DogClassification(ImageClassificationBase):
    def __init__(self):
        super().__init__()
        self.network = nn.Sequential(

            nn.Conv2d(3, 32, kernel_size = 3, padding = 1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.BatchNorm2d(32),

            nn.Conv2d(32,64, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.BatchNorm2d(64),

            nn.Conv2d(64, 128, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU(),
            nn.BatchNorm2d(128),

            nn.Conv2d(128 ,128, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU(),
            nn.BatchNorm2d(128),
```

```
            nn.Conv2d(128, 256, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU(),
            nn.MaxPool2d(2,2),
            nn.BatchNorm2d(256),

            nn.Flatten(),

            nn.Linear(200704,1024),
            nn.ReLU(),
            nn.Dropout(p=0.2),
            nn.BatchNorm1d(1024),

            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(p=0.2),
            nn.BatchNorm1d(512),

            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(p=0.2),
            nn.BatchNorm1d(256),

            nn.Linear(256,8),
            nn.Softmax(1)
        )
        #  ◦ AlexNet
        # resources to read up on architectures:
        # https://medium.com/analytics-vidhya/concept-of-alexnet-convolutional-n
        # modelled after this:
        # https://github.com/abhijeetpujara/AlexNet/blob/main/Alexnet.ipynb

        # more general resources
        # https://medium.com/@siddheshb008/why-convolutions-dd7641d2bf81
        # https://medium.com/@siddheshb008/understanding-convolution-neural-netw
        # https://medium.com/@siddheshb008/understanding-convolutional-neural-ne

    def forward(self, xb):
        return self.network(xb)
```

In [ ]:

```
def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))


def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)


def fit(epochs, lr, model, train_loader, val_loader, opt_func = torch.optim.SGD)

    history = []
    optimizer = opt_func(model.parameters(),lr)
    for epoch in range(epochs):

        model.train()
        train_losses = []
        for batch in train_loader:
```

```
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()

        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        model.epoch_end(epoch, result)
        history.append(result)

    return history
```

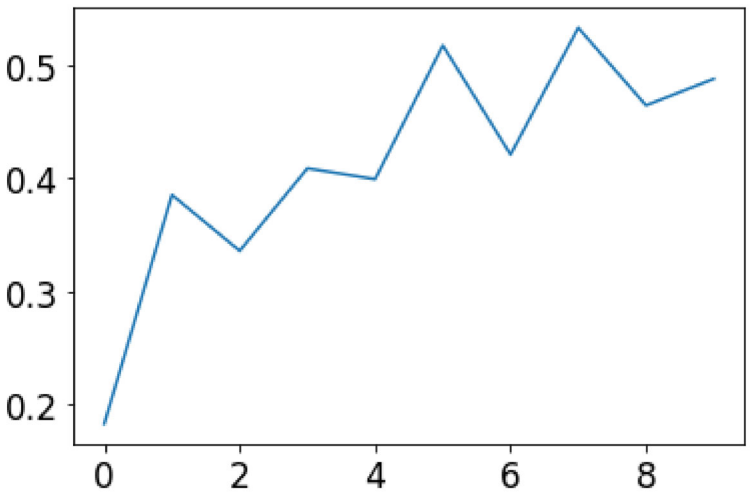
In [ ]:
```
num_epochs = 10
opt_func = torch.optim.Adam
lr = 0.0001
model = DogClassification() # number of breeds

#fitting the model on training data and record the result after each epoch
history = fit(num_epochs, lr, model, train_dl, val_dl, opt_func)
```

```
Epoch [0], train_loss: 2.0187, val_loss: 2.0738, val_acc: 0.1832
Epoch [1], train_loss: 1.9160, val_loss: 1.9405, val_acc: 0.3854
Epoch [2], train_loss: 1.8618, val_loss: 1.9211, val_acc: 0.3359
Epoch [3], train_loss: 1.8208, val_loss: 1.8626, val_acc: 0.4089
Epoch [4], train_loss: 1.7873, val_loss: 1.8656, val_acc: 0.3993
Epoch [5], train_loss: 1.7596, val_loss: 1.8034, val_acc: 0.5174
Epoch [6], train_loss: 1.7459, val_loss: 1.8215, val_acc: 0.4210
Epoch [7], train_loss: 1.6957, val_loss: 1.7812, val_acc: 0.5330
Epoch [8], train_loss: 1.6889, val_loss: 1.8239, val_acc: 0.4644
Epoch [9], train_loss: 1.6716, val_loss: 1.8212, val_acc: 0.4878
```

In [ ]:
```
plotDF = pd.DataFrame.from_records(history)

plt.plot(plotDF["val_acc"])
plt.title("Accuracy over Epochs")
plt.show()
```



Changing optimization function

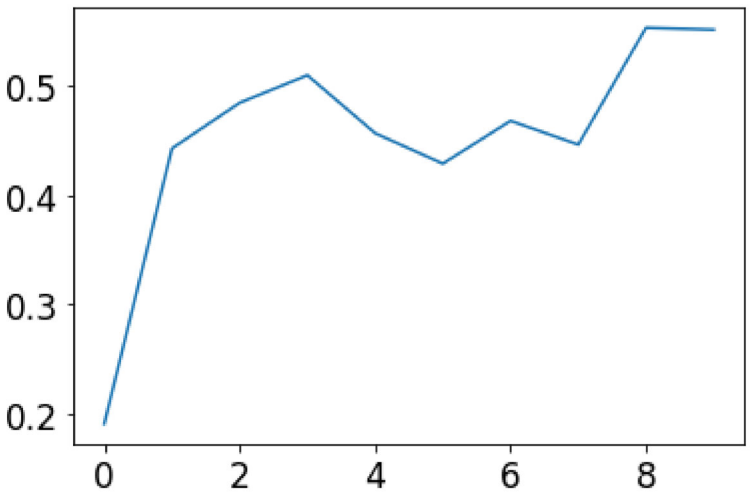
opt_func2 = torch.optim.AdamW

In [ ]:
```
history = fit(num_epochs, lr, model, train_dl, val_dl, opt_func2)
```

```
Epoch [0], train_loss: 1.9918, val_loss: 2.0663, val_acc: 0.1910
Epoch [1], train_loss: 1.9017, val_loss: 1.8502, val_acc: 0.4427
Epoch [2], train_loss: 1.8401, val_loss: 1.8551, val_acc: 0.4844
Epoch [3], train_loss: 1.8073, val_loss: 1.7919, val_acc: 0.5095
Epoch [4], train_loss: 1.7664, val_loss: 1.8292, val_acc: 0.4566
Epoch [5], train_loss: 1.7509, val_loss: 1.8498, val_acc: 0.4288
Epoch [6], train_loss: 1.7089, val_loss: 1.8130, val_acc: 0.4679
Epoch [7], train_loss: 1.6930, val_loss: 1.8254, val_acc: 0.4462
Epoch [8], train_loss: 1.6634, val_loss: 1.7623, val_acc: 0.5530
Epoch [9], train_loss: 1.6508, val_loss: 1.7760, val_acc: 0.5512
```

In [ ]:
```
plotDF = pd.DataFrame.from_records(history)

plt.plot(plotDF["val_acc"])
plt.title("Accuracy over Epochs")
plt.show()
```



In [ ]:
```

```