untitled

May 13, 2023

[38]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the directories for training, validation, and testing data
train_dir = '/path/to/train'
val_dir = '/path/to/validation'
test_dir = '/path/to/test'

# Define the image dimensions and batch size
image_width = 150
image_height = 150
batch_size = 32

# Use ImageDataGenerator for data augmentation and preprocessing
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(rescale=1.0/255.0)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)
```

[39]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Initialize the model
model = Sequential()

# Add a convolutional layer with 32 filters, 3x3 kernel size, and 'relu'␣
 ↪activation
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(image_width,␣
 ↪image_height, 3)))

# Add a max pooling layer with 2x2 pool size
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
# Add a flatten layer
model.add(Flatten())

# Add two dense (fully connected) layers with 128 neurons and 'relu' activation
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))

# Add an output layer with the number of classes and 'softmax' activation
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Print the model summary
model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 148, 148, 32)      896

 max_pooling2d_6 (MaxPooling  (None, 74, 74, 32)       0
 2D)

 flatten_7 (Flatten)         (None, 175232)            0

 dense_18 (Dense)            (None, 128)               22429824

 dense_19 (Dense)            (None, 128)               16512

 dense_20 (Dense)            (None, 90)                11610

=================================================================
Total params: 22,458,842
Trainable params: 22,458,842
Non-trainable params: 0
_________________________________________________________________
```

[44]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Initialize the model
model = Sequential()
```

```
# Add the input layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(image_width,␣
 ↪image_height, 3)))

# Add more layers...
```

[46]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Initialize the model
model = Sequential()

# Add the first Convolutional layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(image_width,␣
 ↪image_height, 3)))

# Add the first Pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))
```

[47]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the directories for training, validation, and testing data
train_dir = '/path/to/train'
val_dir = '/path/to/validation'
test_dir = '/path/to/test'

# Define the image dimensions and batch size
image_width = 150
image_height = 150
batch_size = 32

# Use ImageDataGenerator for data augmentation and preprocessing
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(rescale=1.0/255.0)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)
```

[48]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Initialize the model
```

```
model = Sequential()

# Add the first Convolutional layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(image_width,␣
 ↪image_height, 3)))

# Add the first Pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add the Flatten layer
model.add(Flatten())
```

[50]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the directories for training, validation, and testing data
train_dir = '/path/to/train'
val_dir = '/path/to/validation'
test_dir = '/path/to/test'

# Define the image dimensions and batch size
image_width = 150
image_height = 150
batch_size = 32

# Use ImageDataGenerator for data augmentation and preprocessing
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

val_datagen = ImageDataGenerator(rescale=1.0/255.0)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)
```

[53]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Add the first Hidden layer
model.add(Dense(64, activation='relu'))

# Add the second Hidden layer
model.add(Dense(64, activation='relu'))

# Add more layers...
```

[54]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Initialize the model
model = Sequential()

# Add the first Convolutional layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(image_width,␣
 ↪image_height, 3)))

# Add the first Pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add the Flatten layer
model.add(Flatten())

# Add the first Hidden layer
model.add(Dense(64, activation='relu'))

# Add the second Hidden layer
model.add(Dense(64, activation='relu'))

# Add the Output layer
num_classes = 90  # number of different animal classes in the dataset
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[56]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Initialize the model
model = Sequential()

# Add the first Convolutional layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(image_width,␣
 ↪image_height, 3)))

# Add the first Pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add the Flatten layer
model.add(Flatten())

# Add the first Hidden layer
```

```
model.add(Dense(64, activation='relu'))

# Add the second Hidden layer
model.add(Dense(64, activation='relu'))

# Add the Output layer
num_classes = 90  # number of different animal classes in the dataset
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[61]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the directory for the dataset
data_dir = 'path/to/archive'

# Create an ImageDataGenerator for data augmentation
datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2  # Split the dataset into training and validation
)
```

[62]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Initialize the model
model = Sequential()

# Input layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))

# Convolutional and Pooling layers
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Flatten layer
model.add(Flatten())
```

```
# Hidden layers
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))

# Output layer
model.add(Dense(90, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[70]:
```
import os
import random
import numpy as np
import pandas as pd
from tqdm import tqdm
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import random_split
from torch.utils.data import DataLoader, Dataset, Subset
from torch.utils.data import random_split, SubsetRandomSampler
from torchvision import datasets, transforms, models
from torchvision.datasets import ImageFolder
from torchvision.transforms import ToTensor
from torchvision.utils import make_grid
from pytorch_lightning import LightningModule
from pytorch_lightning import Trainer
import pytorch_lightning as pl
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from PIL import Image
```

[73]:
```
transform=transforms.Compose([
        transforms.RandomRotation(10),      # rotate +/- 10 degrees
        transforms.RandomHorizontalFlip(),  # reverse 50% of images
        transforms.Resize(224),             # resize shortest side to 224 pixels
        transforms.CenterCrop(224),         # crop longest side to 224 pixels␣
 ↪at center
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
])
```

[80]:
```
dataset0=datasets.ImageFolder(root="C:/Users/ELCOT/Downloads/archive/animals/
 ↪animals",transform=None)

class_names=dataset0.classes
print(class_names)
print(len(class_names))
```

```
['antelope', 'badger', 'bat', 'bear', 'bee', 'beetle', 'bison', 'boar',
'butterfly', 'cat', 'caterpillar', 'chimpanzee', 'cockroach', 'cow', 'coyote',
'crab', 'crow', 'deer', 'dog', 'dolphin', 'donkey', 'dragonfly', 'duck',
'eagle', 'elephant', 'flamingo', 'fly', 'fox', 'goat', 'goldfish', 'goose',
'gorilla', 'grasshopper', 'hamster', 'hare', 'hedgehog', 'hippopotamus',
'hornbill', 'horse', 'hummingbird', 'hyena', 'jellyfish', 'kangaroo', 'koala',
'ladybugs', 'leopard', 'lion', 'lizard', 'lobster', 'mosquito', 'moth', 'mouse',
'octopus', 'okapi', 'orangutan', 'otter', 'owl', 'ox', 'oyster', 'panda',
'parrot', 'pelecaniformes', 'penguin', 'pig', 'pigeon', 'porcupine', 'possum',
'raccoon', 'rat', 'reindeer', 'rhinoceros', 'sandpiper', 'seahorse', 'seal',
'shark', 'sheep', 'snake', 'sparrow', 'squid', 'squirrel', 'starfish', 'swan',
'tiger', 'turkey', 'turtle', 'whale', 'wolf', 'wombat', 'woodpecker', 'zebra']
90
```

[81]:
```
class DataModule(pl.LightningDataModule):

    def __init__(self, transform=transform, batch_size=32):
        super().__init__()
        self.root_dir = "/kaggle/input/
 ↪animal-image-dataset-90-different-animals/animals/animals"
        self.transform = transform
        self.batch_size = batch_size

    def setup(self, stage=None):
        dataset = datasets.ImageFolder(root=self.root_dir, transform=self.
 ↪transform)
        n_data = len(dataset)
        n_train = int(0.8 * n_data)
        n_test = n_data - n_train

        train_dataset, test_dataset = torch.utils.data.random_split(dataset,␣
 ↪[n_train, n_test])

        self.train_dataset = DataLoader(train_dataset, batch_size=self.
 ↪batch_size, shuffle=True)
        self.test_dataset = DataLoader(test_dataset, batch_size=self.batch_size)

    def train_dataloader(self):
        return self.train_dataset
```

```
    def test_dataloader(self):
        return self.test_dataset
```

[82]:
```
class ConvolutionalNetwork(LightningModule):

    def __init__(self):
        super(ConvolutionalNetwork, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 3, 1)
        self.conv2 = nn.Conv2d(6, 16, 3, 1)
        self.fc1 = nn.Linear(16 * 54 * 54, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 20)
        self.fc4 = nn.Linear(20, len(class_names))

    def forward(self, X):
        X = F.relu(self.conv1(X))
        X = F.max_pool2d(X, 2, 2)
        X = F.relu(self.conv2(X))
        X = F.max_pool2d(X, 2, 2)
        X = X.view(-1, 16 * 54 * 54)
        X = F.relu(self.fc1(X))
        X = F.relu(self.fc2(X))
        X = F.relu(self.fc3(X))
        X = self.fc4(X)
        return F.log_softmax(X, dim=1)

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=0.001)
        return optimizer

    def training_step(self, train_batch, batch_idx):
        X, y = train_batch
        y_hat = self(X)
        loss = F.cross_entropy(y_hat, y)
        pred = y_hat.argmax(dim=1, keepdim=True)
        acc = pred.eq(y.view_as(pred)).sum().item() / y.shape[0]
        self.log("train_loss", loss)
        self.log("train_acc", acc)
        return loss

    def validation_step(self, val_batch, batch_idx):
        X, y = val_batch
        y_hat = self(X)
        loss = F.cross_entropy(y_hat, y)
        pred = y_hat.argmax(dim=1, keepdim=True)
        acc = pred.eq(y.view_as(pred)).sum().item() / y.shape[0]
        self.log("val_loss", loss)
```

```
        self.log("val_acc", acc)

    def test_step(self, test_batch, batch_idx):
        X, y = test_batch
        y_hat = self(X)
        loss = F.cross_entropy(y_hat, y)
        pred = y_hat.argmax(dim=1, keepdim=True)
        acc = pred.eq(y.view_as(pred)).sum().item() / y.shape[0]
        self.log("test_loss", loss)
        self.log("test_acc", acc)
```

[ ]: