

FLOWER

RECOGNITION

Project by Team 11



PROBLEM statement

We had a large dataset of flowers of different variety and colour and we wanted to sort them with high accuracy while also decreasing the time and energy spent in the sorting process.

OBJECTIVE

Our objective was to create a neural network (An application of machine learning) to automatically classify flowers.



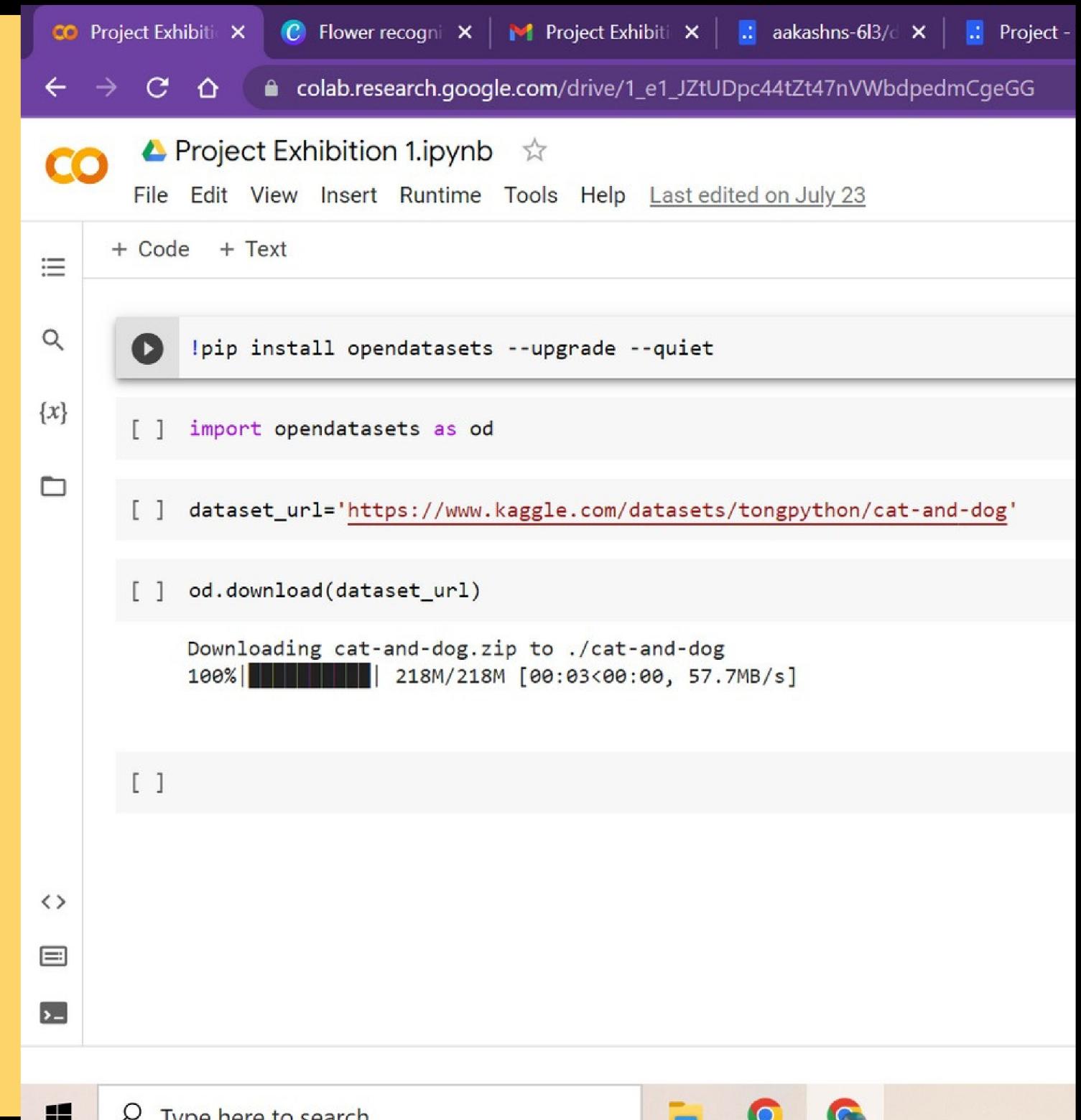
HARDWARE requirement

While projects using ML require a powerful dedicated Graphics Processing Unit (GPU), we leveraged the power of cloud servers to make this task easier for us. Hence any thin and light laptop is enough to execute the code.



SOFTWARE requirement

We used Google Collab to write and run our code since it provides access to free Tensor Processing Units (TPU) and Graphical Processing Units (GPU) and this greatly helped improve the efficiency of our project.



The screenshot shows a Google Colab notebook titled "Project Exhibition 1.ipynb". The code cell contains the following Python code:

```
!pip install opendatasets --upgrade --quiet
import opendatasets as od
dataset_url='https://www.kaggle.com/datasets/tongpython/cat-and-dog'
od.download(dataset_url)
```

The output of the code shows the download progress:

Downloading cat-and-dog.zip to ./cat-and-dog
100%|██████████| 218M/218M [00:03<00:00, 57.7MB/s]

ARCHITECTURE DIAGRAM and PROCESS FLOW

The code is divided into the following important parts:

1. Downloading the dataset
2. Importing the dataset into pytorch
3. Configuring GPU and training utilities
4. Creating the model
5. Training the model
6. Testing the model

**IMPORTANT POINTS AND PLACES WHERE THE
CODE COULD BE OPTIMIZED FOR THE NEXT
PROJECT EXHIBITION HAVE BEEN WRITTEN IN
THE FORM OF COMMENTS..**

DOWNLOADING THE DATASET

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook interface includes a toolbar at the top with various icons, a sidebar on the left with navigation and search tools, and a main workspace with two code cells.

Cell 1:

```
[1] !pip install opendatasets --upgrade
```

Output:

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting opendatasets
  Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from opendatasets) (4.64.1)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from opendatasets) (7.1.2)
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (from opendatasets) (1.5.12)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (2022.6.15)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (1.24.3)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (1.15.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (2.23.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (6.1.2)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages (from python-slugify->kaggle->opendatasets) (1.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle->opendatasets) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle->opendatasets) (2.10)
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22
```

Cell 2:

```
[2] import opendatasets as od
```

The status bar at the bottom shows system information including battery level, network, and date/time.

DOWNLOADING THE DATASET

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook interface includes a toolbar at the top with various icons for file operations, search, and sharing. The main area displays a sequence of Python code cells:

```
[1] Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle->opendatasets) (2.10)
3s
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22

{x}
[2] import opendatasets as od

[3] dataset_url = 'https://www.kaggle.com/alxmamaev/flowers-recognition'

[4] od.download(dataset_url)

    Downloading flowers-recognition.zip to ./flowers-recognition
    100%|██████████| 225M/225M [00:01<00:00, 126MB/s]

[5] data_dir = './flowers-recognition/flowers'

[6] import os
os.listdir(data_dir)

['daisy', 'tulip', 'sunflower', 'rose', 'dandelion']

[7] for cls in os.listdir(data_dir):
    print(cls, ':', len(os.listdir(data_dir + '/' + cls)))
```

The notebook shows the execution of the code, starting with the installation of the "opendatasets" package. It then defines a variable "dataset_url" pointing to a Kaggle dataset. Cell [4] shows the download of the dataset, with a progress bar indicating it's 100% complete at 126MB/s. Cells [5] through [7] demonstrate how to list the contents of the downloaded directory.

DOWNLOADING THE DATASET

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook interface includes a toolbar at the top with various icons, a sidebar on the left with file navigation and search functions, and a main workspace with two code cells.

Code Cell 7:

```
[7] for cls in os.listdir(data_dir):
    print(cls, ':', len(os.listdir(data_dir + '/' + cls)))
```

Output of Code Cell 7:

```
daisy : 764
tulip : 984
sunflower : 733
rose : 784
dandelion : 1052
```

Code Cell 8:

```
[8] from torchvision.datasets import ImageFolder
dataset = ImageFolder(data_dir)
len(dataset)
```

Output of Code Cell 8:

```
4317
```

Code Cell 9:

```
[9] import matplotlib.pyplot as plt
%matplotlib inline
img, label = dataset[376]
```

The bottom of the screen shows the Windows taskbar with icons for File Explorer, Task View, and Start, along with system status indicators like battery level and signal strength. The system tray shows the date and time as 02-10-2022 12:35.

EXPLANATION

In this part we basically downloaded a dataset from kaggle on which we would train our neural network on.

We used the opendataset library to download the dataset directly to the backend servers and after that we used the os library to find the number of flowers of each type.

This pretty much sums up our downloading the dataset part...

IMPORTING THE DATASET INTO PYTORCH

Project Exhibition.ipynb - Colab + colab.research.google.com/drive/11S0Wh8hd2gDivCUCRcsJEc11VxZLeW0P#scrollTo=MfjAEcvHYRVu

Project Exhibition.ipynb ★

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text RAM Disk Editing

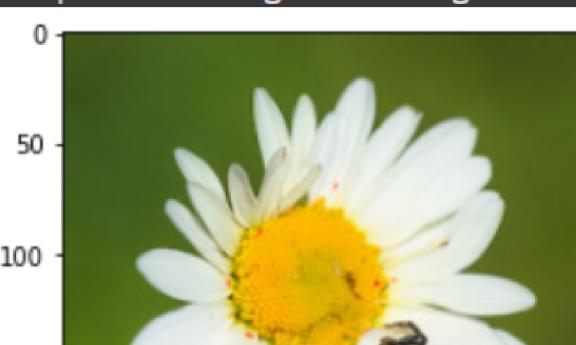
2. Importing the dataset into pytorch

```
[8] from torchvision.datasets import ImageFolder  
dataset = ImageFolder(data_dir)  
len(dataset)
```

4317

```
[9] import matplotlib.pyplot as plt  
%matplotlib inline  
img, label = dataset[376]  
plt.imshow(img)
```

<matplotlib.image.AxesImage at 0x7f8d633cc4d0>



ENG IN 12:35 02-10-2022 1

IMPORTING THE DATASET INTO PYTORCH

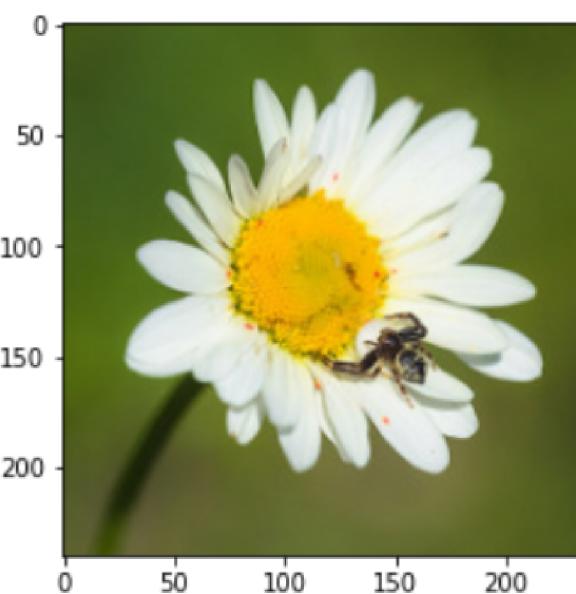
The screenshot shows a Google Colab notebook interface. The title bar reads "Project Exhibition.ipynb - Colab". The URL in the address bar is "colab.research.google.com/drive/11S0Wh8hd2gDivCUCRcsJEc11VxZLeW0P#scrollTo=MfjAEcvHYRVu". The notebook menu bar includes File, Edit, View, Insert, Runtime, Tools, Help, and "All changes saved". On the left, there are sidebar icons for Code (+ Code), Text (+ Text), {x}, and folder. The main workspace contains two code cells and one output cell.

Code Cell 1:

```
[9] plt.imshow(img)
```

Output Cell 1:

```
<matplotlib.image.AxesImage at 0x7f8d633cc4d0>
```



Code Cell 2:

```
[10] a = dataset[756] #1:Find the image with the least size in the database and resize accordingly---will help get a more accurate result.  
b = dataset[0]  
c = dataset[358]  
print (a,b,c)
```

Output Cell 2:

```
(<PIL.Image.Image image mode=RGB size=240x240 at 0x7F8D63541D10>, 0) (<PIL.Image.Image image mode=RGB size=320x263 at 0x7F8D63541B50>, 0) (<PIL.Image.1>, 0)
```

At the bottom, the taskbar shows icons for Windows, Search, Chrome, File, and a checkmark. The system tray shows battery level, signal strength, ENG IN, and the date/time "12:35 02-10-2022".

IMPORTING THE DATASET INTO PYTORCH

Project Exhibition.ipynb - Colab + colab.research.google.com/drive/11S0Wh8hd2gDivCUCRcsJEc11VxZLeW0P#scrollTo=MfjAEcvHYRVu

Project Exhibition.ipynb ★

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text RAM Disk Editing

[10] a = dataset[756] #1:Find the image with the least size in the database and resize accordingly---will help get a more accurate result.
b = dataset[0]
c = dataset[358]
print (a,b,c)

(`PIL.Image` image mode=RGB size=240x240 at 0x7F8D63541D10>, 0) (`PIL.Image` image mode=RGB size=320x263 at 0x7F8D63541B50>, 0) (`PIL.Image` image mode=RGB size=320x263 at 0x7F8D63541B50>, 0)

[11] import torchvision.transforms as tt
dataset = ImageFolder(data_dir, tt.Compose([tt.Resize((64,64)),
tt.RandomCrop((64,64)),
tt.ToTensor()]))

[12] img, label = dataset[376]
plt.imshow(img.permute((1, 2, 0)))

<matplotlib.image.AxesImage at 0x7f8d632ee890>



ENG IN 12:35 02-10-2022 1

IMPORTING THE DATASET INTO PYTORCH

Project Exhibition.ipynb - Colab +

colab.research.google.com/drive/11S0Wh8hd2gDivCUCRcsJEc11VxZLeW0P#scrollTo=MfjAEcvHYRVu

Project Exhibition.ipynb ★

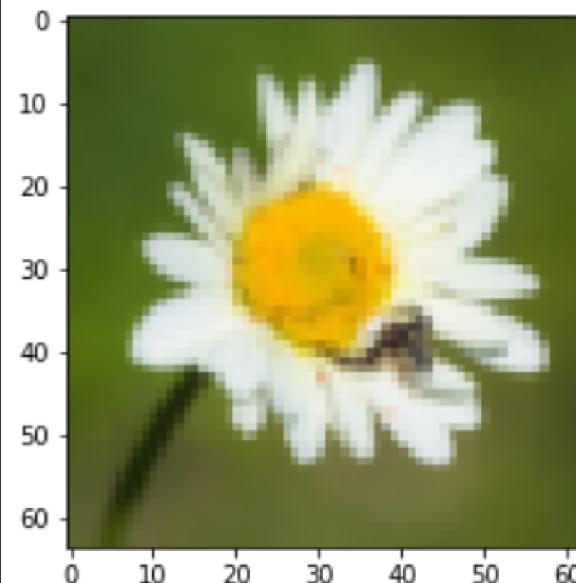
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[12] img, label = dataset[376]
1s plt.imshow(img.permute((1, 2, 0)))

{x}

<matplotlib.image.AxesImage at 0x7f8d632ee890>



[13] val_pct = 0.1
0s val_size = int(val_pct * len(dataset))
train_size = len(dataset) - val_size
train_size, val_size

(3886, 431)

RAM Disk ✓ Editing

Comment Share

12:35 02-10-2022 1

IMPORTING THE DATASET INTO PYTORCH

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The code in the notebook is as follows:

```
[13] val_pct = 0.1
val_size = int(val_pct * len(dataset))
train_size = len(dataset) - val_size
train_size, val_size

(3886, 431)

[14] from torch.utils.data import random_split
train_ds, valid_ds = random_split(dataset, [train_size, val_size])
len(train_ds), len(valid_ds)

(3886, 431)

[15] from torch.utils.data import DataLoader
batch_size = 128 #2:Gradient Descent Batch size to control the stability
train_dl = DataLoader(train_ds,
                     batch_size,
                     shuffle=True,
                     num_workers=2, #3:Check the highest number of num_workers the model is stable for and works.
                     pin_memory=True)
valid_dl = DataLoader(valid_ds,
                     batch_size,
```

The notebook interface includes a toolbar at the top with various icons for file operations, a progress bar for RAM and Disk usage, and a status bar at the bottom showing system information like battery level, network, and date.

IMPORTING THE DATASET INTO PYTORCH

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The code in the notebook is as follows:

```
[15]
    shuffle=True,
    num_workers=2, #3:Check the highest number of num_workers the model is stable for and works.
    pin_memory=True)
valid_dl = DataLoader(valid_ds,
    batch_size,
    num_workers=2,
    pin_memory=True)

"""
4: Instead of loading the dataset into the cpu and then moving it to the gpu for training (pin_memory),
load it directly into the gpu.....
"""

'\n 4: Instead of loading the dataset into the cpu and then moving it to the gpu for training (pin_memory),\n      load it directly into the gpu.....\n'

[16] from torchvision.utils import make_grid

def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(120, 60))
        ax.imshow(make_grid(images, nrow=16).permute(1, 2, 0))
        break
```

The notebook interface includes a toolbar at the top with various icons for file operations, sharing, and settings. The status bar at the bottom shows system information like battery level, signal strength, and the date and time (12:35, 02-10-2022).

IMPORTING THE DATASET INTO PYTORCH

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook interface includes a toolbar at the top with various icons for file operations, search, and sharing. The main workspace contains three code cells:

- Cell [15]: A comment line: '\n 4: Instead of loading the dataset into the cpu and then moving it to the gpu for training (pin_memory),\n load it directly into the gpu.....\n'
- Cell [16]: A function definition:

```
from torchvision.utils import make_grid

def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(120, 60))
        ax.imshow(make_grid(images, nrow=16).permute(1, 2, 0))
        break
```
- Cell [17]: A call to the function: "show_batch(train_dl)"

Below the code cells, a large grid of small images displays various flowers, including sunflowers, tulips, roses, and dandelions, arranged in a 4x4 grid.

At the bottom of the screen, the Windows taskbar is visible with icons for Start, Search, Task View, File Explorer, Google Chrome, and a checkmark icon. System tray icons include ENG IN, battery level, signal strength, and the date/time 02-10-2022 12:35.

IMPORTING THE DATASET INTO PYTORCH

The screenshot shows a Google Colab notebook interface. The title bar reads "Project Exhibition.ipynb - Colab". The main area displays a grid of approximately 100 flower images, likely used for training a PyTorch model. Below the grid, under the heading "3. Configuring GPU and training utilities", is a code cell containing the command `import torch`. The status bar at the bottom indicates "12:35 02-10-2022".

Project Exhibition .ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[17] 15s

{x}

RAM Disk

Comment Share

Editing

3. Configuring GPU and training utilities

[18] import torch

12:35 02-10-2022

EXPLANATION

In this section of code, we found out the details and size of images in the database using the pytorch library and we found out that they were not consistent and hence we then normalized our dataset.

After this, we divided the entire dataset into two parts; one for training and one for validation and created dataloaders for the purpose of dividing the images into smaller batches for the purpose of training and validation.

All of this helps immensely in increasing the accuracy of the neural network.

CONFIGURING GPU and TRAINING UTILITIES

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook is currently displaying a section titled "3. Configuring GPU and training utilities". The code in cell [18] is as follows:

```
[18] import torch

def get_default_device():
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        for b in self.dl:
            yield to_device(b, self.device)
```

The Colab interface includes a sidebar with file navigation, a search bar, and a toolbar with various icons. The status bar at the bottom shows system information like battery level, signal strength, and the date and time (12:36, 02-10-2022).

CONFIGURING GPU and TRAINING UTILITIES

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook interface includes a toolbar at the top with various icons for file operations, search, and sharing. The main area displays a sequence of code cells:

```
[18] 0s def __iter__(self):
        for b in self.dl:
            yield to_device(b, self.device)

[19] 0s torch.cuda.is_available()
      True

[20] 0s device = get_default_device()

[21] 0s device
      device(type='cuda')

[22] 0s img, label = dataset[0]

[23] 0s img.device
      device(type='cpu')
```

The code demonstrates the configuration of a PyTorch DataLoader to work with CUDA tensors and the retrieval of the current default device.

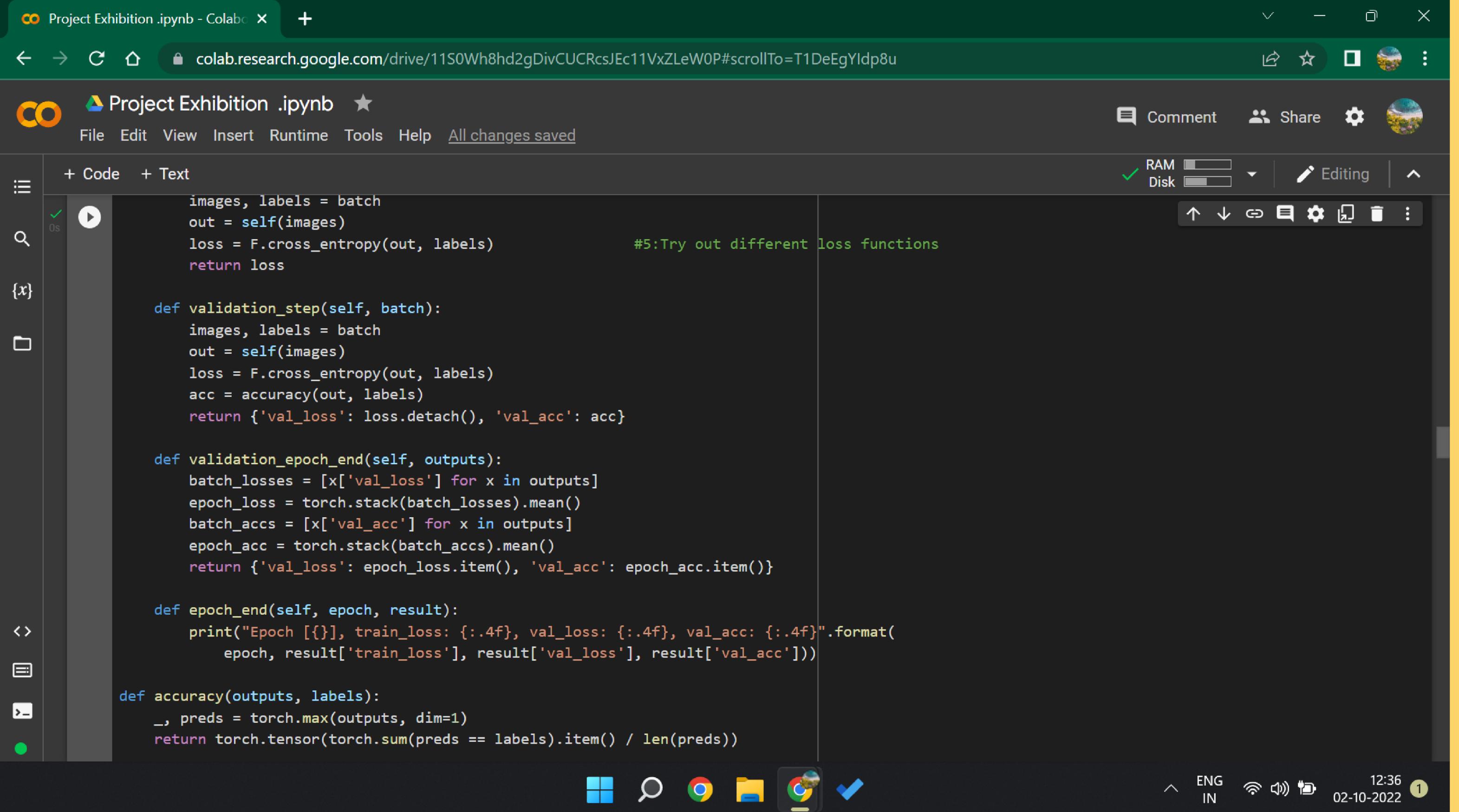
CONFIGURING GPU and TRAINING UTILITIES

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The code in the notebook includes:

```
[23] img.device  
device(type='cpu')  
  
[24] img_gpu = to_device(img, device)  
img_gpu.device  
  
device(type='cuda', index=0)  
  
[25] train_dl = DeviceDataLoader(train_dl, device)  
valid_dl = DeviceDataLoader(valid_dl, device)  
  
[26] import torch.nn as nn  
import torch.nn.functional as F  
  
class ImageClassificationBase(nn.Module):  
    def training_step(self, batch):  
        images, labels = batch  
        out = self(images)  
        loss = F.cross_entropy(out, labels)  
        return loss  
  
    def validation_step(self, batch):  
        images, labels = batch  
        out = self(images)  
        loss = F.cross_entropy(out, labels)  
        acc = accuracy(out, labels)  
        return {"val_loss": loss.item(), "val_accuracy": acc.item()}\n\n#5: Try out different loss functions
```

The notebook interface shows various tools and status indicators at the top and bottom.

CONFIGURING GPU and TRAINING UTILITIES



The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The code in the notebook is as follows:

```
images, labels = batch
out = self(images)
loss = F.cross_entropy(out, labels)
return loss

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

def validation_epoch_end(self, outputs):
    batch_losses = [x['val_loss'] for x in outputs]
    epoch_loss = torch.stack(batch_losses).mean()
    batch_accs = [x['val_acc'] for x in outputs]
    epoch_acc = torch.stack(batch_accs).mean()
    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}].format(
        epoch, result['train_loss'], result['val_loss'], result['val_acc']))"

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

CONFIGURING GPU and TRAINING UTILITIES

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook interface includes a toolbar at the top with various icons for file operations, sharing, and settings. The main area displays Python code for training a machine learning model using PyTorch. The code defines functions for calculating accuracy and evaluating a model's performance on a validation set. It also includes a main loop for training, which involves a training phase where the model is trained on batches of data, followed by an evaluation phase on a validation loader.

```
epoch, result['train_loss'], result['val_loss'], result['val_acc']))\n\n    def accuracy(outputs, labels):\n        _, preds = torch.max(outputs, dim=1)\n        return torch.tensor(torch.sum(preds == labels).item() / len(preds))\n\n[27] @torch.no_grad()\n    def evaluate(model, val_loader):\n        """Evaluates the model's performance on the validation set"""\n        model.eval()\n        outputs = [model.validation_step(batch) for batch in val_loader]\n        return model.validation_epoch_end(outputs)\n\n    def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.SGD):\n        history = []\n        optimizer = opt_func(model.parameters(), lr)\n        for epoch in range(epochs):\n            # Training Phase\n            model.train()\n            train_losses = []\n            for batch in train_loader:\n                loss = model.training_step(batch)\n                train_losses.append(loss)\n                loss.backward()\n                optimizer.step()\n                optimizer.zero_grad()
```

CONFIGURING GPU and TRAINING UTILITIES

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook contains Python code for training and validating a machine learning model. The code includes loops for epochs, training batches, and validation steps, along with loss calculation and history tracking.

```
optimizer = opt_func(model.parameters(), lr)
for epoch in range(epochs):
    # Training Phase
    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()
    # Validation phase
    result = evaluate(model, val_loader)
    result['train_loss'] = torch.stack(train_losses).mean().item()
    model.epoch_end(epoch, result)
    history.append(result)
return history
```

The notebook is currently executing cell 27. Below the main code cell, there is a section titled "4. Creating the model" which contains the following code:

```
def conv_block(in_channels, out_channels, pool=False):
    layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
              nn.BatchNorm2d(out_channels),
              nn.ReLU(inplace=True)]
```

The Colab interface includes a toolbar at the top with various icons for file operations, a sidebar on the left with icons for file, search, and code/text selection, and a bottom navigation bar with system icons like battery, signal, and date/time.

EXPLANATION

This section of code deals with the backend part, we are basically configuring the default device to be selected and used during the operation and also choosing how the data is fed to these devices.

We also define several functions here like epoch,accuracy and history.Apart from functions and modules, we also define select several attributes and properties of the model like the optimizer to be used and the number of iterations that the model has to run.

This section basically controls the efficiency and the time taken by the model to learn.

CREATING THE MODEL

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook interface includes a toolbar at the top with various icons for file operations, search, and sharing. The main workspace displays Python code for defining a convolutional block and a ResNet9 model. The code uses PyTorch's nn module to build the network layers.

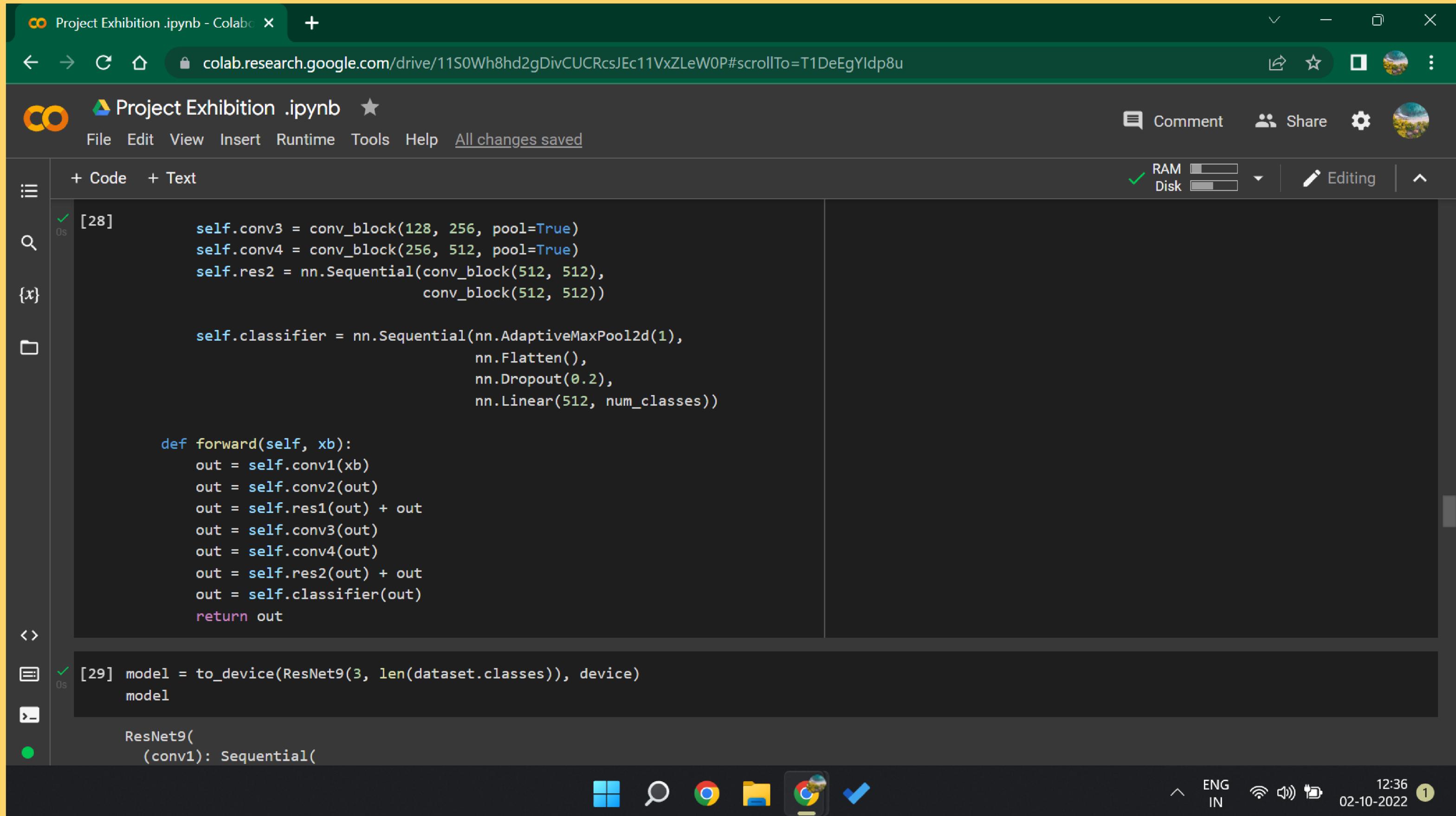
```
[27] return history

[28] def conv_block(in_channels, out_channels, pool=False):
    layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
              nn.BatchNorm2d(out_channels),
              nn.ReLU(inplace=True)]
    if pool: layers.append(nn.MaxPool2d(2))
    return nn.Sequential(*layers)

class ResNet9(ImageClassificationBase):
    def __init__(self, in_channels, num_classes):
        super().__init__()
        self.conv1 = conv_block(in_channels, 64)
        self.conv2 = conv_block(64, 128, pool=True)
        self.res1 = nn.Sequential(conv_block(128, 128),
                                conv_block(128, 128))

        self.conv3 = conv_block(128, 256, pool=True)
        self.conv4 = conv_block(256, 512, pool=True)
        self.res2 = nn.Sequential(conv_block(512, 512),
                                conv_block(512, 512))
```

CREATING THE MODEL



The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The code defines a ResNet9 model with the following structure:

```
[28]
    self.conv3 = conv_block(128, 256, pool=True)
    self.conv4 = conv_block(256, 512, pool=True)
    self.res2 = nn.Sequential(conv_block(512, 512),
                             conv_block(512, 512))

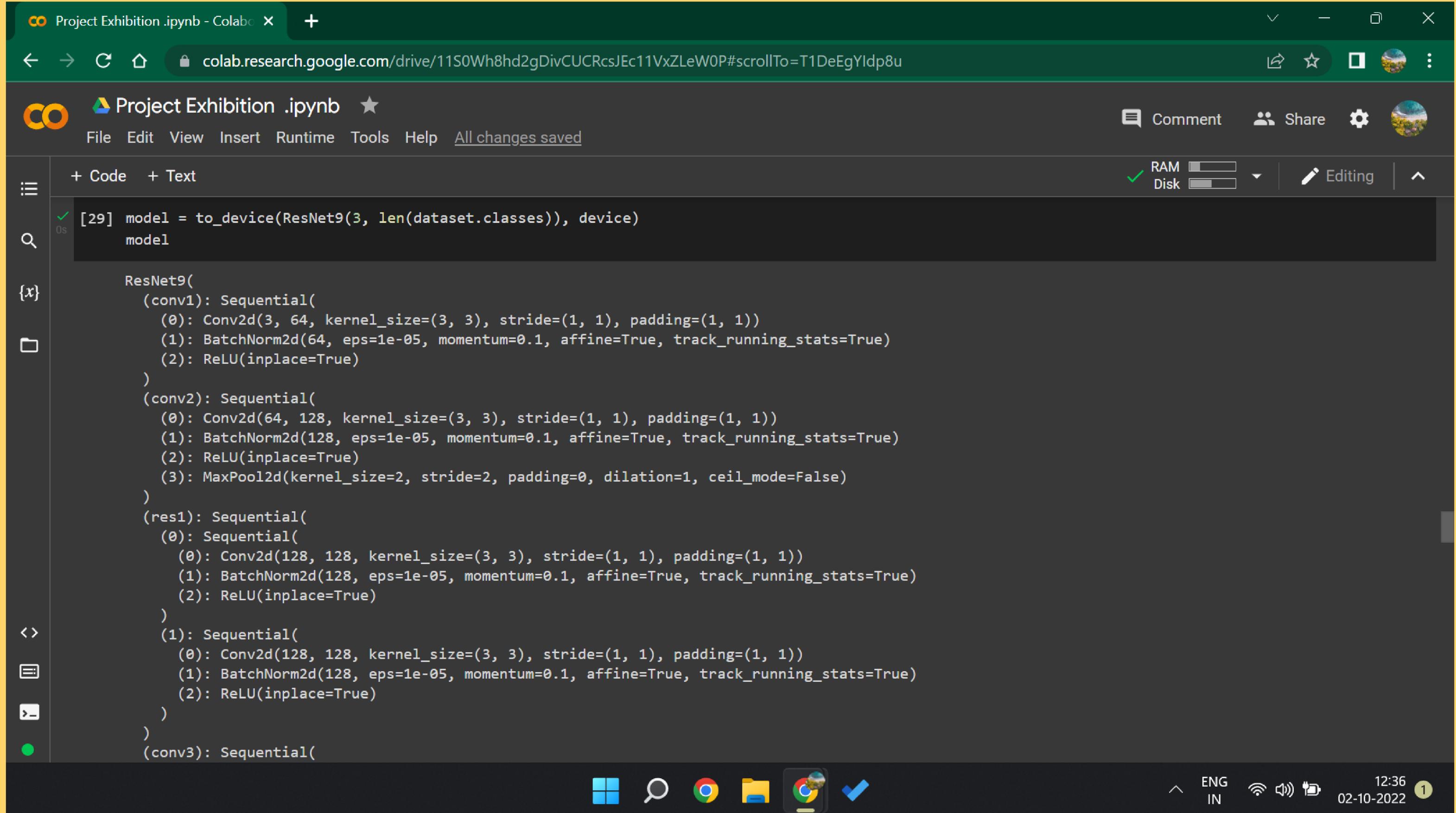
    self.classifier = nn.Sequential(nn.AdaptiveMaxPool2d(1),
                                    nn.Flatten(),
                                    nn.Dropout(0.2),
                                    nn.Linear(512, num_classes))

def forward(self, xb):
    out = self.conv1(xb)
    out = self.conv2(out)
    out = self.res1(out) + out
    out = self.conv3(out)
    out = self.conv4(out)
    out = self.res2(out) + out
    out = self.classifier(out)
    return out

[29] model = to_device(ResNet9(3, len(dataset.classes)), device)
      model
```

The code uses PyTorch's nn.Sequential and nn.Module classes to define the layers. It includes residual connections (res1 and res2) and a classifier consisting of adaptive max pooling, flattening, dropout, and a linear layer. The final output is returned by the forward method.

CREATING THE MODEL



The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The code cell at the top contains the following Python code:

```
[29] model = to_device(ResNet9(3, len(dataset.classes)), device)
      model
```

The main code block defines a `ResNet9` class, which is a sequential model consisting of three main parts: `(conv1)`, `(conv2)`, and `(conv3)`. Each part is a sequential model containing layers of `Conv2d`, `BatchNorm2d`, and `ReLU`. The `(conv1)` and `(conv2)` parts also include a `MaxPool2d` layer. The `(conv3)` part ends with a final `ReLU` layer.

CREATING THE MODEL

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The code in the notebook defines a neural network structure using PyTorch's nn.Sequential API. The model consists of several layers, including Conv2d, BatchNorm2d, and ReLU, with specific parameters like kernel_size, stride, padding, and dilation. The code is well-structured with comments and whitespace.

```
+ Code + Text
[29]   0s
  (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
)
)
{x}
(conv3): Sequential(
  (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(conv4): Sequential(
  (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
  (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(res2): Sequential(
  (0): Sequential(
    (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
)
)
(1): Sequential(
  (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU(inplace=True)
)
)
```

The notebook interface includes a toolbar at the top with various icons for file operations, a sidebar on the left with navigation and search tools, and a status bar at the bottom showing system information like battery level, signal strength, and date/time.

CREATING THE MODEL

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The code in the notebook defines a neural network structure and performs some initial setup steps.

```
(1): Sequential(  
  (0): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))  
  (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
  (2): ReLU(inplace=True)  
)  
{x}  
(classifier): Sequential(  
  (0): AdaptiveMaxPool2d(output_size=1)  
  (1): Flatten(start_dim=1, end_dim=-1)  
  (2): Dropout(p=0.2, inplace=False)  
  (3): Linear(in_features=512, out_features=5, bias=True)  
)  
  
[30] model.conv1[0].weight.device  
  
device(type='cuda', index=0)  
  
[31] torch.cuda.empty_cache()  
for batch in train_dl:  
    images, labels = batch  
    print('images.shape', images.shape)  
    print('images.device', images.device)  
    preds = model(images)  
    print('preds.shape', preds.shape)  
    break
```

CREATING THE MODEL

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook interface includes a toolbar at the top with various icons for file operations, search, and sharing. The main workspace displays two code cells. The first cell contains code to print the shape and device of the input images and the predicted output. The second cell shows the evaluation of the model on a validation dataset and the fitting of the model over 5 epochs. The status bar at the bottom shows system information like battery level, network connection, and the date/time.

```
+ Code + Text
[31] 6s
print('images.shape', images.shape)
print('images.device', images.device)
preds = model(images)
print('preds.shape', preds.shape)
break

images.shape torch.Size([128, 3, 64, 64])
images.device cuda:0
preds.shape torch.Size([128, 5])

▼ 5.Training the model

[32] 1s
history = [evaluate(model, valid_dl)]
history

[{'val_loss': 1.614917516708374, 'val_acc': 0.14768949151039124}]

<> 48s
[33] history += fit(5, 0.001, model, train_dl, valid_dl, torch.optim.Adam)

Epoch [0], train_loss: 1.6175, val_loss: 1.4370, val_acc: 0.3985
Epoch [1], train_loss: 1.0898, val_loss: 1.0155, val_acc: 0.5621
Epoch [2], train_loss: 0.8920, val_loss: 0.8275, val_acc: 0.6945
Epoch [3], train_loss: 0.8021, val_loss: 0.6785, val_acc: 0.7470
Epoch [4], train_loss: 0.7088, val_loss: 0.7893, val_acc: 0.7057
```

EXPLANATION

In the model, we define the type of neural network and in our case it is a CNN (Convolutional Neural Network) and after this we configure the number of neurons we want at each layer and the weights and biases between them. We also define the activation function over here.

Changing any attribute/property over here would result in a huge change in the final network.

TRAINING THE MODEL

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook interface includes a toolbar with file operations, a search bar, and a sidebar with various icons. The main content area displays a code cell output:

```
[32] history = [evaluate(model, valid_dl)]
    history
[{'val_loss': 1.614917516708374, 'val_acc': 0.14768949151039124}]

[33] history += fit(5, 0.001, model, train_dl, valid_dl, torch.optim.Adam)
Epoch [0], train_loss: 1.6175, val_loss: 1.4370, val_acc: 0.3985
Epoch [1], train_loss: 1.0898, val_loss: 1.0155, val_acc: 0.5621
Epoch [2], train_loss: 0.8920, val_loss: 0.8275, val_acc: 0.6945
Epoch [3], train_loss: 0.8021, val_loss: 0.6785, val_acc: 0.7470
Epoch [4], train_loss: 0.7088, val_loss: 0.7893, val_acc: 0.7057

[34] history += fit(5, 0.001, model, train_dl, valid_dl, torch.optim.Adam)
Epoch [0], train_loss: 0.8937, val_loss: 0.6920, val_acc: 0.7398
Epoch [1], train_loss: 0.7137, val_loss: 1.2567, val_acc: 0.6091
Epoch [2], train_loss: 0.6160, val_loss: 0.7897, val_acc: 0.7233
Epoch [3], train_loss: 0.5666, val_loss: 0.6281, val_acc: 0.7948
Epoch [4], train_loss: 0.5034, val_loss: 0.5846, val_acc: 0.7987
```

The notebook also shows status indicators for RAM and Disk usage.

TRAINING THE MODEL

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook interface includes a toolbar at the top with file operations like "File", "Edit", "View", "Insert", "Runtime", "Tools", "Help", and a status bar indicating "All changes saved". The main area displays a series of code cells and their outputs:

- Cell [34]:

```
Epoch [1], train_loss: 0.7157, val_loss: 1.2587, val_acc: 0.6691
[34] Epoch [2], train_loss: 0.6160, val_loss: 0.7897, val_acc: 0.7233
49s Epoch [3], train_loss: 0.5666, val_loss: 0.6281, val_acc: 0.7948
Epoch [4], train_loss: 0.5034, val_loss: 0.5846, val_acc: 0.7987
```
- Cell [35]:

```
[35] history += fit(5, 0.0001, model, train_dl, valid_dl, torch.optim.Adam)
```
- Cell [36]:

```
48s Epoch [0], train_loss: 0.3682, val_loss: 0.4564, val_acc: 0.8470
Epoch [1], train_loss: 0.3120, val_loss: 0.4408, val_acc: 0.8358
Epoch [2], train_loss: 0.2893, val_loss: 0.4407, val_acc: 0.8548
Epoch [3], train_loss: 0.2661, val_loss: 0.4489, val_acc: 0.8378
Epoch [4], train_loss: 0.2543, val_loss: 0.4417, val_acc: 0.8344
```
- Cell [37]:

```
[37] def plot_accuracies(history):
    accuracies = [x['val_acc'] for x in history]
    plt.plot(accuracies, '-x')
    plt.xlabel('epoch')
    plt.ylabel('accuracy')
```

The bottom of the screen shows the Windows taskbar with icons for File Explorer, Task View, and Start, along with system status icons for battery, signal, and volume. The system tray shows the date and time as "02-10-2022 12:37".

TRAINING THE MODEL

TRAINING THE MODEL

Project Exhibition.ipynb - Colab + colab.research.google.com/drive/11S0Wh8hd2gDivCUCRcsJEc11VxZLeW0P#scrollTo=T1DeEgYldp8u Comment Share ⚙️

Project Exhibition.ipynb ★

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[38] 0s [39] 0s [40] 0s

```
[38] 0s
[39] def plot_losses(history):
    train_losses = [x.get('train_loss') for x in history]
    val_losses = [x['val_loss'] for x in history]
    plt.plot(train_losses, '-bx')
    plt.plot(val_losses, '-rx')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend(['Training', 'Validation'])
    plt.title('Loss vs. No. of epochs');

[40] plot_losses(history)
```

RAM Disk ✓ Editing

{x}

Loss vs. No. of epochs

Training Validation

16.0
14.0
12.0
10.0
8.0
6.0
4.0
2.0
0.0

12:37 02-10-2022 1

TRAINING THE MODEL

Project Exhibition .ipynb - Colab

colab.research.google.com/drive/11S0Wh8hd2gDivCUCRcsJEc11VxZLeW0P#scrollTo=T1DeEgYldp8u

Project Exhibition .ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share Settings User Picture

RAM Disk Editing

+ Code + Text

[40] loss vs epoch

[41] history[-1]

```
{'val_loss': 0.4361819922924042, 'val_acc': 0.844165563583374, 'train_loss': 0.22680029273033142}
```

6. Testing the model

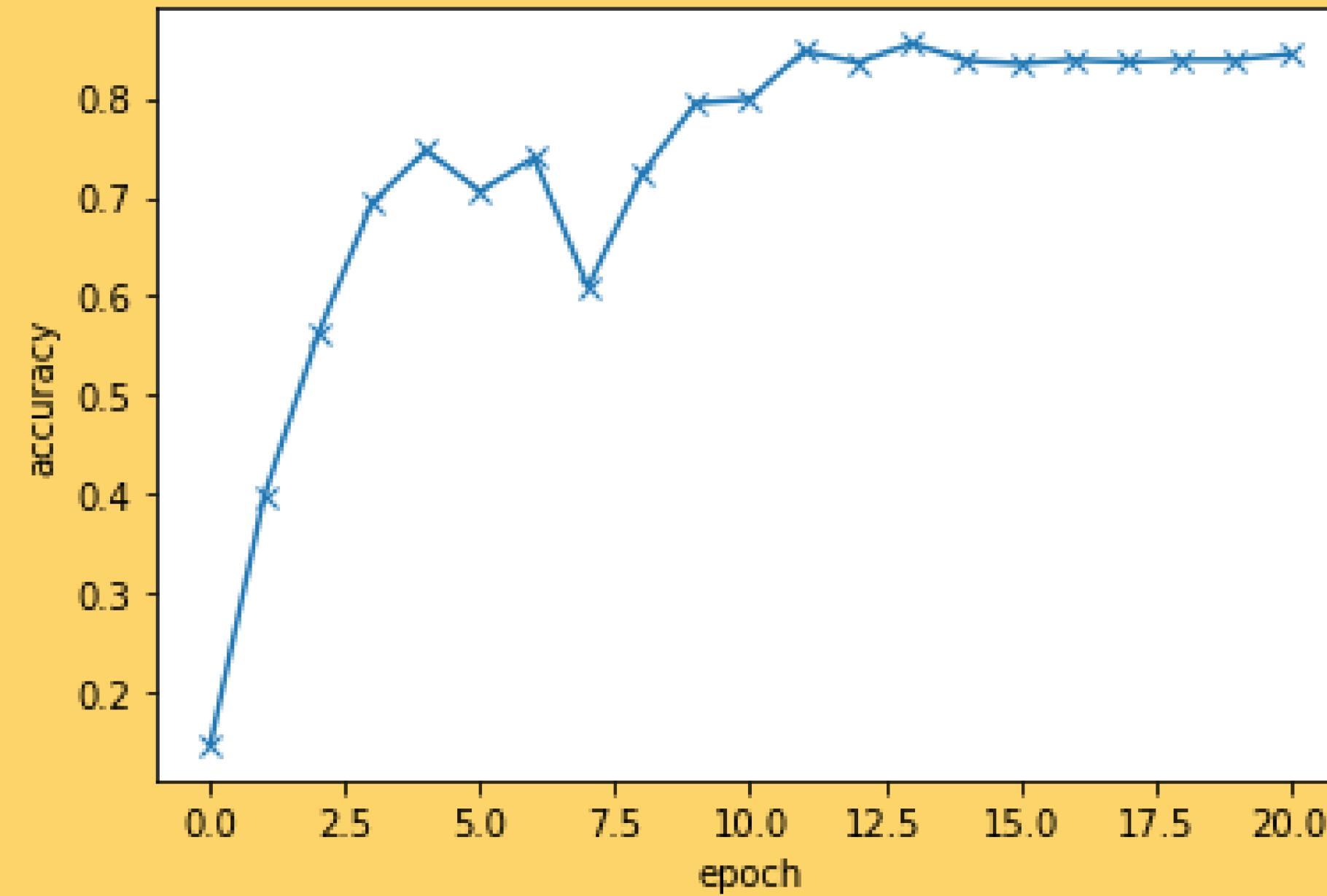
[42] def predict_image(img, model, classes):
 xb = to_device(img.unsqueeze(0), device)
 xb = model1(xb)

Windows Taskbar icons: File Explorer, Google Chrome, Task View, Taskbar settings.

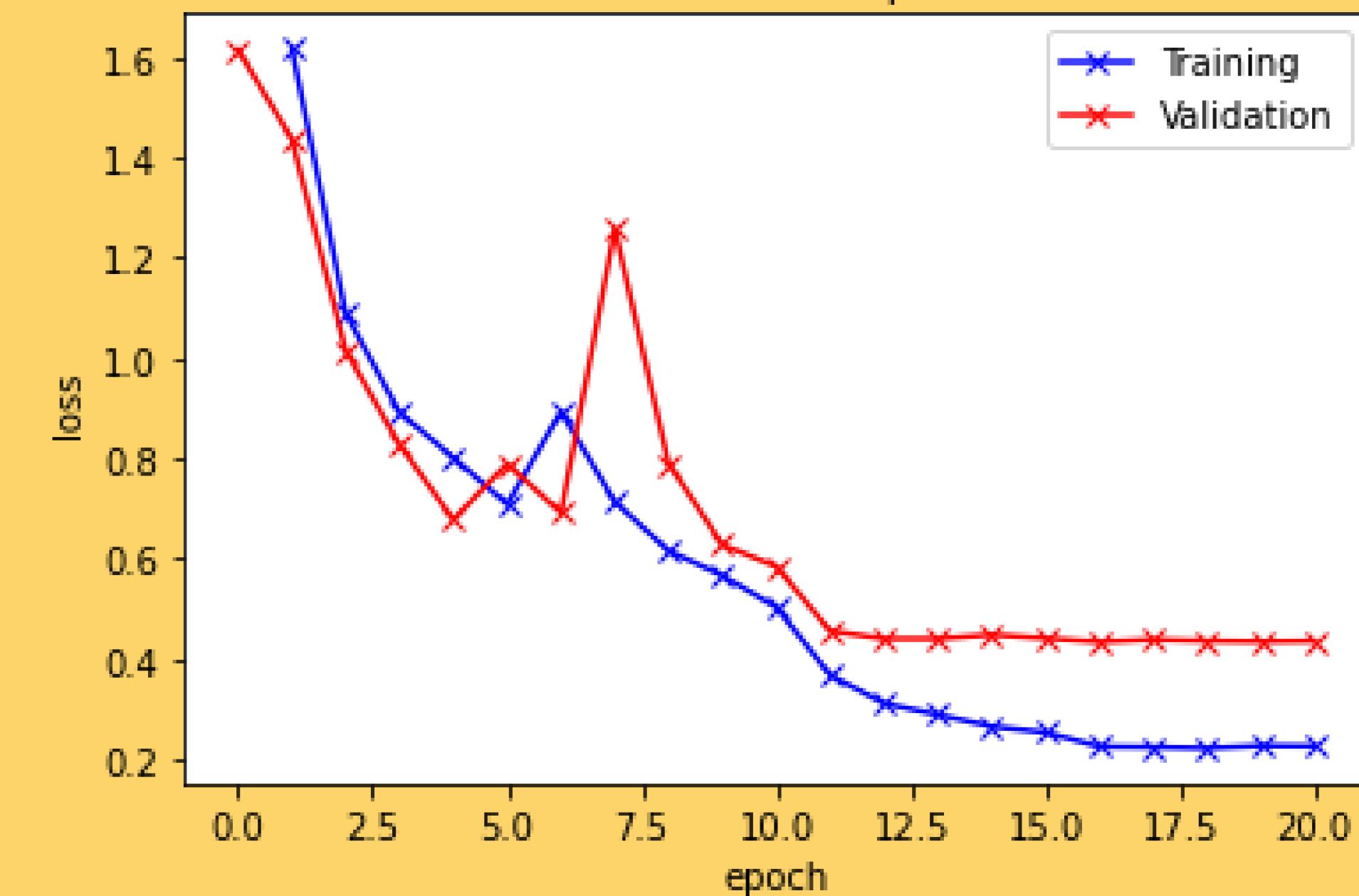
System tray icons: ENG IN, Wi-Fi, Battery, 12:37, 02-10-2022, 1 notification.

TRAINING THE MODEL

Accuracy vs. No. of epochs



Loss vs. No. of epochs



EXPLANATION

In this part we do a few final modifications like the number of epochs and plotting the accuracy vs the number of epochs, we also plot the history of the loss against the number of epochs.

TESTING THE MODEL

The screenshot shows a Google Colab notebook titled "Project Exhibition.ipynb". The notebook interface includes a toolbar at the top with various icons for file operations, search, and sharing. The main area displays the notebook's content, which consists of three code cells:

- Cell 42:** A function named `predict_image` that takes an image, a model, and a list of classes. It uses PyTorch to move the image to the device, performs inference, and returns the predicted class index.
- Cell 43:** A function named `show_image_prediction` that takes an image and its label. It displays the image, prints the target label, and prints the predicted label.
- Cell 44:** A call to `show_image_prediction` with the argument `*valid_ds[120]`, which triggers the execution of the previous two cells. The output shows that the target label is "sunflower" and the predicted label is also "sunflower". Below the output, a small thumbnail image of a sunflower is visible.

The status bar at the bottom of the screen shows system information like battery level, signal strength, and the date and time (02-10-2022, 12:37).

```
[42] def predict_image(img, model, classes):
    xb = to_device(img.unsqueeze(0), device)
    yb = model(xb)
    _, preds = torch.max(yb, dim=1)
    return classes[preds[0].item()]

[43] def show_image_prediction(img, label):
    plt.imshow(img.permute((1, 2, 0)))
    pred = predict_image(img, model, dataset.classes)
    print('Target:', dataset.classes[label])
    print('Prediction:', pred)

[44] show_image_prediction(*valid_ds[120])
```

TESTING THE MODEL

Project Exhibition .ipynb - Colab

colab.research.google.com/drive/11S0Wh8hd2gDivCUCRcsJEc11VxZLeW0P#scrollTo=T1DeEgYldp8u

Project Exhibition .ipynb

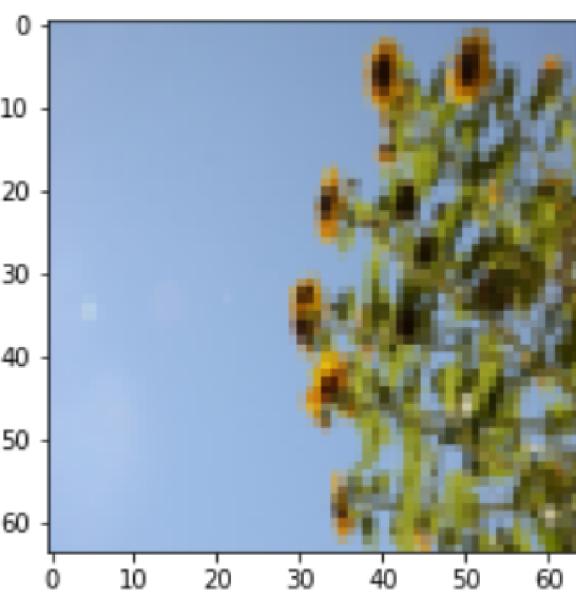
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[44] show_image_prediction(*valid_ds[120])

0s

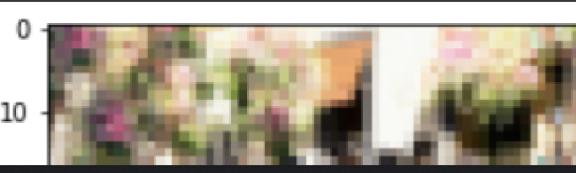
Target: sunflower
Prediction: sunflower



[45] show_image_prediction(*valid_ds[310])

0s

Target: rose
Prediction: rose



RAM Disk

Comment Share

Editing

12:37 02-10-2022 1

TESTING THE MODEL

Project Exhibition .ipynb - Colab + colab.research.google.com/drive/11S0Wh8hd2gDivCUCRcsJEc11VxZLeW0P#scrollTo=T1DeEgYldp8u

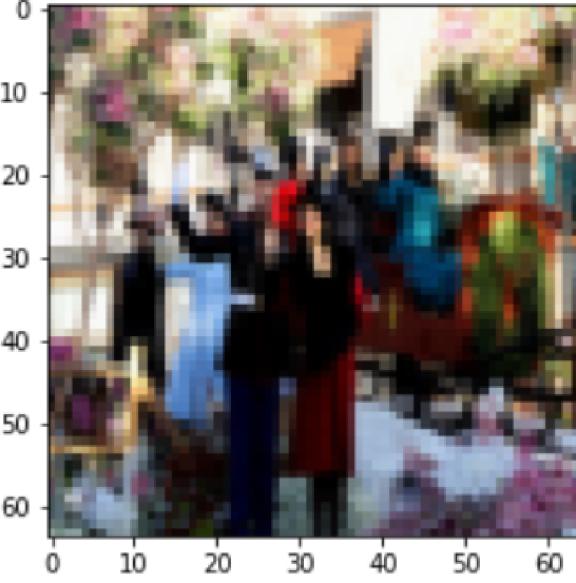
Project Exhibition .ipynb ★

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text RAM Disk Editing

[45] show_image_prediction(*valid_ds[310])

0s Target: rose
Prediction: rose



[46] show_image_prediction(*valid_ds[152])

0s Target: sunflower
Prediction: sunflower

12:37 02-10-2022 1

TESTING THE MODEL

Project Exhibition.ipynb - Colab + colab.research.google.com/drive/11S0Wh8hd2gDivCUCRcsJEc11VxZLeW0P#scrollTo=T1DeEgYldp8u

Project Exhibition.ipynb ★

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[46] show_image_prediction(*valid_ds[152])

0s Target: sunflower
Prediction: sunflower

{x}

0 10 20 30 40 50 60

0 10 20 30 40 50 60

[47] torch.save(model.state_dict(), 'flowers-resnet9.pth')

0s

ENG IN 12:37 02-10-2022 1

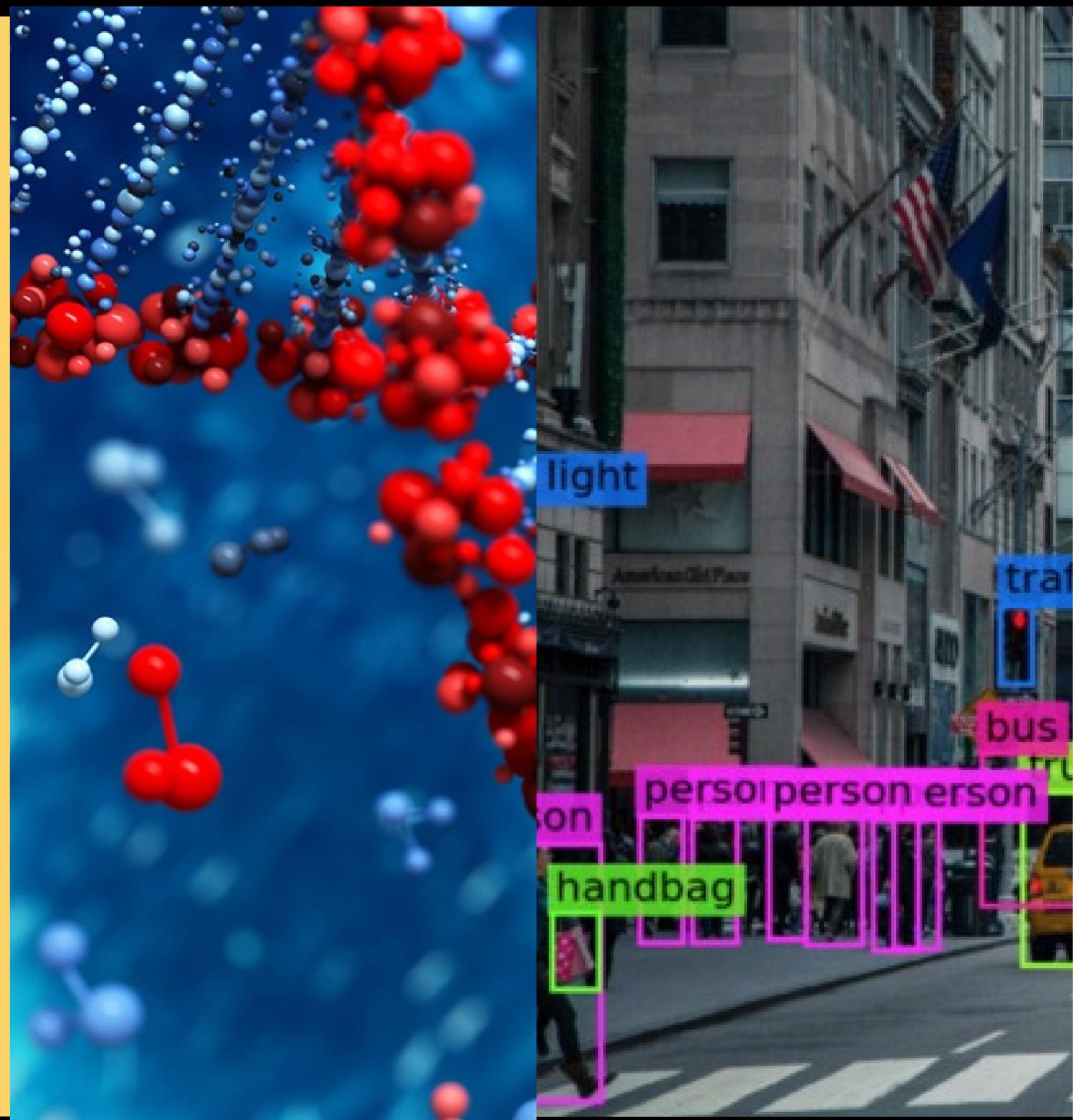
EXPLANATION

In the final part of the code, we are testing our model to see if it has successfully learned to identify different types of flowers and for this we are testing it with random images.

With the completion of the testing phase we have finally completed our whole code.

APPLICATION

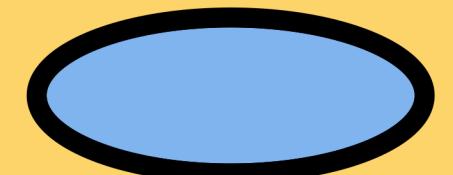
Classification of flowers is a relatively simple type of neural network since it uses a pretty typical Convolutional Neural Network (CNN). This model can be further modified to solve different types of problems like classifying different kinds of biomolecules or using computer vision to recognize and identify objects in real life.



CONTRIBUTION

All 5 members of this group contributed towards the code in every step of the process and has a deep and clear understanding of the working principle and the syntax.





THANK YOU!