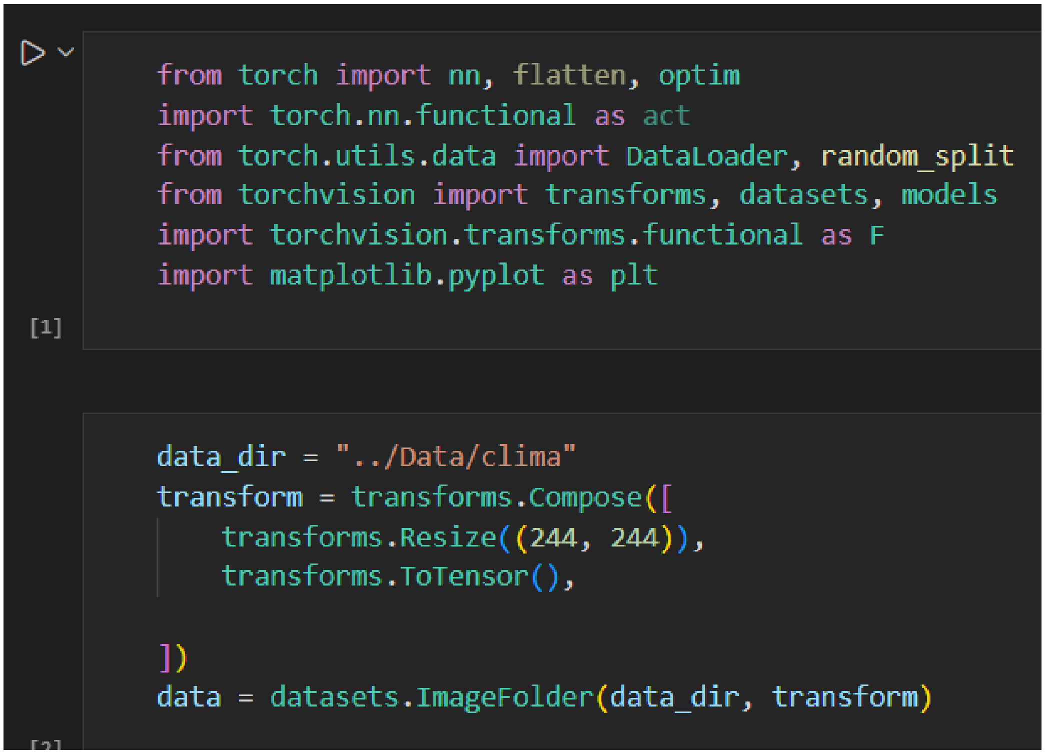
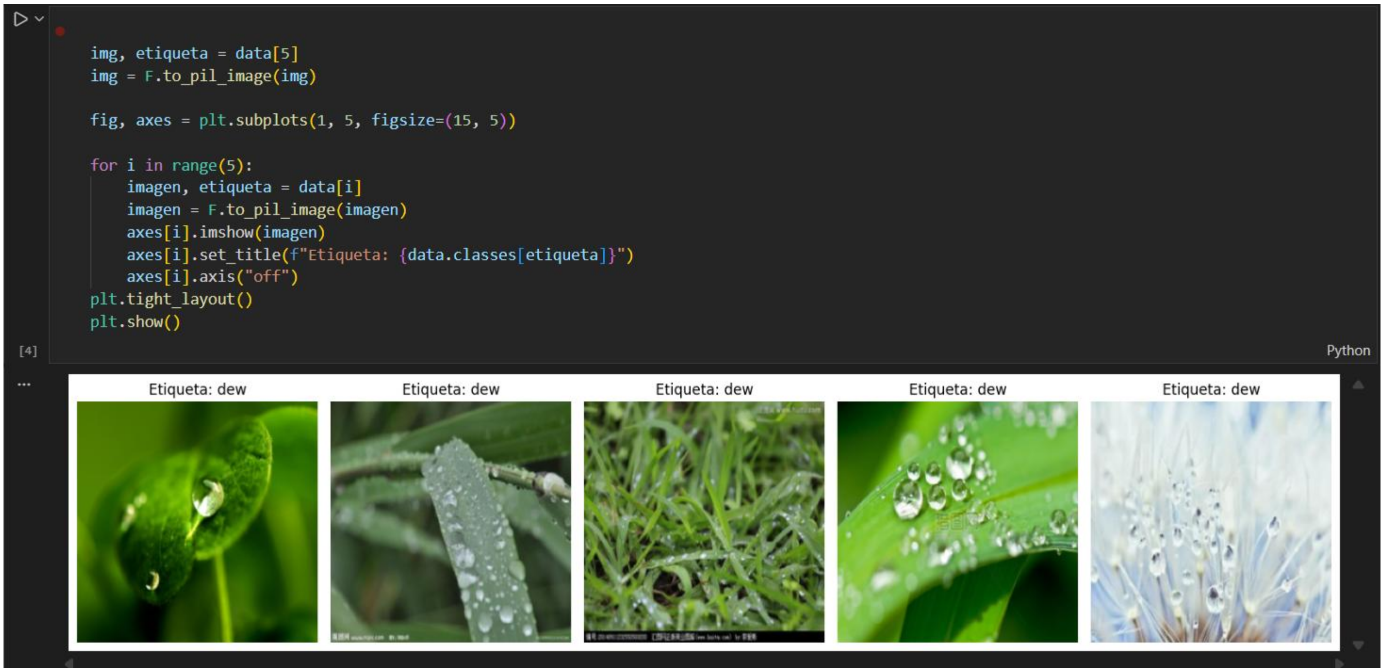
Importaciones y datos

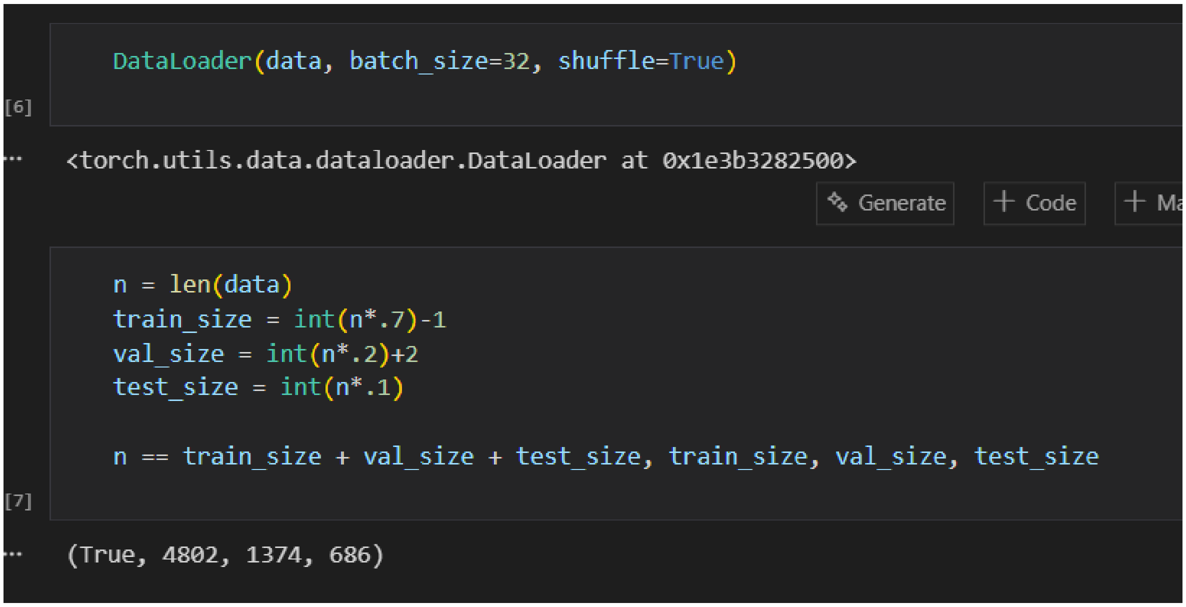


```python
from torch import nn, flatten, optim
import torch.nn.functional as act
from torch.utils.data import DataLoader, random_split
from torchvision import transforms, datasets, models
import torchvision.transforms.functional as F
import matplotlib.pyplot as plt
```

```python
data_dir = "../Data/clima"
transform = transforms.Compose([
    transforms.Resize((244, 244)),
    transforms.ToTensor(),

])
data = datasets.ImageFolder(data_dir, transform)
```



```python
img, etiqueta = data[5]
img = F.to_pil_image(img)

fig, axes = plt.subplots(1, 5, figsize=(15, 5))

for i in range(5):
    imagen, etiqueta = data[i]
    imagen = F.to_pil_image(imagen)
    axes[i].imshow(imagen)
    axes[i].set_title(f"Etiqueta: {data.classes[etiqueta]}")
    axes[i].axis("off")
plt.tight_layout()
plt.show()
```

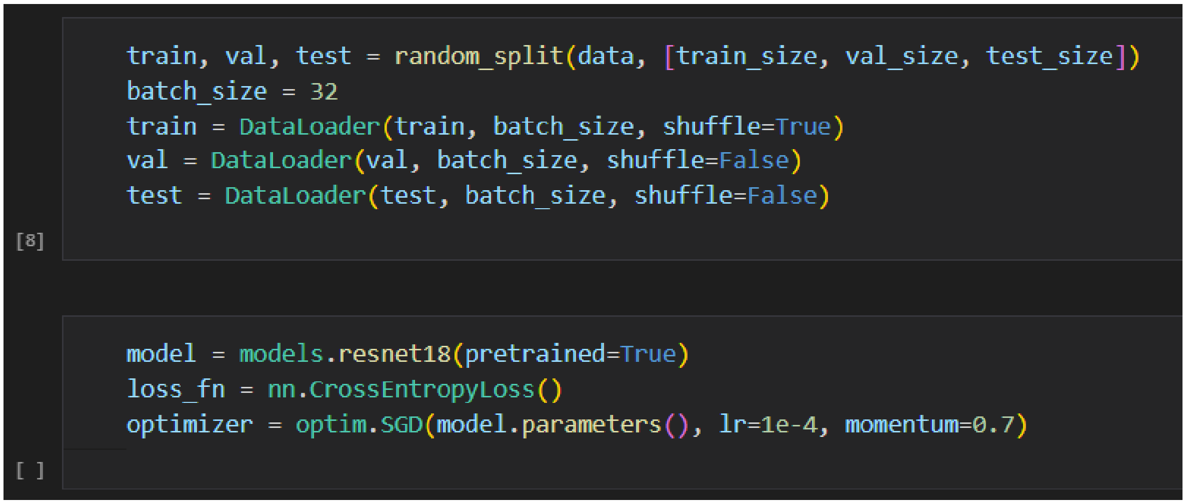


```
DataLoader(data, batch_size=32, shuffle=True)
```

```
<torch.utils.data.dataloader.DataLoader at 0x1e3b3282500>
```

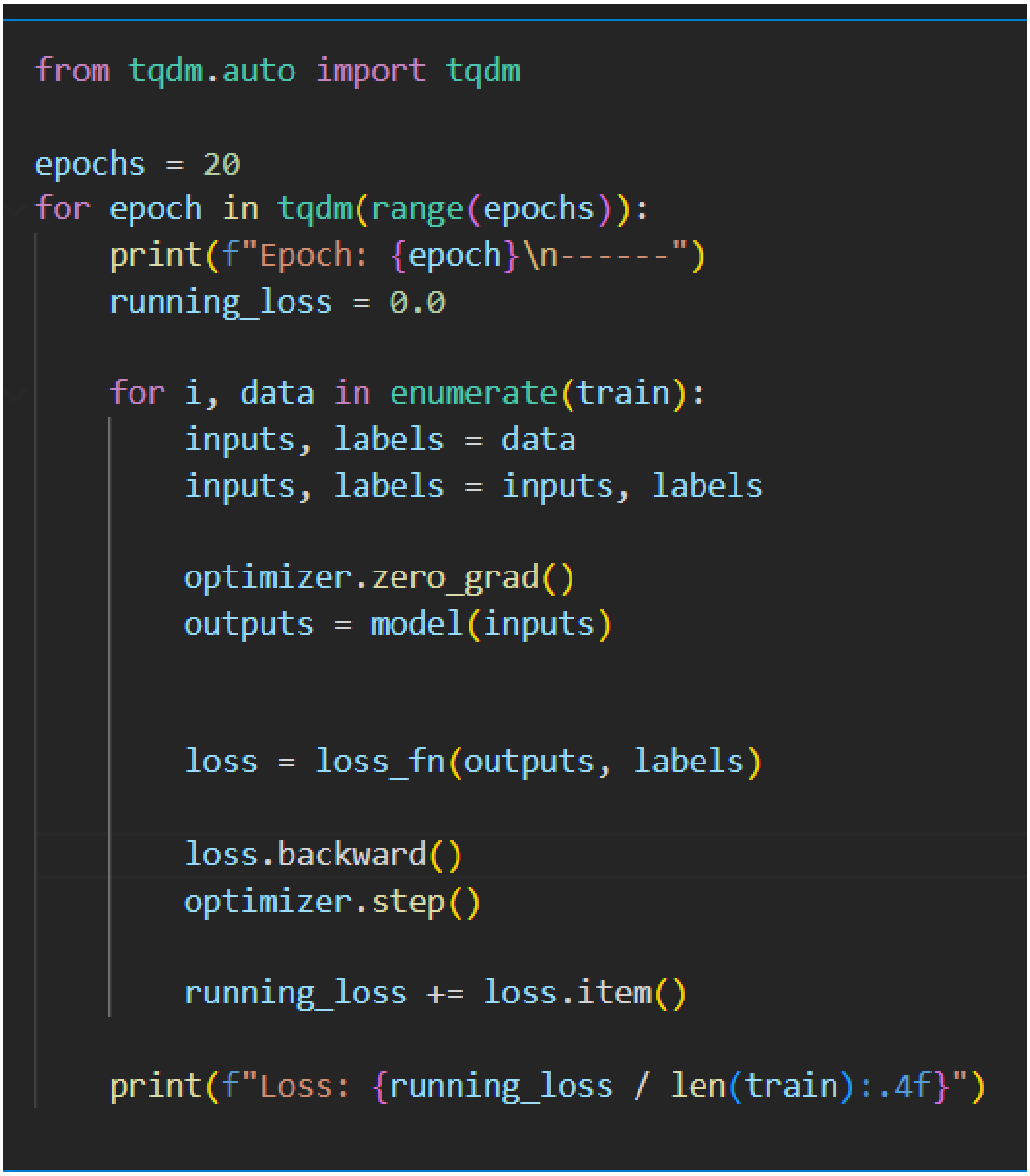
```
n = len(data)
train_size = int(n*.7)-1
val_size = int(n*.2)+2
test_size = int(n*.1)

n == train_size + val_size + test_size, train_size, val_size, test_size
```

```
(True, 4802, 1374, 686)
```

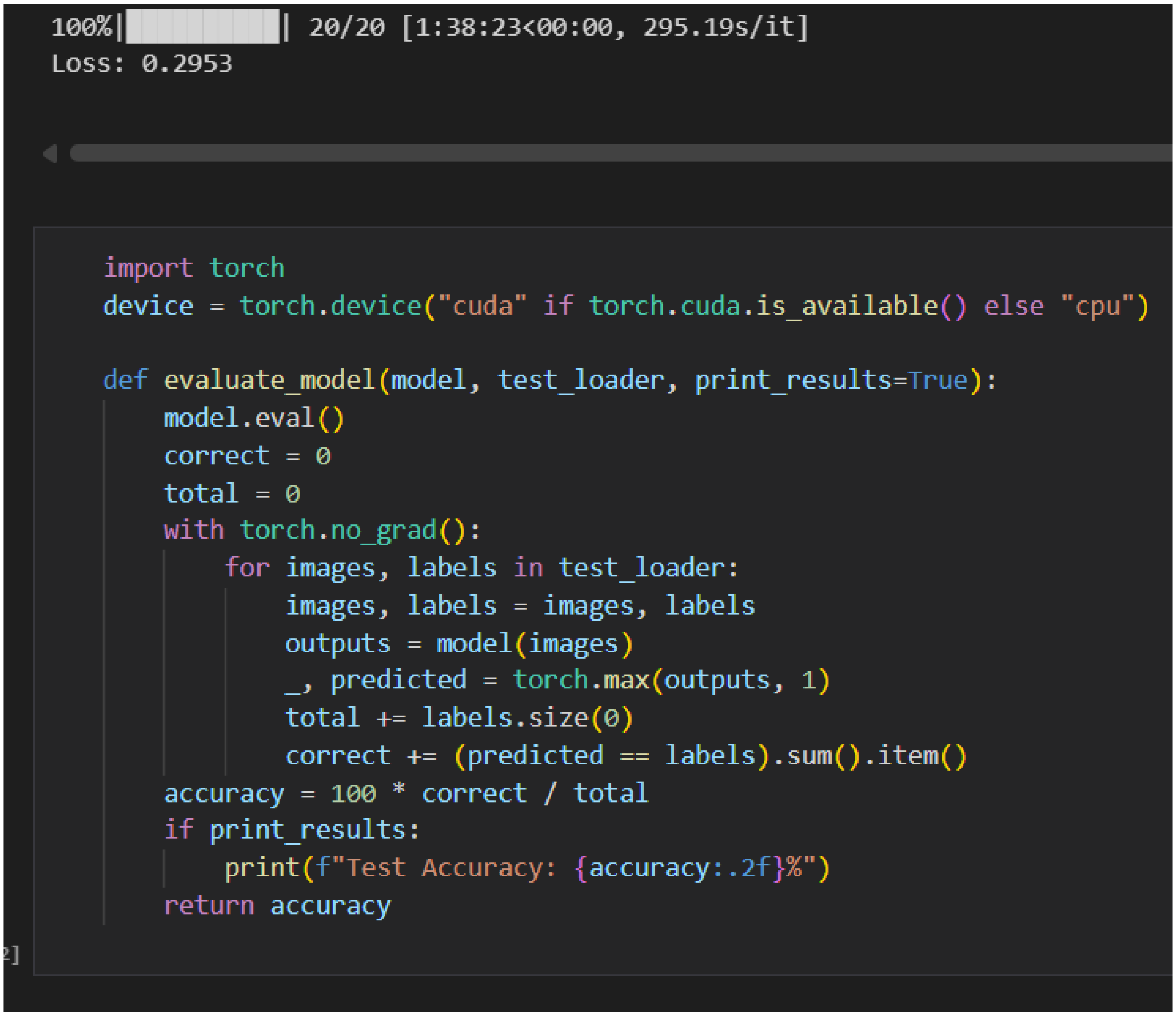


```
train, val, test = random_split(data, [train_size, val_size, test_size])
batch_size = 32
train = DataLoader(train, batch_size, shuffle=True)
val = DataLoader(val, batch_size, shuffle=False)
test = DataLoader(test, batch_size, shuffle=False)
```

```
model = models.resnet18(pretrained=True)
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=1e-4, momentum=0.7)
```

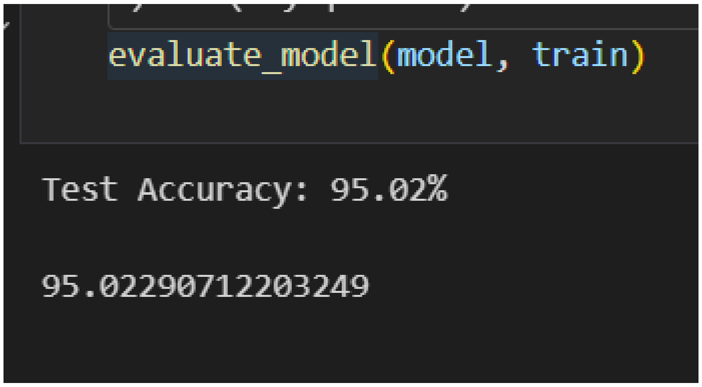


```python
from tqdm.auto import tqdm

epochs = 20
for epoch in tqdm(range(epochs)):
    print(f"Epoch: {epoch}\n------")
    running_loss = 0.0

    for i, data in enumerate(train):
        inputs, labels = data
        inputs, labels = inputs, labels

        optimizer.zero_grad()
        outputs = model(inputs)


        loss = loss_fn(outputs, labels)

        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f"Loss: {running_loss / len(train):.4f}")
```



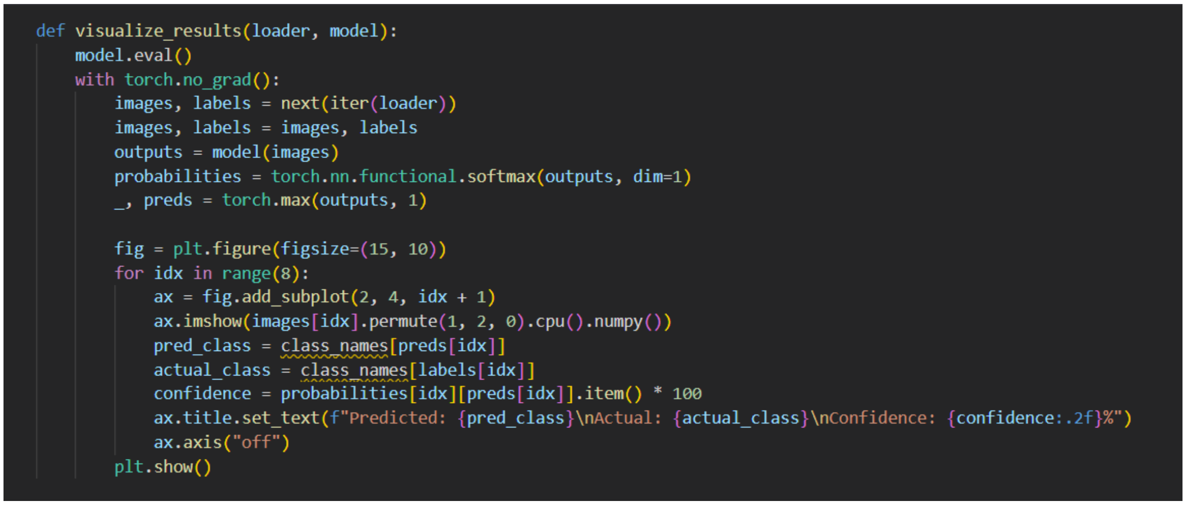
```
100%|██████████| 20/20 [1:38:23<00:00, 295.19s/it]
Loss: 0.2953
```

```python
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

def evaluate_model(model, test_loader, print_results=True):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images, labels
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    if print_results:
        print(f"Test Accuracy: {accuracy:.2f}%")
    return accuracy
```

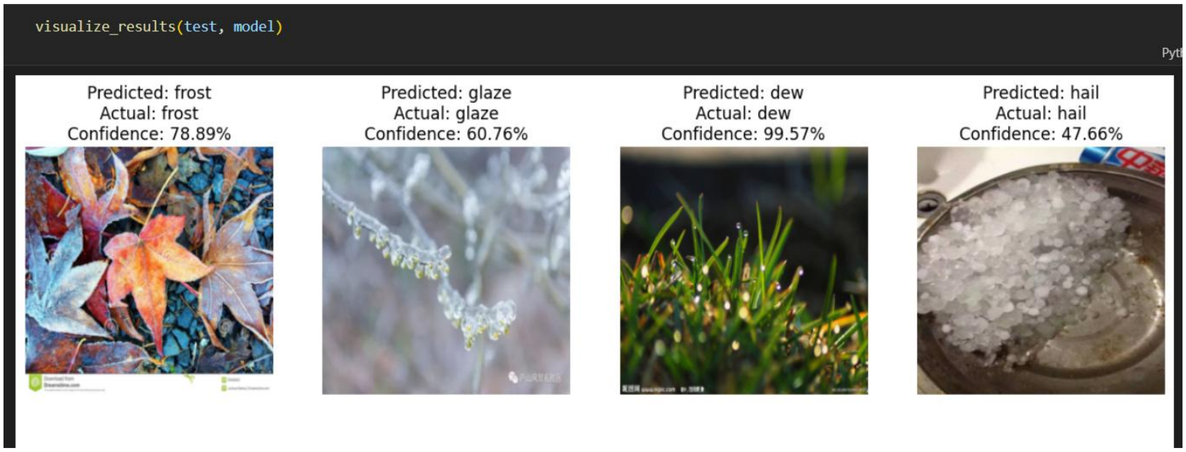


```python
evaluate_model(model, train)
```

```
Test Accuracy: 95.02%

95.02290712203249
```



```python
def visualize_results(loader, model):
    model.eval()
    with torch.no_grad():
        images, labels = next(iter(loader))
        images, labels = images, labels
        outputs = model(images)
        probabilities = torch.nn.functional.softmax(outputs, dim=1)
        _, preds = torch.max(outputs, 1)

        fig = plt.figure(figsize=(15, 10))
        for idx in range(8):
            ax = fig.add_subplot(2, 4, idx + 1)
            ax.imshow(images[idx].permute(1, 2, 0).cpu().numpy())
            pred_class = class_names[preds[idx]]
            actual_class = class_names[labels[idx]]
            confidence = probabilities[idx][preds[idx]].item() * 100
            ax.title.set_text(f"Predicted: {pred_class}\nActual: {actual_class}\nConfidence: {confidence:.2f}%")
            ax.axis("off")
        plt.show()
```

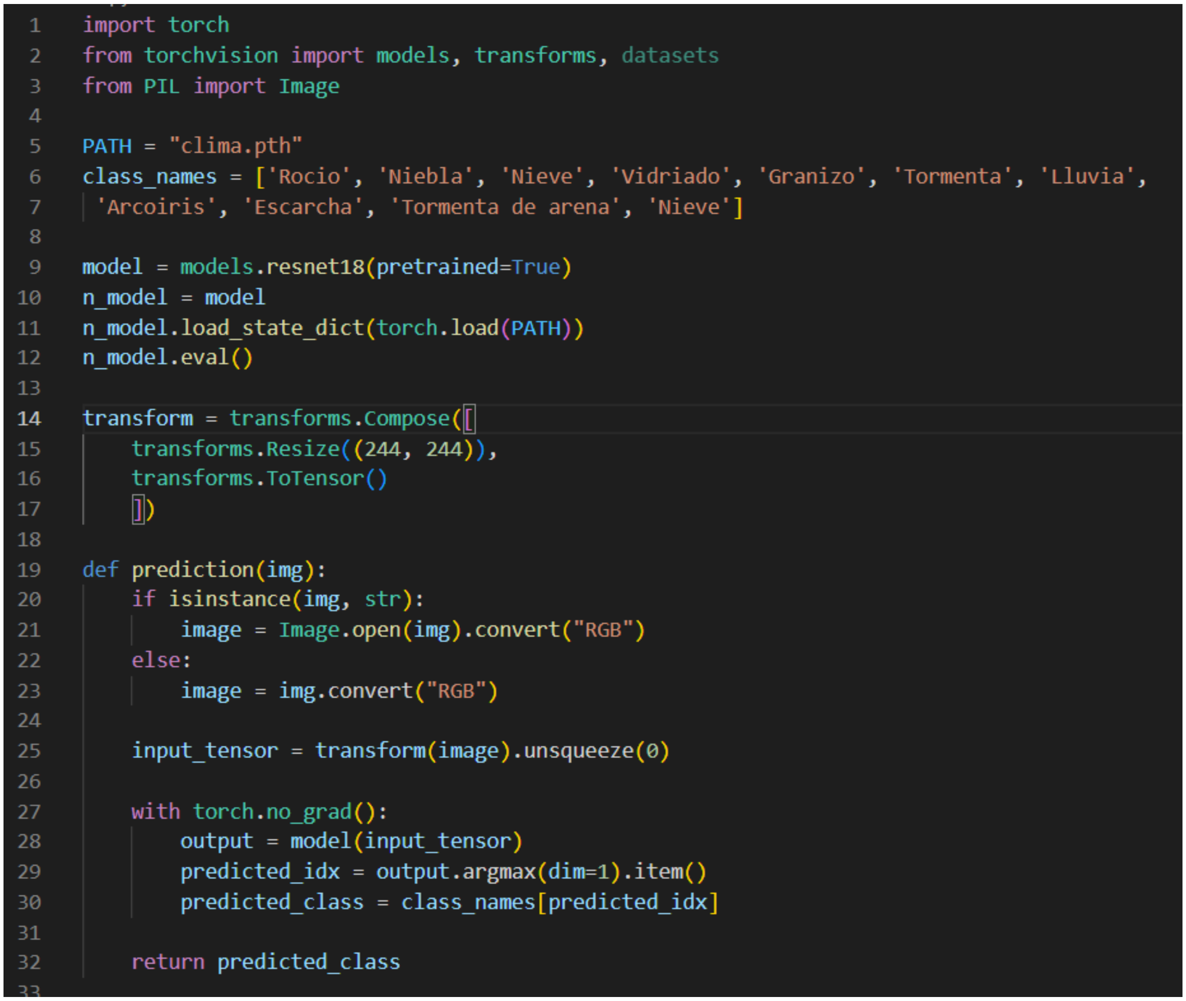


```python
visualize_results(test, model)
```

```
Predicted: frost
Actual: frost
Confidence: 78.89%

Predicted: glaze
Actual: glaze
Confidence: 60.76%

Predicted: dew
Actual: dew
Confidence: 99.57%

Predicted: hail
Actual: hail
Confidence: 47.66%
```

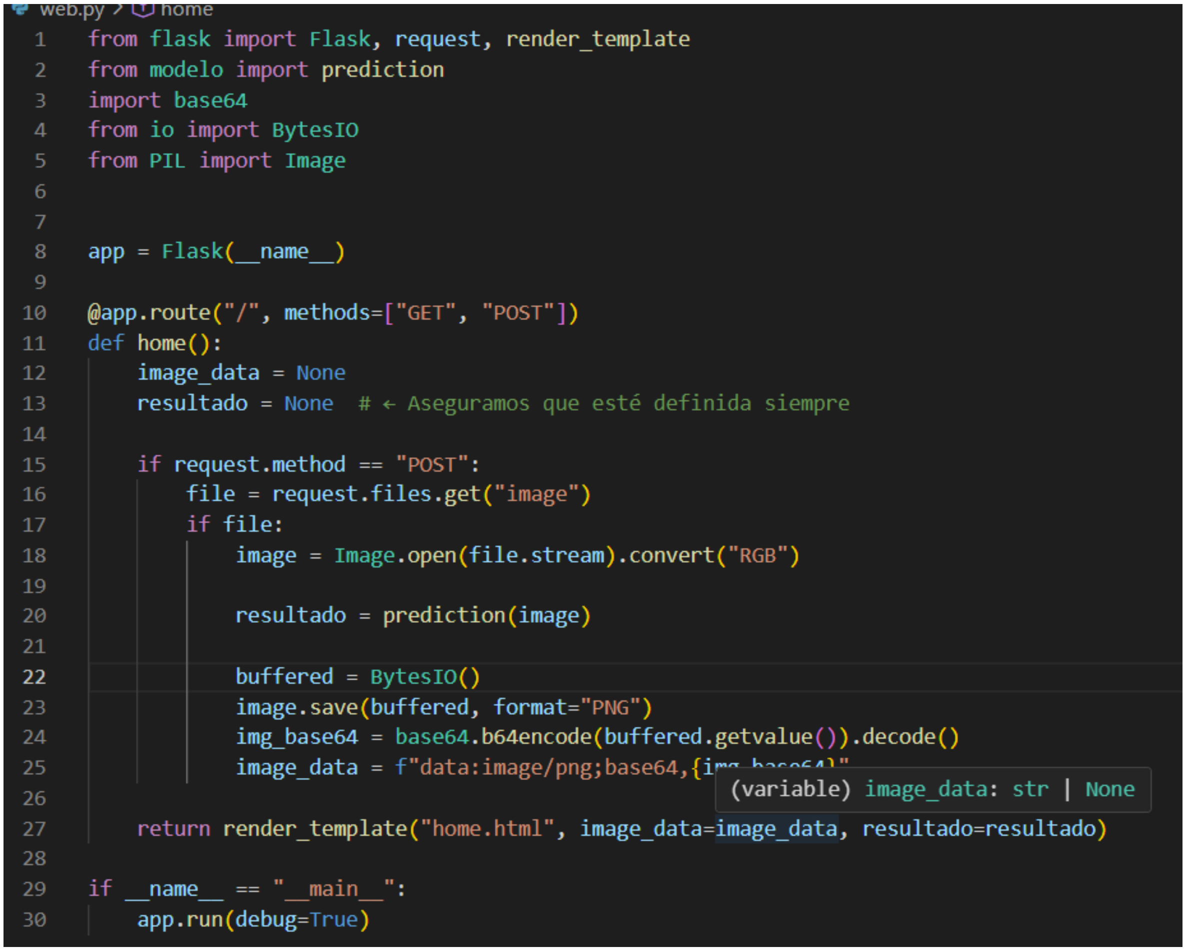
Implementación a web

1. ARCHIVO DE PREDICCION



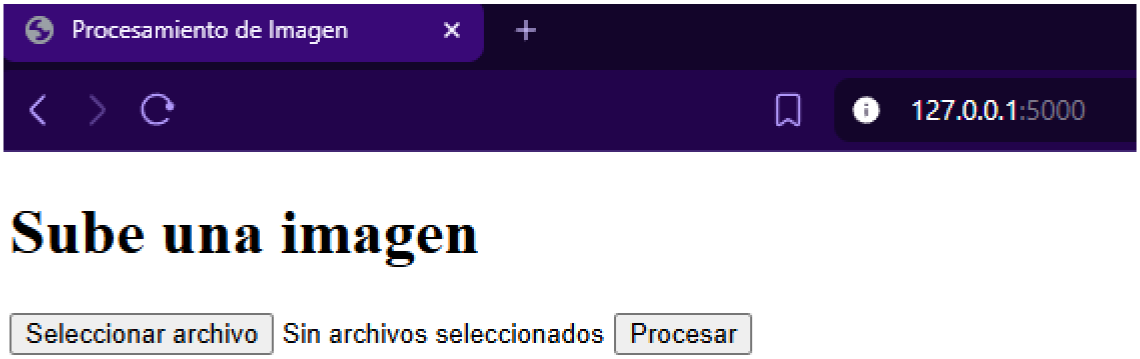
```python
import torch
from torchvision import models, transforms, datasets
from PIL import Image

PATH = "clima.pth"
class_names = ['Rocio', 'Niebla', 'Nieve', 'Vidriado', 'Granizo', 'Tormenta', 'Lluvia',
 'Arcoiris', 'Escarcha', 'Tormenta de arena', 'Nieve']

model = models.resnet18(pretrained=True)
n_model = model
n_model.load_state_dict(torch.load(PATH))
n_model.eval()

transform = transforms.Compose([
    transforms.Resize((244, 244)),
    transforms.ToTensor()
    ])

def prediction(img):
    if isinstance(img, str):
        image = Image.open(img).convert("RGB")
    else:
        image = img.convert("RGB")

    input_tensor = transform(image).unsqueeze(0)

    with torch.no_grad():
        output = model(input_tensor)
        predicted_idx = output.argmax(dim=1).item()
        predicted_class = class_names[predicted_idx]

    return predicted_class
```

2. ARCHIVO WEB

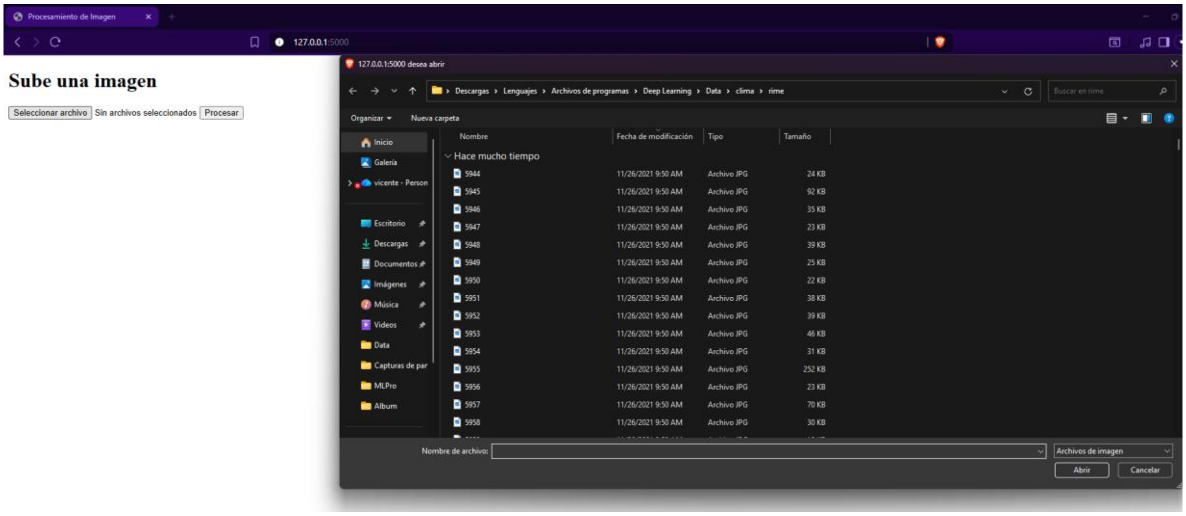


```python
from flask import Flask, request, render_template
from modelo import prediction
import base64
from io import BytesIO
from PIL import Image


app = Flask(__name__)

@app.route("/", methods=["GET", "POST"])
def home():
    image_data = None
    resultado = None  # ← Aseguramos que esté definida siempre

    if request.method == "POST":
        file = request.files.get("image")
        if file:
            image = Image.open(file.stream).convert("RGB")

            resultado = prediction(image)

            buffered = BytesIO()
            image.save(buffered, format="PNG")
            img_base64 = base64.b64encode(buffered.getvalue()).decode()
            image_data = f"data:image/png;base64,{img_base64}"

    return render_template("home.html", image_data=image_data, resultado=resultado)

if __name__ == "__main__":
    app.run(debug=True)
```

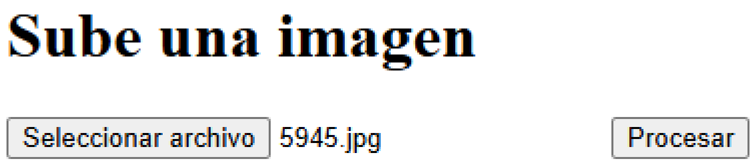
En funcionamiento



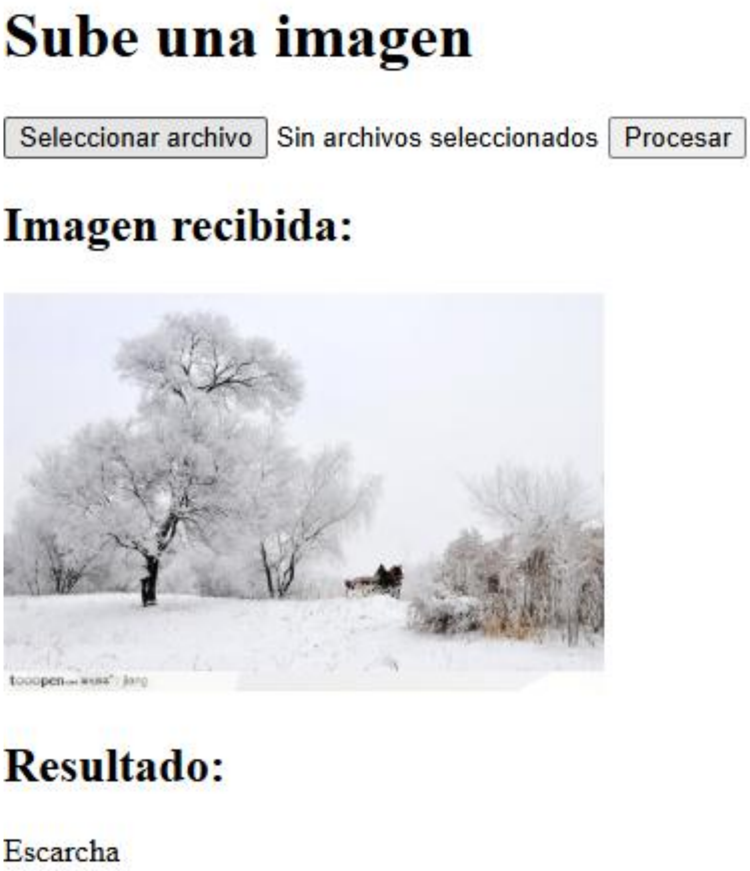
Sube una imagen



Nombre	Fecha de modificación	Tipo	Tamaño
5944	11/26/2021 9:50 AM	Archivo JPG	24 KB
5945	11/26/2021 9:50 AM	Archivo JPG	92 KB
5946	11/26/2021 9:50 AM	Archivo JPG	35 KB
5947	11/26/2021 9:50 AM	Archivo JPG	23 KB
5948	11/26/2021 9:50 AM	Archivo JPG	39 KB
5949	11/26/2021 9:50 AM	Archivo JPG	25 KB
5950	11/26/2021 9:50 AM	Archivo JPG	22 KB
5951	11/26/2021 9:50 AM	Archivo JPG	38 KB
5952	11/26/2021 9:50 AM	Archivo JPG	39 KB
5953	11/26/2021 9:50 AM	Archivo JPG	46 KB
5954	11/26/2021 9:50 AM	Archivo JPG	31 KB
5955	11/26/2021 9:50 AM	Archivo JPG	252 KB
5956	11/26/2021 9:50 AM	Archivo JPG	23 KB
5957	11/26/2021 9:50 AM	Archivo JPG	70 KB
5958	11/26/2021 9:50 AM	Archivo JPG	30 KB



Sube una imagen



Sube una imagen

Imagen recibida:

Resultado:

Escarcha