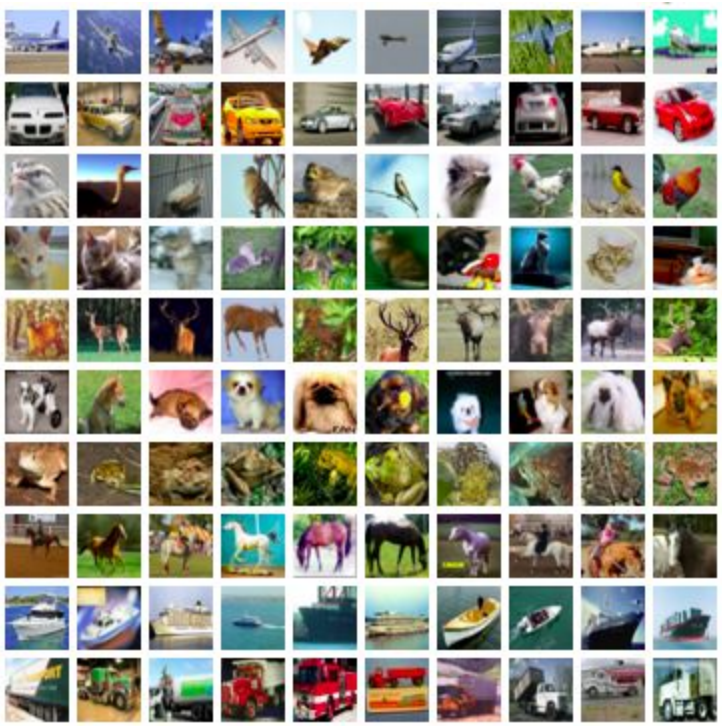
Presenter: Hayley



Technology: Flask and ML models

Goal: To use a machine learning model without making an API call

The Original Plan:

The Original Plan:

Creating images of dogs with CGANS.

This didn't work, because the dogs didn't look like dogs and also I could only ever get it to run on Kaggle.com

The New Plan

Classify Images with ResNet.

Training my own model had many limitations.

Decided to use a pretrained resnet model.

- Accept a file input with Flask
- Process the image for ResNet
- Make a prediction with ResNet

How to use ResNet in Flask

1) Pick a model that is right for you

I picked ResNet 18, a smaller network, because of computational resources.
Some of the larger ResNet models are difficult for my computer to import.

2) Load the model:

```
def load_model():
    global model
    model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet18', pretrained=True)
    model.eval()
```

How to use ResNet in Flask

3) Collect your input image in the Flask App

```
# ensure an image was properly uploaded to our endpoint
if flask.request.method == "POST":
    if flask.request.files.get("image"):

        image1 = flask.request.files["image"]
        # save the image to the upload folder, for display on the webpage.
        image = image1.save(os.path.join(app.config['UPLOAD_FOLDER'], image1.filename))

        # read the image in PIL format
        with open(os.path.join(app.config['UPLOAD_FOLDER'], image1.filename), 'rb') as f:
            image = Image.open(io.BytesIO(f.read()))
```

This portion of the code looks for a POST request from the user, and collects the image file. It saves the image to a specific Upload folder, and then reads in the image using the PIL python library

How to use ResNet in Flask

4) Collect the ResNet class labels as a txt file

```
with open("imagenet_classes.txt", "r") as f:
    categories = [s.strip() for s in f.readlines()]
```

```
 9  hen
10  ostrich
11  brambling
12  goldfinch
13  house finch
14  junco
15  indigo bunting
16  robin
17  bulbul
18  jay
19  magpie
20  chickadee
```

This can be downloaded from the pytorch documentation and isn't super large

How to use ResNet in Flask

5) Prepare your input image

Ensures the image is in RGB format
Then it performs some preprocessing

1. Resizes the image's short side to 256 pixels while maintaining the aspect ratio,
2. Crops the image at the center to a size of 224x224 pixels.
3. Converts the image into a PyTorch tensor.
4. Normalizes the tensor by subtracting the mean and dividing by the standard deviation.

Prepares the input tensor as a batch by unsqueezing it along the first dimension (input_tensor.unsqueeze(0)).

```
def prepare_image(image):
    if image.mode != "RGB":
        image = image.convert("RGB")
    preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])
    input_tensor = preprocess(image)
    input_batch = input_tensor.unsqueeze(0)
    if torch.cuda.is_available():
        input_batch = input_batch.to('cuda')
        model.to('cuda')
    # print(input_batch)
    return input_batch
```

How to use ResNet in Flask

6) Make your predictions

- Run the model on your image
- Create an array of probabilities with the softmax function
- For all probabilities, I have my app print the class and probability
- I take the top 100 and render my Flask web app with these top 100

```
with torch.no_grad():
    output = model(input_batch)
# print("works here? 2",output)
if output == None:
    data["success"] = False
else:
    data["success"] = "Your Image!"

probabilities = torch.nn.functional.softmax(output[0], dim=0)
```

```
for i in range(len(probabilities)):
    if probabilities[i] > 0:
        print("Class ", categories[i], "probability   ", probabilities[i])

# # Show top categories per image
top5_prob, top5_catid = torch.topk(probabilities, 100)
# data = []
for i in range(top5_prob.size(0)):
    r={"label":categories[top5_catid[i]], "probability":top5_prob[i].item()}
    # r = {"label": label, "probability": float(prob)}
    data["predictions"].append(r)


return render_template('index.html', data=data, title = "ResNet Result!", name=image1.filename)
```

Demo!