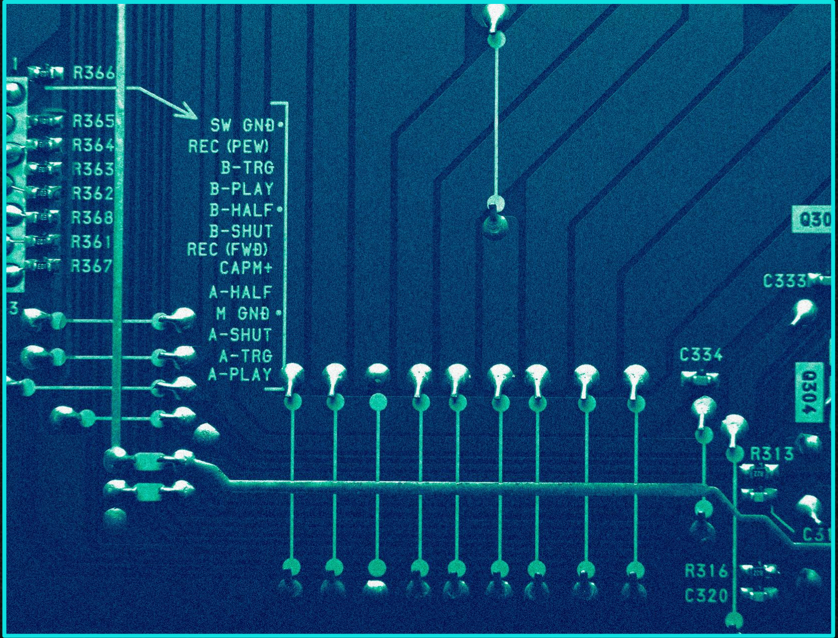
SASE x DSI Presents:

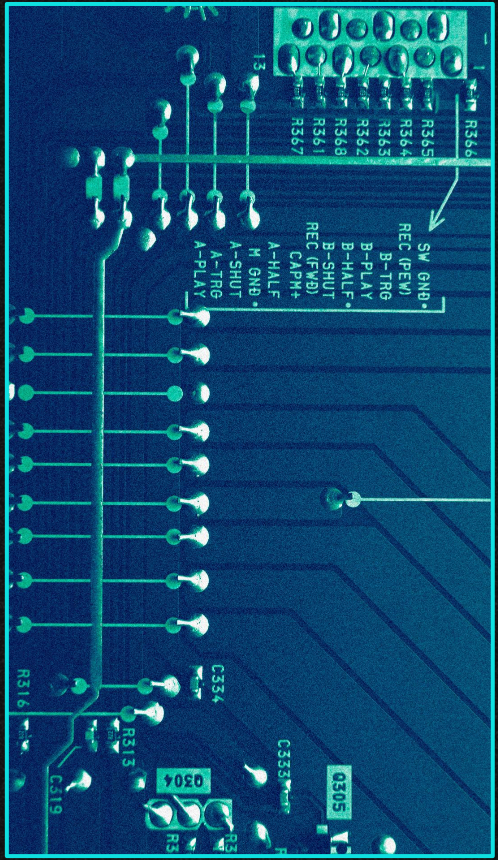
Technical Workshop #2!

THE BASICS OF DEEP LEARNING



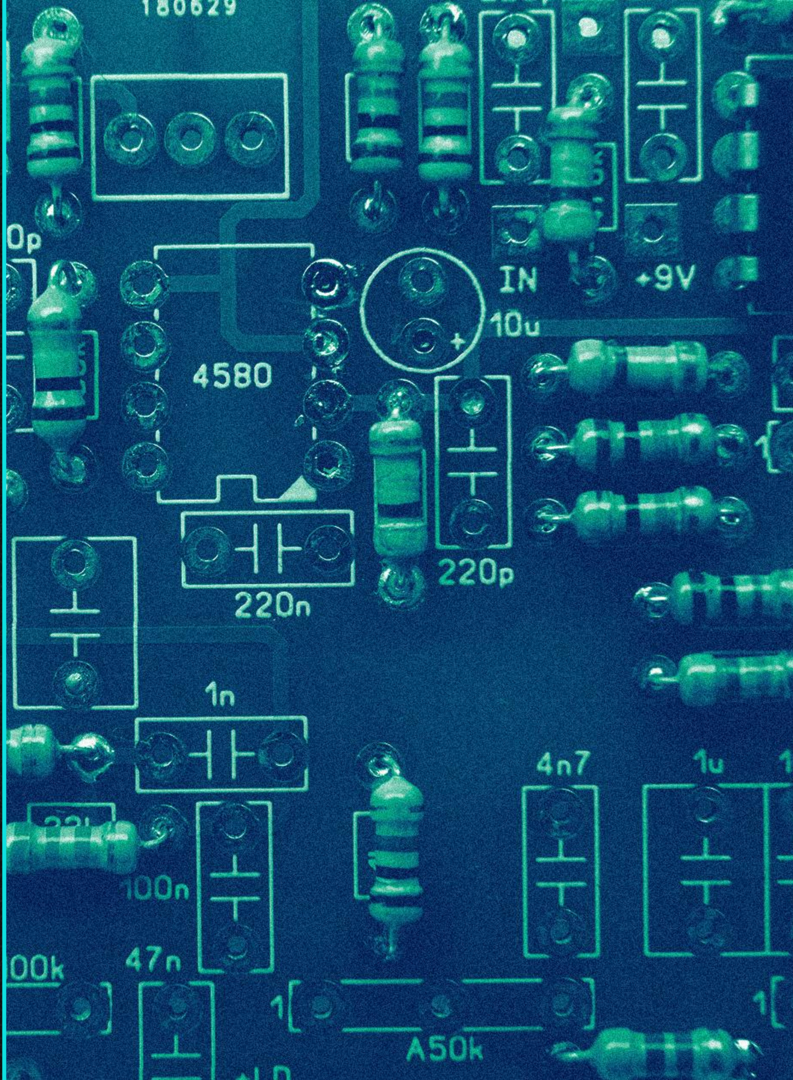
Sign in here!

Table of contents



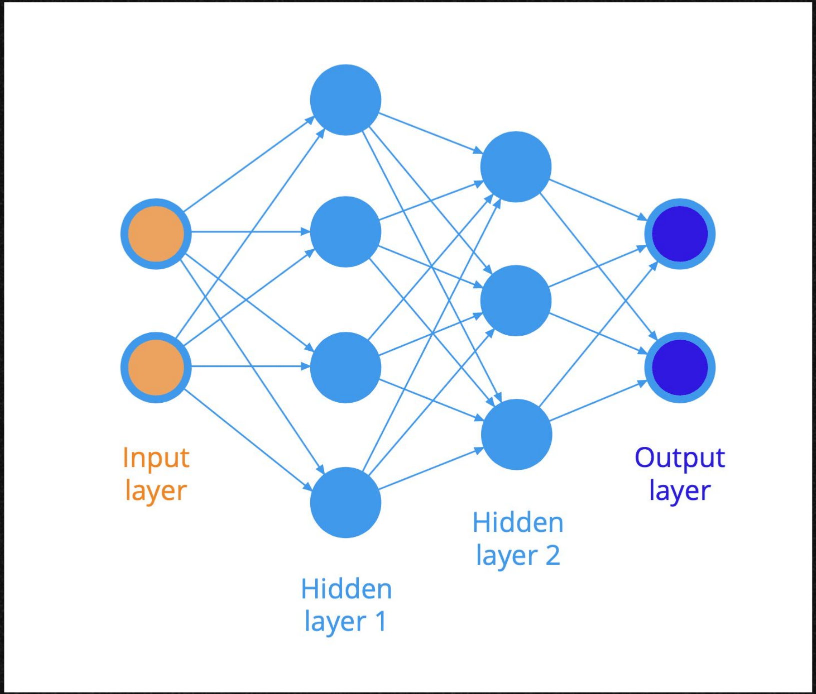
01

The Introduction



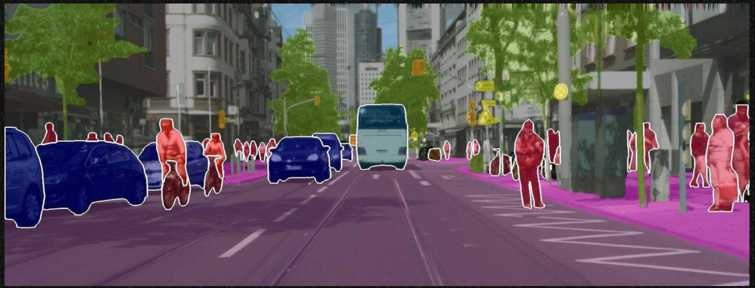
What is Deep Learning?

- Deep learning is a branch of AI that teaches computers how to process data in a similar fashion to the human brain.
- A neural network is used to achieve this
  - “Nodes” are circles in the neural network, which represent a neuron
  - “Deep” meaning the network has many “layers”
- Layers basically transform the input (image, text, etc.) over and over again until the computer understands what it is. (i.e an image is a cat)

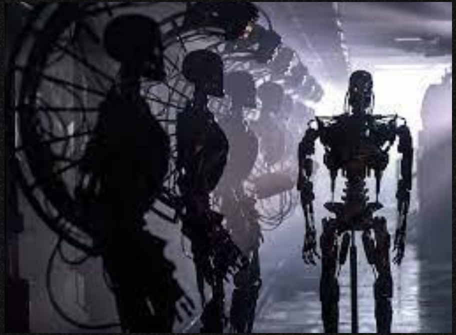


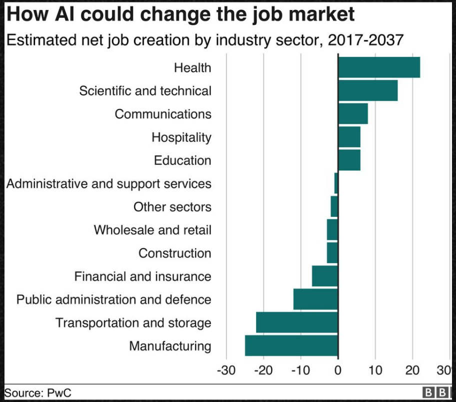
Applications of Deep Learning

- Image Classification/Recognition
- Image Segmentation/Object Detection
- Natural Language Processing
- Fraud Detection, Vocal AI, Recommendations, and much more!



So... why should you care?



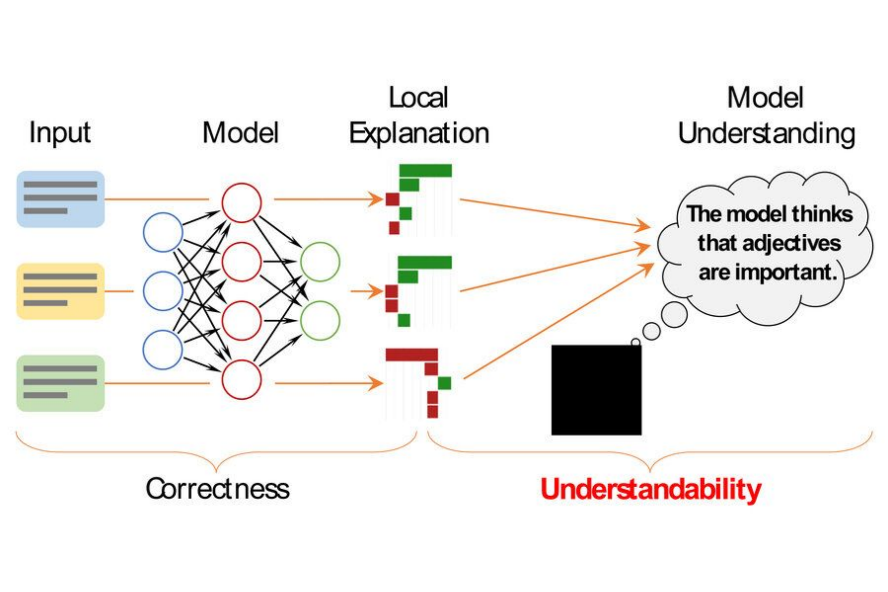


- Deep learning has countless applications across many industries
- Disruptive technology; has the potential to revolutionize how businesses operate
- “68.5% of college graduates believe AI could take their job or make it irrelevant”

How to Train Your Model

- In regular programming, you write code to program your app. In Machine Learning, you are essentially “coding” your model with the data you give it
    - **This is very important to note,** because for example, if you trained a face detection model, but only used Asian people in your data, this model would not work well with non-Asian people
    - This is called bias in training data, and very important when making a model and something people should consider before training.
    - Overfitting - another common problem in deep learning, models with many parameters (millions, billions!) can learn “too well” which means they can’t generalize

How to Train Your Model



- Once you have collected your data, someone has to manually label each one, and that could be putting it in a folder named “Cat”, or manually naming it Cat12.jpg
- Kaggle is an open source site that has data pre labeled for everyone to use!
- Once you are sure your data is ready, you can start training your model

Different kinds of Neural Networks

Recurrent

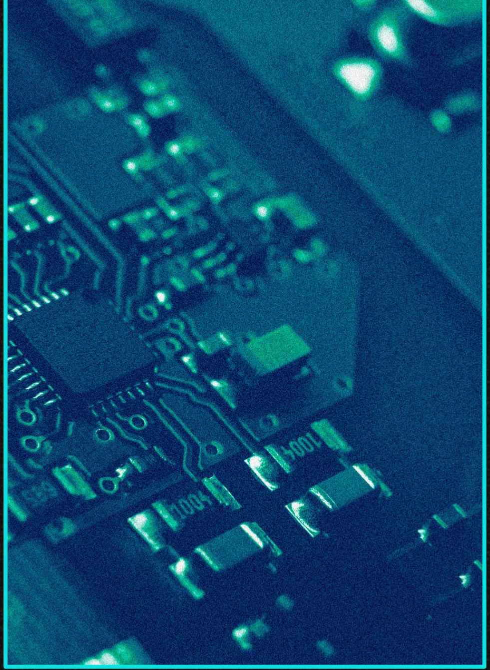
Saves a layer's output and feeds it back to the input to improve predictions; "remembers info"

Sequence to Sequence

Two recurrent neural networks consisting of an encoder and decoder

Modular

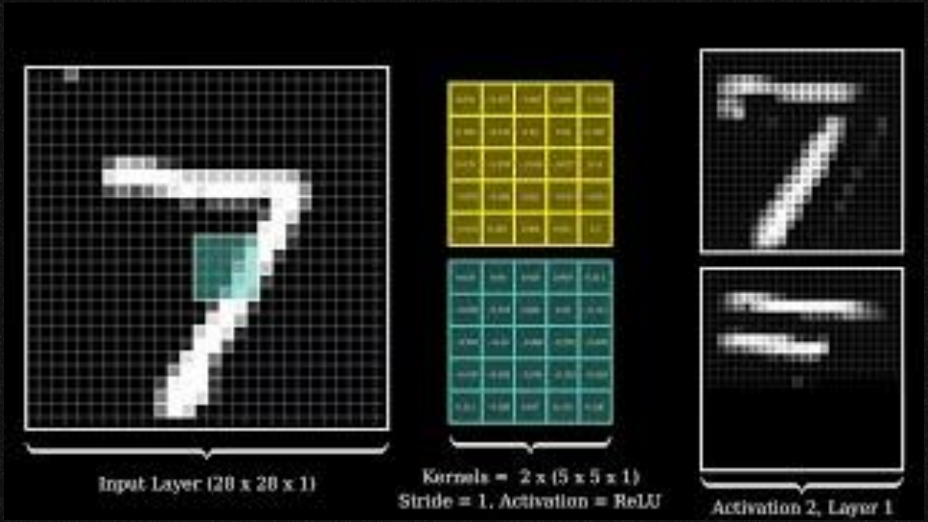
Model has different networks that function independently and perform sub-tasks



02

Convolutional Neural Network

What is a Convolutional Neural Network (CNN)?

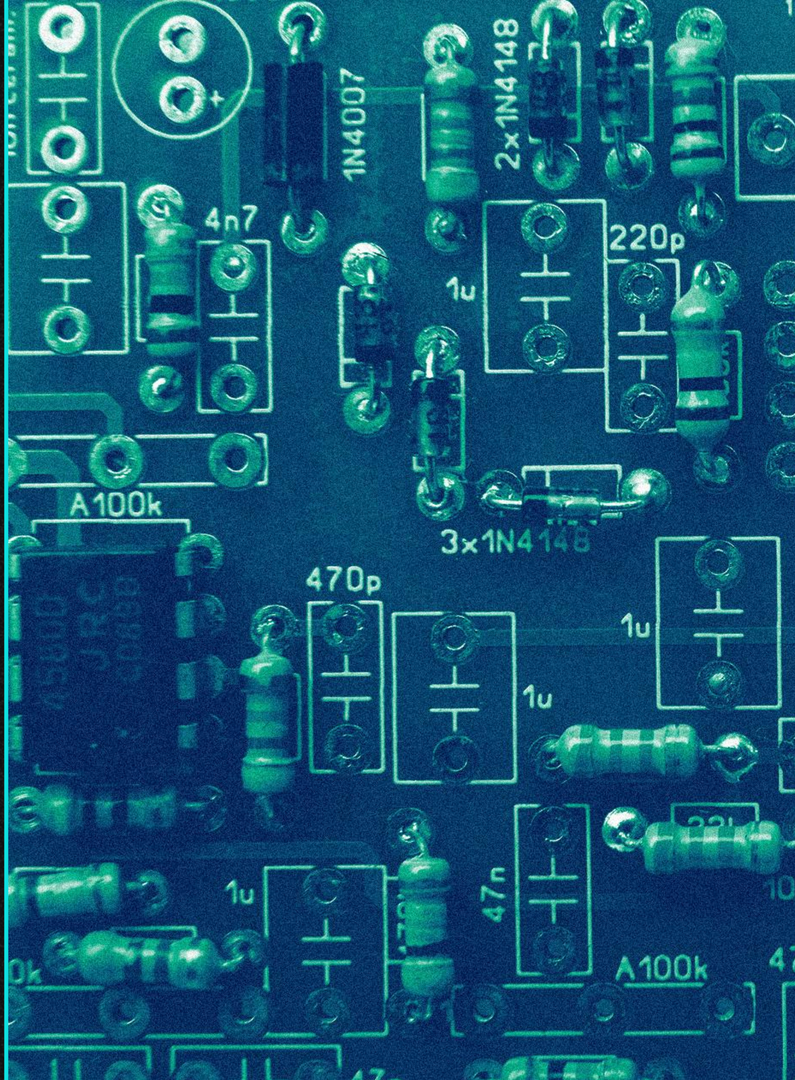


- A neural network that has a window that essentially “slides” over images to extract features from them.
- Key layers in a CNN: Conv2D, Flatten, Linear/FC
  - Conv2D: scans the image w/ filter
  - Flatten: converts the 2D image into linear data
  - Linear/FC: last step to classify data

03

Live Demo!

Let’s learn how to differentiate between cats and dogs!



Follow along with us!

Training Co-Lab

Inference Co-Lab

Device?

- Device - hardware where the computer will be doing the calculations for the model during training/inference
  - Is performed on the CPU by default
  - ‘CUDA’ is the GPU; performs much faster since the GPU enables multiple processes to be done at once, speeding up training significantly
- On Google Colab, you can enable GPU by doing Runtime › Change Runtime Type

```
#checking for device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
[ ] print(device)
```

```
cuda
```

Transforming Data

- Transformations modify the input data (e.g. resizing it, normalization, performing random crops, blurring it, cutting parts out, etc.)
    - Not everyone is Meta or Google and can collect mountains of user data
    - Data augmentation means training better models by creating additional data from what we already have, which means the model is less likely to overfit and will perform better
    - ToTensor() - turns an image into a tensor, which is the basic data structure for deep learning (it's like an array with more than 2 dimensions)

```
# Transforms
# All images are different in size, design, etc. We need to make them all "the same
transformer = transforms.Compose([
    transforms.Resize((150, 150)),
    transforms.RandomHorizontalFlip(), # 50% chance of flipping our image (essentia
    transforms.ToTensor(), #0-255 to 0-1, numpy to tensors(required for pytorch)
    transforms.Normalize([0.5, 0.5, 0.5], # 0 - 1 to [-1, 1] , formula (x-mean) /st
                         [0.5, 0.5, 0.5])
])
```

DataLoaders

- DataLoader - PyTorch class used for loading data from a dataset during training/testing, during training the dataloader will shuffle the data and apply transformations, so during one epoch the same image might be flipped horizontally but might not be in another one
    - Models will process every image in a batch before updating itself to “learn”, so batch size specifies how many images are in one batch (larger batch size requires larger computational resources, but will train faster)

```
#Dataloader

#Path for training and testing data
train_path = '/content/drive/MyDrive/WorkshopData/train'
test_path = '/content/drive/MyDrive/WorkshopData/test'

# Loading the training data
train_loader = DataLoader(
    torchvision.datasets.ImageFolder(train_path, transform=transformer),
    batch_size=16, shuffle=True
)

#Loading test data
test_loader = DataLoader(
    torchvision.datasets.ImageFolder(test_path, transform=transformer),
    batch_size=16, shuffle=True
)
```

Neural Network Structure

- Coding a neural network means defining its constructor (__init__) and the forward() function.
  - __init__ will define the layers of the neural network
    - Conv2d - 2-D convolutional layer that scans over the image using the given parameters to find important features
    - BatchNorm2d - normalizes the data to improve model performance/reduce overfitting
    - ReLU - stands for rectified linear unit, pretty much just allows the model to learn non-linear data
    - MaxPool2d - extracts the largest value from a patch to create a smaller tensor
  - forward() passes the input through each layer of the network and returns the output as logits (probabilities that the image is of a certain class)

Optimizers and Loss Functions

- Optimizers find the values that minimize a loss function as much as possible.
  - This allows for the model to be updated and thus improve its performance during training
- Loss functions calculate how far off the model is from the actual truth, lower is better

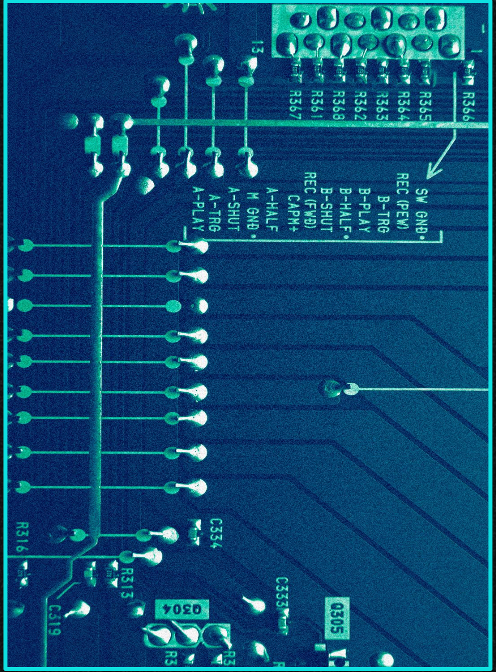
```
#Optimizer and loss function
optimizer=Adam(model.parameters(), lr=0.001, weight_decay=0.0001)
loss_function=nn.CrossEntropyLoss()
```

Training Loop

- Five Steps During Training:
    1. Forward Pass/outputs = model(images) - passing the batch of images through the layers of the model and generating predictions
    2. Calculate Loss/loss = loss_function(outputs, labels) - loss function will compare the model's outputs to the ground truth and determine how wrong the model is
    3. Zero Gradients/optimizer.zero_grad() - sets the optimizer's values to zero so they can be recalculated for the specific training step
    4. Backpropagation/loss.backward() - estimates how much the loss will change after each parameter of the model is updated
    5. Update the optimizer/optimizer.step()- updates the parameters of the model so they can be better for next batch

Validation

- Validation - monitoring the progress of the model with additional data that the model hasn’t been trained on
    - Allows us to see whether the model is overfitting, learning successfully, or not learning yet
    - If the model performs really well on training but is garbage during validation, the model is probably overfitting



Making Predictions

04

Neural Network Structure Pt 2

To use our model we saved, we will need to define the training path + prediction path. Then, the code for the CNN constructor needs to be copied over as well as the transformer.

Variables that need to be Defined:

- Training Path: To get the classes
- Prediction Path: Path to where the pictures are to predict them

Code to Copy over:

- Transformer: Used to transform new images to feed into our CNN
- CNN Structure: Used to define the structure that we will load the model to.

Loading the Model

```
checkpoint=torch.load('/content/drive/MyDrive/WorkshopData/saved_checkpoint.model')
model=ConvNet(num_classes=2)
model.load_state_dict(checkpoint)
model.eval()
```

```
ConvNet(
  (conv1): Conv2d(3, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(12, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu1): ReLU()
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(12, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (relu2): ReLU()
  (conv3): Conv2d(20, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu3): ReLU()
  (fc): Linear(in_features=180000, out_features=2, bias=True)
)
```

Previously in our training loop, we saved the model as we went with the best accuracy. We need to load it again with the torch.load('…') function

Then, we initialize our model with 2 classes and load the checkpoint into it.

model.eval() is used to ensure that the structure of our CNN is still the same

Prediction Function

```
#prediction function
def prediction(img_path,transformer):

    image=Image.open(img_path)

    image_tensor=transformer(image).float()


    image_tensor=image_tensor.unsqueeze_(0)

    if torch.cuda.is_available():
        image_tensor.cuda()

    input=Variable(image_tensor)


    output=model(input)

    index=output.data.numpy().argmax()

    pred=classes[index]

    return pred
```

- Uses the transformer to get our image to the same specification as our model
- We pass the image as input into the model and get an output and return it.

Using the model

```
[29] images_path=glob.glob(pred_path+'/*.jpg')

[30] pred_dict={}

     for i in images_path:
         pred_dict[i[i.rfind('/')+1:]]=prediction(i,transformer)

[31] pred_dict
```

[29] Gets the paths to all the images we have in a list called images_path

[30] For every image path, pass that image into the prediction function, and store the result in the predictions dictionary

[31] View the content of 'pred_dict'

Resources

If you want to learn more or follow along, here's some resources to look at!

Source Code:

- Training: https://colab.research.google.com/drive/1c_Xm6DHD8wRL73NAXDNrQS3mWhG2NxI-?usp=sharing
- Inference: https://colab.research.google.com/drive/1T3r2vT7ayN_aUvBYYgpUgEUIKGaJuMAk?usp=sharing

More Resources:

- Deep Learning Basics by Lex Fridman
- Deep Learning: State of the Art by Lex Fridman
- Zero to Mastery (in-depth PyTorch tutorials)

Thanks!

For any questions, please contact

Sharika, Tam or Matthew on Discord!