Diffusion Generative AI for Computer Vision and Science

—— Lecture 6：Machine Learning Frameworks and Tools

Teaching Assistants: Kaihui Cheng

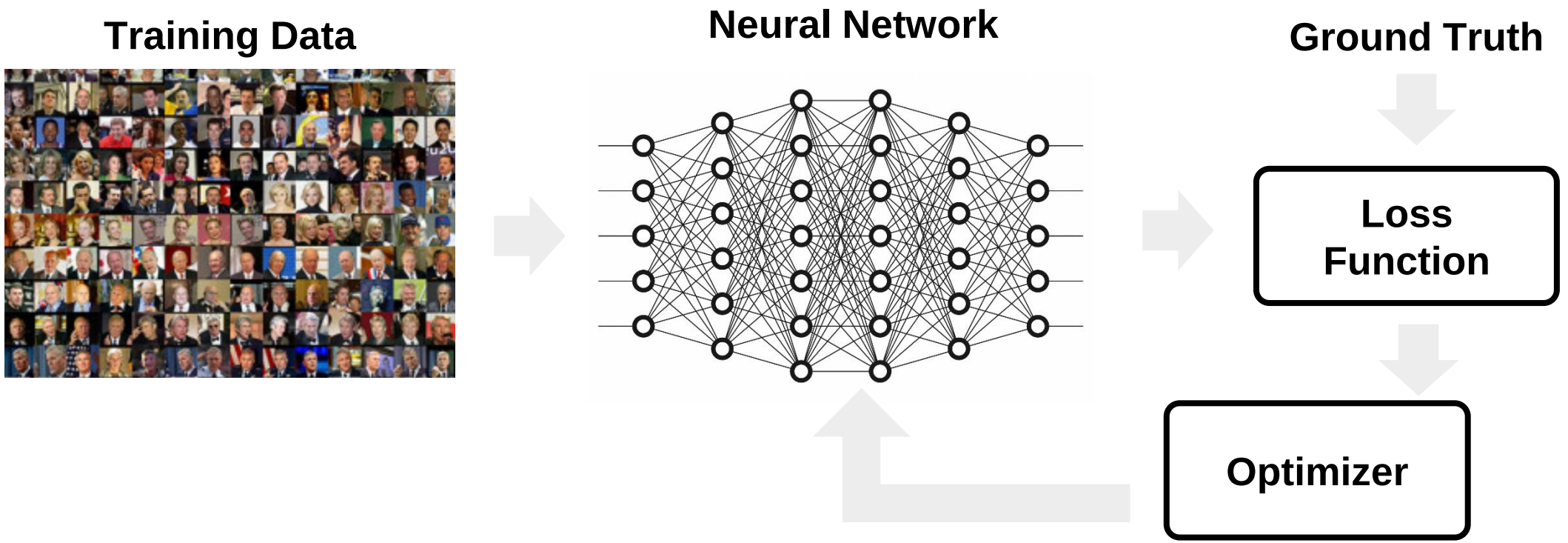
AI institute, Fudan University

Contents

- **Introduce to Pytorch**
- Project Guidelines
  - Project Expectations
  - Diffusion Demo
  - Picking a project idea
  - Final Report

Pytorch

- **Pipeline to train your model.**



Initializing a Tensor

Directly from data

Tensors can be created directly from data. The data type is automatically inferred.

```
data = [[1, 2],[3, 4]]
x_data = torch.tensor(data)
```

From a NumPy array

Tensors can be created from NumPy arrays (and vice versa - see Bridge with NumPy).

```
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
```

From another tensor:

The new tensor retains the properties (shape, datatype) of the argument tensor, unless explicitly overridden.

```
x_ones = torch.ones_like(x_data) # retains the properties of x_data
print(f"Ones Tensor: \n {x_ones} \n")

x_rand = torch.rand_like(x_data, dtype=torch.float) # overrides the datatype
of x_data
print(f"Random Tensor: \n {x_rand} \n")
```

Out:

```
Ones Tensor:
 tensor([[1, 1],
        [1, 1]])

Random Tensor:
 tensor([[0.8823, 0.9150],
        [0.3829, 0.9593]])
```

data from https://pytorch.org/tutorials/

Attributes of a Tensor

Tensor attributes describe their shape, datatype, and the device on which they are stored.

```
tensor = torch.rand(3,4)

print(f"Shape of tensor: {tensor.shape}")
print(f"Datatype of tensor: {tensor.dtype}")
print(f"Device tensor is stored on: {tensor.device}")
```

Out:

```
Shape of tensor: torch.Size([3, 4])
Datatype of tensor: torch.float32
Device tensor is stored on: cpu
```

data from https://pytorch.org/tutorials/

Operations on Tensors

By default, tensors are created on the CPU. We need to explicitly move tensors to the GPU using `.to` method (after checking for GPU availability). Keep in mind that copying large tensors across devices can be expensive in terms of time and memory!

```
# We move our tensor to the GPU if available
if torch.cuda.is_available():
    tensor = tensor.to("cuda")
```

data from https://pytorch.org/tutorials/

Operations on Tensors

Standard numpy-like indexing and slicing:

```
tensor = torch.ones(4, 4)
print(f"First row: {tensor[0]}")
print(f"First column: {tensor[:, 0]}")
print(f"Last column: {tensor[..., -1]}")
tensor[:,1] = 0
print(tensor)
```

Out:

```
First row: tensor([1., 1., 1., 1.])
First column: tensor([1., 1., 1., 1.])
Last column: tensor([1., 1., 1., 1.])
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])
```

Arithmetic operations

```
# This computes the matrix multiplication between two tensors. y1, y2, y3 will
have the same value
# ``tensor.T`` returns the transpose of a tensor
y1 = tensor @ tensor.T
y2 = tensor.matmul(tensor.T)

y3 = torch.rand_like(y1)
torch.matmul(tensor, tensor.T, out=y3)


# This computes the element-wise product. z1, z2, z3 will have the same value
z1 = tensor * tensor
z2 = tensor.mul(tensor)

z3 = torch.rand_like(tensor)
torch.mul(tensor, tensor, out=z3)
```

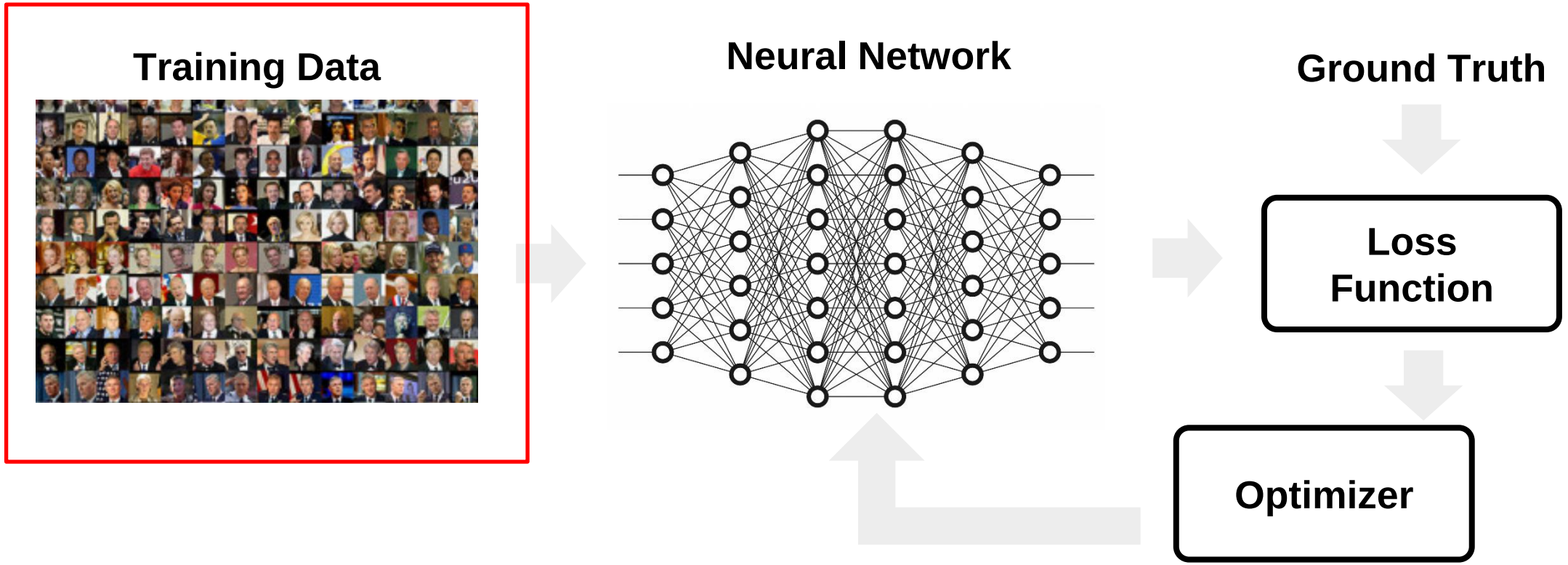
Out:

```
tensor([[1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.],
        [1., 0., 1., 1.]])
```

data from https://pytorch.org/tutorials/

Pytorch

- **Pipeline to train your model.**



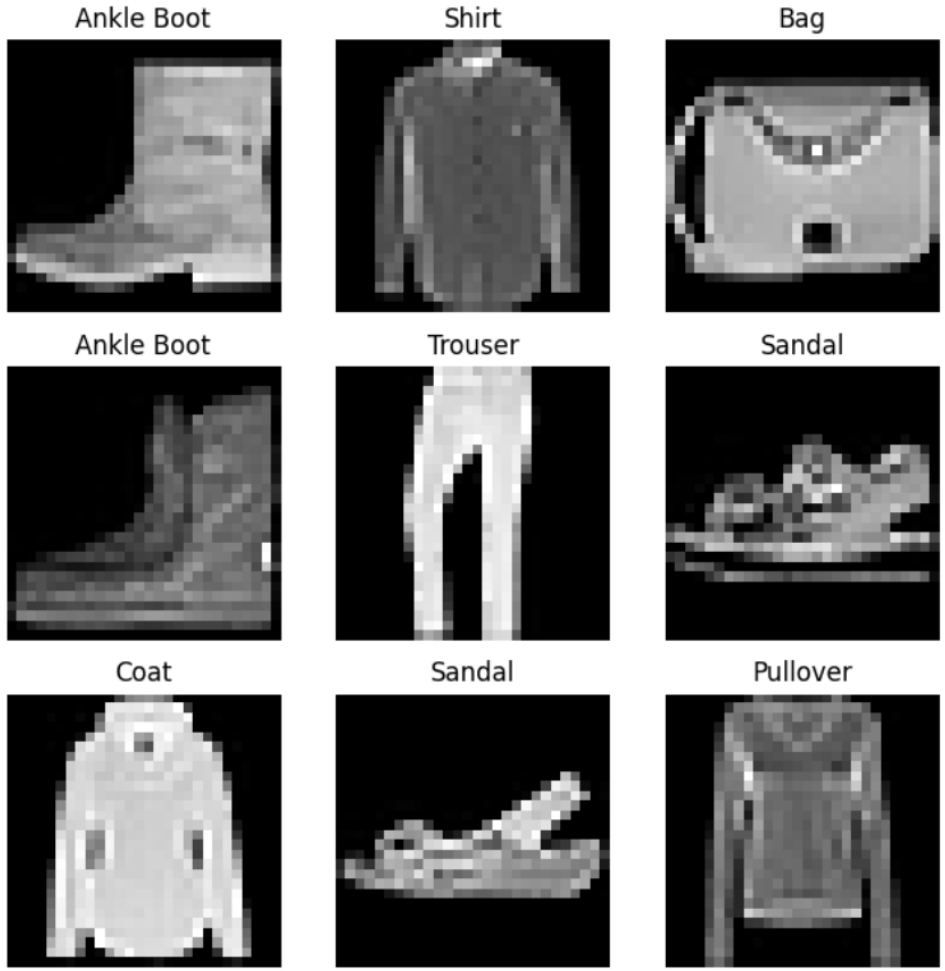
Loading a Dataset

```
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt


training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

data from https://pytorch.org/tutorials/

Iterating and Visualizing the Dataset

```
labels_map = {
    0: "T-Shirt",
    1: "Trouser",
    2: "Pullover",
    3: "Dress",
    4: "Coat",
    5: "Sandal",
    6: "Shirt",
    7: "Sneaker",
    8: "Bag",
    9: "Ankle Boot",
}
figure = plt.figure(figsize=(8, 8))
cols, rows = 3, 3
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(training_data), size=(1,)).item()
    img, label = training_data[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(labels_map[label])
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```



data from https://pytorch.org/tutorials/

Creating a Custom Dataset for your files

```python
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

data from https://pytorch.org/tutorials/

Creating a Custom Dataset for your files

```python
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

data from https://pytorch.org/tutorials/

Creating a Custom Dataset for your files

```python
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

data from https://pytorch.org/tutorials/

Creating a Custom Dataset for your files

```python
import os
import pandas as pd
from torchvision.io import read_image

class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        self.img_labels = pd.read_csv(annotations_file)
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image = read_image(img_path)
        label = self.img_labels.iloc[idx, 1]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

data from https://pytorch.org/tutorials/

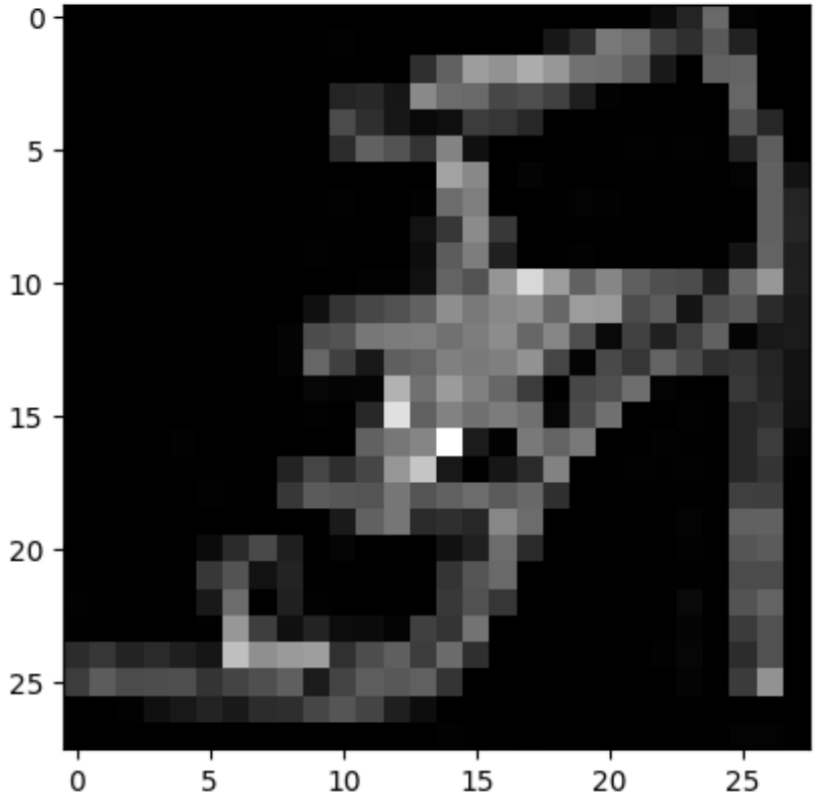
Preparing your data for training with DataLoaders

The `Dataset` retrieves our dataset's features and labels one sample at a time. While training a model, we typically want to pass samples in "minibatches", reshuffle the data at every epoch to reduce model overfitting, and use Python's `multiprocessing` to speed up data retrieval.

`DataLoader` is an iterable that abstracts this complexity for us in an easy API.

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

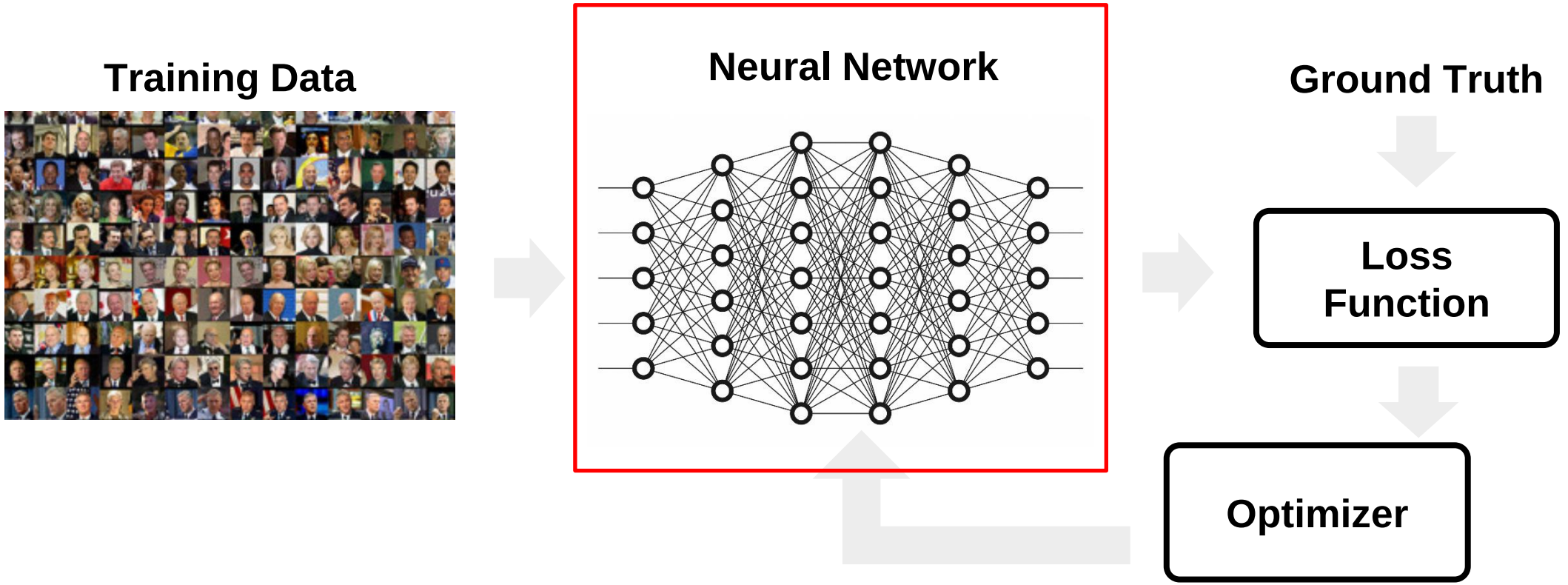
```
# Display image and label.
train_features, train_labels = next(iter(train_dataloader))
print(f"Feature batch shape: {train_features.size()}")
print(f"Labels batch shape: {train_labels.size()}")
img = train_features[0].squeeze()
label = train_labels[0]
plt.imshow(img, cmap="gray")
plt.show()
print(f"Label: {label}")
```



data from https://pytorch.org/tutorials/

Pytorch

- **Pipeline to train your model.**



data from https://pytorch.org/tutorials/

Build a Neural Network

```
import os
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
```

We want to be able to train our model on a hardware accelerator like the GPU or MPS, if available. Let's check to see if torch.cuda or torch.backends.mps are available, otherwise we use the CPU.

```
device = (
    "cuda"
    if torch.cuda.is_available()
    else "mps"
    if torch.backends.mps.is_available()
    else "cpu"
)
print(f"Using {device} device")
```

data from https://pytorch.org/tutorials/

Build a Neural Network

We define our neural network by subclassing `nn.Module`, and initialize the neural network layers in `__init__`. Every `nn.Module` subclass implements the operations on input data in the `forward` method.

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

data from https://pytorch.org/tutorials/

Build a Neural Network

We create an instance of `NeuralNetwork`, and move it to the `device`, and print its structure.

```
model = NeuralNetwork().to(device)
print(model)
```

Out:

```
NeuralNetwork(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

To use the model, we pass it the input data. This executes the model's `forward`, along with some background operations. Do not call `model.forward()` directly!

Calling the model on the input returns a 2-dimensional tensor with dim=0 corresponding to each output of 10 raw predicted values for each class, and dim=1 corresponding to the individual values of each output. We get the prediction probabilities by passing it through an instance of the `nn.Softmax` module.

```
X = torch.rand(1, 28, 28, device=device)
logits = model(X)
pred_probab = nn.Softmax(dim=1)(logits)
y_pred = pred_probab.argmax(1)
print(f"Predicted class: {y_pred}")
```

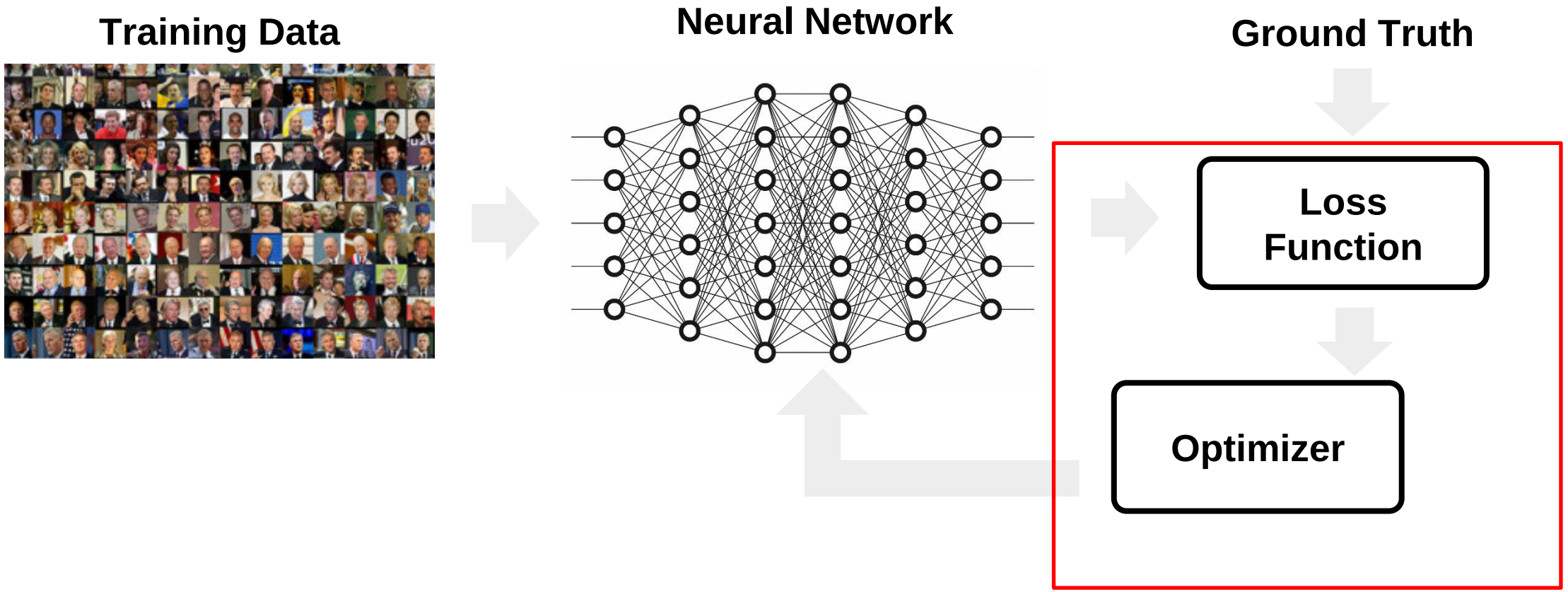
Out:

```
Predicted class: tensor([7], device='cuda:0')
```

data from https://pytorch.org/tutorials/

Pytorch

- **Pipeline to train your model.**



Automatic Differentiation

```
import torch

x = torch.ones(5)  # input tensor
y = torch.zeros(3)  # expected output
w = torch.randn(5, 3, requires_grad=True)
b = torch.randn(3, requires_grad=True)
z = torch.matmul(x, w)+b
loss = torch.nn.functional.binary_cross_entropy_with_logits(z, y)
```

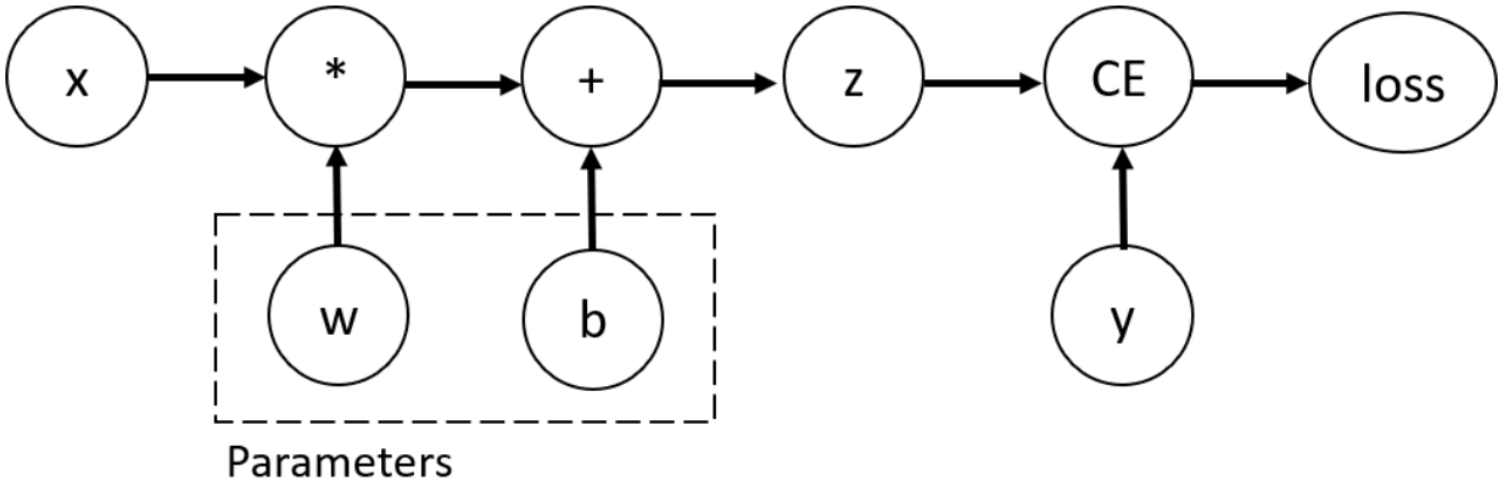
```
print(f"Gradient function for z = {z.grad_fn}")
print(f"Gradient function for loss = {loss.grad_fn}")
```

Out:

```
Gradient function for z = <AddBackward0 object at 0x7f55696a38b0>
Gradient function for loss = <BinaryCrossEntropyWithLogitsBackward0
object at 0x7f55696a0e20>
```

Tensors, Functions and Computational graph

This code defines the following **computational graph**:



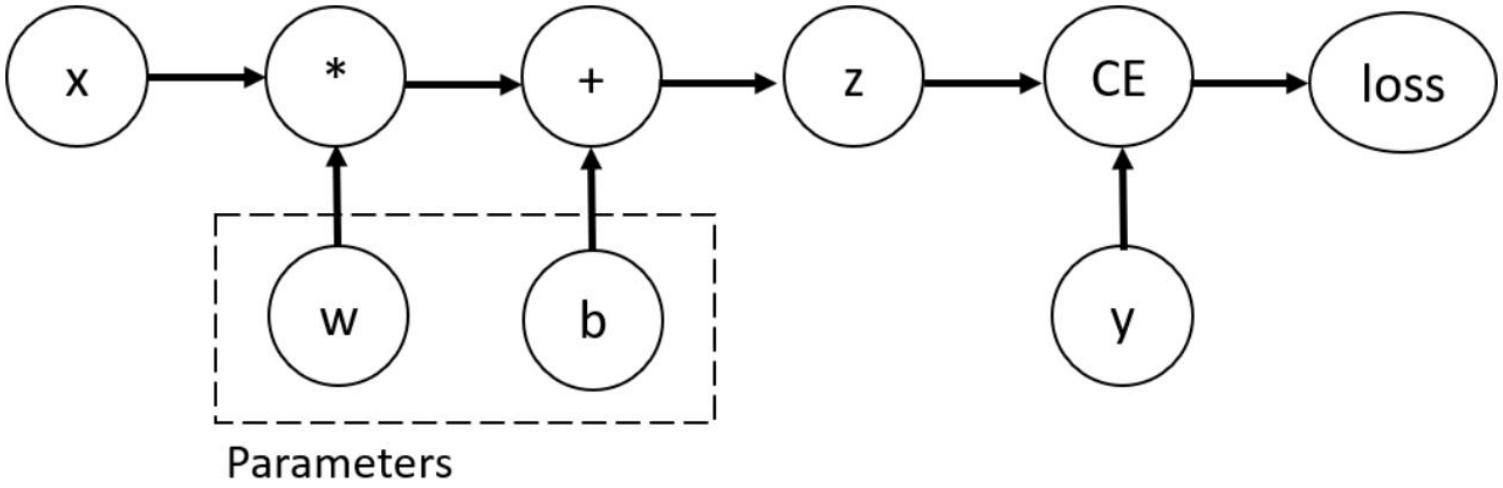

```
loss.backward()
print(w.grad)
print(b.grad)
```

Out:

```
tensor([[0.3313, 0.0626, 0.2530],
        [0.3313, 0.0626, 0.2530],
        [0.3313, 0.0626, 0.2530],
        [0.3313, 0.0626, 0.2530],
        [0.3313, 0.0626, 0.2530]])
tensor([0.3313, 0.0626, 0.2530])
```

data from https://pytorch.org/tutorials/

Automatic Differentiation




```
print(f"Gradient function for z = {z.grad_fn}")
print(f"Gradient function for loss = {loss.grad_fn}")
```

Out:

```
Gradient function for z = <AddBackward0 object at 0x7f55696a38b0>
Gradient function for loss = <BinaryCrossEntropyWithLogitsBackward0
object at 0x7f55696a0e20>
```

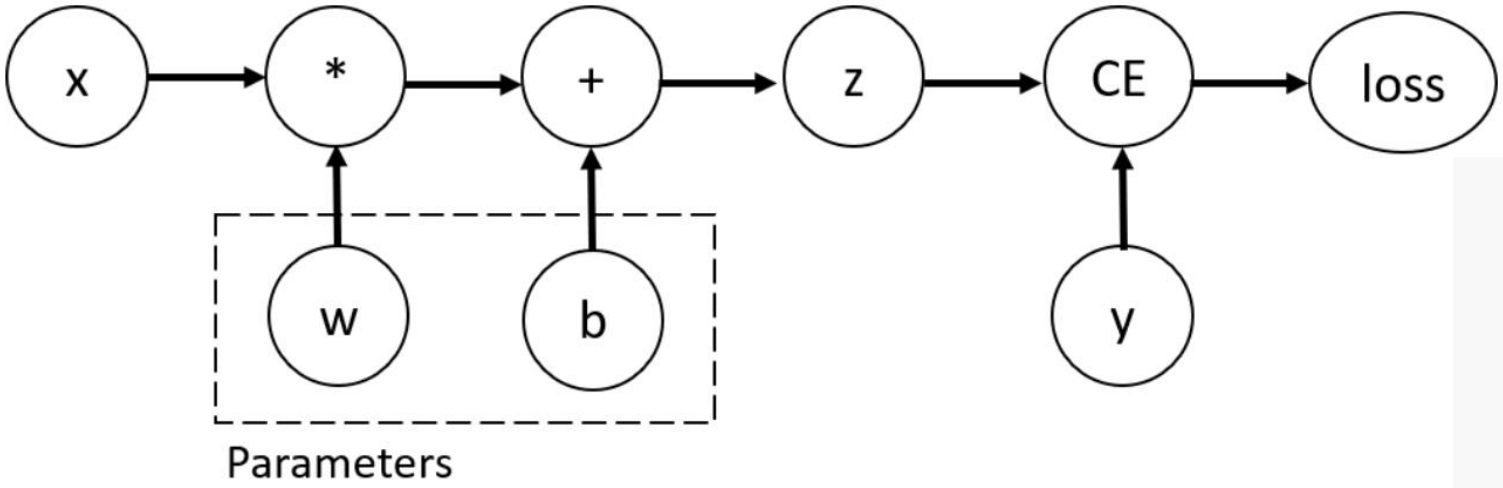
```
loss.backward()
print(w.grad)
print(b.grad)
```

Out:

```
tensor([[0.3313, 0.0626, 0.2530],
        [0.3313, 0.0626, 0.2530],
        [0.3313, 0.0626, 0.2530],
        [0.3313, 0.0626, 0.2530],
        [0.3313, 0.0626, 0.2530]])
tensor([0.3313, 0.0626, 0.2530])
```

data from https://pytorch.org/tutorials/

Automatic Differentiation




```
z = torch.matmul(x, w)+b
z_det = z.detach()
print(z_det.requires_grad)
```

Out:

```
False
```

```
z = torch.matmul(x, w)+b
print(z.requires_grad)

with torch.no_grad():
    z = torch.matmul(x, w)+b
print(z.requires_grad)
```

Out:

```
True
False
```

data from https://pytorch.org/tutorials/

Loss Function & Optimizer

```python
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

```python
# Initialize the loss function
loss_fn = nn.CrossEntropyLoss()
```

```python
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

Inside the training loop, optimization happens in three steps:

- Call `optimizer.zero_grad()` to reset the gradients of model parameters. Gradients by default add up; to prevent double-counting, we explicitly zero them at each iteration.
- Backpropagate the prediction loss with a call to `loss.backward()`. PyTorch deposits the gradients of the loss w.r.t. each parameter.
- Once we have our gradients, we call `optimizer.step()` to adjust the parameters by the gradients collected in the backward pass.

data from https://pytorch.org/tutorials/

Train Loop

```python
def train_loop(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    # Set the model to training mode - important for batch normalization and dropout layers
    # Unnecessary in this situation but added for best practices
    model.train()
    for batch, (X, y) in enumerate(dataloader):
        # Compute prediction and loss
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * batch_size + len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

data from https://pytorch.org/tutorials/

Test Loop

```python
def test_loop(dataloader, model, loss_fn):
    # Set the model to evaluation mode - important for batch normalization and dropout layers
    # Unnecessary in this situation but added for best practices
    model.eval()
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    test_loss, correct = 0, 0

    # Evaluating the model with torch.no_grad() ensures that no gradients are computed during test mode
    # also serves to reduce unnecessary gradient computations and memory usage for tensors with requires_grad=True
    with torch.no_grad():
        for X, y in dataloader:
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) == y).type(torch.float).sum().item()

    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_loss:>8f} \n")
```

data from https://pytorch.org/tutorials/

Full Implementation

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

epochs = 10
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader, model, loss_fn, optimizer)
    test_loop(test_dataloader, model, loss_fn)
print("Done!")
```

Out:

```
Epoch 1
-------------------------------
loss: 2.298730  [   64/60000]
loss: 2.289123  [ 6464/60000]
loss: 2.273286  [12864/60000]
loss: 2.269406  [19264/60000]
loss: 2.249603  [25664/60000]
```

data from https://pytorch.org/tutorials/

Save & Load the Model Weights

PyTorch models store the learned parameters in an internal state dictionary, called `state_dict`. These can be persisted via the `torch.save` method:

```
model = models.vgg16(weights='IMAGENET1K_V1')
torch.save(model.state_dict(), 'model_weights.pth')
```

In the code below, we set `weights_only=True` to limit the functions executed during unpickling to only those necessary for loading weights. Using `weights_only=True` is considered a best practice when loading weights.

```
model = models.vgg16() # we do not specify ``weights``, i.e. create untrained model
model.load_state_dict(torch.load('model_weights.pth', weights_only=True))
model.eval()
```

data from https://pytorch.org/tutorials/

Save & Load the Model with Shapes

When loading model weights, we needed to instantiate the model class first, because the class defines the structure of a network. We might want to save the structure of this class together with the model, in which case we can pass `model` (and not `model.state_dict()`) to the saving function:

```
torch.save(model, 'model.pth')
```

We can then load the model as demonstrated below.

As described in Saving and loading torch.nn.Modules, saving `state_dict``s is considered the best practice. However, below we use ``weights_only=False` because this involves loading the model, which is a legacy use case for `torch.save`.

```
model = torch.load('model.pth', weights_only=False),
```

Contents

- Introduce to Pytorch
- **Project Guidelines**
  - **Project Expectations**
  - Diffusion Demo
  - Picking a project idea
  - Final Report

Course Description

- **Objective**
  - Master the principles and theories of diffusion model-based generative AI.
  - Build programming skills and project development experience.
  - Apply diffusion-based generative AI to real-world problems.

- **Coursework**
  - Attendance
  - Participation
  - Diffusion Demo
  - Course Project
  - **Students are encouraged to develop innovative projects that combine generative models with their academic disciplines.**

Project Expectations

- **Explore concepts on a topic of your choice with diffusion**
  - We will build a project page, see it in the later notifications.

- **Completed in groups, no more than 3**
  - The number of teams can be adjusted according to the difficulty of the task, but the project expectation are higher with more people

- **Three tracks:**
  - **Survey:** you can explore existing literature and methodologies in the diffusion algorithm space, identifying gaps and opportunities for innovation.
  - **Applications:** you can apply diffusion algorithm to a specific interest (e.g. vision, biology, physics, geosciences), it is great to see solve problems on your particular domains.
  - **Models:** you can build a new model or algorithm, test and verify your design. This might be challenging, but could come up with a publishable work or works !!!

Final Reports

- **The final reports (6-8 pages) need to include:**
  - **Title:** The main theme of your work (method name/ domain/ tasks)
  - **Author(s):** List of all contributors
  - **Abstract:** Brief summary of your work (problem, method, key findings)
  - **Introduction:** Outline the objectives of your work, motivation and the problem you seek to solve.
  - **Related Work:** Discuss prior similar or related work, and what is the difference of your work.
  - **Methods:** Provide a detailed description of your methods

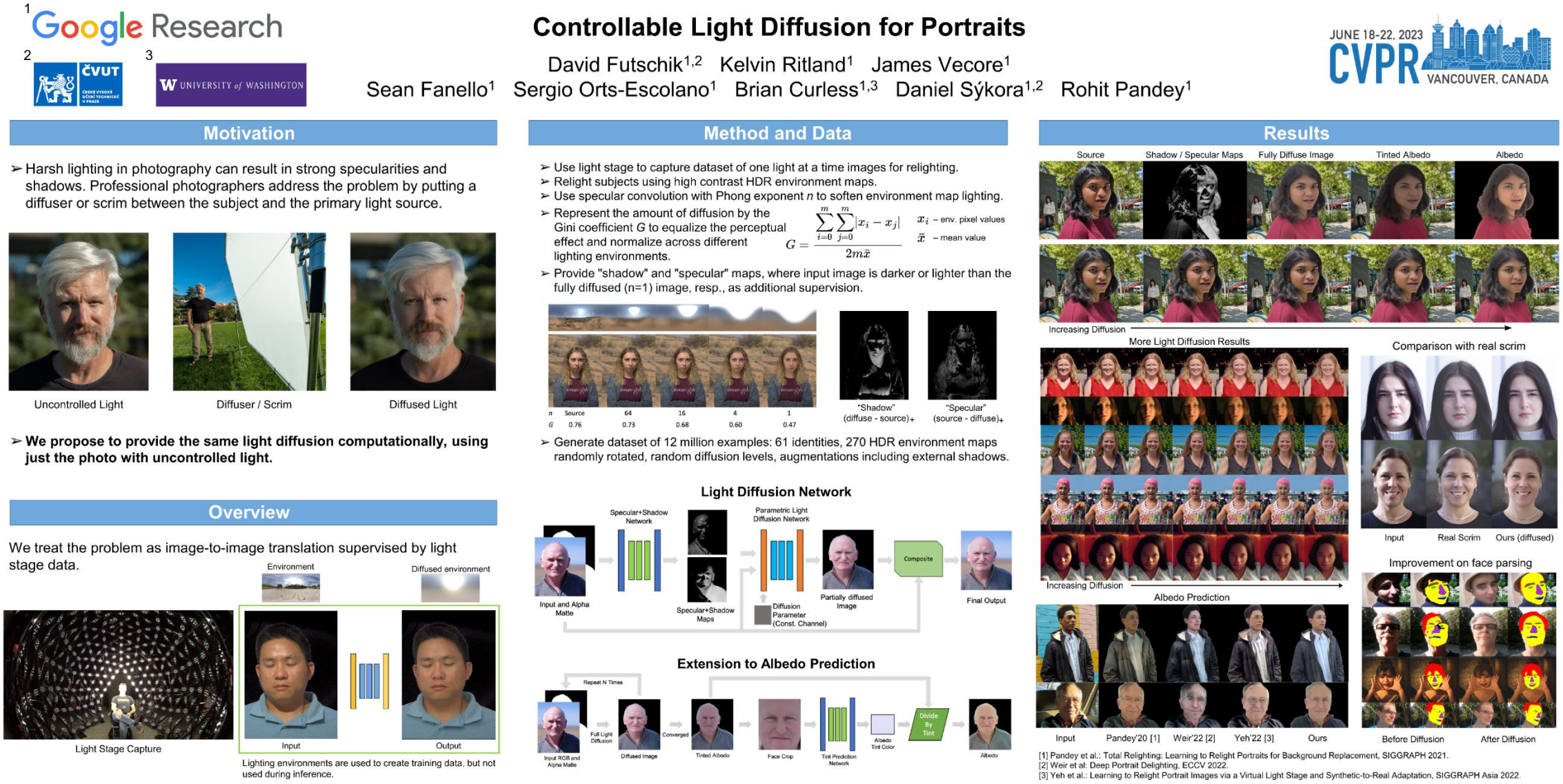
Final Reports

- **The final reports (6-8 pages) need to include:**
  - **Implementation:** Explain what datasets and tools/devices you used
  - **Experiments:** Outline the experimental design, and present your findings clearly with figures or tables.
  - **Conclusion:** Summarize the main findings of method.
  - **Supplementary Materials (optional) :** Include any additional data, code, or resources that support your method
  - **Poster:** Include title, authors, introduction, methods, results and conclusion in a visually appealing poster.

Final Reports

- **The final reports (6-8 pages) need to include:**
  - **Title**
  - **Author(s)**
  - **Abstract**
  - **Introduction**
  - **Related Work**
  - **Methods**
  - **Implementation**
  - **Experiments**
  - **Conclusion**
  - **Supplementary Materials (optional)**
  - **Poster**

Examples of Posters



[CVPR Poster Controllable Light Diffusion for Portraits](#)

Examples of Posters

Stanford University

Medical Image Dataset Rebalancing via Conditional Deep Convolutional Generative Adversarial Networks (cDCGANs)

Adam Sun and Kevin Li
{adsun, kevinli8}@stanford.edu

Background

Imbalanced datasets are common in the medical imaging context where normal data outnumbers disease data.
Deep learning models trained on imbalanced data tend to be biased towards majority class, leading to more false negatives, which is particularly problematic since this means medical conditions may be left undetected.
Classic approaches include:

- Oversampling
- Undersampling
- Class weights
- SMOTE methods
- Data augmentation

To avoid overfitting that comes with classic oversampling methods we propose using a GAN to generate synthetic examples from random noise, adding to the diversity of minority class.

Experiments

	Precision	Recall/ Sensitivity	Specificity	F1
Baseline	0.730	0.842	0.850	0.782
Class Weights	0.859	0.855	0.915	0.857
Under-sam-pling	0.709	0.865	0.843	0.779
Over-sam-pling	0.863	0.845	0.917	0.854
DCGAN	0.821	**0.910**	0.898	0.863
cDCGAN $\alpha = 0.50$	**0.893**	0.857	0.934	**0.874**
cDCGAN $\alpha = 0.25$	0.710	0.706	**0.949**	0.803
cDCGAN $\alpha = 0.75$	0.821	0.901	0.898	0.859

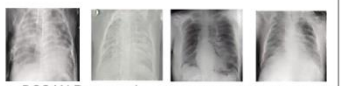
Problem

Our goal is to rebalance image datasets while reducing bias and improving overall model performance.

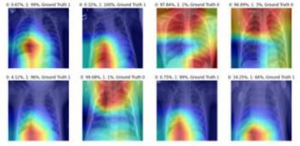
Our GAN uses random noise to generate new synthetic pneumonia chest x-rays that are added to the dataset, and we assess the different methods by training a ResNet-18 model that takes the chest x-rays as input and outputs a binary classification of healthy or pneumonia.

Due to our imbalanced dataset, we evaluate performance based not on accuracy, but precision, recall/sensitivity, specificity, and F1 score.

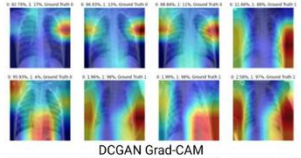
Visualizations



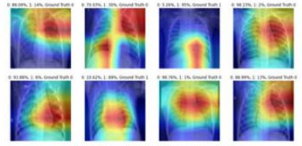
DCGAN Pneumonia cDCGAN Healthy vs Pneumonia



Baseline Grad-CAM



DCGAN Grad-CAM

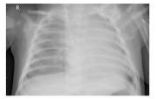


cDCGAN Grad-CAM, $\alpha = 0.5$

Dataset

Dataset was taken from Kermany et al. consisting of pediatric chest X-rays from Guangzhou. We supplemented with healthy chest X-rays from public NIH dataset ChestX-Ray8. We randomly removed pneumonia images to created imbalanced situation: 4385 normal, 1460 pneumonia. The test set is more balanced to be an unbiased evaluation of model performance.

Healthy



Pneumonia

Analysis

We see that:

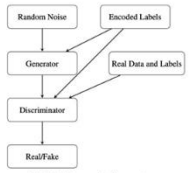
- DCGAN and the cDCGAN variants were all able to outperform the baseline in almost all metrics.
- In particular, both our DCGAN and cDCGAN ($\alpha = 0.5$) approaches outperform class weights, oversampling, and undersampling (the most commonly used methods) in terms of F1 score.
- DCGAN and cDCGAN ($\alpha = 0.25$) have the highest sensitivity and specificity, respectively. This uncovers the possibility of using these models in conjunction to generate a better dataset.
- We find that α can be used to tune sensitivity and specificity.
- Our Grad-CAMs uncover that the DCGAN ResNet is not learning relevant features inside the chest cavity, while the baseline and cDCGAN do a bit better in this respect.

Methods

We use a Generative Adversarial Network, which is used to learn a mapping from random noise to the data space of a distribution. A generator network, which tries to generate a realistic data point, plays a minimax game against a discriminator network which seeks to discriminate between real and fake data. The objective function from the original paper:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

We use a deep convolutional GAN (DCGAN), which has strided convolution, batchnorm, and ReLU to improve GAN stability and image comprehensibility. We train it on the minority class of pneumonia images and then use the generator to generate realistic examples from the minority class distribution and oversample the minority class to rebalance an imbalanced dataset.



CGAN architecture

Additionally, we explore conditional DCGAN as a way to utilize the entire dataset. By conditioning on labels, we can then drive the generated images towards the pneumonia class. We also introduce a hyperparameter α that represents the probability of a randomly generated healthy label.
We compare ResNet-18 performance on the rebalanced dataset with its performance on datasets rebalanced by more classic methods.

Conclusions and Future Work

In this work, we explored the usage of Deep Convolutional Generative Adversarial Networks (DCGANs) and conditional Deep Convolutional Generative Adversarial Networks (cDCGANs) for oversampling of the minority class in imbalanced medical image datasets.

Based on the results, we conclude that GAN-based approaches are a valid way to oversample an imbalanced training dataset, despite the Grad-CAMs indicating that the model may be learning irrelevant features.

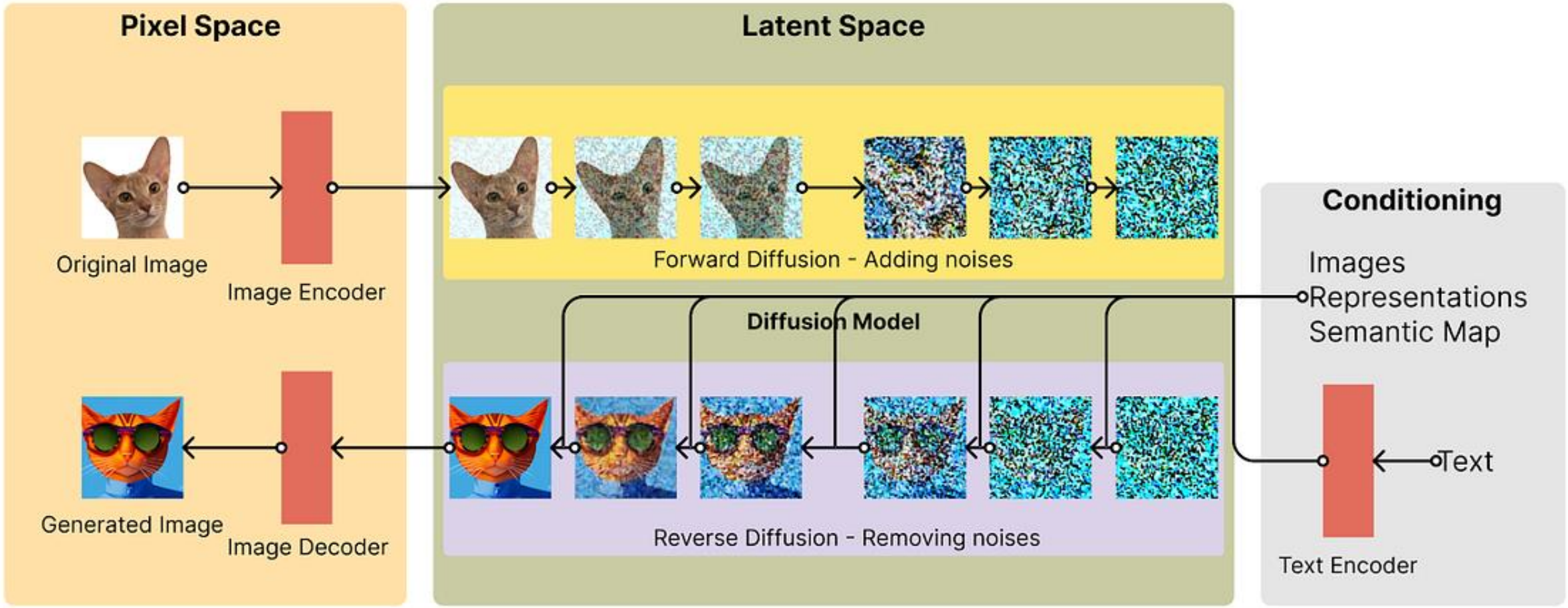
In the future, we would like to tune the GAN models more carefully, and train models multiple times to compensate for the stochastic nature of model training, as well as curate a more diverse imbalanced dataset. We may offer our GAN-generated images to an actual experienced radiologist for review for usefulness, relevance, and realisticness, resulting in a better annotated dataset.

CS231N Final Project Poster - Adam Sun and Kevin Li

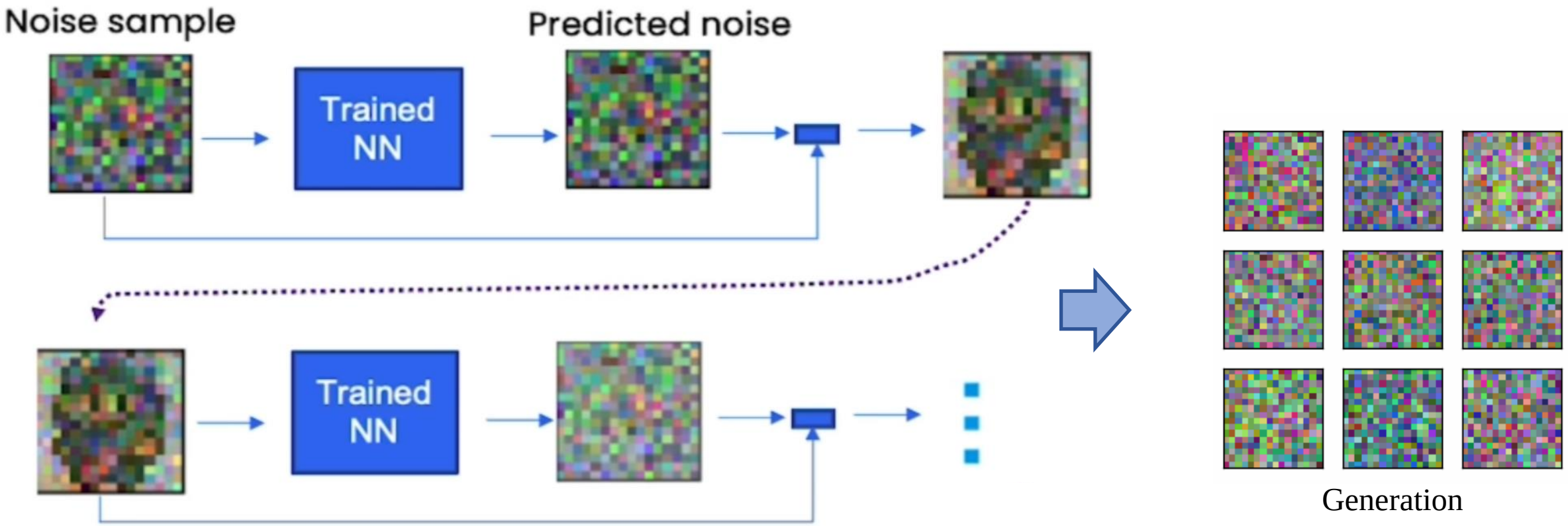
Contents

- Introduce to Pytorch
- **Project Guidelines**
  - Project Expectations
  - **Diffusion Demo**
  - Picking a project idea
  - Final Report

Diffusion


Pixel Space
Latent Space
Original Image
Image Encoder
Forward Diffusion - Adding noises
Conditioning
Images
Representations
Semantic Map
Diffusion Model
Generated Image
Image Decoder
Reverse Diffusion - Removing noises
Text
Text Encoder

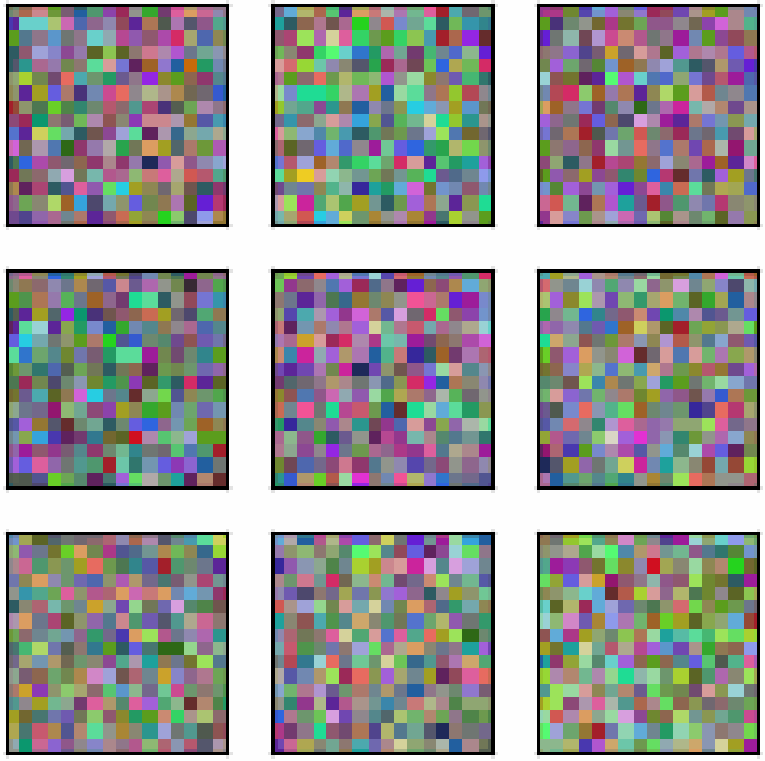
Diffusion Inference

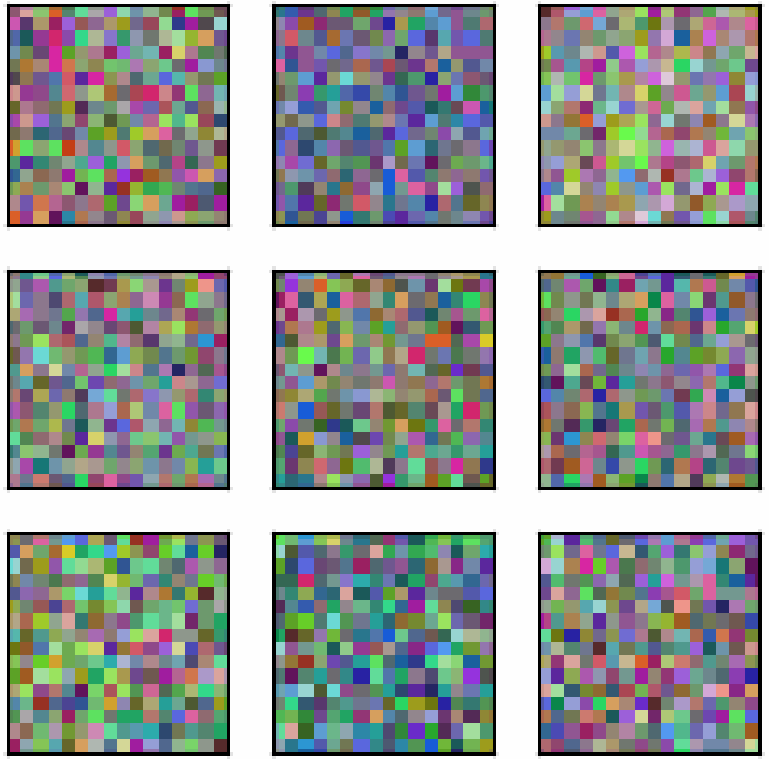


How Diffusion Models Work - DeepLearning.AI

Diffusion Inference

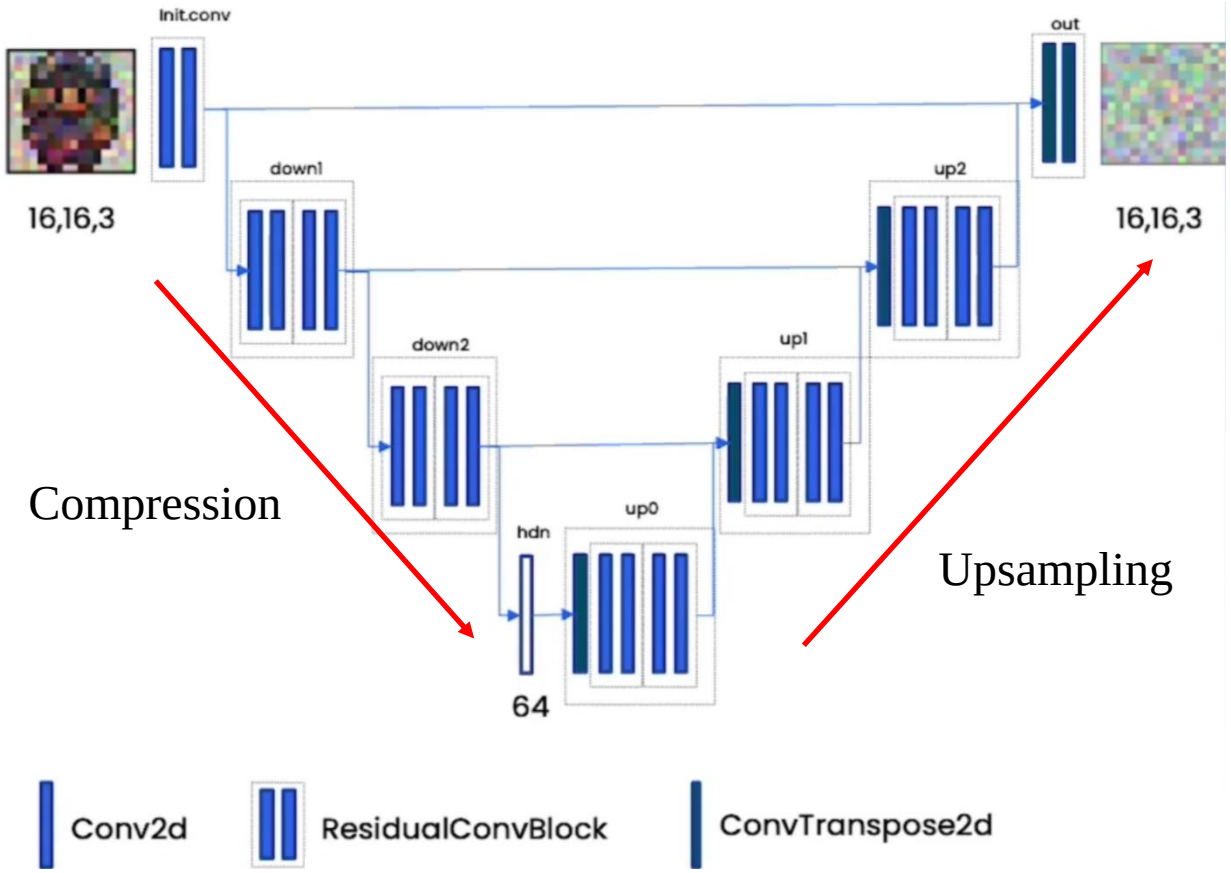
- **Trick/Empirically**
  - Add in additional noise before it gets passed to the next step
  - Will not collapse to something closer to the average of the dataset

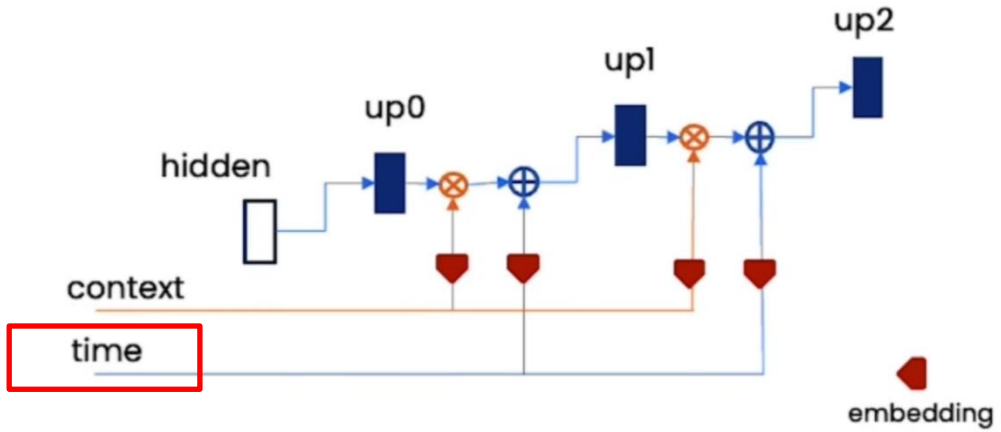




Neural Network

- **Input & outputs have the same shape.**
- **Can take additional inputs.**



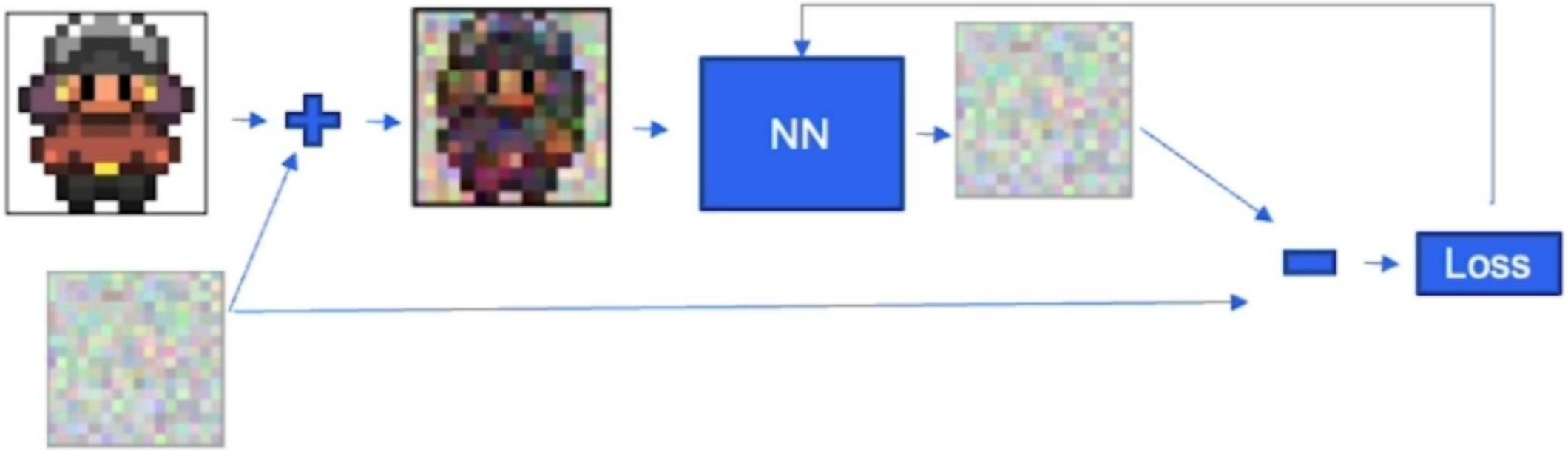



```
# embed context and timestep
cemb1 = self.contextembed1(c).view(-1, self.n_feat * 2, 1, 1)
# (batch, 2*n_feat, 1,1)
temb1 = self.timeembed1(t).view(-1, self.n_feat * 2, 1, 1)
cemb2 = self.contextembed2(c).view(-1, self.n_feat, 1, 1)
temb2 = self.timeembed2(t).view(-1, self.n_feat, 1, 1)


up1 = self.up0(hiddenvec)
# add and multiply embeddings
up2 = self.up1(cemb1*up1 + temb1, down2)
up3 = self.up2(cemb2*up2 + temb2, down1)
out = self.out(torch.cat((up3, x), 1))
return out
```

How Diffusion Models Work - DeepLearning.AI

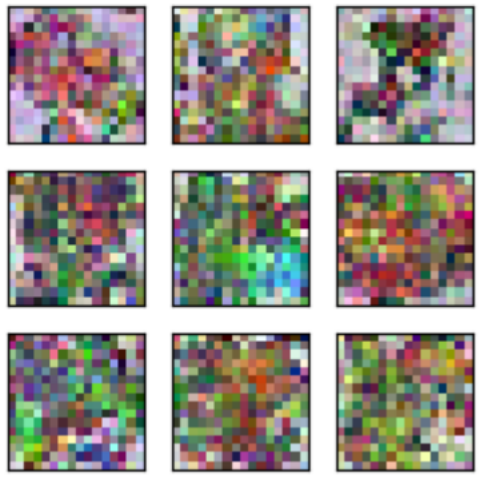
Training

- **Neural network learns to predict noise**
  - The distribution of what is not noise
  - Sample random timestep (a.k.a noise level) per image to train more stably

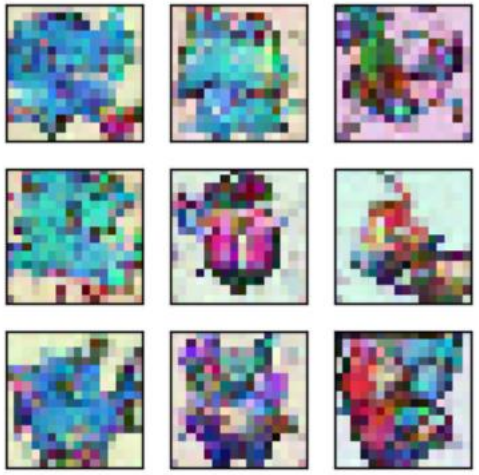


How Diffusion Models Work - DeepLearning.AI

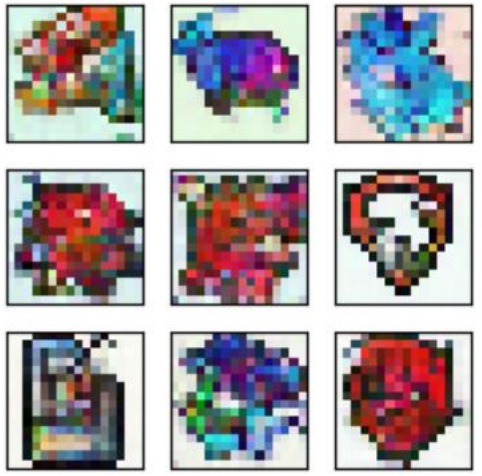
Training



Epoch 0



Epoch 4

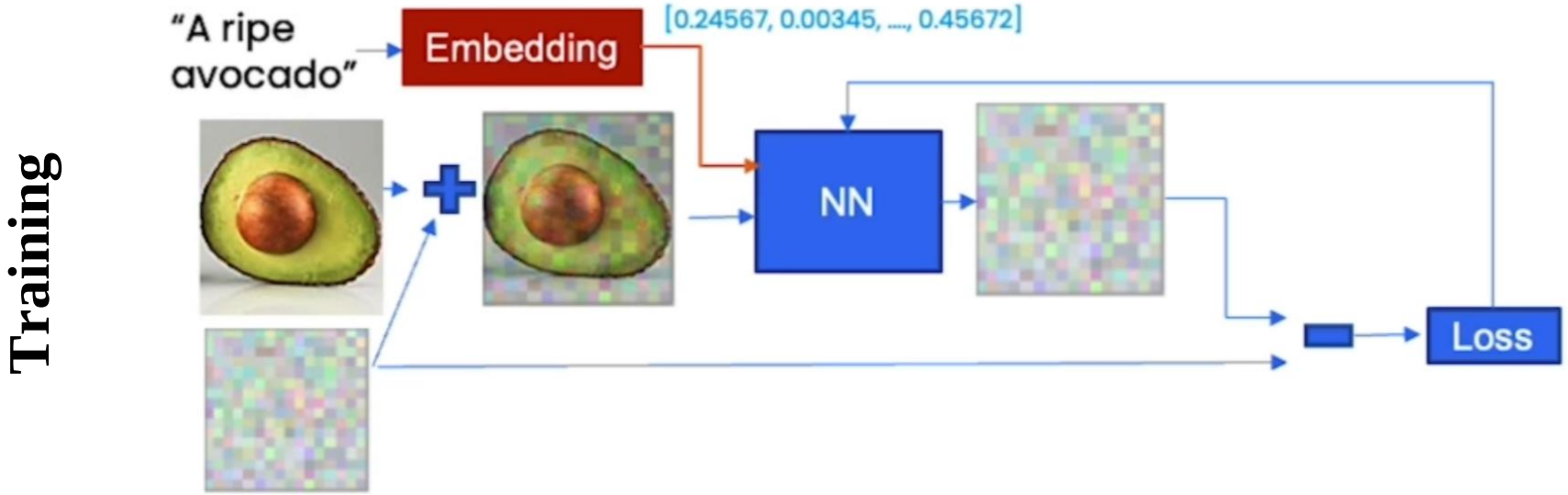


Epoch 8

Epoch 31

Controlling

Embedding vector captures meaning



How Diffusion Models Work - DeepLearning.AI

Controlling

- **Context Definition:**
  - hero, non-hero, food, spell, side-facing

Input Context:

Food Hero Side-facing
Hero Food Side-facing
Spell Food No-hero

Combination:

Food Hero Side-facing
Hero Food Side-facing
Spell Food No-hero

```
[1,0,0,0,0],
[1,0,0.6,0,0],
[0,0,0.6,0.4,0],
[1,0,0,0,1],
[1,1,0,0,0],
[1,0,0,1,0],
[1,0,1,0,0],
[1,1,0,0,0],
[1,0,0,1,0],
```

Homework

Diffusion Demo

- **Project #1 (For everyone)**
  - **Introduction:** explore the training, sampling and controlling in the diffusion models.
  - **Implementation:**

    How-Diffusion-Models-Work
  - **Tasks:**
    - Training your diffusion models.
    - Explore the hyper-parameter in the diffusion models.
    - Sampling algorithm: DDPM, DDIM.
    - Control strategy.
    - ……
  - **Members:** for everyone.

Contents

- Introduce to Pytorch
- **Project Guidelines**
  - Project Expectations
  - Diffusion Demo
  - **Picking a project idea**
  - Final Report

Picking a project idea

- **Do what is interesting to you.**
- **Do not directly what seems easiest.**

- **Considerations:**
  - **Data:**
    - ✓ Existing data for your problem.
    - ✓ Need to collect and preprocess data by yourself.
  - **Code:**
    - ✓ Based on an existing implementation.
    - ✓ Implement from scratch.

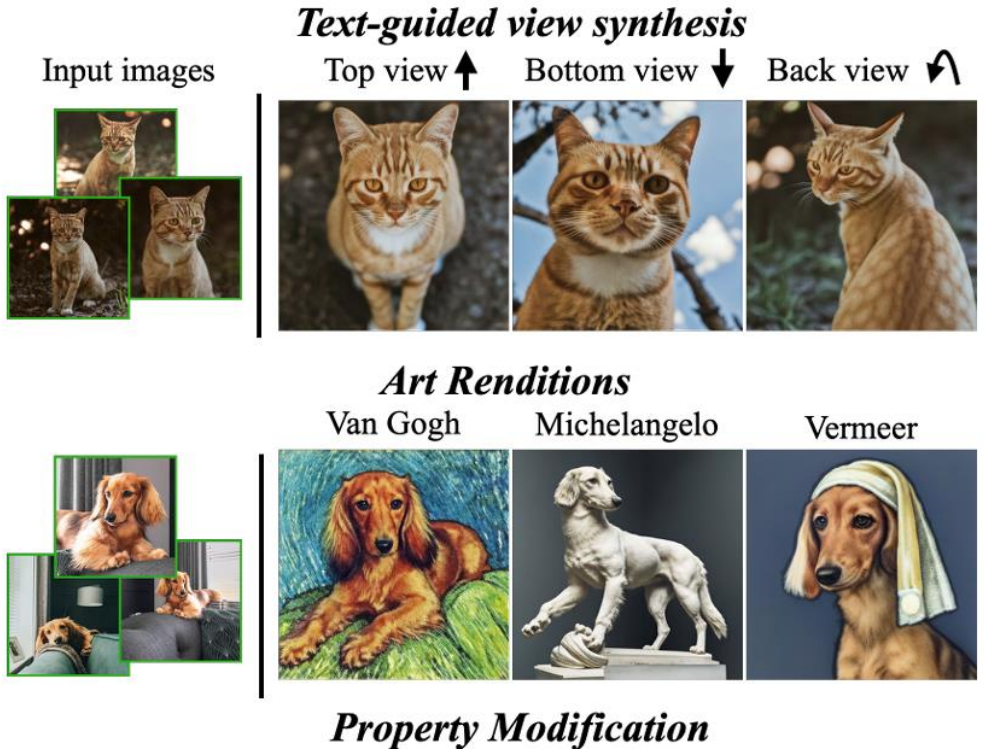
Picking a project idea

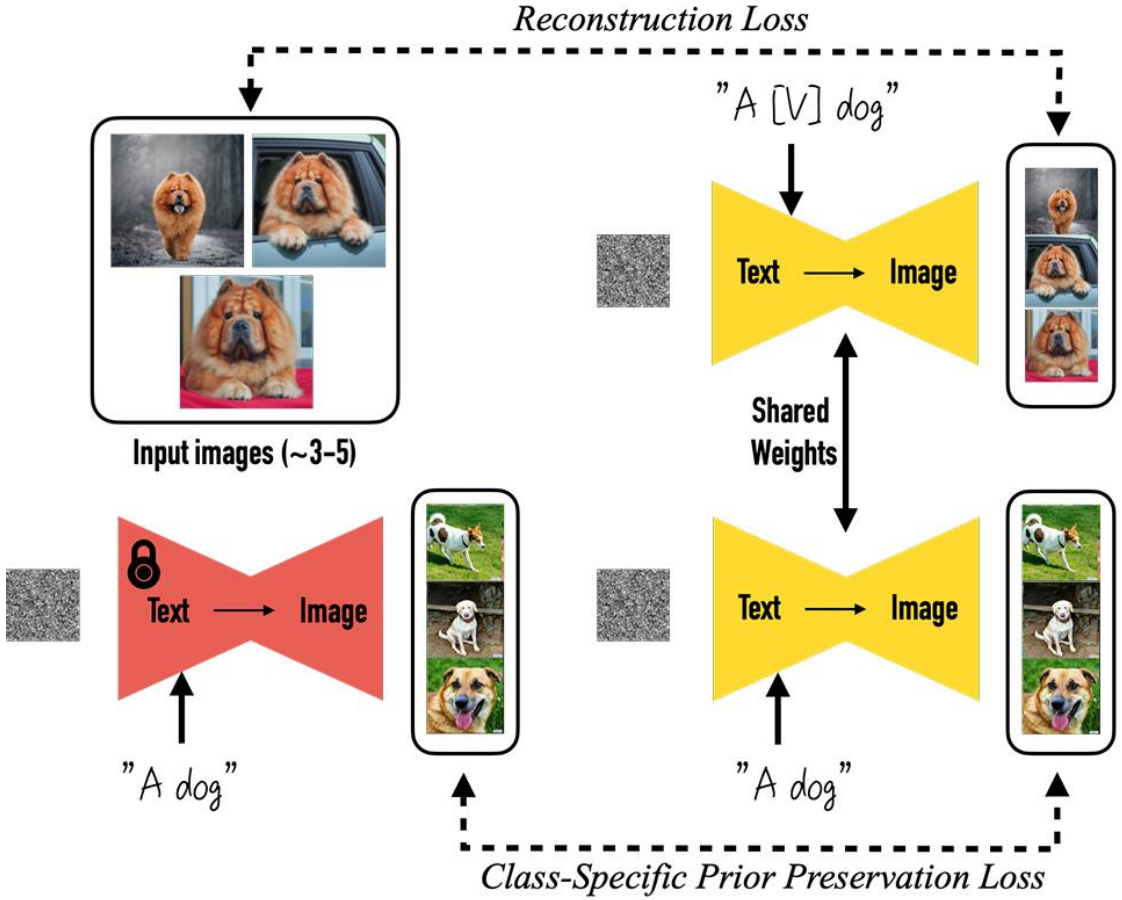
- **Attachable Project (Dreambooth)**
  - Text Prompts (DALL-E2/Imagen): retro style yellow alarm clock with a white clock face and a yellow number three on the right part of the clock face in the jungle
  - Dreambooth: a [V] clock in the jungle

DreamBooth (Ruiz, CVPR 2023)

Picking a project idea

- **Attachable Project (Dreambooth)**
  - **Introduction:** Given 3-5 images, fine-tuning a text-to-image diffusion model with a unique identifier [V] and the name of class.





Dreambooth (Ruiz, ICCV2023)

Picking a project idea

- **Attachable Project (Dreambooth)**

Color modification (*"A [color] [V] car"*)

Input

purple

red

yellow

blue

pink

Hybrids (*"A cross of a [V] dog and a [target species]"*)

Input

bear

panda

koala

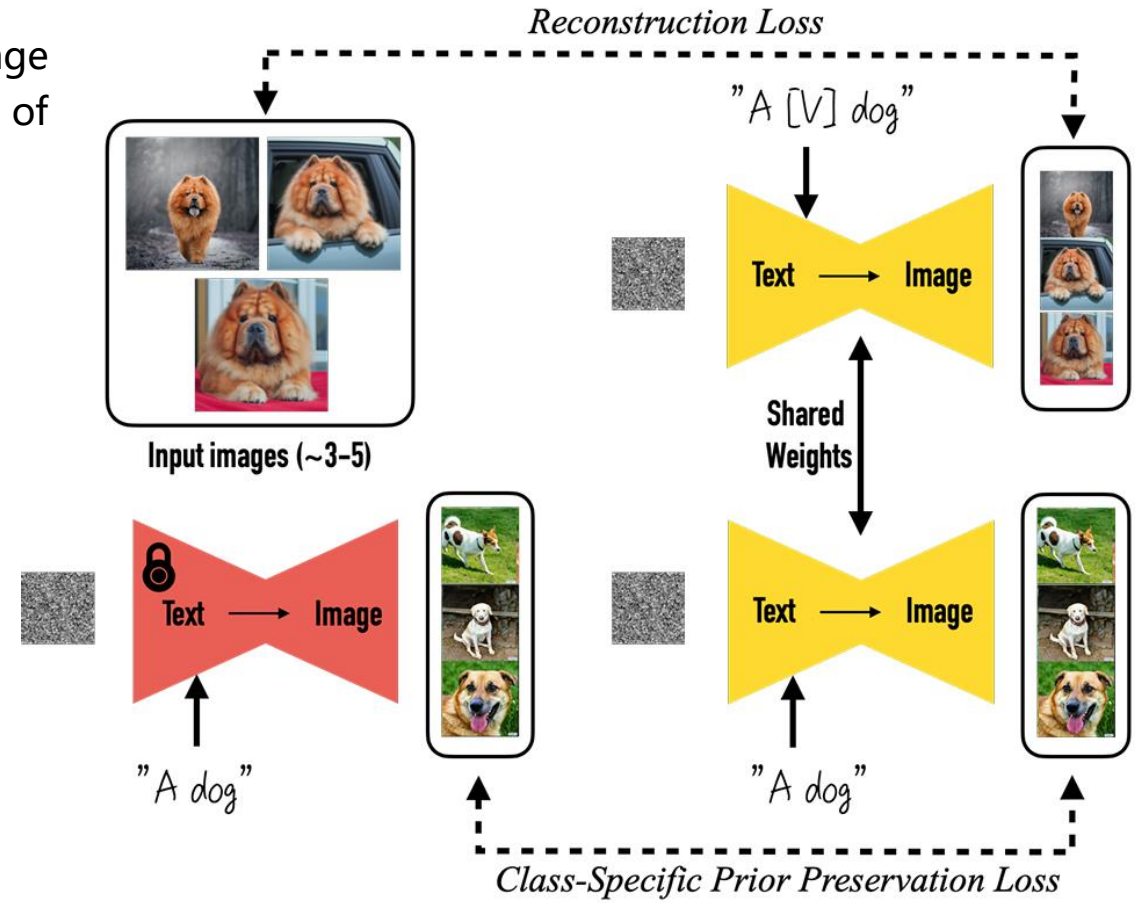
lion

hippo

Picking a project idea

- **Attachable Project (Dreambooth)**
  - **Introduction:** Given 3-5 images, fine-tuning a text-to-image diffusion model with a unique identifier [V] and the name of class.
  - **Implementation:**

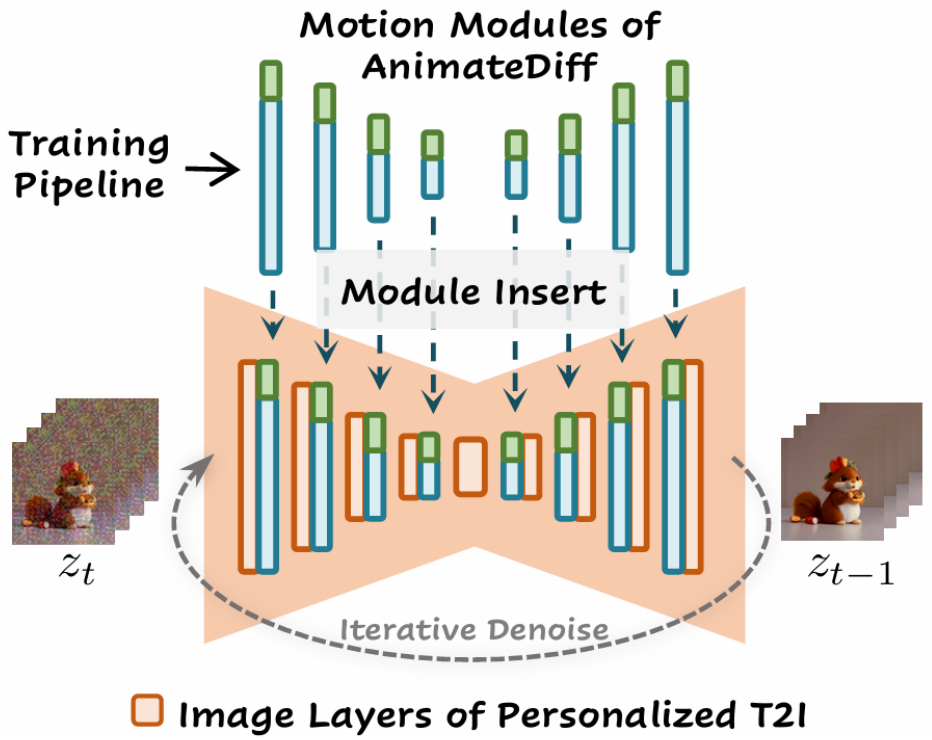
    google/dreambooth (github.com)
  - **Tasks:**
    - Build your own fine-tuning datasets.
    - Explore the training process of diffusion model.
    - Analysis the inferences of your models.
    - How to select the identifier [V].
    - ……
  - **Members:** no more than 3.



Dreambooth (Ruiz, ICCV2023)

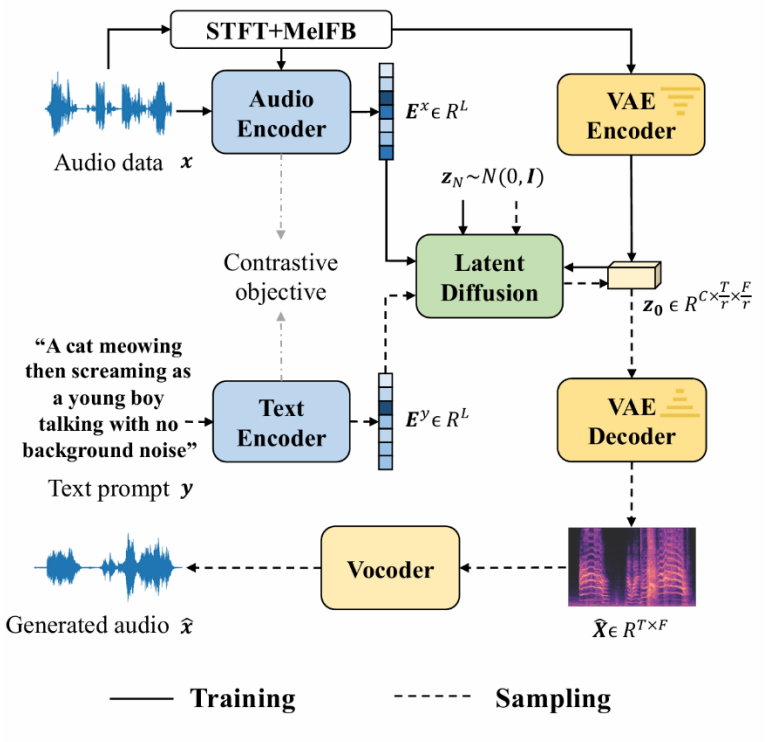
Picking a project idea

- **Project based on your interests (no more than 6 members)**



Video Generation

AnimateDiff (Guo, ICLR2024)

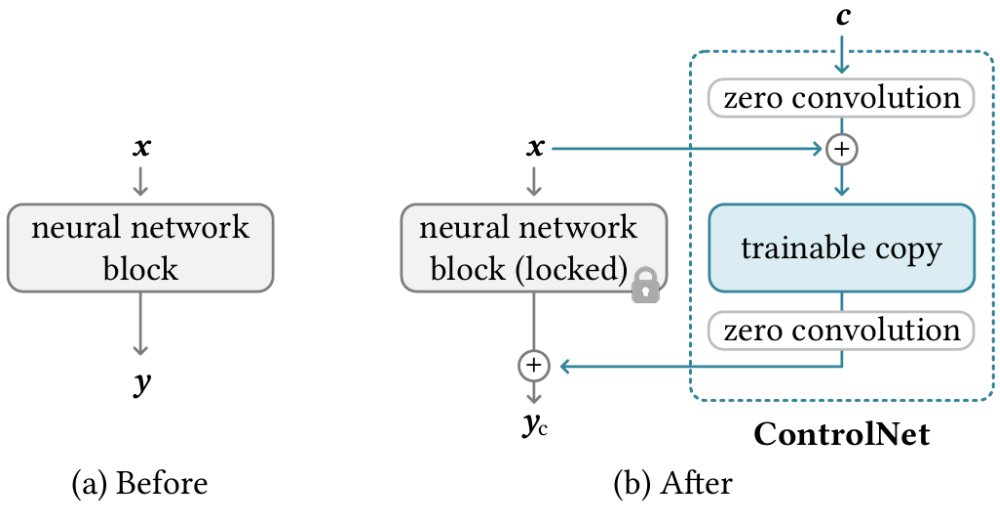


Audio Generation

AudioLDM (Liu, ICML2023)

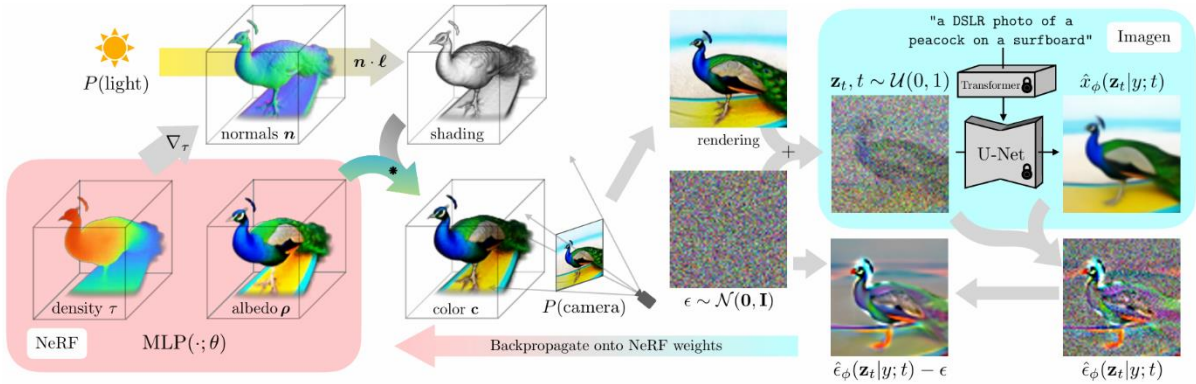
Picking a project idea

- **Project based on your interests (no more than 3 members)**



Controllable Diffusion

[ControlNet (Zhang ICCV 2023)]



Diffusion in 3D

[DreamFusion (Google)]

Picking a project idea

- **Literature review about diffusion models (algorithm or applications)**
  - **Contents**
    - Introduction:
    - Preliminary
    - Literature Review (applications in specific domains)
    - Key Challenges
    - Future Direction
    - …..
    - Conclusions
  - **Members:** only for 1.

Contents

- Introduce to Pytorch
- **Project Guidelines**
  - Project Expectations
  - Diffusion Demo
  - Picking a project idea
  - **Final Report**

Course Schedule

Lecture	Topic	Content
1	Introduction	Overview of this Course
2	Basics of Generative AI (1)	VAE, GAN, Flow-based models, and applications
3	Basics of Generative AI (2)	Probability distributions, random variables
4	Fundamentals of Diffusion Models (1)	Diffusion model principles, DDPMs, DDIMs, SGMs, Score SDEs, VDMs
5	Fundamentals of Diffusion Models (2)	Model structures for generation processes: Unet, DiT, Latent space
6	Machine Learning Frameworks and Tools	Machine learning frameworks and tools: training, inference, optimization
7	Images + Diffusion Models	LDM、Stable Diffusion、DALL-E、GigaGAN
8	Audio + Diffusion Models	Audio Diffusion Models、VideoDiffusion、Sora
9	3D + Diffusion Models	NeRF、3D-VAE、DreamFusion
10	Biology (Structure) + Diffusion Models	AlphaFold3、ESMFold、RFdiffusion、SE(3) Diffusion
11	Physics and Meteorology + Diffusion Models	GraphCast、Pangu-Meteorology、NowcastNet、Fuxi
12	Advanced Topics in Diffusion Models (1)	External expert talk
13	Advanced Topics in Diffusion Models (2)	External expert talk
14	Paper and Project Presentations	
15	Paper and Project Presentations	
16	Paper and Project Presentations	

Final Report

- **Report Proposal**
  - **Brief Introduction**
  - **Problems**
  - **Methods**
  - **Members**
  - **Framework** (optional in this stage)
- **Report Proposal**
  - **Slides**
  - **Reports**
  - **Posters**
- **More details in the course homepage**

Lecture	Topic
1	Introduction
2	Basics of Generative AI (1)
3	Basics of Generative AI (2)
4	Fundamentals of Diffusion Models (1)
5	Fundamentals of Diffusion Models (2)
6	Machine Learning Frameworks and Tools
7	Images + Diffusion Models
8	Audio + Diffusion Models
9	3D + Diffusion Models
10	Biology (Structure) + Diffusion Models
11	Physics and Meteorology + Diffusion Models
12	Advanced Topics in Diffusion Models (1)
13	Advanced Topics in Diffusion Models (2)
14	**Paper and Project Presentations**
15	**Paper and Project Presentations**
16	**Paper and Project Presentations**

Thanks!