







Pytorch basics

1. Dataset and DataLoader
2. Transforms
3. Build Model
4. Training and validation
5. Save and Load model
6. Testing

Pytorch basics

Image	label
	0
	0
	0
	1
	1
	1

Feature Matrix for classification task

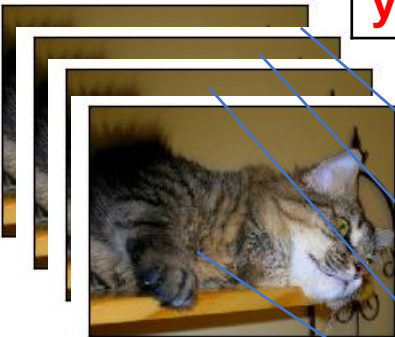
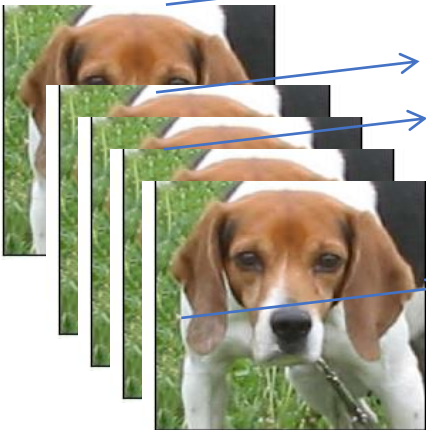


Features							label
Energy	entropy	mean	variance	area	color	texture	0

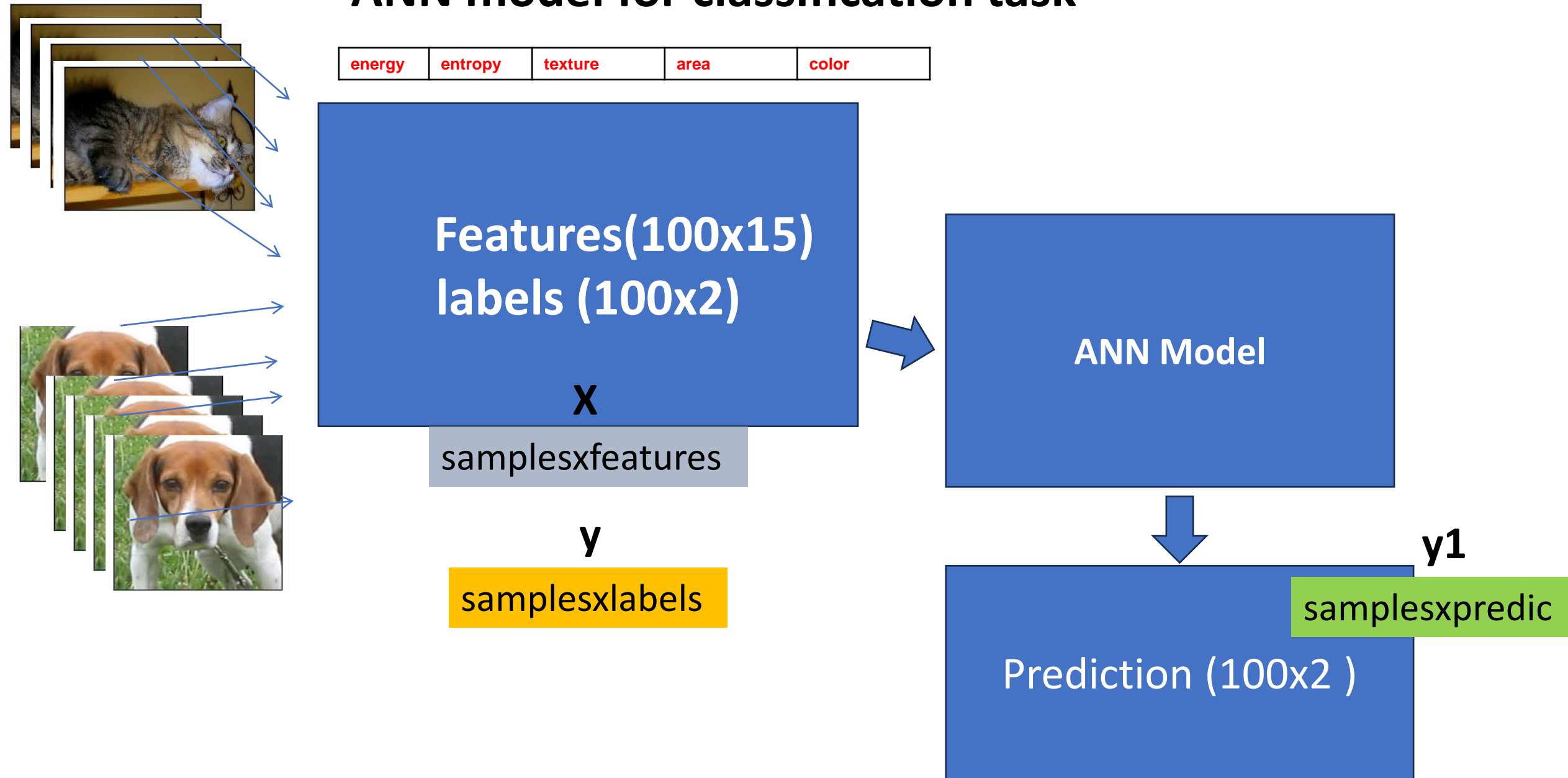


Features							label
Energy	entropy	mean	variance	area	color	texture	1

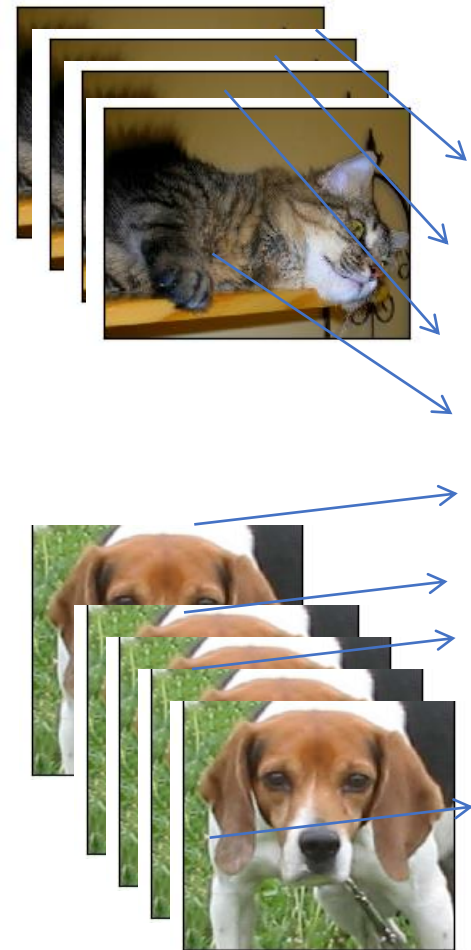
Feature Matrix for classification task

											energy y	entropy y	texture	area	color
samples			Features								Labels							
	Cat_samp 1		f1	f2	f3	f4	f15	0							
	Cat_samp 1		f1	f2	f3	f4	f15	0							
	Cat_samp 1		f1	f2	f3	f4	f15	0							
	Cat_samp 1		f1	f2	f3	f4	f15	0							
	dog_samp 1		f1	f2	f3	f4	f15	1							
	dog_samp 1		f1	f2	f3	f4	f15	1							
	dog_samp 1		f1	f2	f3	f4	f15	1							

ANN model for classification task



ANN model for classification task



energy	entropy	texture	area	color
--------	---------	---------	------	-------

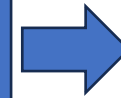
Features(100x15)
labels (100x2)

X

samplesxfeatures

y

samplesxlabels



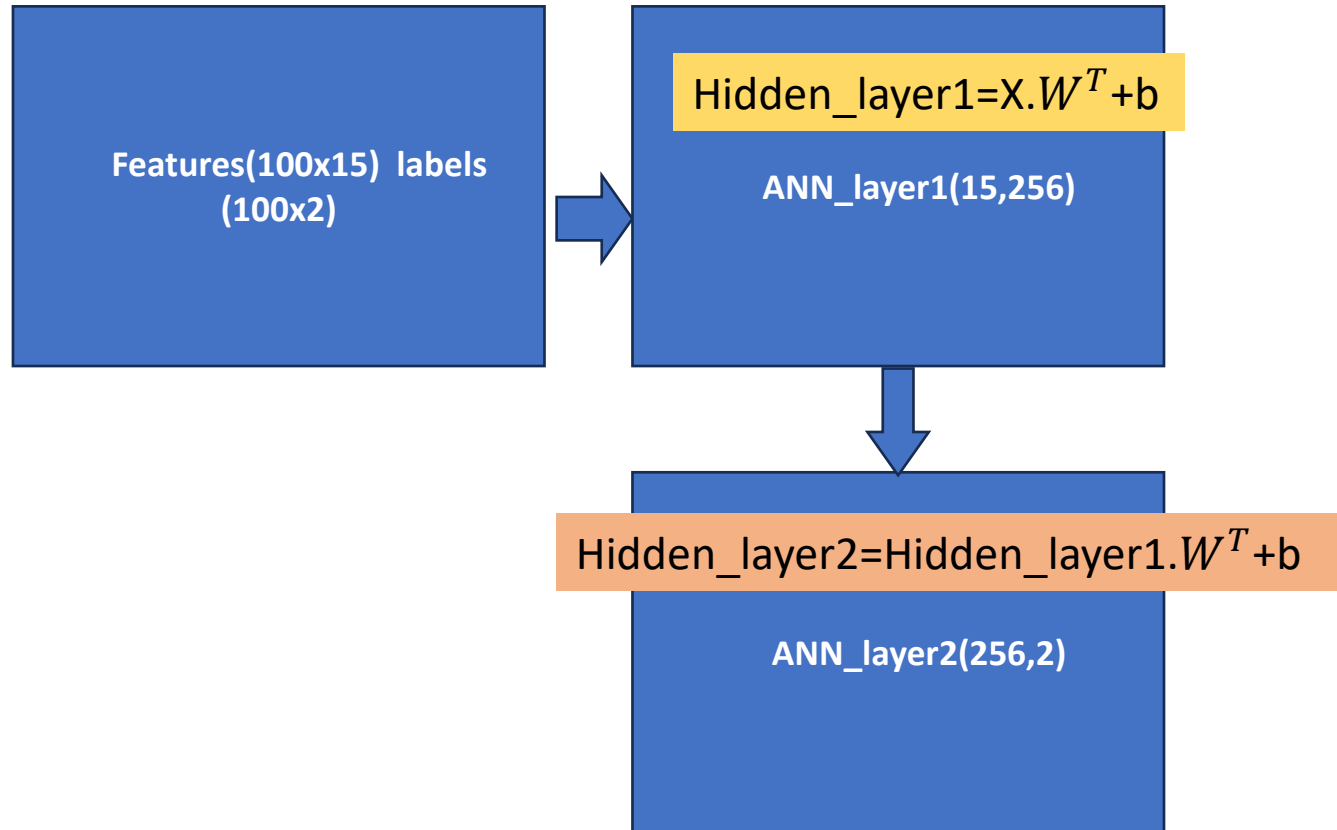
```
# ANN network with two FC layers
## define input feature matrix
X=torch.rand(100,15) ##
samplesxfeatures(100,15)
ANN_layer1=nn.Linear(15,256) #
in_features=15,out_features=256
hidden_features=ANN_layer1(X)
#samplesxoutfeatures(100,256)
#print(hidden_features.shape)
ANN_layer2=nn.Linear(256,2) #
in_features=256,out_features=2
y1=ANN_layer2(hidden_features)
print(y1.shape) ###
samplesxoutfeatures(100,2)
```

y1

samplesxpredic

ANN model for classification task

energy	entropy	texture	area	color
--------	---------	---------	------	-------



$X = (\text{samples}, \text{in_features})$

$W = (\text{out_features}, \text{in_features})$

$W^T = (\text{in_features}, \text{out_features})$

$b = (\text{out_features})$

$X = (100, 15)$

$W = (256, 15)$

$W^T = (15, 256)$

$b = (256)$

$\text{Hidden_layer1} = X.W^T + b$
 $= (100 \times 15) \times (15 \times 256) + 256 = 100 \times 256$

$\text{Hidden_layer2} = \text{Hidden_layer1}.W^T + b$
 $= (100 \times 256) \times (256 \times 2) + 2 = 100 \times 2$

ANN model for classification task

Way1

```
import torch
import torch.nn as nn
# ANN network with two FC layers
## define input feature matrix
X=torch.rand(100,15) ##
samplesxfeatures(100,15)
ANN_layer1=nn.Linear(15,256) #
in_features=15,out_features=256
hidden_features=ANN_layer1(X)
#samplesxoutfeatures(100,256)
#print(hidden_features.shape)
ANN_layer2=nn.Linear(256,2) #
in_features=256,out_features=2
y=ANN_layer2(hidden_features)
print(y.shape) ###
samplesxoutfeatures(100,2)
```

Way2

```
##### nn.Sequential module
ANN_model=nn.Sequential(ANN_layer1,ANN_layer2
,nn.ReLU())
y=ANN_model(X)
print(y.shape)
```

Way3

```
##### another way to define
the model
class ANN_model(nn.Module):

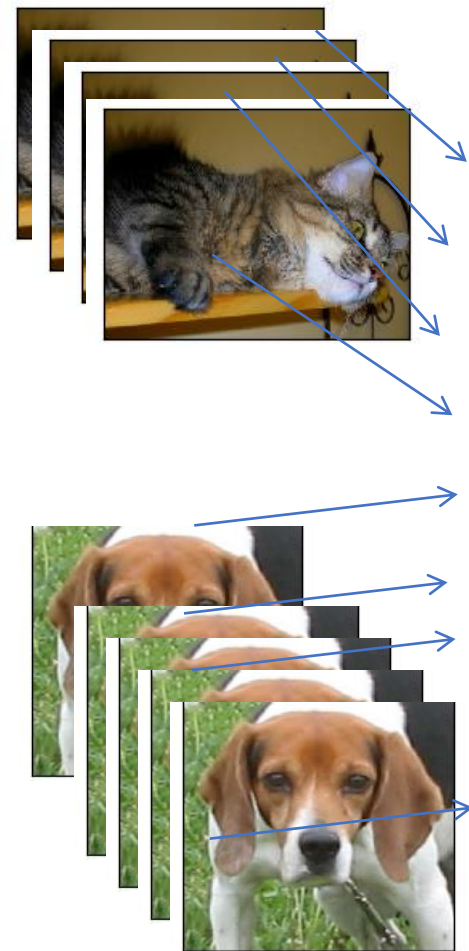
    def __init__(self):
        super(ANN_model,self).__init__()

        self.layer1=nn.Linear(15,256)
        self.layer2=nn.Linear(256,2)
        self.relu=nn.ReLU()

    def forward(self,x):
        x=self.layer1(x)
        x=self.layer2(x)
        x=self.relu(x)
        return x

model=ANN_model()
X=torch.rand(100,15)
y1=model(X)
```


CNN+ ANN model for classification task



samples	Features								Labels
Cat_samp 1	Automatic Feature extraction by CNN layers and Transforms features using ANN layers								0
Cat_samp 1									0
Cat_samp 1									0
Cat_samp 1									0
dog_samp 1									1
dog_samp 1	f1	f2	f3	f4	f15	1
dog_samp 1	f1	f2	f3	f4	f15	1
dog_samp 1	f1	f2	f3	f4	f15	1
dog_samp 1	f1	f2	f3	f4	f15	1

CNN+ ANN model for classification task

Automatic Features extraction using CNN
layers

Transform Features using ANN models

2D CNN model

ANN Model



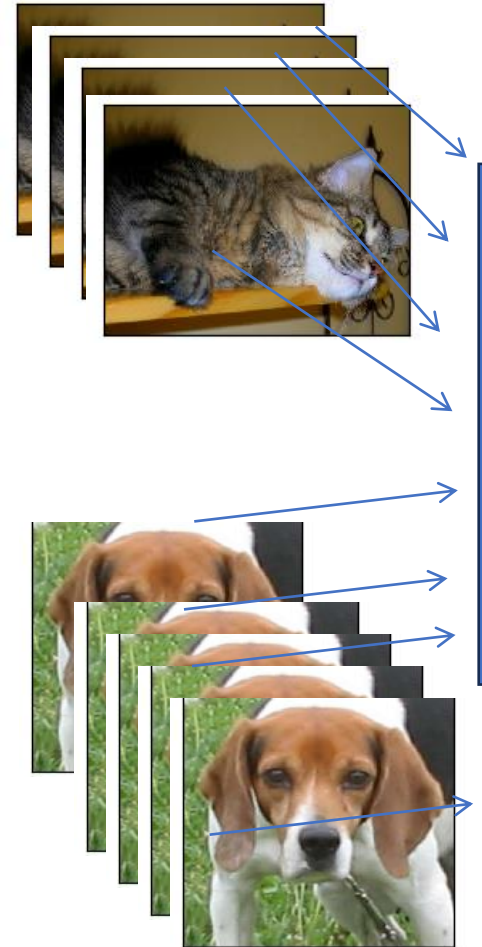
y

samplesxlabels

Prediction (100x2)

y1

samplesxpredic



Dataset Conversions



dataset



Read dataset



Convert dataset
into batches



Pass dataset and
labels to model



Batch_data: **10x3x224x224**(Batchx channel xHx W)

Batch_label:**1x10**

Training Steps

batchxCxHxW

batchxlabels

Forward pass

Out=Model(input)

Compute loss

loss=loss_f(output, labels)

Model parameters optimization

1. Zero gradient optimizer
2. Perform backpropagation on the loss
3. Update the model parameters with respect to gradient

% training loop

For I in range(0,100):

For batch, labels in dataloader:

Imges_batch, labels

%Forward pass

Out=model(images_batch)

%backward pass and update grad

loss=loss_f(Out, labels)

optimizer.zero_grad()

Loss.backward()

Optimizer.step()

Training and validation Steps

% training loop

For batch, labels in dataloader_Train:

Imges_batch, labels

%Forward pass

Out=model(images_batch)

%backward pass and update grad

loss=loss_f(Out, labels)

% Model parameters optimization

optimizer.zero_grad()

Loss.backward()

Optimizer.step()

% Validation/Testing loop

For batch, labels in dataloader_valid:

Imges_batch, labels

%Forward pass

Out=model(images_batch)

Pred=torch.max(out)

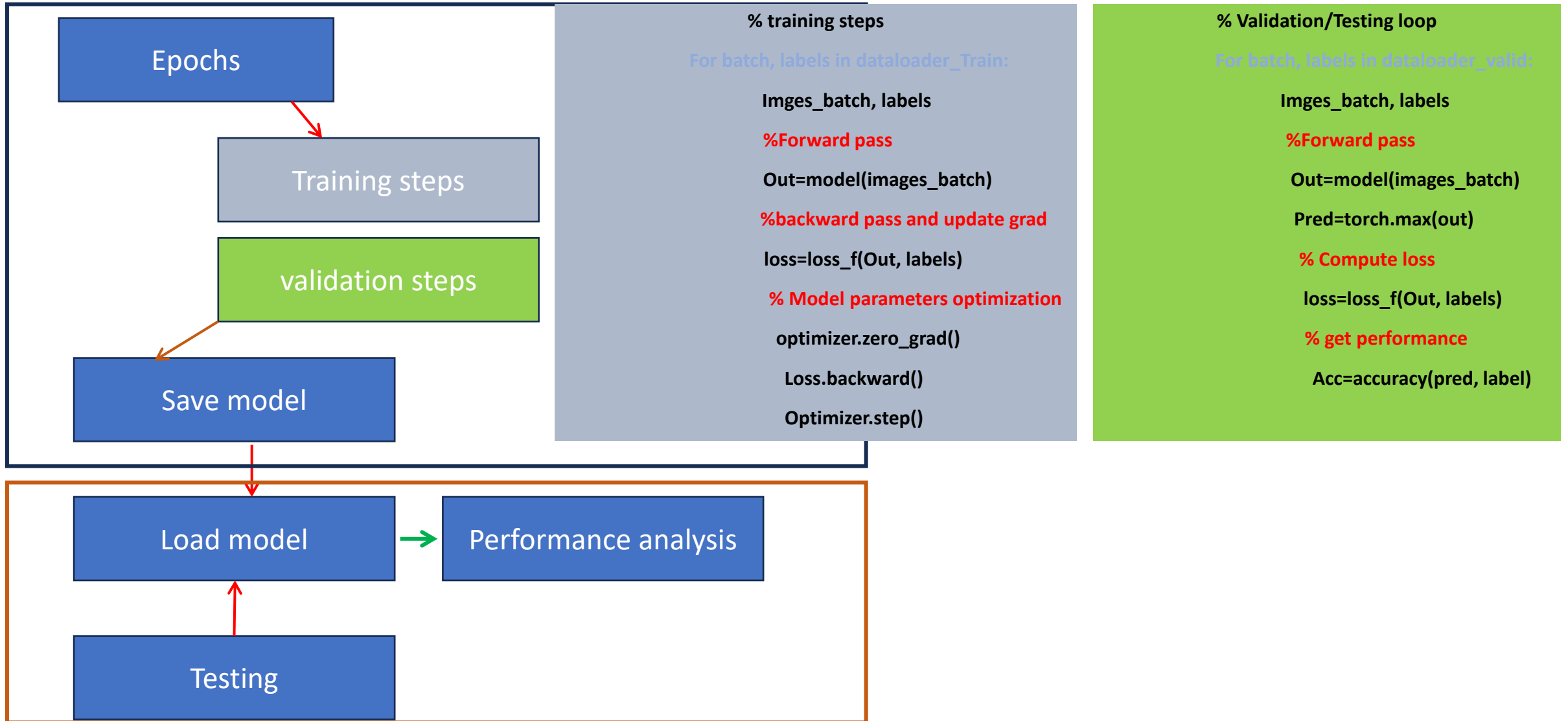
% Compute loss

loss=loss_f(Out, labels)

% get performance

Acc=accuracy(pred, label)

Standard Steps for training and optimization of ANN, CNN



Pytorch basics

PyTorch training loop

```
1 # Pass the data through the model for a number of epochs (e.g. 100)
2 for epoch in range(epochs):
3
4     # Put model in training mode (this is the default state of a model)
5     model.train()
6
7     # 1. Forward pass on train data using the forward() method inside
8     y_pred = model(X_train)
9
10    # 2. Calculate the loss (how different are the model's predictions to the true values)
11    loss = loss_fn(y_pred, y_true)
12
13    # 3. Zero the gradients of the optimizer (they accumulate by default)
14    optimizer.zero_grad()
15
16    # 4. Perform backpropagation on the loss
17    loss.backward()
18
19    # 5. Progress/step the optimizer (gradient descent)
20    optimizer.step()
```

Note: all of this can be turned into a function

Pass the data through the model for a number of **epochs** (e.g. 100 for 100 passes of the data)

Pass the data through the model, this will perform the **forward()** method located within the model object

Calculate the loss value (how wrong the model's predictions are)

Zero the optimizer gradients (they accumulate every epoch, zero them to start fresh each forward pass)

Perform **backpropagation** on the loss function (compute the gradient of every parameter with `requires_grad=True`)

Step the optimizer to update the model's parameters with respect to the gradients calculated by `loss.backward()`

Pytorch basics

PyTorch testing loop

```
1 # Setup empty lists to keep track of model progress
2 epoch_count = []
3 train_loss_values = []
4 test_loss_values = []
5
6 # Pass the data through the model for a number of epochs (e.g. 100) pchcs):
7 for epoch in range(epochs):
8
9     ### Training loop code here ###
10
11     ### Testing starts ###
12
13     # Put the model in evaluation mode
14     model.eval()
15
16     # Turn on inference mode context manager
17     with torch.inference_mode():
18         # 1. Forward pass on test data
19         test_pred = model(X_test)
20
21         # 2. Caculate loss on test data
22         test_loss = loss_fn(test_pred, y_test)
23
24     # Print out what's happening every 10 epochs
25     if epoch % 10 == 0:
26         epoch_count.append(epoch)
27         train_loss_values.append(loss)
28         test_loss_values.append(test_loss)
29         print(f"Epoch: {epoch} | MAE Train Loss: {loss} | MAE Test Loss: {test_loss} ")
```

Note: all of this can be turned into a function

Create empty lists for storing useful values (helpful for tracking model progress)

Tell the model we want to **evaluate** rather than train (this turns off functionality used for training but not evaluation)

Turn on `torch.inference_mode()` context manager to disable functionality such as gradient tracking for inference (gradient tracking not needed for inference) *(faster performance!)*

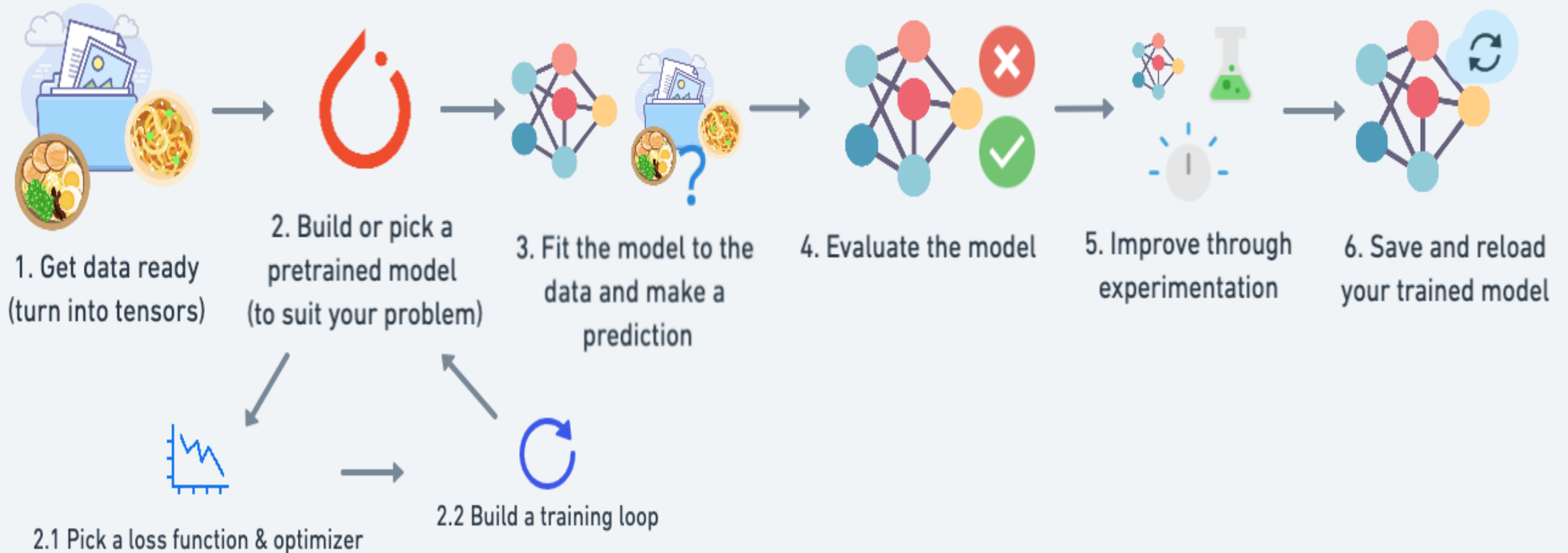
Pass the test data through the model (this will call the model's implemented `forward()` method)

Calculate the **test loss value** (how wrong the model's predictions are on the test dataset, lower is better)

Display **information outputs** for how the model is doing during training/testing every ~10 epochs (note: what gets printed out here can be adjusted for specific problems)

Pytorch basics

A PyTorch Workflow



Pytorch basics

