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

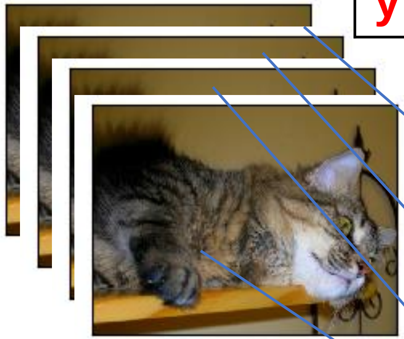
Feature Matrix for classification task

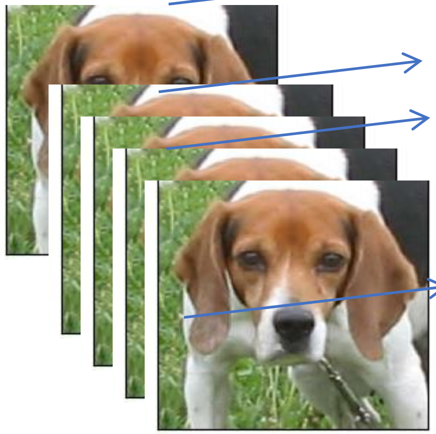
Features							label
Energy	entropy	mean	variance	area	color	texture	0

Features							label
Energy	entropy	mean	variance	area	color	texture	1

Feature Matrix for classification task

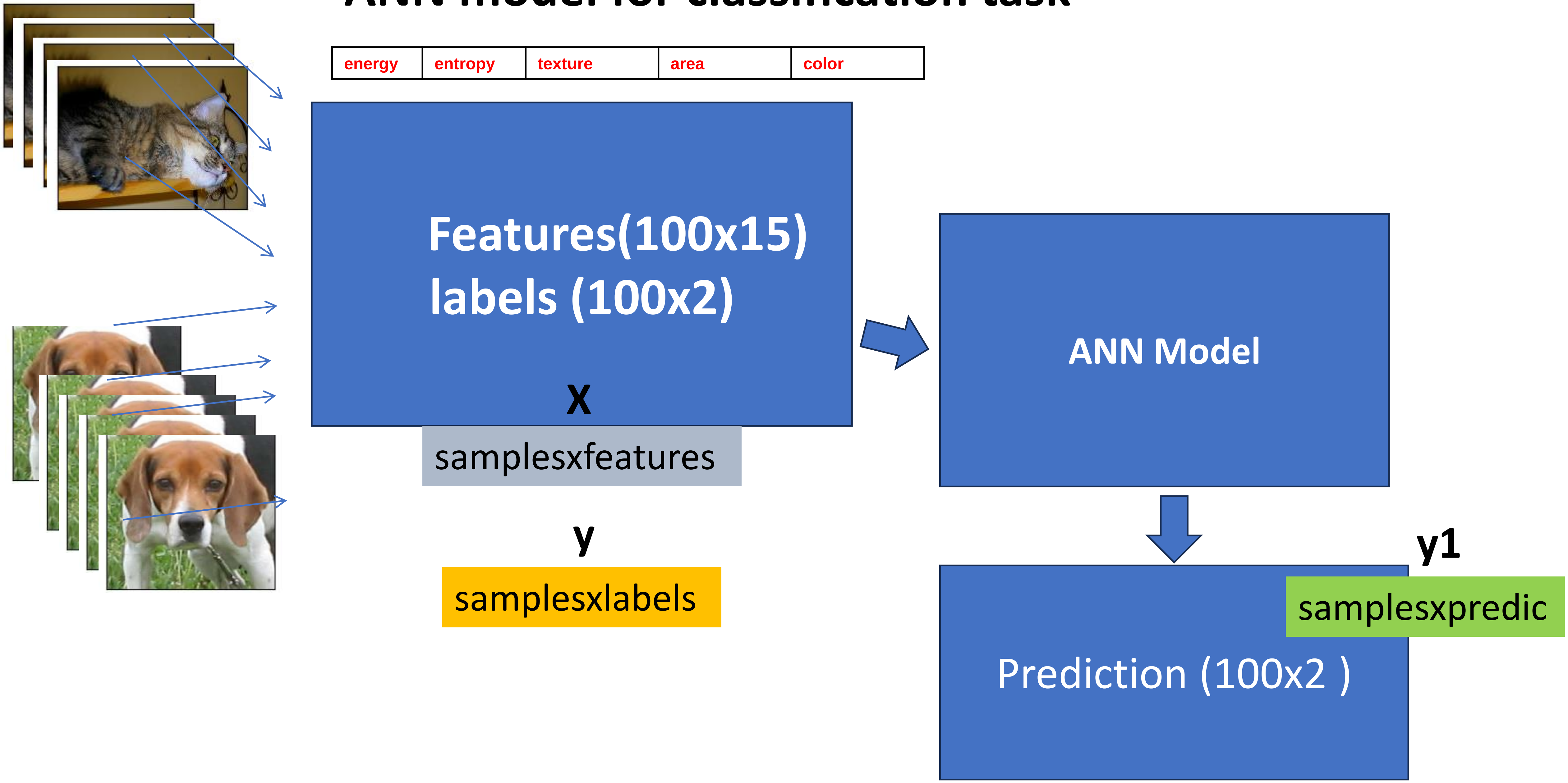
energy	entropy	texture	area	…	…..	…..	color



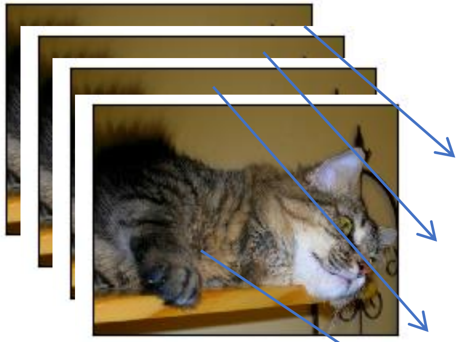


samples	Features								Labels
Cat_samp1	f1	f2	f3	f4	…	…..	…..	f15	0
Cat_samp1	f1	f2	f3	f4	…	…..	…..	f15	0
Cat_samp1	f1	f2	f3	f4	…	…..	…..	f15	0
Cat_samp1	f1	f2	f3	f4	…	…..	…..	f15	0
dog_samp1	f1	f2	f3	f4	…	…..	…..	f15	1
dog_samp1	f1	f2	f3	f4	…	…..	…..	f15	1
dog_samp1	f1	f2	f3	f4	…	…..	…..	f15	1

ANN model for classification task



ANN model for classification task

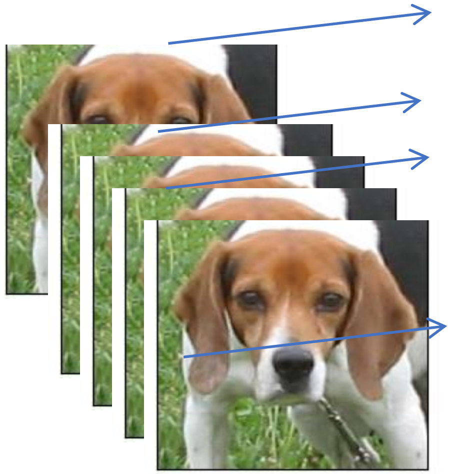


energy	entropy	texture	area	color

Features(100x15)
labels (100x2)

X

samplesxfeatures



y

samplesxlabels

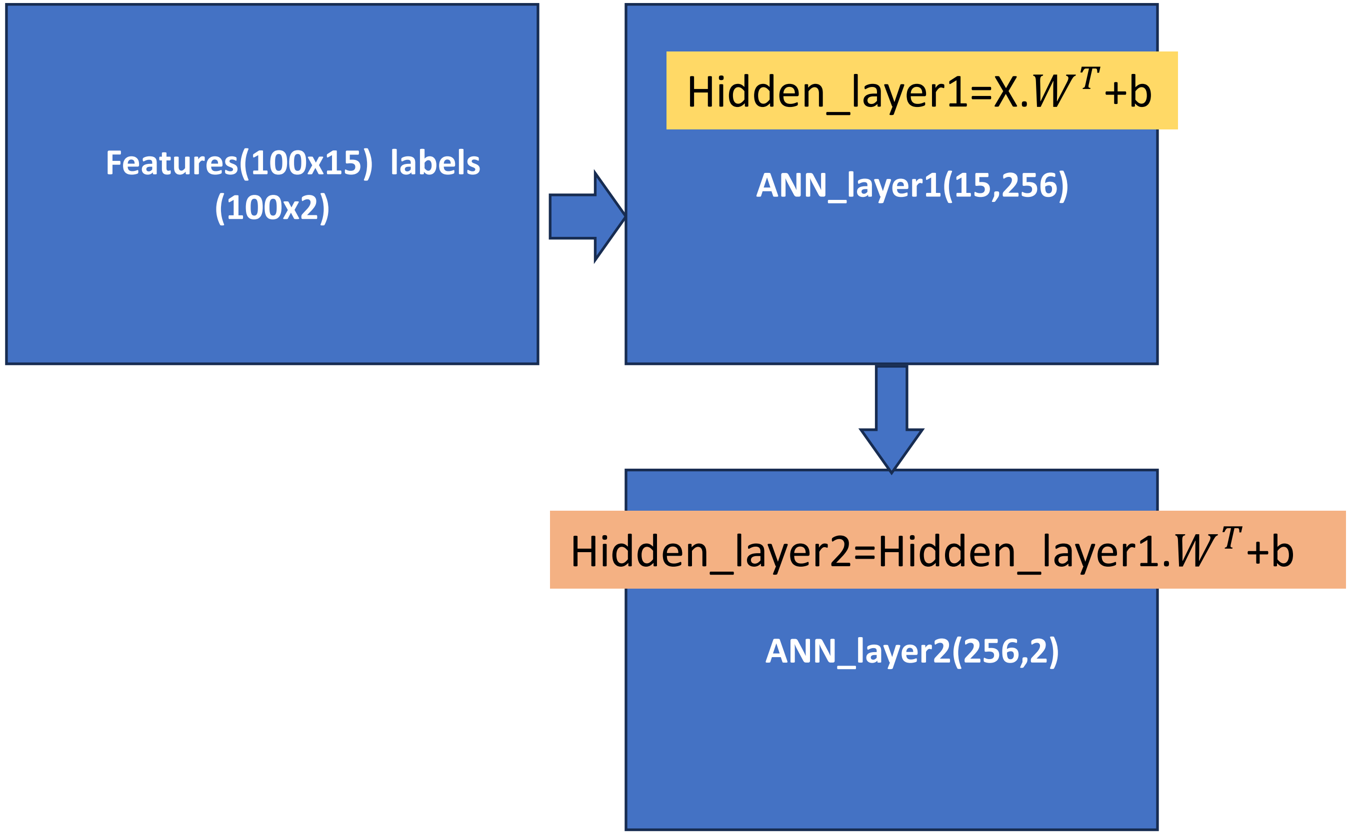
```
# ANN network with two FC layers
## define input feature matrix
X=torch.rand(100,15) ##
samplesxfeatures(100,15)
ANN_layer1=nn.Linear(15,256) #
in_features=15,out_features=256
hidden_features=ANN_layer1(X)
#samplesxoutfeatures(100,256)
#print(hidden_features.shape)
ANN_layer2=nn.Linear(256,2) #
in_features=256,out_features=2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

y1

samplesxpredic

ANN model for classification task

energy	entropy	texture	area	color



X=(samples, in_features)

W=(out_features, in_features)

W^T =(in_features, out_features)

b=(out_features)

X=(100, 15)

W=(256, 15)

W^T =(15, 256)

b=(256)

Hidden_layer1=X.W^T+b
=(100x15)x(15x256)+256=100x256

Hidden_layer2= Hidden_layer1.W^T+b
=(100x256)(256x2)+2=100x2

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)
```

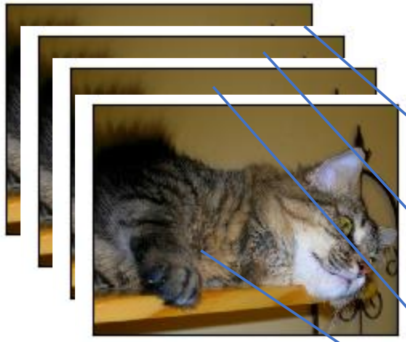
Way2

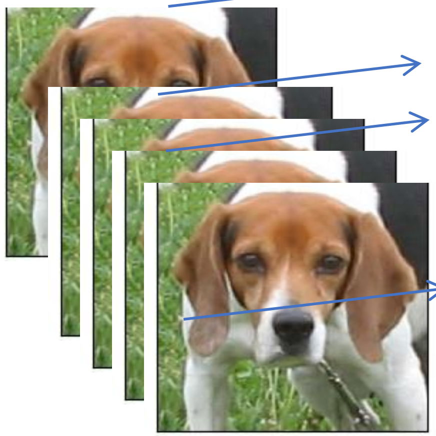
```
########## nn.Sequential module
ANN_model=nn.Sequential(ANN_layer1,ANN_layer2,nn.ReLU())
y=ANN_model(X)
print(y.shape)
```

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

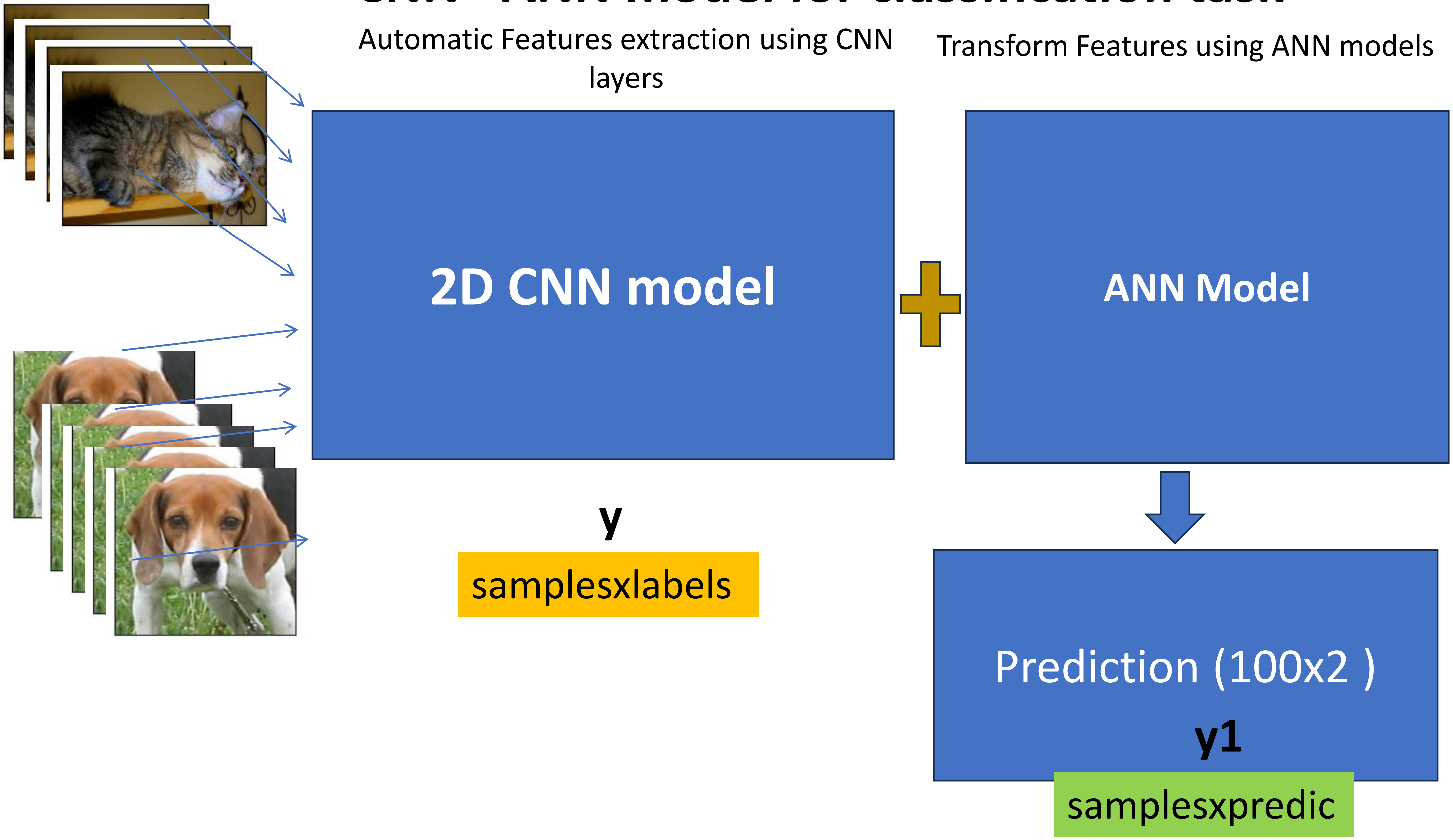




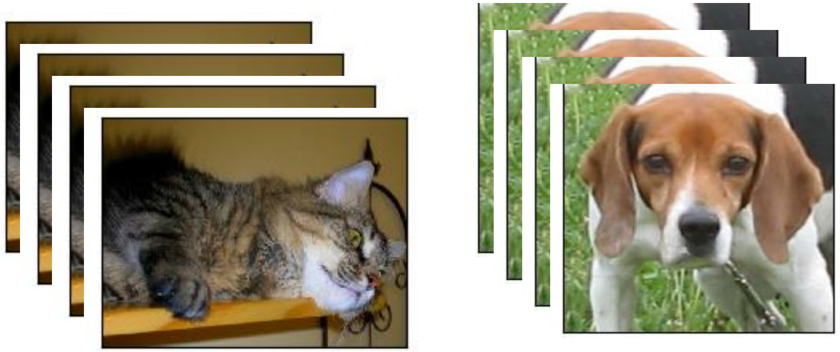
Automatic Feature extraction by **CNN** layers and Transforms features using **ANN** layers

samples	Features								Labels
Cat_samp 1									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	f1	f2	f3	f4	…	…..	…..	f15	1

CNN+ ANN model for classification task



Dataset Conversions



dataset

Read dataset

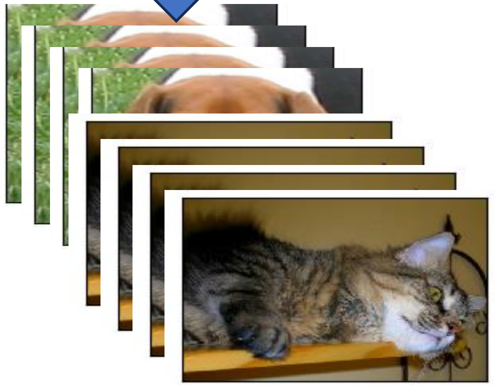
↓

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

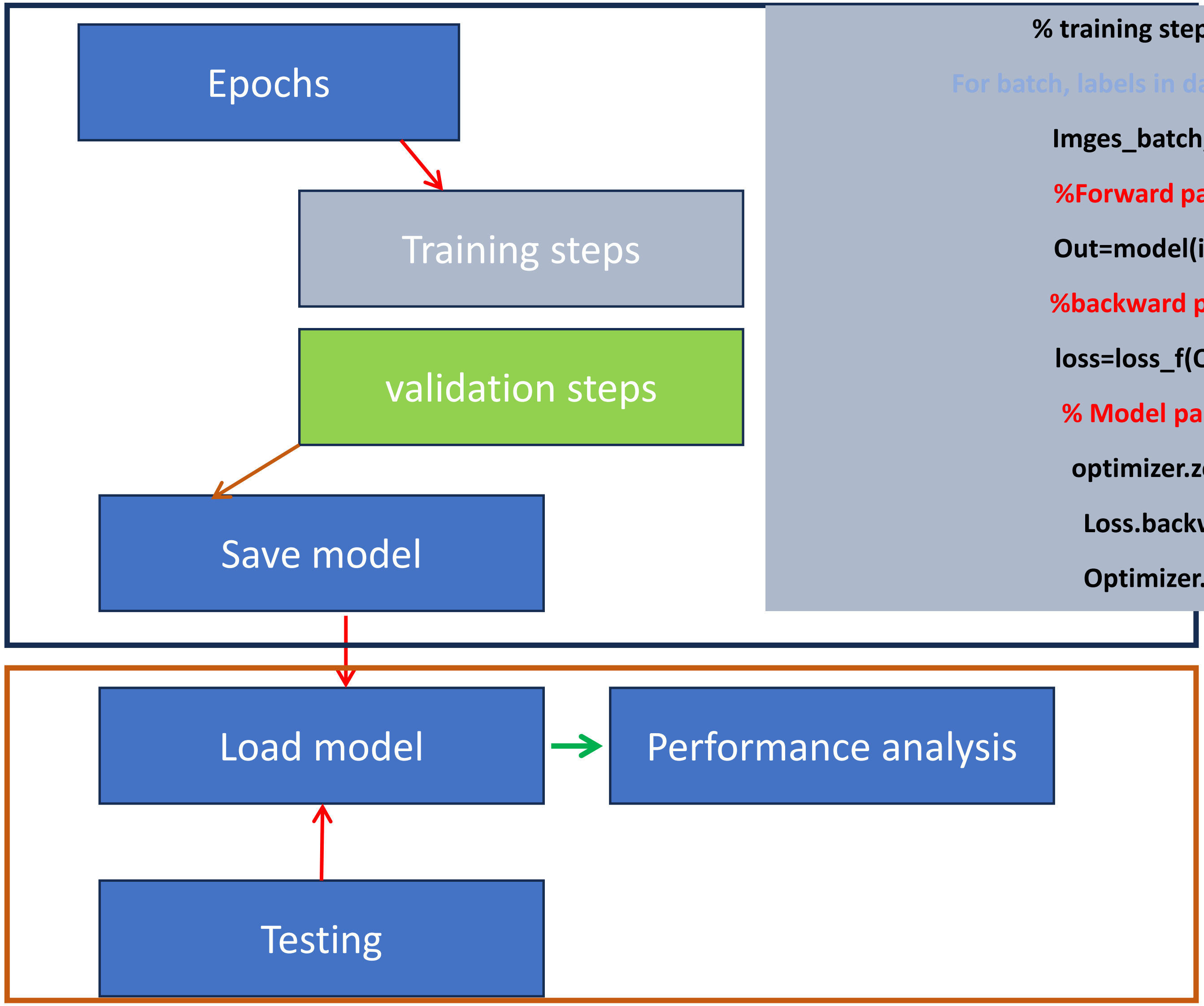
```
% training loop
For I in range(0,100):
For batch, labels in dataloader:
Imges_batch, labels
%Forward pass
Out=model(images_batch)
%backward pass and update grad
loss=loss_f(Out, labels)
optimizer.zero_grad()
Loss.backward()
Optimizer.step()
```

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




```
% training steps
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)
```

Pytorch basics

PyTorch training loop

```python
# Pass the data through the model for a number of epochs (e.g. 100)
for epoch in range(epochs):

    # Put model in training mode (this is the default state of a model)
    model.train()

    # 1. Forward pass on train data using the forward() method inside
    y_pred = model(X_train)

    # 2. Calculate the loss (how different are the model's predictions to the true values)
    loss = loss_fn(y_pred, y_true)

    # 3. Zero the gradients of the optimizer (they accumulate by default)
    optimizer.zero_grad()

    # 4. Perform backpropagation on the loss
    loss.backward()

    # 5. Progress/step the optimizer (gradient descent)
    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

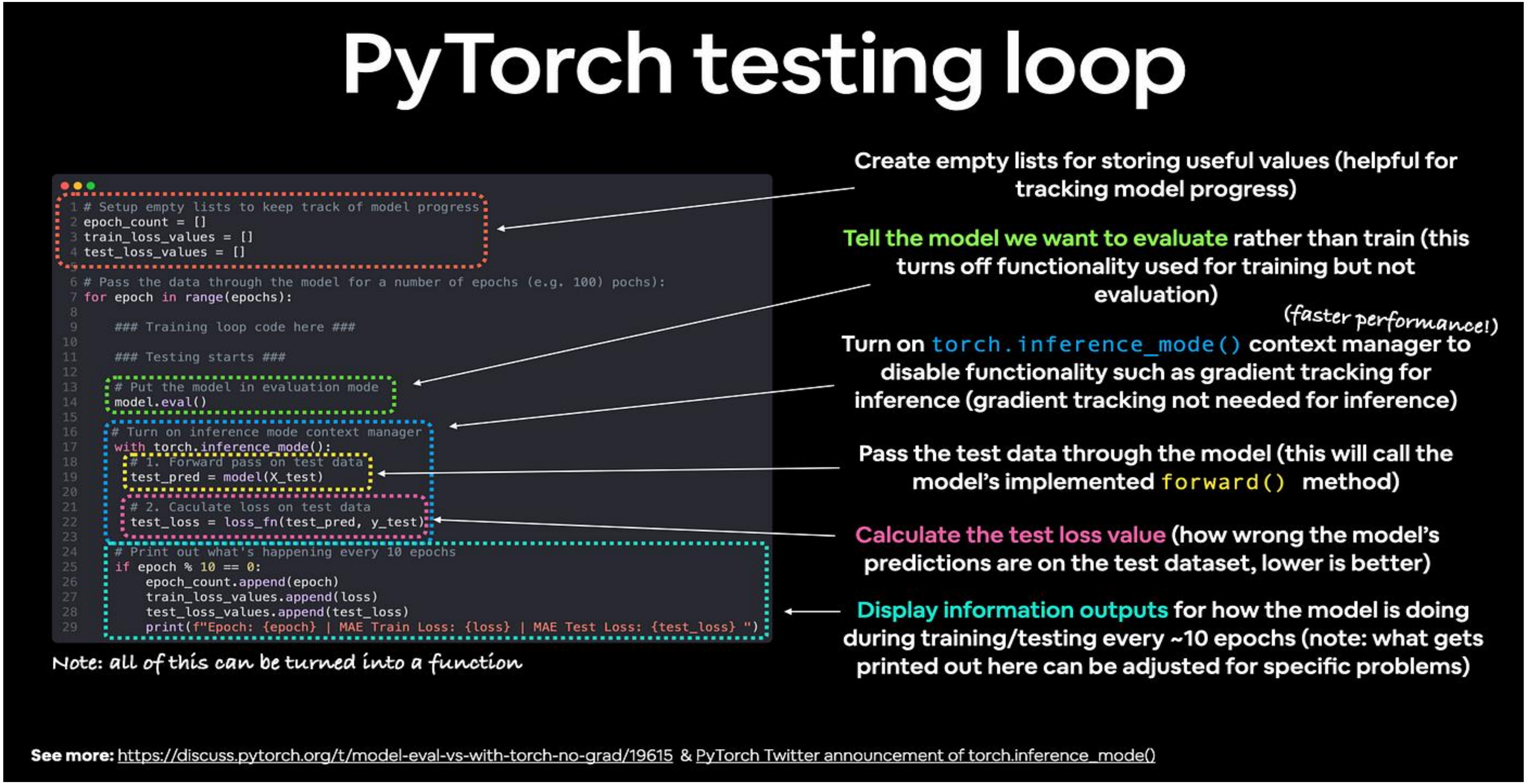
Calculate the loss value (how wrong the model's predictions are)

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

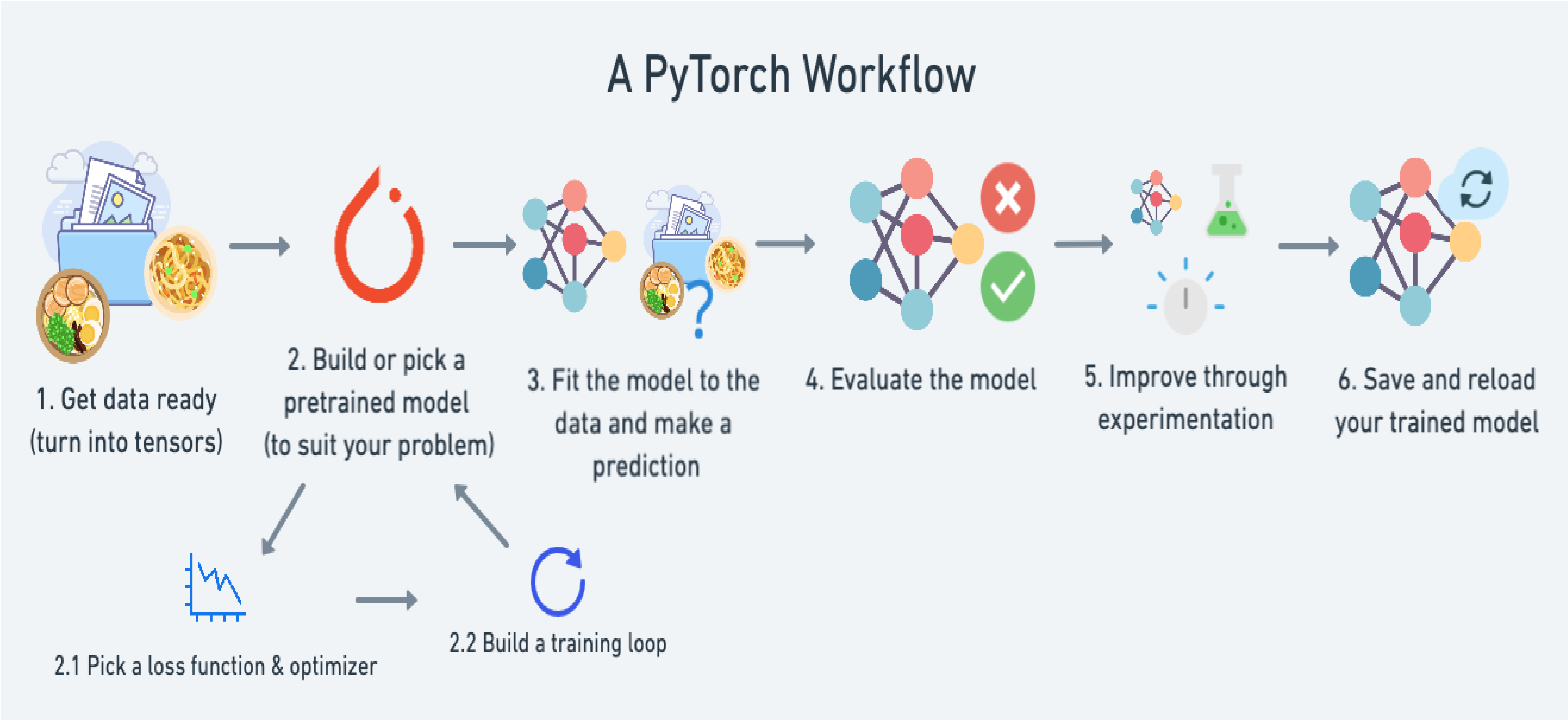
Perform backpropagation on the loss function (compute the gradient of every parameter with `requires_grad=True`)

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

Pytorch basics



Pytorch basics



Pytorch basics