FIT3181/5215 Deep Learning

Convolutional Neural Networks

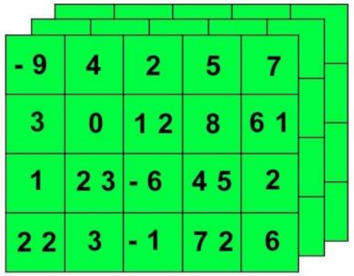
Trung Le and Teaching team
Department of Data Science and AI
Faculty of Information Technology, Monash University
Email: **trunglm@monash.edu**

Outline

- Fundamentals of convolutional neural networks (Tute 3a)
- Implementing CNNs in PyTorch (Tute 3b)
  - MiniVGG network
- Implementing CNNs with Data Augmentation (Tute 3c)
- Visualize feature maps and filters (Tute 3c)

Fundamentals of convolutional neural networks

Conv2D in PyTorch

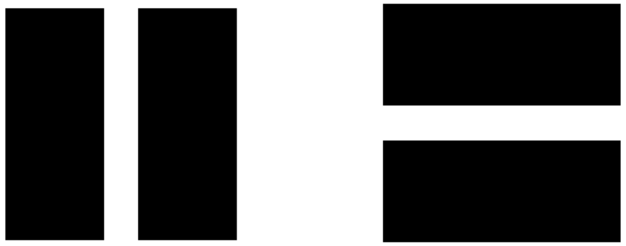


Examples of tensors. [Source: datacamp]

```
import numpy as np
from sklearn.datasets import load_sample_image

# Load sample images
china = load_sample_image("china.jpg")[80:360, 70:390]
flower = load_sample_image("flower.jpg")[80:360, 130:450]
batch = np.array([china, flower], dtype=np.float32)
print(batch.shape)
```

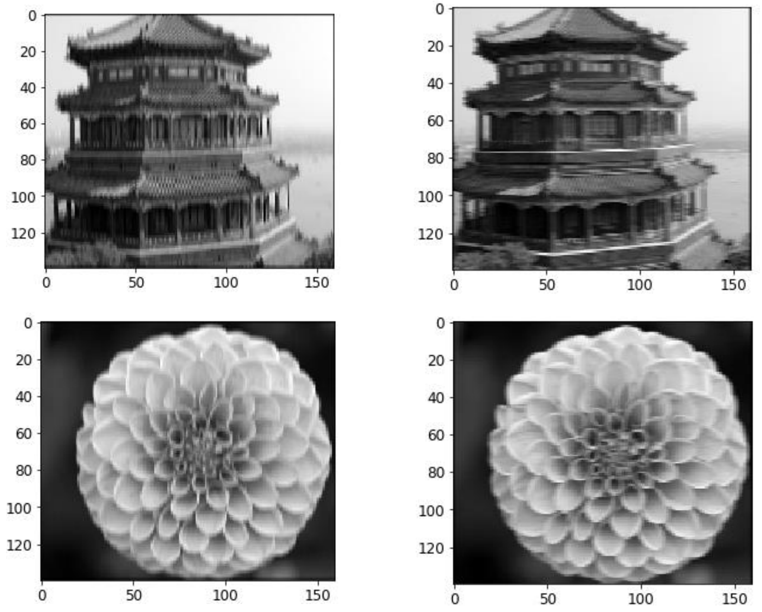
```
(2, 280, 320, 3)
```

```
# Create 2 filters
filters = np.zeros(shape=(7, 7, channels, 2), dtype=np.float32)
filters[:, 3, :, 0] = 1  # vertical line, why?
filters[3, :, :, 1] = 1  # horizontal line, why?
```



```
output = torch.nn.functional.conv2d(input = batch_tensor, weight= filters_tensor,
                                    stride =(2,2), padding=3) #padding means number of padding to left and right
output = output.numpy()

print("Output shape:" + str(output.shape))

plot_image(output[0, 0, :, :], axis=True) # plot 1st image's 1nd feature map, channel 0
plot_image(output[0, 1, :, :], axis=True) # plot 1st image's 2nd feature map, channel 1
plot_image(output[1, 0, :, :], axis=True) # plot 2nd image's 1nd feature map, channel 0
plot_image(output[1, 1, :, :], axis=True) # plot 2nd image's 2nd feature map, channel 1
```

```
Output shape:(2, 2, 140, 160)
```



$$out\_height = \left\lfloor \frac{in\_height + 2p - kernel\_size}{stride} \right\rfloor + 1 \qquad out\_width = \left\lfloor \frac{in\_width + 2p - kernel\_size}{stride} \right\rfloor + 1$$

Example of pooling with PyTorch

Max pooling

```
output = torch.nn.functional.max_pool2d(input = batch_tensor,
                                        kernel_size=(2,2), stride= (2,2))
output = output.numpy()
print(batch[0].shape)
print(output[0].shape)
plot_image(batch[0].transpose((1,2,0)).astype(np.int32), scale=True) # plot the 1st original image
plot_image(output[0].transpose((1,2,0)).astype(np.int32), scale=True) # plot the output for the 1st image
```

```
(3, 280, 320)
(3, 140, 160)
```

Average pooling

```
output = torch.nn.functional.avg_pool2d(input = batch_tensor,
                                        kernel_size=(2,2), stride= (2,2), padding =0)
output = output.numpy()
print(batch[0].shape)
print(output[0].shape)
plot_image(batch[0].transpose((1,2,0)).astype(np.int32), scale=True) # plot the 1st original image
plot_image(output[0].transpose((1,2,0)).astype(np.int32), scale=True) # plot the output for the 1st image
```

```
(3, 280, 320)
(3, 140, 160)
```

Additional reading

- The effects of kernels to the input image
  - Small blur, large blur filters
  - Sharpening filter
  - Laplacian filter
  - SobelX, sobelY filters
  - Emboss filter

Implementing CNNs in PyTorch

MiniVGG for CIFAR-10

Our Tutorial

Layer Type	Output Size	Filter Size / Stride
INPUT IMAGE	$32 \times 32 \times 3$	
CONV	$32 \times 32 \times 32$	$3 \times 3, K = 32$
ACT	$32 \times 32 \times 32$	
BN	$32 \times 32 \times 32$	
CONV	$32 \times 32 \times 32$	$3 \times 3, K = 32$
ACT	$32 \times 32 \times 32$	
BN	$32 \times 32 \times 32$	
POOL	$16 \times 16 \times 32$	2×2
DROPOUT	$16 \times 16 \times 32$	
CONV	$16 \times 16 \times 64$	$3 \times 3, K = 64$
ACT	$16 \times 16 \times 64$	
BN	$16 \times 16 \times 64$	
CONV	$16 \times 16 \times 64$	$3 \times 3, K = 64$
ACT	$16 \times 16 \times 64$	
BN	$16 \times 16 \times 64$	
POOL	$8 \times 8 \times 64$	2×2
DROPOUT	$8 \times 8 \times 64$	
FC	512	
ACT	512	
BN	512	
DROPOUT	512	
FC	10	
SOFTMAX	10	

50,000 images
10 classes

MiniVGG for CIFAR-10 dataset

Download CIFAR-10
Process dataset

```
transform = transforms.Compose([
    transforms.ToTensor(),  # Convert images to PyTorch tensors
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalize the images, each R,G,B value is
])

full_train_set = torchvision.datasets.CIFAR10("./data", download=True, transform=transform)
full_test_set = torchvision.datasets.CIFAR10("./data", download=True, train=False, transform=transform)
```

```
total_num_train = len(full_train_set)
total_num_test = len(full_test_set)
train_valid_idx = torch.randperm(total_num_train)
train_set = torch.utils.data.Subset(full_train_set, train_valid_idx[:5000])
valid_set = torch.utils.data.Subset(full_train_set, train_valid_idx[5000:10000])

test_idx = torch.randperm(total_num_test)
test_set = torch.utils.data.Subset(full_test_set, test_idx[:5000])

print("Traing set\n\t-Number of samples:\t{}\n\t-Shape of 1 sample:\t{}".format(len(train_set), list(train_set[0][0].shape)))
print("Valid set\n\t-Number of samples:\t{}\n\t-Shape of 1 sample:\t{}".format(len(valid_set), list(valid_set[0][0].shape)))
print("Test set\n\t-Number of samples:\t{}\n\t-Shape of 1 sample:\t{}".format(len(test_set), list(test_set[0][0].shape)))

train_loader = torch.utils.data.DataLoader(train_set, batch_size=32, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=32)
valid_loader = torch.utils.data.DataLoader(valid_set, batch_size=32)
```

Visualize dataset

MiniVGG for CIFAR-10 dataset

Build up MiniVGG

```
def create_vgg():
    vgg_model = nn.Sequential(
        #nn.LazyConv2d(32, kernel_size=3, padding=1),
        nn.Conv2d(3, 32, kernel_size=3, padding=1), #[32,32,32]
        nn.BatchNorm2d(32, momentum=0.1),
        nn.ReLU(),
        #nn.LazyConv2d(32, kernel_size=3, padding=1),
        nn.Conv2d(32, 32, kernel_size=3, padding=1), #[32,32,32]
        nn.BatchNorm2d(32, momentum=0.1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2), #down-sample by two #[32,16,16]
        nn.Dropout(p=0.25),

        #nn.LazyConv2d(64, kernel_size=3, padding=1),
        nn.Conv2d(32, 64 , kernel_size=3, padding=1), #[64,16,16]
        nn.BatchNorm2d(64, momentum=0.1),
        nn.ReLU(),
        #nn.LazyConv2d(64, kernel_size=3, padding=1)
        nn.Conv2d(64, 64, kernel_size=3, padding=1), #[64,16,16]
        nn.BatchNorm2d(64, momentum=0.1),
        nn.ReLU(),
        nn.LazyConv2d(64, kernel_size=3, padding=1),
        nn.BatchNorm2d(64, momentum=0.1),
        nn.ReLU(),
        nn.MaxPool2d(kernel_size=2), #down-sample by two [64,8,8]
        nn.Dropout(p=0.25),

        nn.Flatten(1), #64x8x8
        #nn.LazyLinear(512),
        nn.Linear(64*8*8, 512), #512
        nn.ReLU(),
        #nn.LazyLinear(10)
        nn.Linear(512, 10),
      )
    return vgg_model
```

Declare loss and optimizer

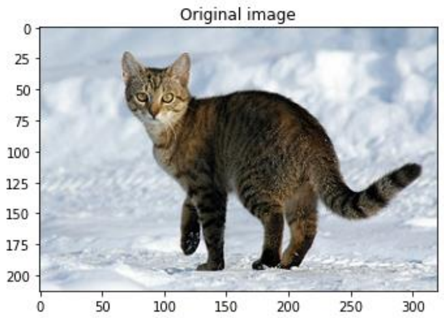
```
# Loss and optimizer
learning_rate = 0.001
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(vgg_model.parameters(), lr=learning_rate)
```

Train the model

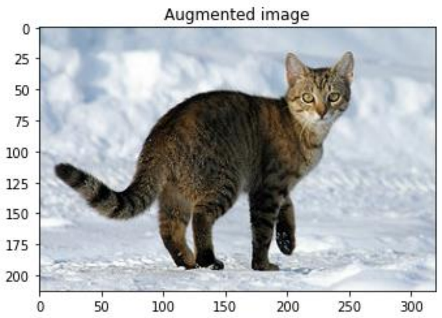
```
def fit(model= None, train_loader = None, valid_loader= None, optimizer = None,
        num_epochs = 50, verbose = True, seed= 1234):
  torch.manual_seed(seed)
  # Move the model to the device before initializing the optimizer
  model.to(device)
  if optimizer == None:
    optim = torch.optim.Adam(model.parameters(), lr = 0.001) # Now initialize optimizer with model on GPU
  else:
    optim = optimizer
  history = dict()
  history['val_loss'] = list()
  history['val_acc'] = list()
  history['train_loss'] = list()
  history['train_acc'] = list()

  for epoch in range(num_epochs):
      for (X, y) in train_loader:
          # Move input data to the same device as the model
          X,y = X.to(device), y.to(device)
          # Forward pass
          outputs = model(X.type(torch.float32))
          loss = loss_fn(outputs, y.type(torch.long))
          # Backward and optimize
          optim.zero_grad()
          loss.backward()
          optim.step()
      #losses and accuracies for epoch
      val_loss = compute_loss(model, loss_fn, valid_loader)
      val_acc = compute_acc(model, valid_loader)
      train_loss = compute_loss(model, loss_fn, train_loader)
      train_acc = compute_acc(model, train_loader)
      history['val_loss'].append(val_loss)
      history['val_acc'].append(val_acc)
      history['train_loss'].append(train_loss)
      history['train_acc'].append(train_acc)
      if not verbose: #verbose = True means we do show the training information during training
        print(f"Epoch {epoch+1}/{num_epochs}")
        print(f"train loss= {train_loss:.4f} - train acc= {train_acc*100:.2f}% - valid loss= {val_loss:.4f} - valid acc= {val_acc*100:.2f}%")
  return history
```

Data augmentation

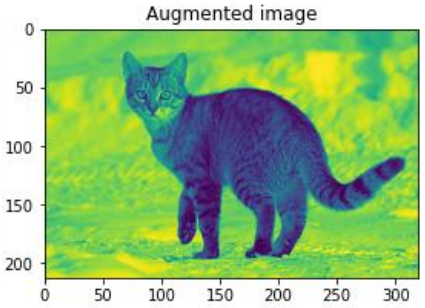
- Apply the **simple transformations** over **training images** to augment data. Model will be **challenged** with **diverge data examples** which might be **encountered** in the testing set
  - Rotation, Width Shifting, Height Shifting, Brightness, Shear Intensity, Zoom, Channel Shift, Horizontal Flip, Vertical Flip
  - https://medium.com/mlait/image-data-augmentation-image-processing-in-tensorflow-part-2-b77237256df0



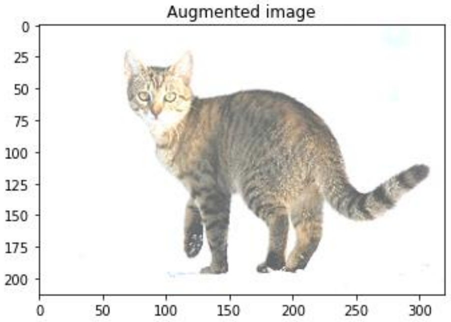
Original image



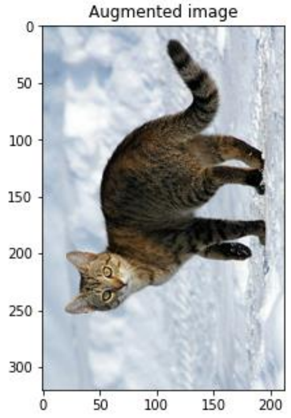
Flip image



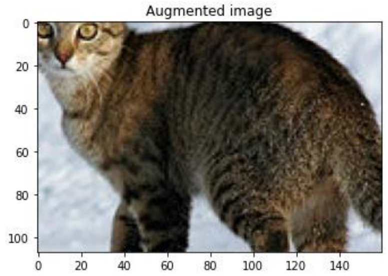
Grey-scale image



Change image brightness



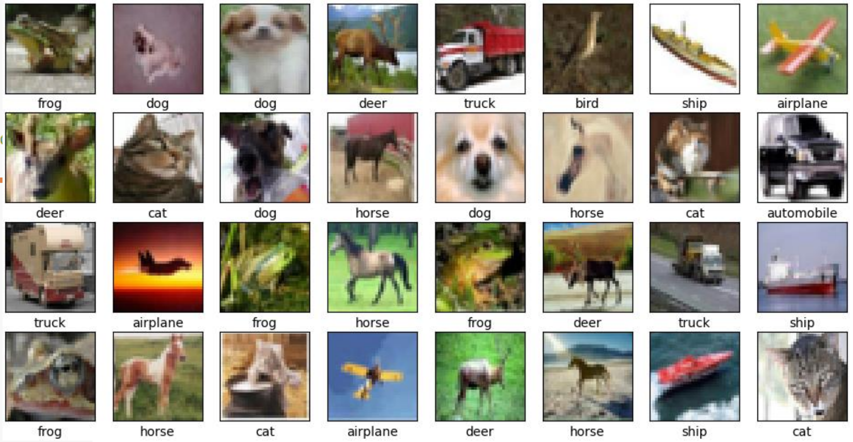
Rotate image



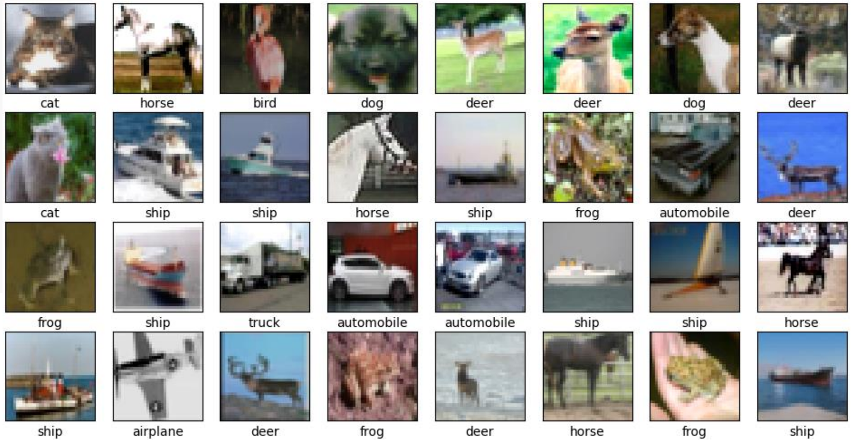
Center crop image

Data augmentation for training our model

```
test_transform = transforms.Compose([transforms.ToTensor(),
                                     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),  # Normalize the images, each R,G,B value is normalized with mean=0.5 and std=0.5
                                     transforms.Resize((32,32)),  #resises the image so it can be perfect for our model.
                                     ])

train_transform = transforms.Compose( transforms.Resize((32,32)),  #resises the image so it can be perfect for our model.
                                      transforms.RandomHorizontalFlip(), # FLips the image w.r.t horizontal axis
                                      #transforms.RandomRotation(4),     #Rotates the image to a specified angel
                                      #transforms.RandomAffine(0, shear=10, scale=(0.8,1.2)), #Performs actions like zooms, change shear angles.
                                      transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2), # Set the color params
                                      transforms.ToTensor(), # convert the image to tensor so that it can work with torch
                                      transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),  # Normalize the images, each R,G,B value is normalized with mean=0.5 an
                                      ])

full_train_set = torchvision.datasets.CIFAR10("./data", download=True, transform=train_transform)     # Apply train_transform to train set
full_valid_set = torchvision.datasets.CIFAR10("./data", download=True, transform=test_transform)      # Apply test_transform to generate valid set
full_test_set = torchvision.datasets.CIFAR10("./data", download=True, train=False, transform=test_transform)

n_train, n_valid, n_test = 5000, 5000, 5000

total_num_train = len(full_train_set)
total_num_test = len(full_test_set)
train_valid_idx = torch.randperm(total_num_train)
train_set = torch.utils.data.Subset(full_train_set, train_valid_idx[:n_train])
valid_set = torch.utils.data.Subset(full_valid_set, train_valid_idx[n_train:n_train+n_valid])

test_idx = torch.randperm(total_num_test)
test_set = torch.utils.data.Subset(full_test_set, test_idx[:n_test])

print("Traing set\n\t-Number of samples:\t{}\n\t-Shape of 1 sample:\t{}".format(len(train_set), list(train_set[0][0].shape)))
print("Valid set\n\t-Number of samples:\t{}\n\t-Shape of 1 sample:\t{}".format(len(valid_set), list(valid_set[0][0].shape)))
print("Test set\n\t-Number of samples:\t{}\n\t-Shape of 1 sample:\t{}".format(len(test_set), list(test_set[0][0].shape)))

train_loader = torch.utils.data.DataLoader(train_set, batch_size=32, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_set, batch_size=32)
valid_loader = torch.utils.data.DataLoader(valid_set, batch_size=32)

img_train = [train_set[idx][0].numpy().transpose((1,2,0)) for idx in range(32)]
label_train = [train_set[idx][1] for idx in range(32)]
```



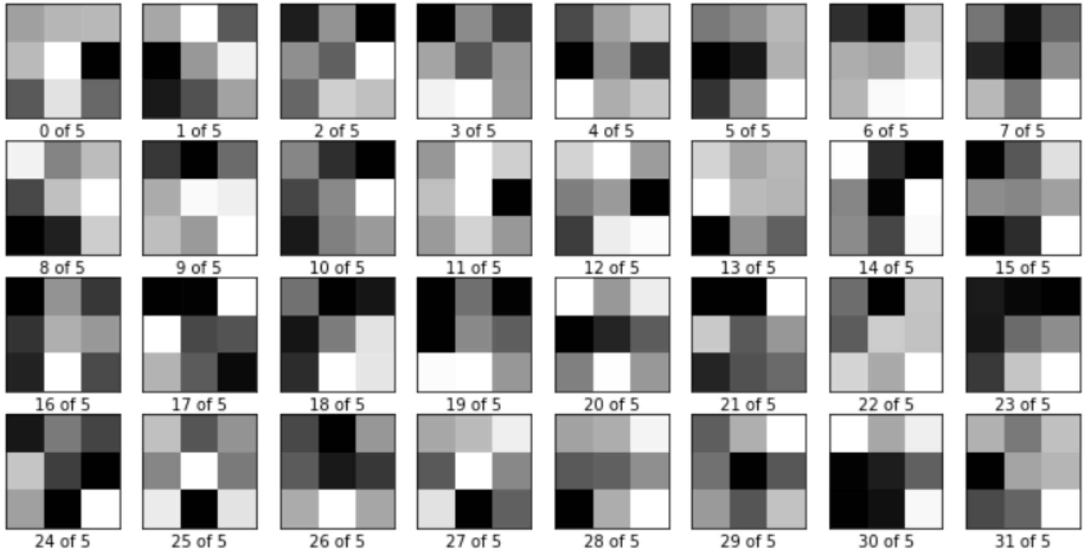
Without augmentation



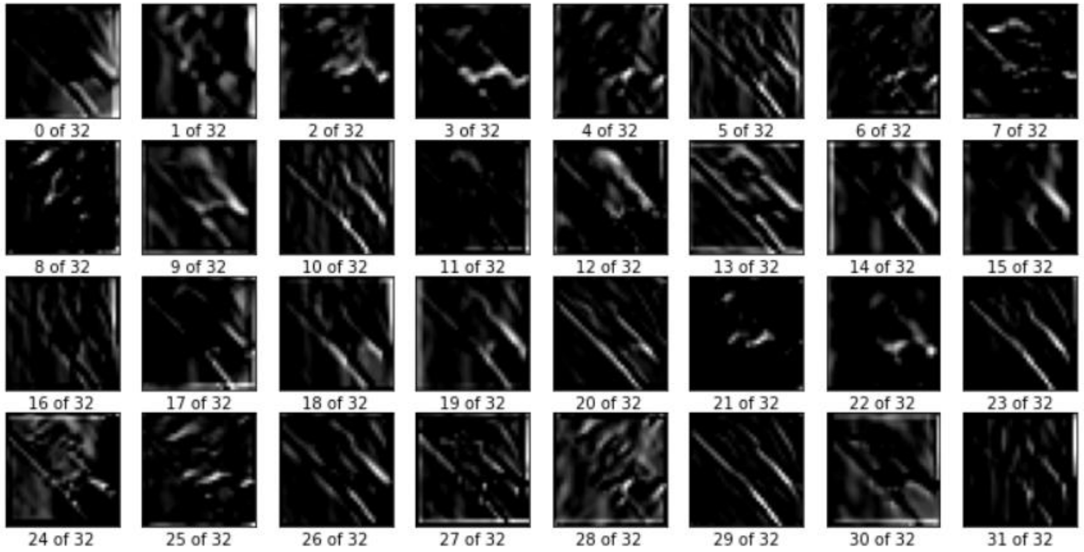
With augmentation

Visualize feature maps and filters

Visualize filters



Visualize feature maps



Thanks for your attention!

Any question?