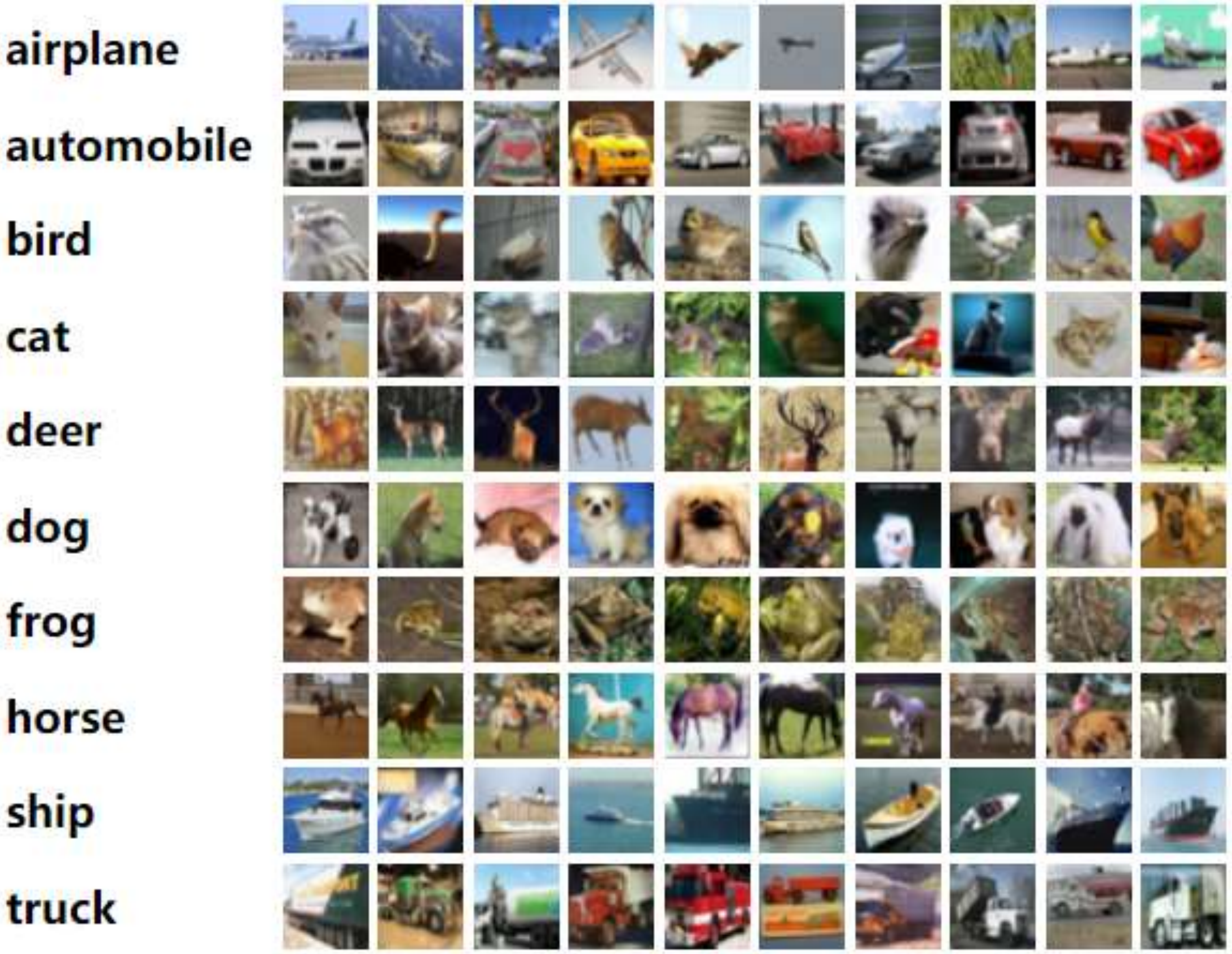
Enhanced Object Recognition Using CIFAR-10:

A comparative Study of TensorFlow, PyTorch and Fine-Tuned AlexNet

Prepared By:
Maryam Ibrahim – 23012132
Mohamed Walied – 23011501

CIFAR-10 Dataset



Total 60,000 images

50,000 for training
(5000 images per class)
10,000 for testing
(1000 images per class)

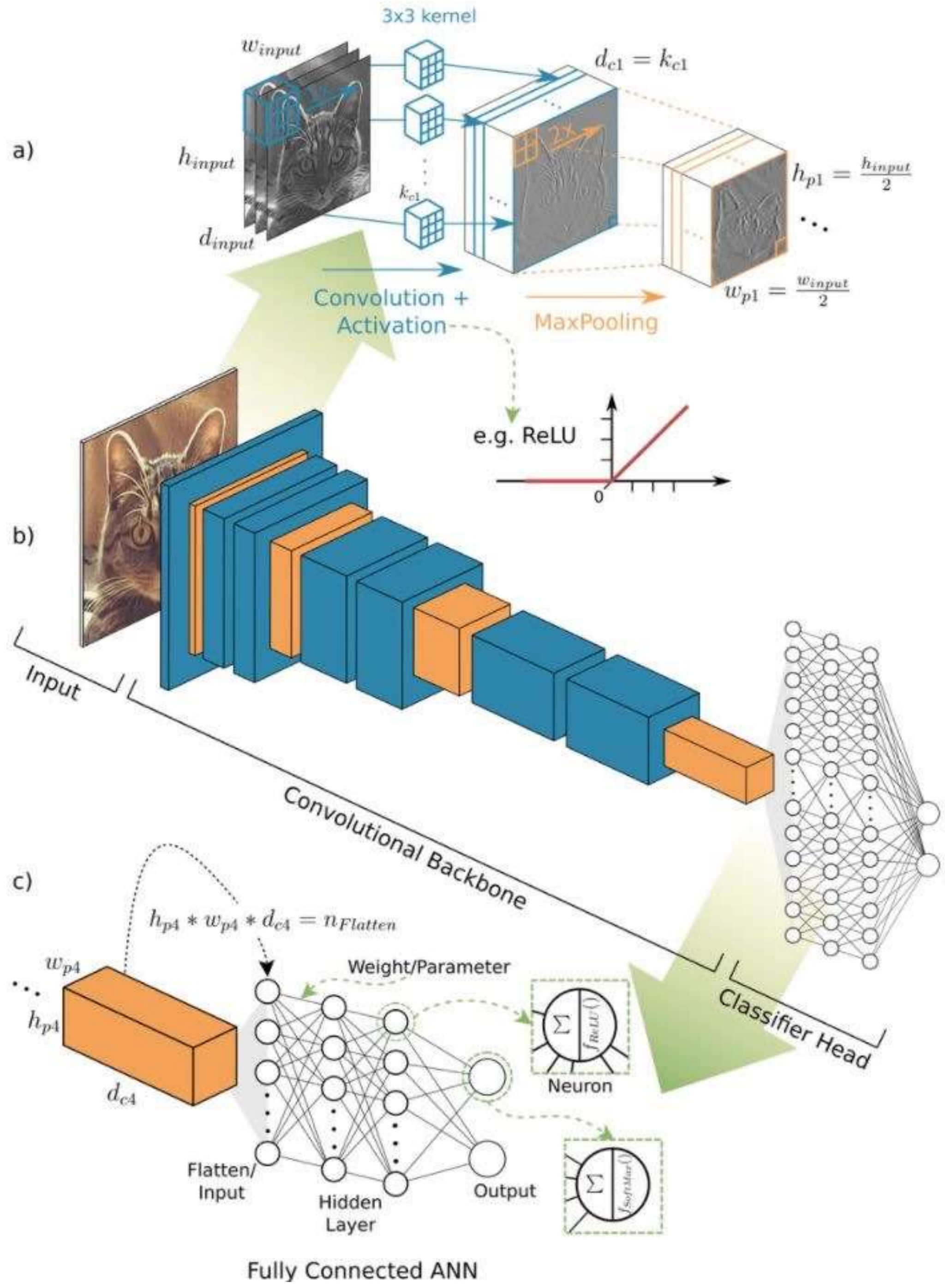
Challenges in Dataset

- The brightness of some images was very low so we add in the augmentations some changes in the brightness with 20%.
- We also make in the augmentations a horizontal flip and random cropping to add more diversity on the dataset.
- therefore we resize it.

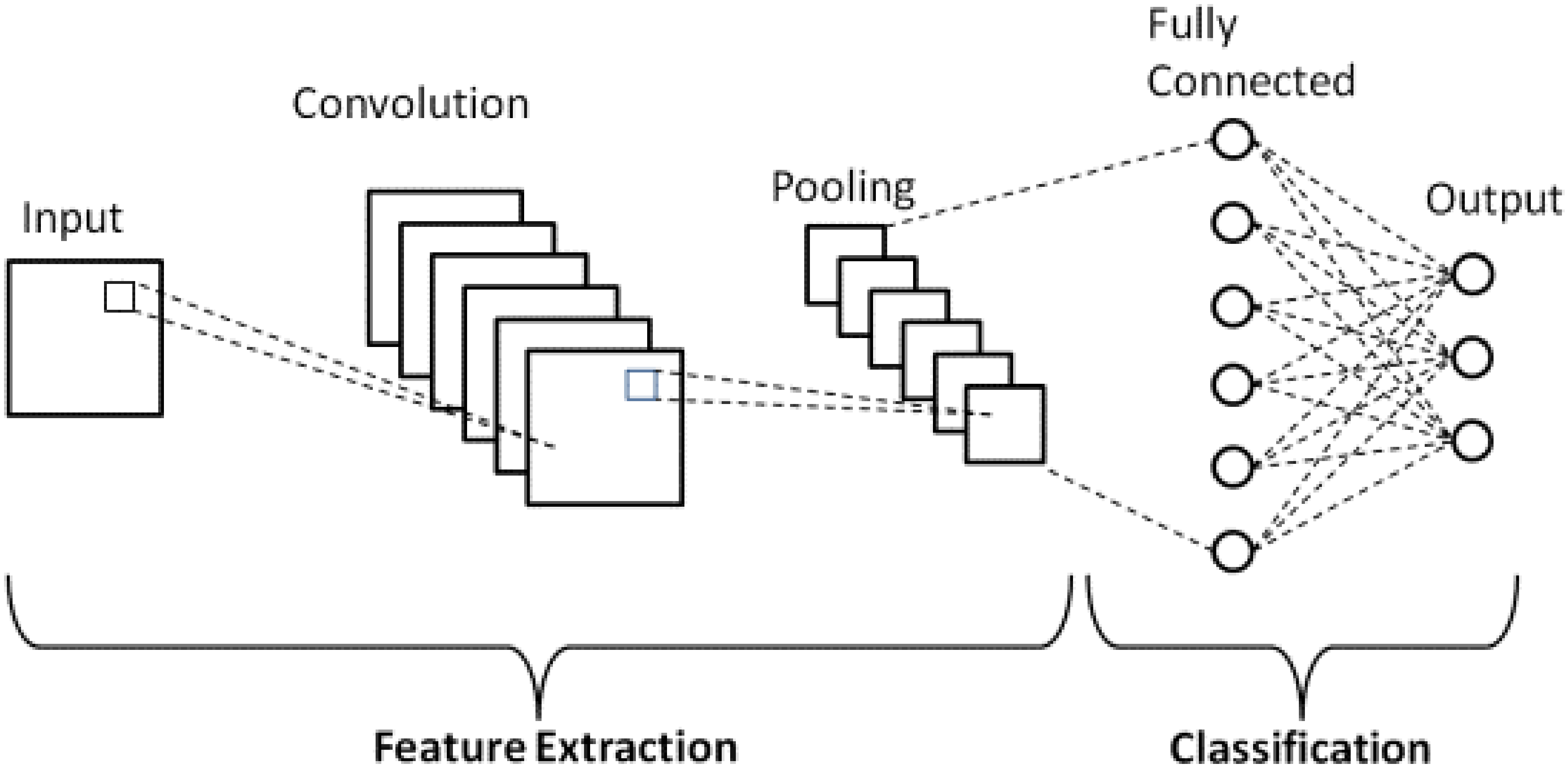
- Images should be normalized for good results in Neural Network.
- Images have to be transformed to 3D-Tensor so PyTorch can deal with it.
- AlexNet must take the data in dimensions of 224x224

CNN Models

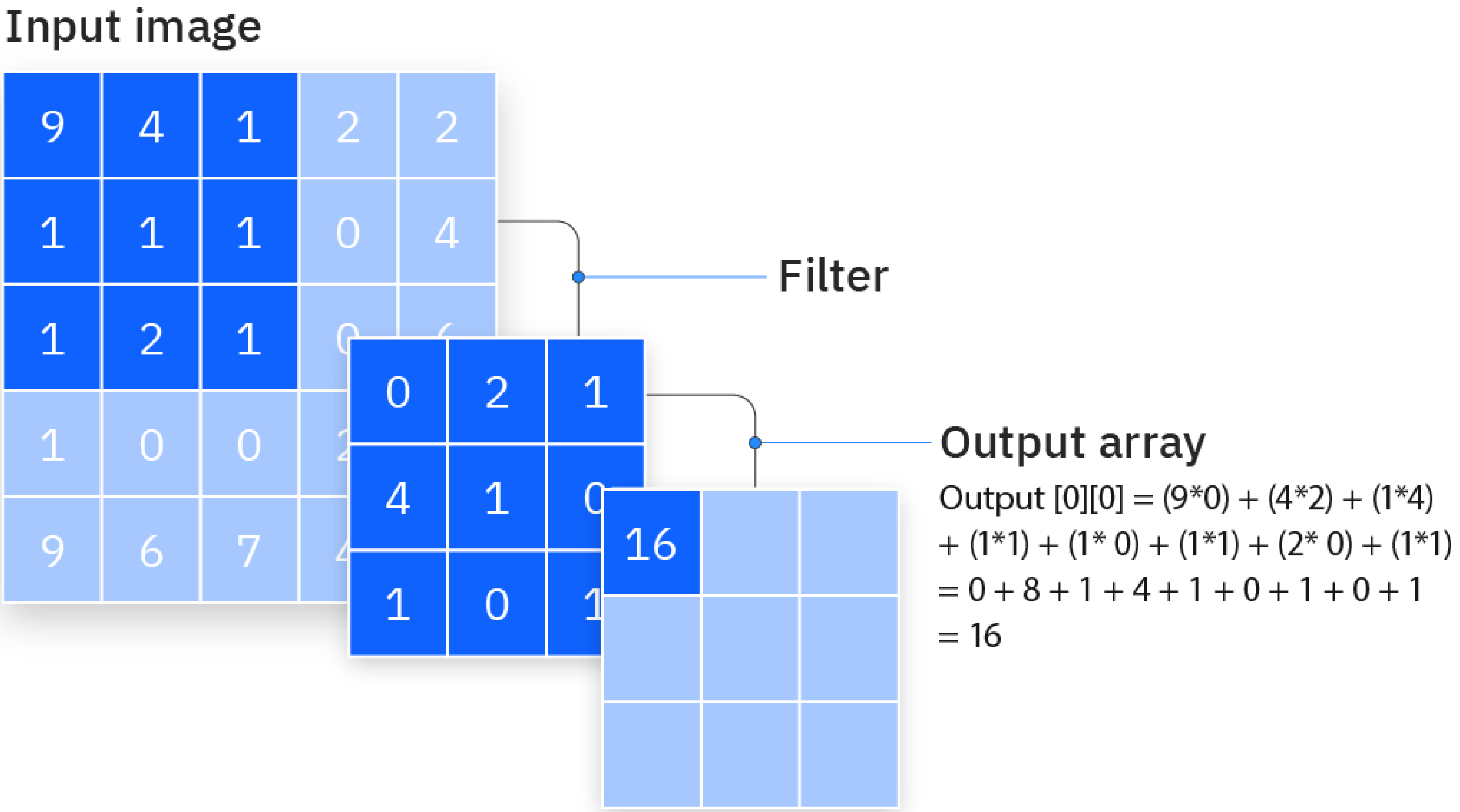
1. CNN architecture in PyTorch.
2. CNN architecture in TensorFlow.
3. Fine-Tuning AlexNet in PyTorch.



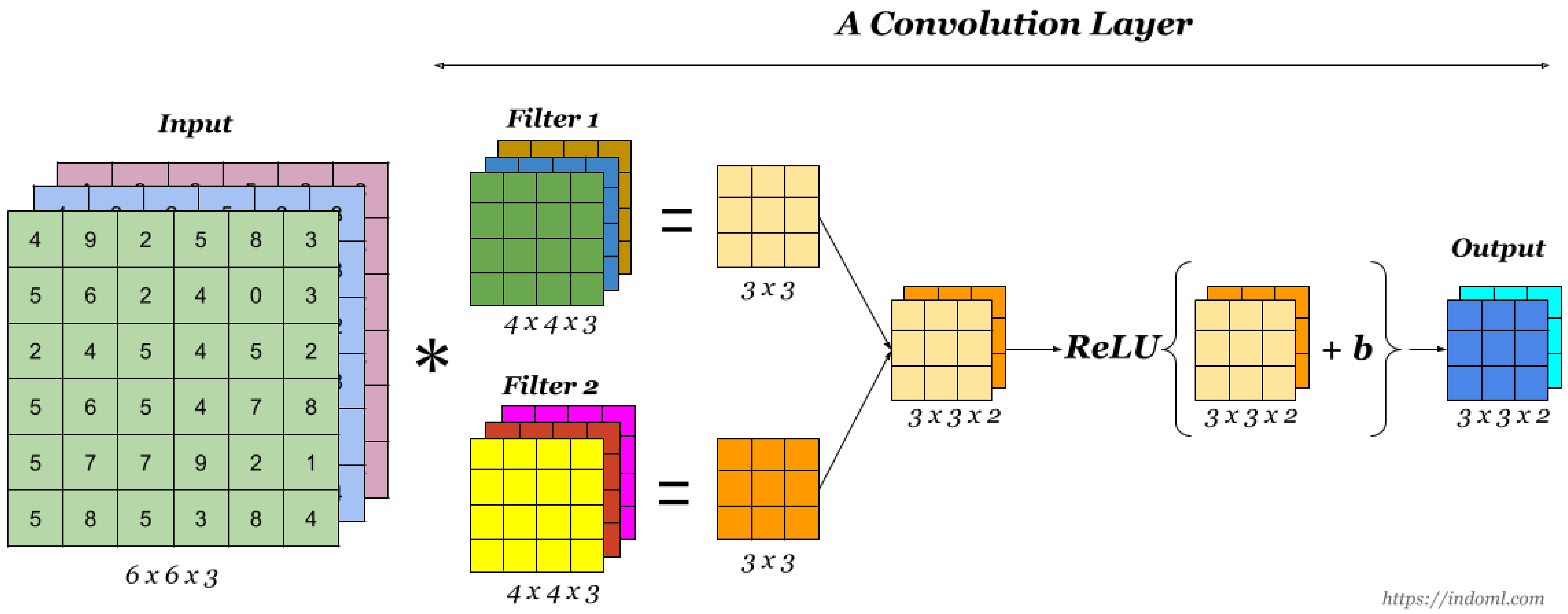
PyTorch CNN Model



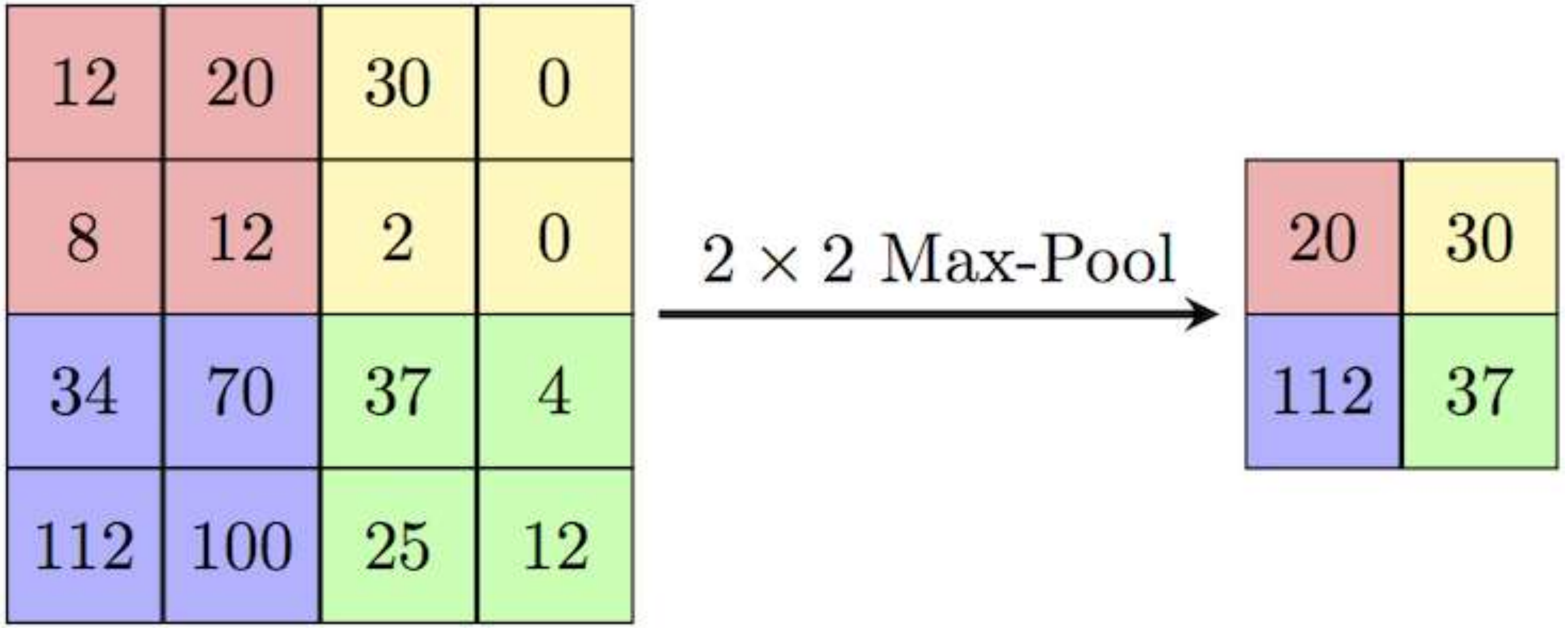
Convolutional Opertator

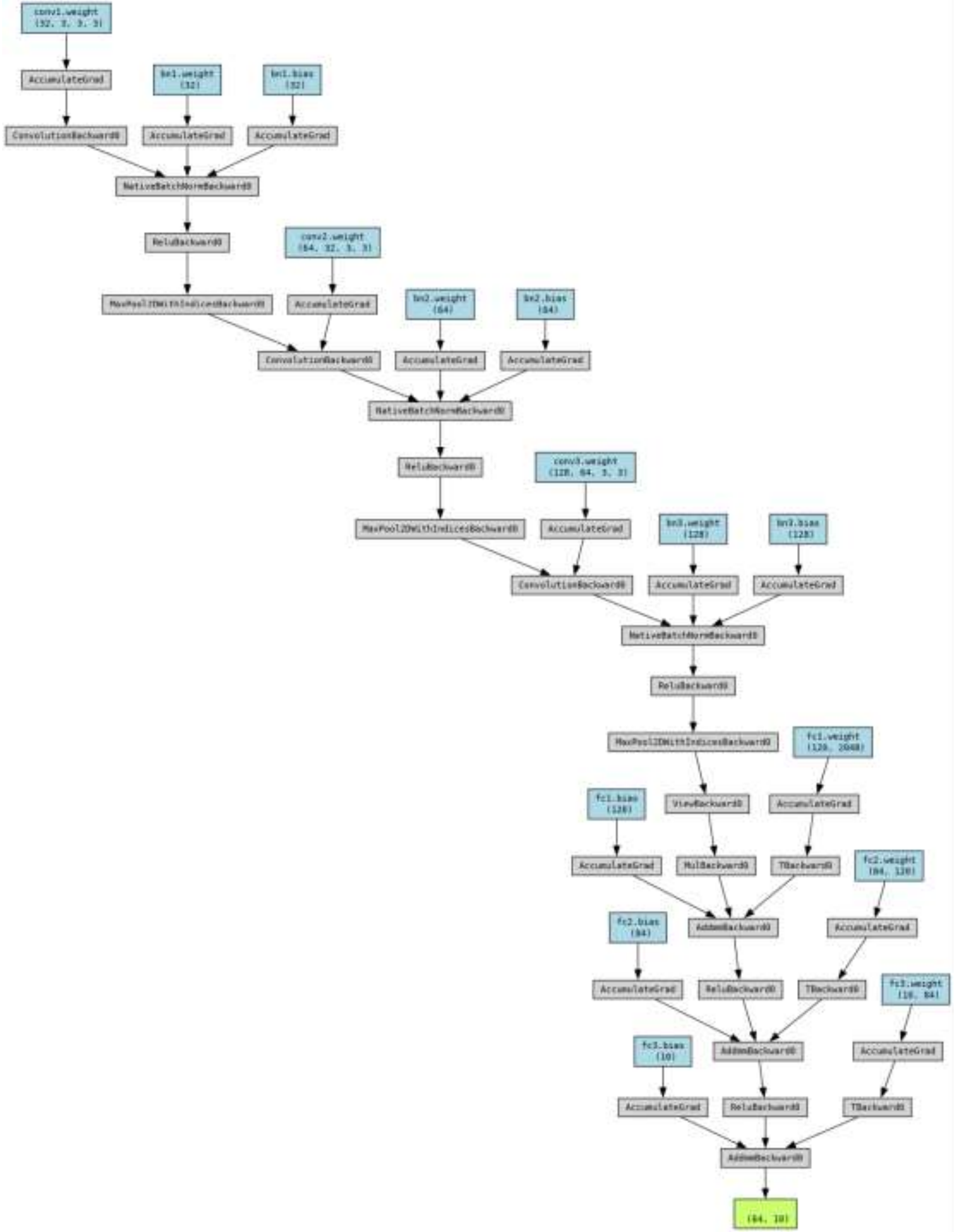


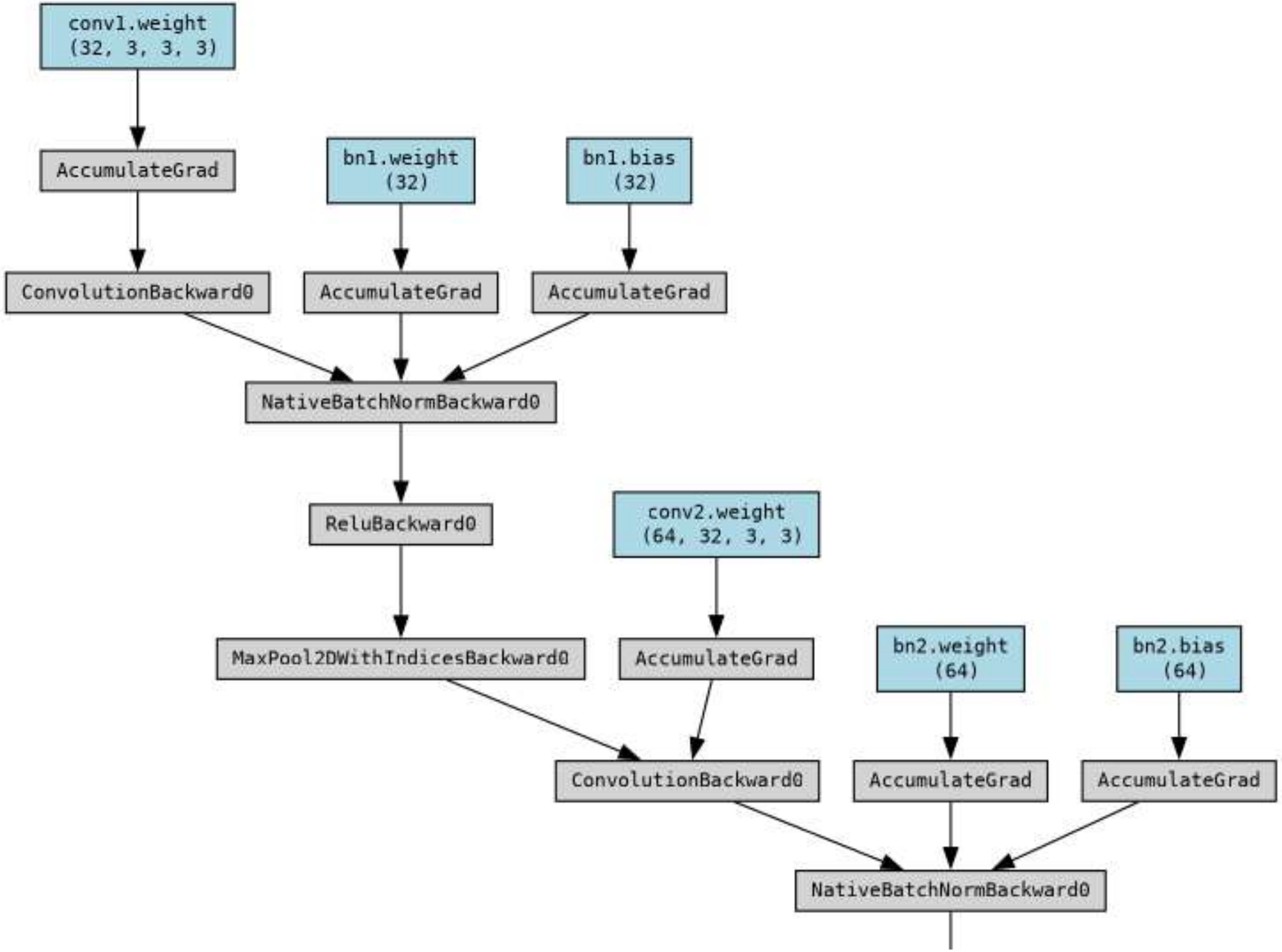
Convolutional Opertator

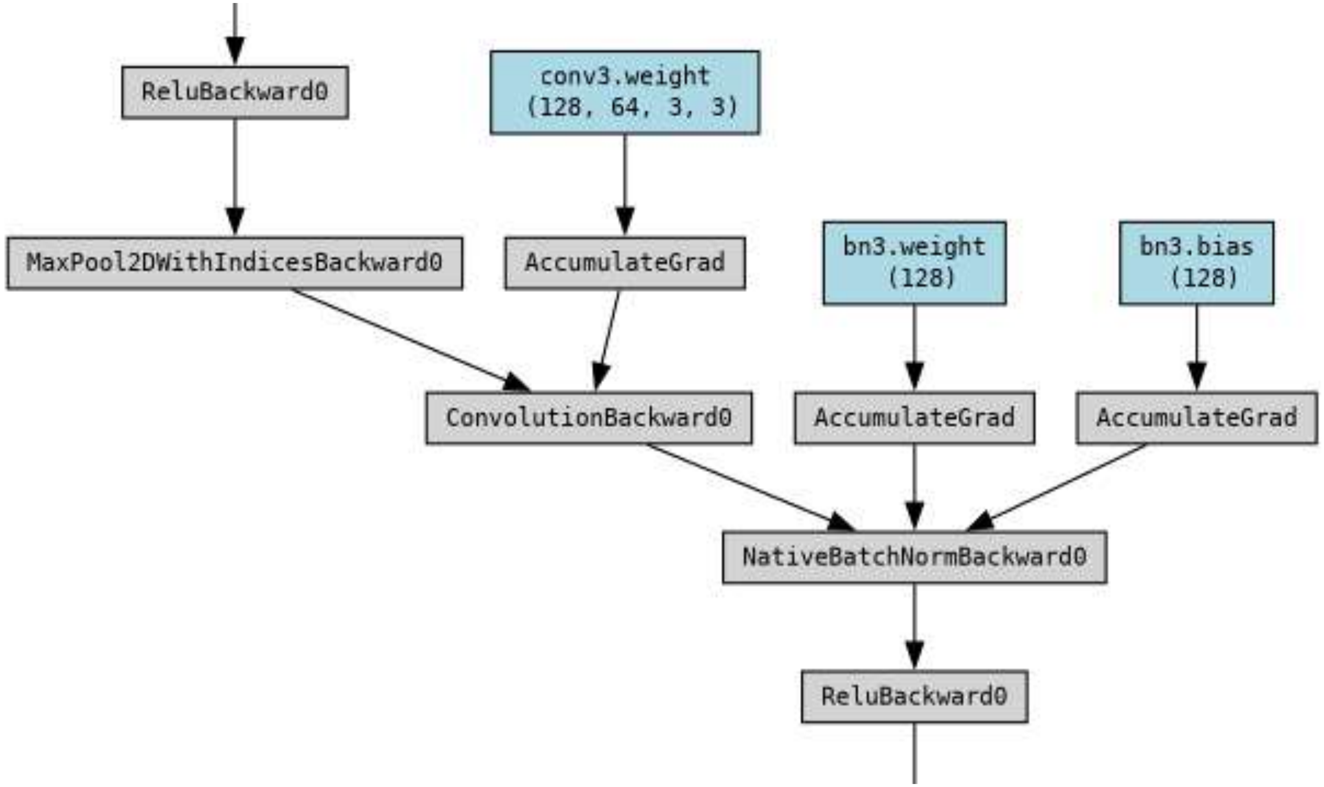


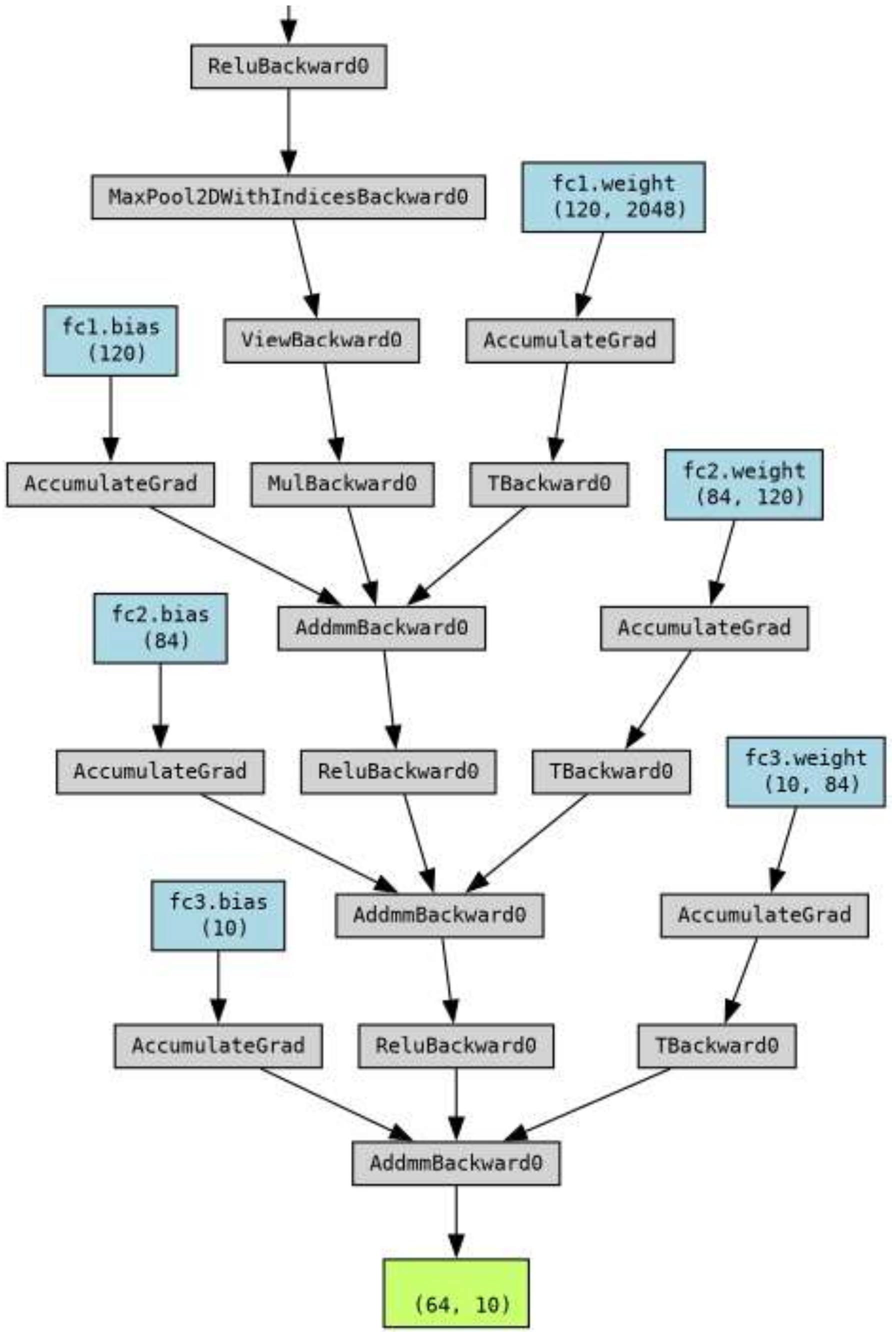
MaxPooling



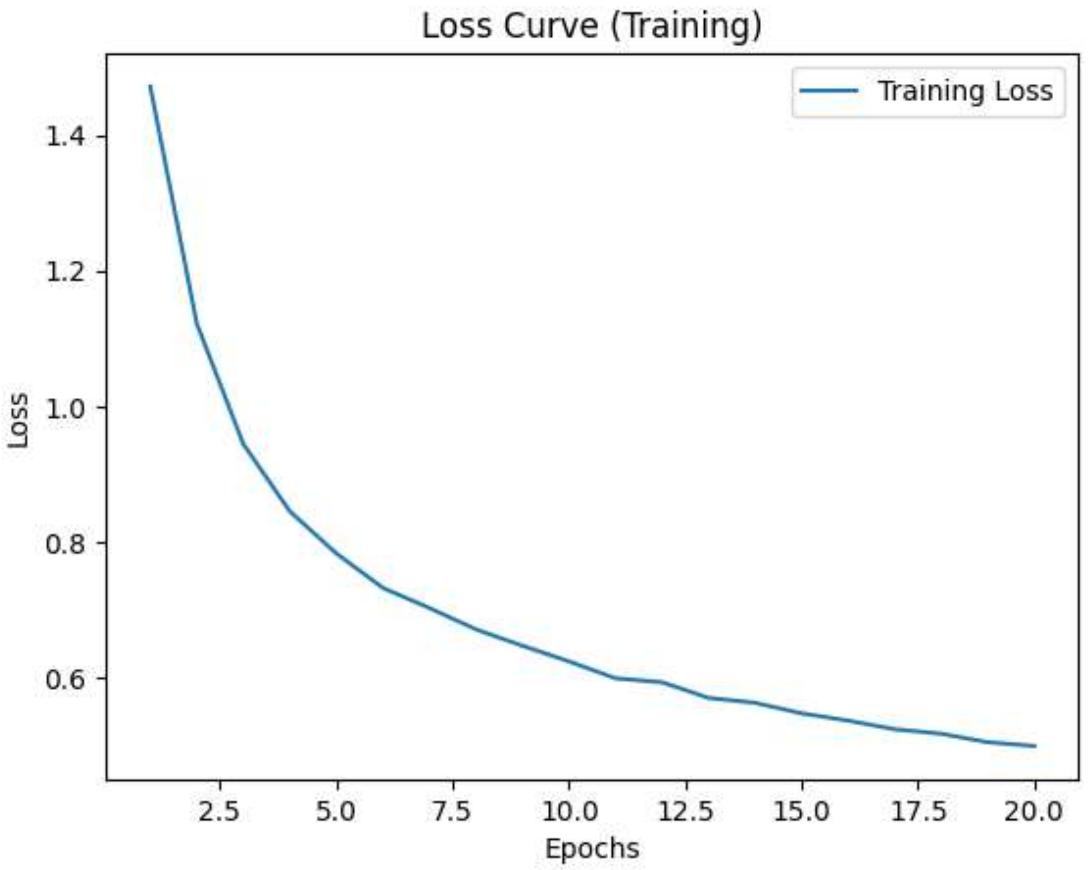

conv1.weight
(32, 3, 3, 3)
AccumulateGrad
bn1.weight
(32)
bn1.bias
(32)
ConvolutionBackward0
AccumulateGrad
AccumulateGrad
NativeBatchNormBackward0
ReluBackward0
conv2.weight
(64, 32, 3, 3)
MaxPool2DWithIndicesBackward0
AccumulateGrad
bn2.weight
(64)
bn2.bias
(64)
ConvolutionBackward0
AccumulateGrad
AccumulateGrad
NativeBatchNormBackward0
ReluBackward0
conv3.weight
(128, 64, 3, 3)
MaxPool2DWithIndicesBackward0
AccumulateGrad
bn3.weight
(128)
bn3.bias
(128)
ConvolutionBackward0
AccumulateGrad
AccumulateGrad
NativeBatchNormBackward0
ReluBackward0
MaxPool2DWithIndicesBackward0
fc1.weight
(128, 2048)
fc1.bias
(128)
ViewBackward0
AccumulateGrad
AccumulateGrad
MulBackward0
TBackward0
fc2.weight
(84, 128)
fc2.bias
(84)
AddmmBackward0
AccumulateGrad
AccumulateGrad
ReluBackward0
TBackward0
fc3.weight
(10, 84)
fc3.bias
(10)
AddmmBackward0
AccumulateGrad
AccumulateGrad
ReluBackward0
TBackward0
AddmmBackward0
(64, 10)


conv1.weight
(32, 3, 3, 3)
AccumulateGrad
bn1.weight
(32)
bn1.bias
(32)
ConvolutionBackward0
AccumulateGrad
AccumulateGrad
NativeBatchNormBackward0
ReluBackward0
conv2.weight
(64, 32, 3, 3)
MaxPool2DWithIndicesBackward0
AccumulateGrad
bn2.weight
(64)
bn2.bias
(64)
ConvolutionBackward0
AccumulateGrad
AccumulateGrad
NativeBatchNormBackward0

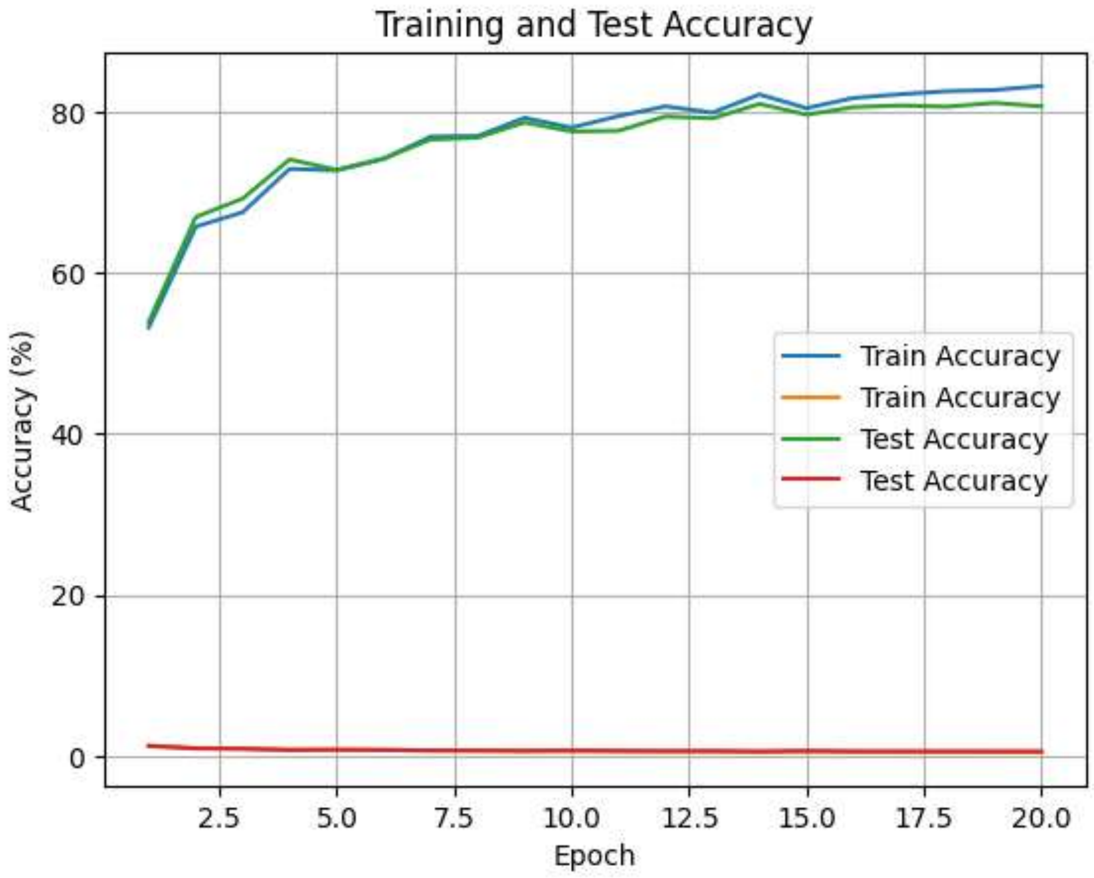

ReluBackward0
conv3.weight
(128, 64, 3, 3)
MaxPool2DWithIndicesBackward0
AccumulateGrad
bn3.weight
(128)
bn3.bias
(128)
ConvolutionBackward0
AccumulateGrad
AccumulateGrad
NativeBatchNormBackward0
ReluBackward0


ReluBackward0
MaxPool2DWithIndicesBackward0
fc1.weight
(120, 2048)
fc1.bias
(120)
ViewBackward0
AccumulateGrad
AccumulateGrad
MulBackward0
TBackward0
fc2.weight
(84, 120)
fc2.bias
(84)
AddmmBackward0
AccumulateGrad
AccumulateGrad
ReluBackward0
TBackward0
fc3.weight
(10, 84)
fc3.bias
(10)
AddmmBackward0
AccumulateGrad
AccumulateGrad
ReluBackward0
TBackward0
AddmmBackward0
(64, 10)

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        # Convolutional and batchnorm layers
        self.conv1 = nn.Conv2d(3, 32, 3, padding='same', bias=False)
        self.bn1 = nn.BatchNorm2d(32)
        self.conv2 = nn.Conv2d(32, 64, 3, padding='same',bias=False)
        self.bn2 = nn.BatchNorm2d(64)
        self.conv3 = nn.Conv2d(64, 128, 3, padding='same',bias=False)
        self.bn3 = nn.BatchNorm2d(128)

        # Max pooling layers
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Calculate the output size after the convolutional layers
        self.fc1 = nn.Linear(128 * 4 * 4, 120)  # Adjust input size accordingly
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

        # Adding Dropout layer to avoid underfitting
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):

        x = self.pool(F.relu(self.bn1(self.conv1(x))))
        x = self.pool(F.relu(self.bn2(self.conv2(x))))
        x = self.pool(F.relu(self.bn3(self.conv3(x))))
        x = x.view(-1, 128 * 4 * 4)  # Adjust input size accordingly
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```



Loss Curve (Training)
Training Loss
Loss
Epochs
1.4
1.2
1.0
0.8
0.6
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



Training and Test Accuracy
Train Accuracy
Train Accuracy
Test Accuracy
Test Accuracy
Accuracy (%)
Epoch
80
60
40
20
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

Training dataset

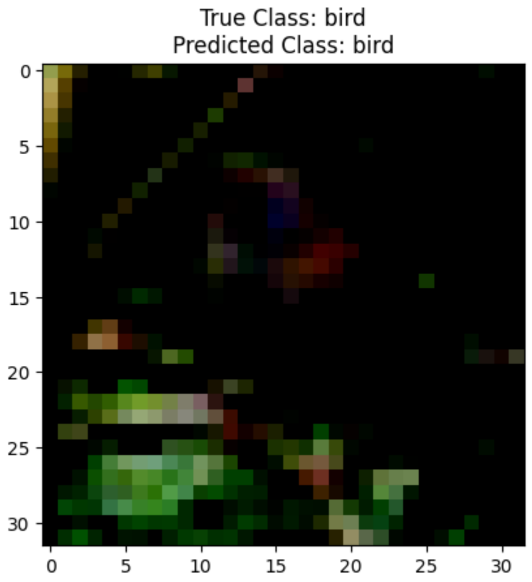
Average loss: **0.4829**
Accuracy: **82.86%**

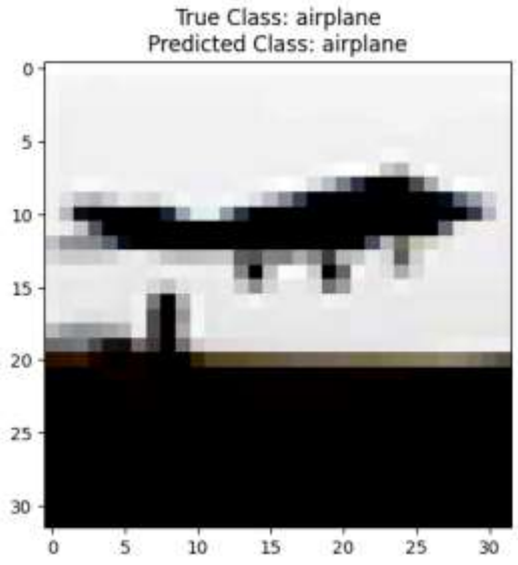
Testing dataset

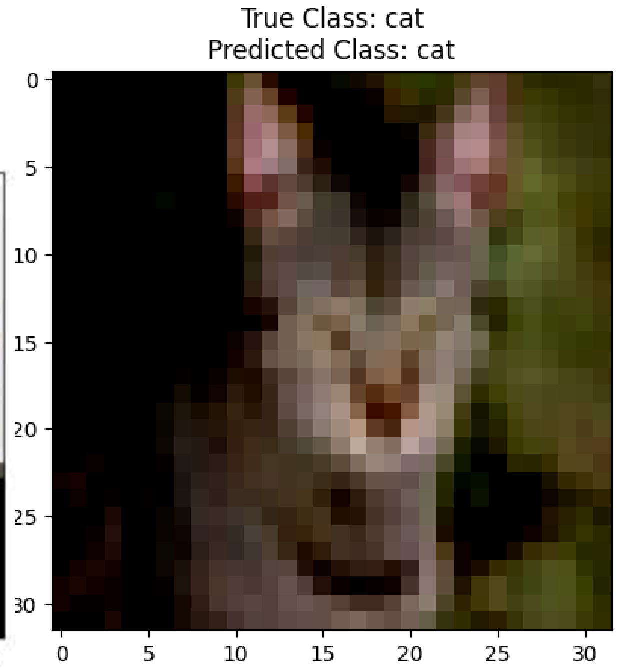
Average loss: **0.5722**
Accuracy: **80.64%**

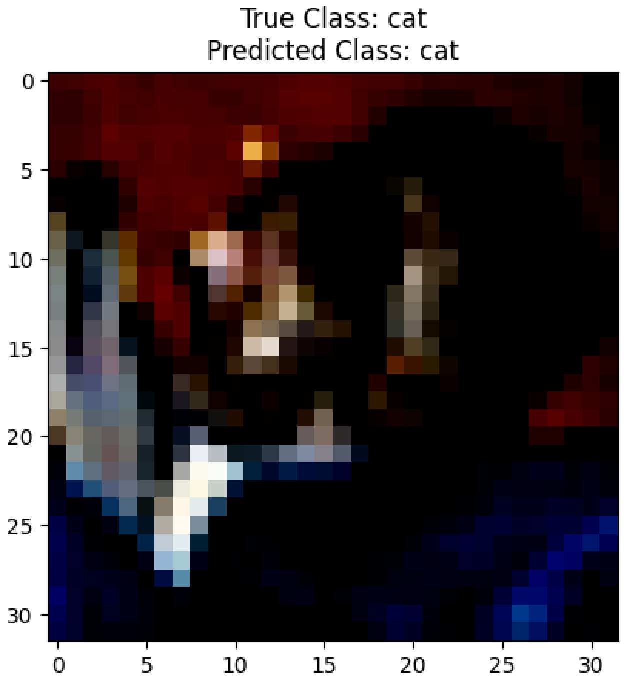
Training time: **663.81 seconds**
No. of parameters: **350,342**
Epochs: **20**
Batchsize: **64**
Learning rate: **0.01**
Optimizer: **Adam**
Accelerator: **Tesla P100-PCIE-16GB (Kaggle)**

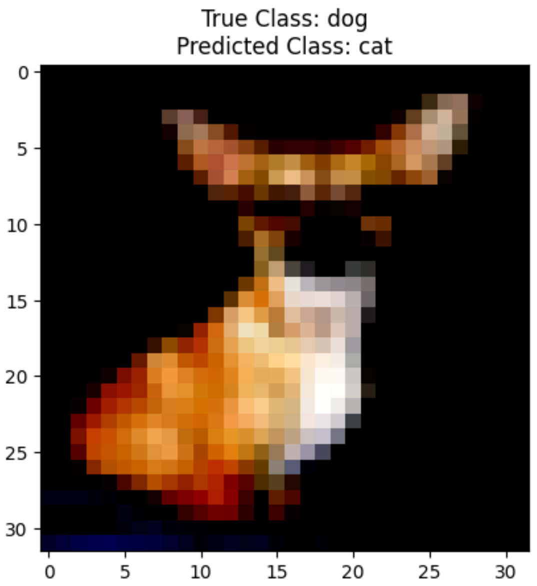
Some results

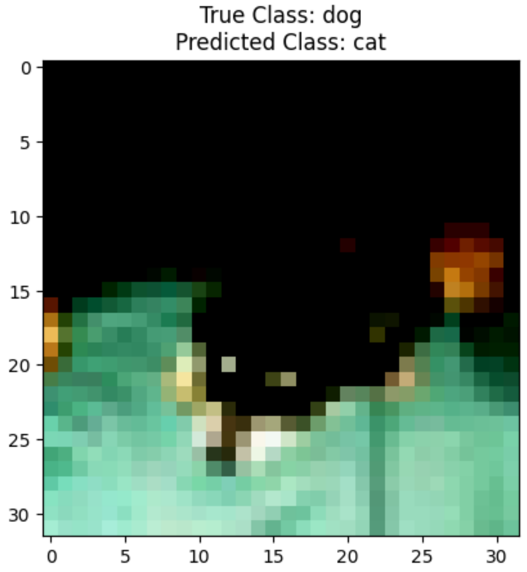


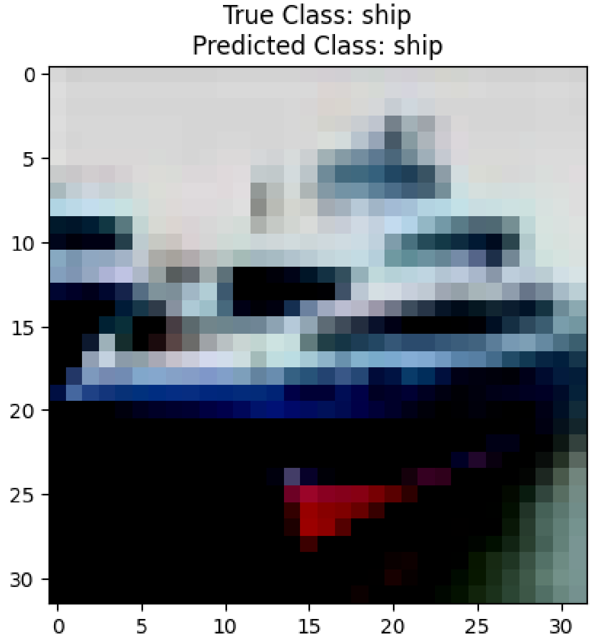


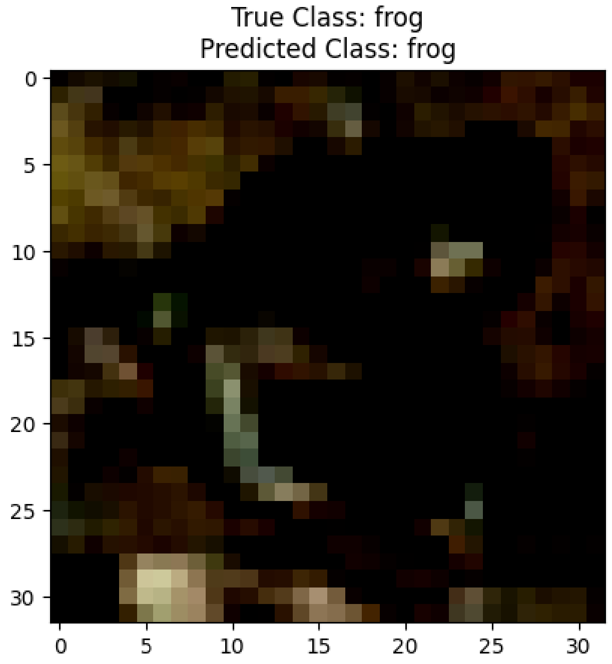












TensorFlow CNN Model

```
# Build a simple CNN
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu',),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
```

Training dataset

Average loss: **0.2692**
Accuracy: **90.43%**

Testing dataset

Average loss: **1.2214**
Accuracy: **70.92%**

Training time: **84 seconds**
Epochs: **20**
No. of parameters: **224,842**
Batchsize: **32 (default)**
Learning rate: **0.01 (default)**
Optimizer: **Adam**
Accelerator: **Tesla P100-PCIE-16GB (Kaggle)**

Some

True: automobile, Pred: automobile

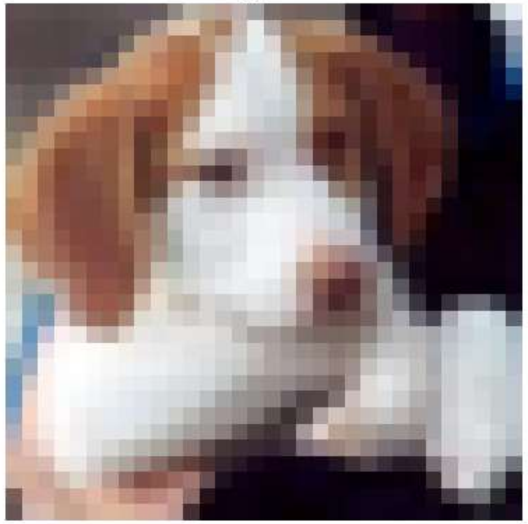
True: horse, Pred: horse

True: ship, Pred: ship

True: cat, Pred: cat

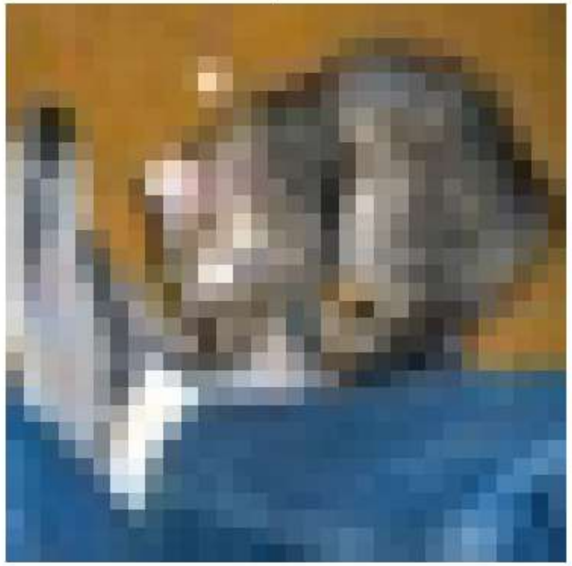
True: automobile, Pred: truck

True: dog, Pred: cat



True: frog, Pred: frog

True: cat, Pred: cat



AlexNet Fine-Tuning

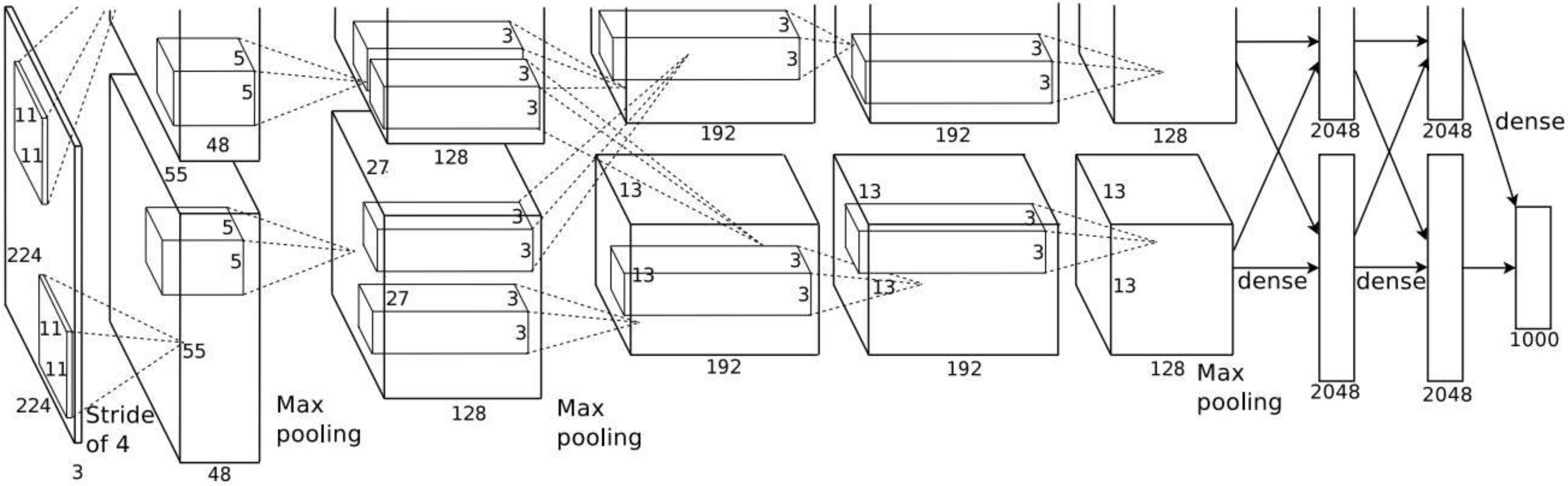
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

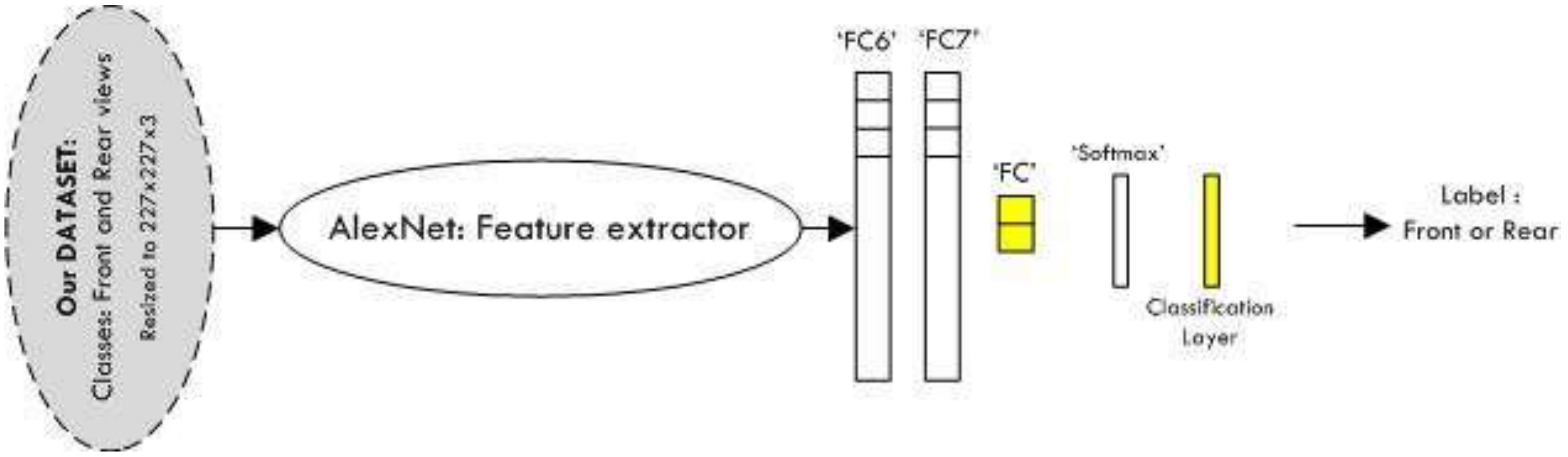
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

2012



224
224
3
11
11
11
11
Stride
of 4
55
55
48
48
5
5
5
5
Max
pooling
27
27
128
128
3
3
3
3
3
3
3
3
Max
pooling
13
13
192
192
3
3
3
3
13
13
192
192
3
3
3
3
13
13
128
128
Max
pooling
dense
dense
dense
2048
2048
2048
2048
1000

Fine-Tuning step



Pre-trained model

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
  (1): ReLU(inplace=True)
  (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (4): ReLU(inplace=True)
  (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): ReLU(inplace=True)
  (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (9): ReLU(inplace=True)
  (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
)
---------------------------------------------------
Sequential(
  (0): Dropout(p=0.5, inplace=False)
  (1): Linear(in_features=9216, out_features=4096, bias=True)
  (2): ReLU(inplace=True)
  (3): Dropout(p=0.5, inplace=False)
  (4): Linear(in_features=4096, out_features=4096, bias=True)
  (5): ReLU(inplace=True)
  (6): Linear(in_features=4096, out_features=1000, bias=True)
)
```

Fine-Tuned model

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=1024, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=1024, out_features=10, bias=True)
  )
)
```

Training dataset

Average loss: **0.1290**
Accuracy: **95.97%**

Testing dataset

Average loss: **0.5556**
Accuracy: **83.49%**

Training time: **740 seconds**
Epochs: **10**
No. of parameters: **60M (4,211,082 trainable)**
Batchsize: **64**
Learning rate: **0.01**
Optimizer: **Adam**
Accelerator: **Tesla P100-PCIE-16GB (Kaggle)**

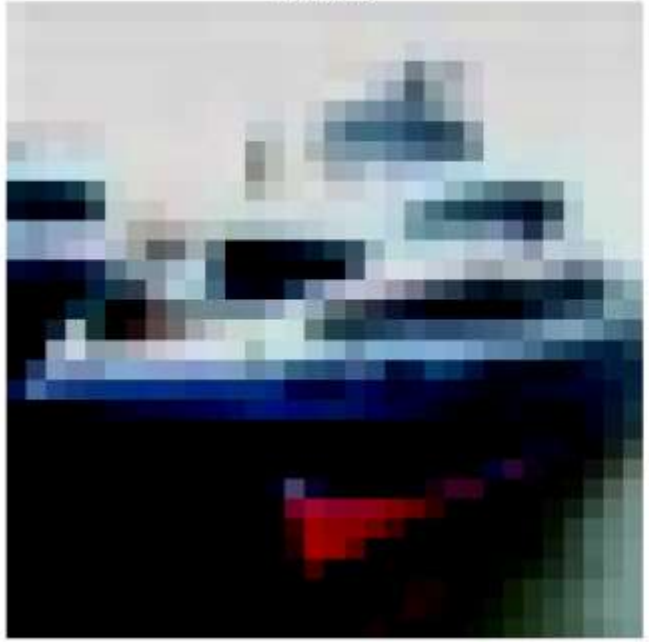
Some results

True: cat
Pred: cat

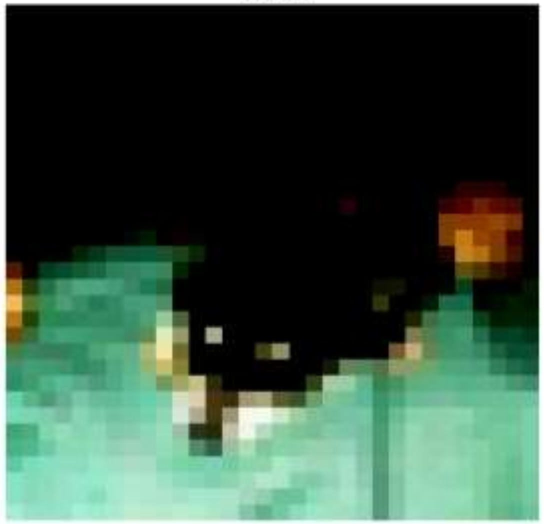
True: airplane
Pred: airplane

True: cat
Pred: dog

True: ship
Pred: ship



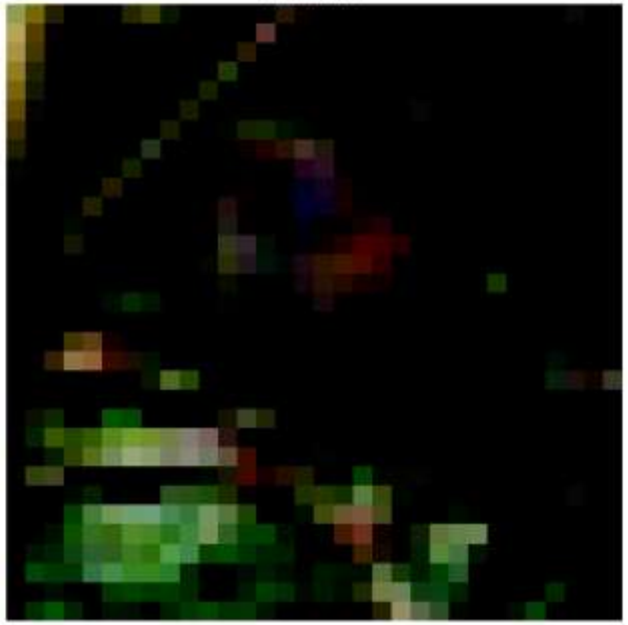
True: dog
Pred: cat



True: bird
Pred: bird

True: frog
Pred: frog

True: bird
Pred: bird



Last Comparison

Metric	Model 1	Model 2	Model 3
Training Accuracy	82.86%	90.43%	95.97%
Testing Accuracy	80.64%	70.92%	83.49%
Training Loss	0.4829	0.2692	0.1290
Testing Loss	0.5722	1.2214	0.5556
Training Time (s)	663.81	84	740
Parameters	350,342	224,842	60M (4,211,082 trainable)
Epochs	20	20	10
Batch Size	64	32	64
Learning Rate	0.01	0.01 (default)	0.01
Optimizer	Adam	Adam	Adam
Accelerator	Tesla P100-PCIE-16GB	Tesla P100-PCIE-16GB	Tesla P100-PCIE-16GB

Thank You !