Benchmark Studies of Various Deep Learning Architecture

Irfan Nur Afif

January 17, 2018

Table of contents

Context & Motivation

- Deep Learning has proven to be doing well in solving complex classification task such as image and speech recognition.
- In this research, we are interested to investigate deep learning for image data (CNN).
- Main problem: no exact guidelines on designing a deep learning architecture.
- Another problem: we rarely see the performance comparison of various deep learning architecture for the same standard datasets

Context & Motivation

To solve this problem, we propose a benchmarking studies of multiple deep learning architecture on many image datasets. The goal of the research is to have a benchmark analysis of various deep learning architecture performance on multiple image datasets.

Research Question

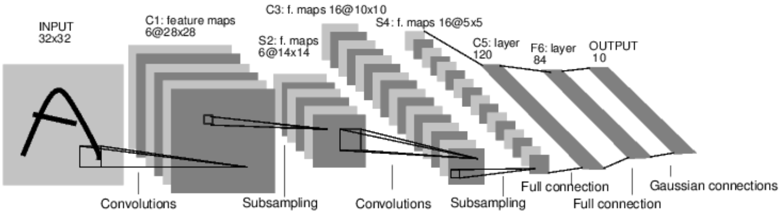
- The works tries to answer the following research question: "How does the performance comparison of deep learning architecture model looks like for a given image classification dataset?"
- There are also some sub-questions to solve the main research questions, which are:
  1. Which architecture that works bests for a given datasets?
  2. What kind of datasets characteristics that makes a deep learning architecture works well?
  3. Is there any architecture that generally works well for image classification?

Convolutional Neural Network

- Deep learning technique for image data.
- Three lypes of layers: convolitional layer, subsampling layer (maxpooling, average pooling) and fully connected layers.
- Sometimes there is a dropout layer.

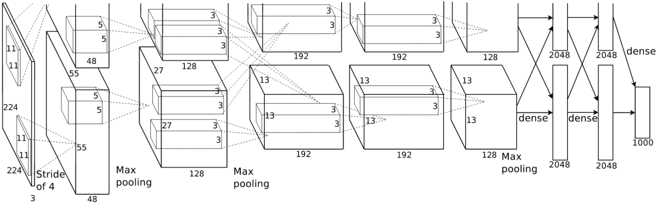
LeNet-5

- Proposed by Yann LeCun, et al. (1998)
- Key idea: 7 layers, combination of 5x5 filter size in convolutional layer and 2x2 subsampling layer.



Alex-Net

- Proposed by Alex Krizhevsky, et al. (2012)
- Winner of ILSVRC 2012
- Tested on ILSVRC 2012 dataset, achieves top 5 test error rate of 15.4%



VGG-Net

- Proposed by Karen Simonyan and Andrew Zisserman.
- Key idea: 3x3 filters (stride and pad of 1), 2x2 maxpooling layers (stride of 2), 11-19 layers.
- Tested on ILSVRC 2012 dataset, achieves top 5 test error rate of 6.8%

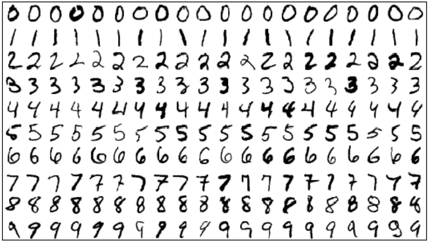
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Res-Net

- Proposed by Microsoft, winner of ILSVRC 2015.
- Key idea: very deep network: up to 1202 layers using 3x3 filter size in convolutional layer.
- Achieved 6.43% error rate on CIFAR-10 (110 layers) and 3.6% error rate on ILSVRC 2015 dataset.

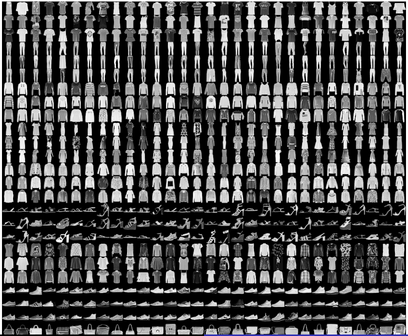
MNIST

- Handwritten digits datasets with 784 features (28x28 grayscale images)
- 10 classes, 60,000 training examples and 10,000 testing examples.



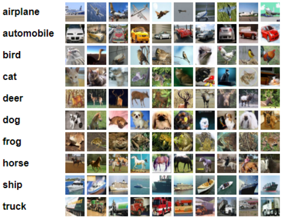
Fashion MNIST

- Zalando's article images datasets with 784 features (28x28 grayscale images)
- Very similar to MNIST
- 10 classes, 60,000 training examples and 10,000 testing examples.



CIFAR-10

- Labeled subset of the 80 million tiny images dataset, collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton
- 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.
- Colored images with a size of 32x32 pixels.
- 50,000 training examples and 10,000 testing examples.



SVHN

- The Street View House Numbers (SVHN) Dataset is a dataset obtained from house numbers in Google Street View images
- Colored images with a size of 32x32 pixels.
- 10 classes, 73257 digits for training, 26032 digits for testing.

LeNet

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 26, 26, 6)         60
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 13, 13, 6)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 12, 12, 16)        400
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 6, 6, 16)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_8 (Dense)              (None, 256)               147712
_________________________________________________________________
dense_9 (Dense)              (None, 10)                2570
=================================================================
```

Figure: LeNet architecture overview

VGG-Net

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 32)	320
conv2d_2 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 64)	0
dropout_1 (Dropout)	(None, 14, 14, 64)	0
conv2d_3 (Conv2D)	(None, 14, 14, 128)	73856
conv2d_4 (Conv2D)	(None, 12, 12, 256)	295168
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_1 (Dense)	(None, 256)	1048832
leaky_re_lu_1 (LeakyReLU)	(None, 256)	0
dropout_3 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 256)	65792
leaky_re_lu_2 (LeakyReLU)	(None, 256)	0
dense_3 (Dense)	(None, 10)	2570

ResNetV1

- We implement the simplest form of ResNetV1 which is ResNet20V1 due to hardware constraint.
- In this variation, there are 20 stacks of "resnet block". Each block consists of 2 x (3 x 3) Convolution-Batch Normalization-ReLU layer.
- In applying ResNet, we use ResNet builder library that are available at https://github.com/keras-team/keras/blob/master/examples/cifar10_resnet.py.

ResNetV2

- The difference of ResNet V2 and V1 is that, in V2 each block consists of (1 x 1)-(3 x 3)-(1 x 1) Batch Normalization-ReLU-Convolution layer.
- At the beginning of each 'stage', the feature map size is halved (downsampled) by a convolutional layer with strides=2, while the number of filter maps is doubled. Within each 'stage', the layers have the same number filters and the same filter map sizes.

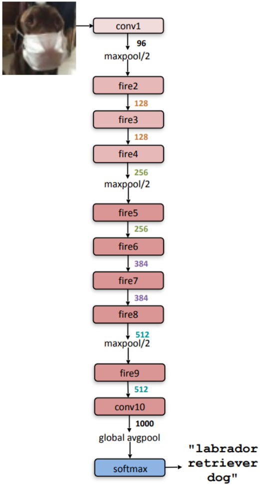
Alex-Net/SqueezeNet

- Initially we plan to implement AlexNet, but when we try to implement it on the current machine we failed to implement it because the number of trainable parameters are too big.
- Then, we discover SqueezeNet, a simpler version of AlexNet that could achieve similar accuracy with less memory resource and training time.

Alex-Net/SqueezeNet

- The architecture consists of convoultional layer and fire module.
- A fire module consists of 1x1 convolutional layer followed by a mix of 1x1 and 3x3 convolution layer.
- Then, it gradually increase the number of filters per fire module from the beginning to the end of the network.
- SqueezeNet performs max-pooling with a stride of 2 after layers conv1, fire4, fire8, and conv10.

Alex-Net/SqueezeNet



MNIST, Fashion-MNIST, CIFAR-10

```
from keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
print("x_train shape: {}".format(x_train.shape))
print("y_train shape: {}".format(y_train.shape))
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

y_train = np_utils.to_categorical(y_train, 10)
y_test = np_utils.to_categorical(y_test, 10)
```

Listing A.5: Code for loading MNIST dataset

```
from keras.datasets import fashion_mnist

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
print("x_train shape: {}".format(x_train.shape))
print("y_train shape: {}".format(y_train.shape))
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

y_train = np_utils.to_categorical(y_train, 10)
y_test = np_utils.to_categorical(y_test, 10)
```

Listing A.6: Code for loading Fashion MNIST dataset

```
from keras.datasets import cifar10

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

Listing A.7: Code for loading CIFAR-10 dataset

SVHN

```
def load_data(path):
""" Helper function for loading a MAT-File"""
data = loadmat(path)
return data['X'], data['y']

X_train, y_train = load_data('train_32x32.mat')
X_test, y_test = load_data('test_32x32.mat')

X_train, y_train = X_train.transpose((3,0,1,2)), y_train[:,0]
X_test, y_test = X_test.transpose((3,0,1,2)), y_test[:,0]

print("Training", X_train.shape)
print("Test", X_test.shape)
print('')

# Calculate the total number of images
num_images = X_train.shape[0] + X_test.shape[0] + X_extra.shape[0]

y_train[y_train == 10] = 0
y_test[y_test == 10] = 0

def balanced_subsample_categorical(y, s):
"""Return a balanced subsample of the population"""
sample = []
# For every label in the dataset
for ii in range(10):
# Get the index of all images with a specific label
images = np.ndarray.flatten(np.where((y[:, ii] == 1))[0])
print(images)
# Draw a random sample from the images
random_sample = np.random.choice(images, size=s, replace=False)
# Add the random sample to our subsample list
sample += random_sample.tolist()
return sample

# Pick 1000 samples per class from the training samples
train_samples = balanced_subsample_categorical(y_train, 6000)
X_train = np.take(X_train, train_samples, axis=0)
y_train = np.take(y_train, train_samples, axis=0)
```

Listing A.8: Code for loading SVHN dataset

Accuracy

	MNIST	Fashion MNIST	CIFAR-10	SVHN
LeNet-5	0.9834	0.8816	0.6561	0.809
VGG-like	0.9946	0.91	0.4075	0.067
Resnet20v1	0.9246	0.592	0.7567	0.866
Resnet20v2	0.9222	0.8425	0.679	0.893
SqueezeNet	0.9858	0.8813	0.555	0.775

Table: Accuracy table

Memory Resource

		MNIST	Fashion MNIST	CIFAR-10	SVHN
LeNet-5	total params	150,742	150,742	204,098	204,098
	trainable params	150,742	150,742	204,098	204,098
	non-trainable params	0	0	0	0
VGG-like	total params	1,505,034	1,505,034	1,505,610	1,505,610
	trainable params	1,505,034	1,505,034	1,505,610	1,505,610
	non-trainable params	0	0	0	0
resnet20v1	total params	274,090	274,090	274,442	274,442
	trainable params	272,746	272,746	273,066	273,066
	non-trainable params	1,344	1,344	1,376	1,376
resnet20v2	total params	573,738	573,738	574,090	574,090
	trainable params	570,282	570,282	570,602	570,602
	non-trainable params	3,456	3,456	3,488	3,488
squeezenet	total params	711,956	711,956	720,084	720,084
	trainable params	711,956	711,956	720,084	720,084
	non-trainable params	0	0	0	0

Time Consumption

	MNIST	Fashion MNIST	CIFAR-10	SVHN
LeNet-5	220s	220s	330s	220s
VGG-like	4986s	4469s	6816s	6526s
Resnet20v1	7970s	7909s	7204s	9310s
Resnet20v2	13900s	13910s	12060s	15693s
SqueezeNet	13800s	13907s	13630s	17550s

Table: Execution time table

"How does the performance comparison of deep learning architecture model looks like for a given image classification dataset?"

- The architecture consists of convoultional layer and fire module.
- A fire module consists of 1x1 convolutional layer followed by a mix of 1x1 and 3x3 convolution layer.
- Then, it gradually increase the number of filters per fire module from the beginning to the end of the network.
- SqueezeNet performs max-pooling with a stride of 2 after layers conv1, fire4, fire8, and conv10.