Machine Learning Practical

Coursework 4

Ridlo Nur Rahman
s1669912
s1669912@sms.ed.ac.uk

Coursework 4

Ridlo Nur Rahman - s1669912

Machine Learning Practical

Contents

1 Research Question

In the previous coursework, we have already built a simple feedforward neural network as the baseline. The goal of this coursework is to explore and design a more advanced neural network for the classification problem of chosen dataset. The big question is how to achieve a high classification accuracy for the dataset? We use a different approach from the previous coursework to answer the question. In the previous coursework, we explored various activation functions, hidden layer depths and widths, learning rules, normalisation, and regularisation one after another. Convolutional neural network is the state of the art for this problem. And since there are already several popular well-performed architectures of convolutional neural network, we will start to answer our main question by exploring them. Then, to answer our big question, we must answer the following questions: Which convolutional neural network architectures suit the best for this problem with our resource? How many convolutional layers needed? And how is the configuration? Does dropout and batch normalisation help to reduce overfitting in our convolutional neural network? Does interesting approaches done by some of the leaderboards such as using ELU as the activation function or replacing pooling with convolutional layers help improving the accuracy?

2 The Dataset Used

For the experiment, we use CIFAR-10 and CIFAR-100 [1, 2] as our dataset. CIFAR-10/100 is an images dataset widely used for object recognition. The dataset is a pair of image and its label collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 consists of 60,000 32x32 colour images in 10 classes, while CIFAR-100 has 100 classes and 20 super classes. Each of the datasets has 50,000 training images and 10,000 test images. We split the training images into 40,000 training images and 10,000 validation images.

3 Baseline Model

We have already created our baseline in the prior coursework. Our baseline is a simple feedforward neural network with the properties as shown in Table 1. We built our baseline by investigating the best combination of activation functions, hidden layer depths and widths, learning rules, normalisation, and regularisation for classifying CIFAR dataset. Our baseline achieved classification accuracy at 52.51% on CIFAR-10 and 24.02% on CIFAR-100. In the end of this report, we will compare our convolutional neural network performance with the baseline.

Table 1: Baseline system's properties

Hidden layers	2
Unit per layer	500
Activation function	ReLU
Learning Rule	Adam (0.001)
Weight Initialisation	Xavier Init

4 Designing The Convolutional Neural Network

4.1 Existing Architectures and State of The Art

A typical Convolutional Neural Network (CNN/ConvNet) is composed of a sequence of convolutional layer followed by a pooling layer that performs feature extraction. Since 2012, CNNs/ConvNets have been widely used for object recognition due to its outstanding performance in the Imagenet Large Scale Visual Recognition Challenge (ILSVRC) 2012. At that time, AlexNet [3], a work by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton; outperformed the runner-up by a significant margin. However, AlexNet was not the first successful application of CNN. It was a work before AlexNet, LeNet [4], which was introduced by Yann LeCunn in 90's that was the first successful application of CNN.

After AlexNet outstanding performance in ILSVRC 2012, many new and better architectures such as ZFNet [5], VGGNet [6], GoogLeNet [7], and ResNet [8] appeared. ZFNet is an improvement of AlexNet and was the winner of ILSVRC 2013. Both GoogLeNet and VGGNet competed in ILSVRC 2014 and finished as winner and runner-up respectively. GoogLeNet introduced the inception module, while VGGNet contributed by showing that the depth of a network is essential to give a good classification performance. In the next year, ResNet (residual network) was the winner. ResNet uses many batch normalisation and special skip connection. And just in the last year, ResNet was outperformed by DenseNet. In DenseNet architecture, each layer is connected to every other layer. It is currently state of the art of CNN/ConvNet architecture.

4.2 CIFAR-10 and CIFAR-100 Kaggle Leaderboards

A result collection in [9] presents several interesting techniques to achieve a very high classification accuracy in CIFAR dataset. Ben Graham presented fractional max pooling [10] and spatially-sparse CNN [11]. Fractional max-pooling and spatially-sparse CNN is rather simple to be implemented, because Tensorflow has the library for fractional max pooling, while spatially-sparse CNN is done simply by extending input size using padding zero to introduce input sparsity to our network. Clevert et al. implemented exponential linear unit (ELU) [12] as the activation function to improve the classification accuracy. Springenberg et al. introduced a new CNN architecture without pooling layer. In their work, pooling layer is substituted using convolutional layer with increased stride [13]. The method achieved more than 95% in accuracy using aggressive data augmentation in a deep network.

Most of the techniques on the leaderboards rely on heavy data augmentation. According to [9], about 92% accuracy is the best result without data augmentation. And from a short article written by Andrej Karpathy [14], we know that the human accuracy on CIFAR dataset is about 94%.

4.3 Which Architecture?

We choose a different approach to design our architecture in this coursework. Instead of one-by-one varying depths, widths, activation functions, learning rules, we will explore a few notable architectures such as AlexNet and VGGNet. This approach will save our time and effort rather than designing the architecture

from scratch. Then, we will evaluate the best configuration, add regularisation, and try some of the leaderboards' approaches to improving the performance.

4.3.1 AlexNet

AlexNet was proposed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton [3]. The architecture is similar to LeNet [4] but with much deeper and wider layers. AlexNet contributes to promoting ReLU and dropout to be widely used. It also introduces overlapping max-pooling.

Large convolutional filter/kernel and overlapping max-pooling are the characteristic of AlexNet. The first convolutional layer of AlexNet has 11x11 filter with stride 4, the second convolutional layer has 5x5 filter, and it uses 3x3 filter for the third layer and so forth. It uses max pooling with the size of 3 and stride 2. This overlapping max pooling was proposed by Alex because he observed that it helps to reducing the overfitting. The max pooling layer is always followed by local response normalisation layer. AlexNet has a few fully connected layers before the softmax layer. The architecture used for ImageNet is presented in Figure 1.

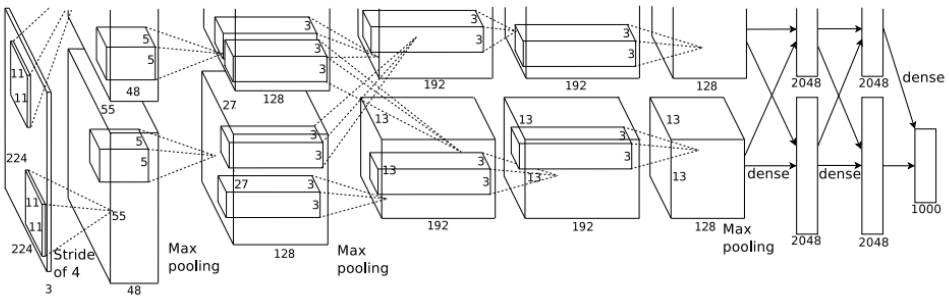

Figure 1: The illustration of AlexNet architecture used for ImageNet

AlexNet was designed originally for the ImageNet dataset with 224x224 images as the input. Consequently, implementing AlexNet for CIFAR dataset needs some adjustments. First, we cannot use the 11x11 convolutional filter due to our small 32x32 images input. So, we will use a 5x5 convolutional filter with stride 1. We employ the very similar max pooling with 3x3 filter size and stride 2 and followed by local response normalisation. We will only use two convolutional layers because after the second max pooling, the spatial size is already small enough. Then, it is connected by two fully connected layers with 256 hidden units. AlexNet configuration used for our evaluation is shown in Table 2. The learning used is use stochastic gradient descent with momentum of 0.9 as defined in the paper. We also follow the paper to apply weight decay of 0.0005.

4.3.2 VGGNet

VGGnet, proposed by Karen Simonyan and Andrew Zisserman, was the runner-up of ILSVRC 2014. Instead of using large convolutional filters like AlexNet, VGGNet uses a smaller 3x3 convolutional filter with stride 1. But, it applies

Table 2: AlexNet configuration used

Configuration
Input (32x32 RGB image)
Convolutional layer. Depth = 64/24. Size = 5x5.
Max-pooling. Size = 3x3. Stride = 2.
Convolutional layer. Depth = 92/64. Size = 5x5.
Max-pooling. Size = 3x3. Stride = 2.
Fully-connected layer - 256 units
Fully-connected layer - 256 units
Fully-connected layer - 10/100 units
Softmax

Table 3: VGGNet configuration used

Configuration	
VGG-B (9 weight layers)	VGG-D (13 weight layers)
Input (32x32 RGB image)	
Convolutional layer. 64. 3x3.	Convolutional layer. 64. 3x3.
Convolutional layer. 64. 3x3.	Convolutional layer. 64. 3x3.
Max-pooling. 2x2. Stride = 2.	
Convolutional layer. 128. 3x3.	Convolutional layer. 128. 3x3.
Convolutional layer. 128. 3x3.	Convolutional layer. 128. 3x3.
Max-pooling. 2x2. Stride = 2.	
Convolutional layer. 256. 3x3.	Convolutional layer. 256. 3x3.
Convolutional layer. 256. 3x3.	Convolutional layer. 256. 3x3.
	Convolutional layer. 256. 3x3.
Max-pooling. 2x2. Stride = 2.	
	Convolutional layer. 256. 3x3.
	Convolutional layer. 256. 3x3.
	Convolutional layer. 256. 3x3.
Max-pooling. 2x2. Stride = 2.	
Fully-connected layer - 1024 units	
Fully-connected layer - 1024 units	
Fully-connected layer - 10/100 units	
Softmax	

multiple small convolution layers in sequence to mimic the effect of using large convolution and to represent complex features. VGGNet uses max pooling with size 2x2 and stride 2. In the paper [6], Simonyan and Zisserman propose a few VGG architecture, see Figure 6 in the appendix.

Because VGG was originally designed for the ImageNet dataset which has large 224x224 image size, we need some adjustments for making VGG work with small CIFAR images. Our VGG network will have the configuration of VGG-B and VGG-D. The detail configuration of VGG-B and VGG-D are shown in Figure 6. We will evaluate two different networks to evaluate the effect of the depth of the network. But, due to the images size in CIFAR is small, we cannot use the exactly same VGG-B and VGG-D design because after the third or fourth max pooling, the spatial size of our dataset is already small. It is useless to subsample the spatial size that cannot be subsampled again. We then use our adjusted VGG-B and VGG-D as shown in Table 3. We also decrease the number of neurons in the fully connected layers to 1024 hidden units per layer. The original architecture has 4096 hidden units per layer and 1000 hidden units in the last layer. We decrease the number of the hidden units because unlike ImageNet dataset that has 1000 classes, CIFAR has much fewer classes, 10 or 100 classes.

In short, we will try and evaluate an adjusted 9 weight layers VGG-B and an adjusted 13 weight layers VGG-D. **We will refer those architecture as 9-layers VGG and 13-layers VGG**. According to the paper, the learning rule for VGGNet is stochastic gradient descent with momentum of 0.9 and weight decay of 0.0005 [6].

4.3.3 Why not GoogLeNet, ResNet, or DenseNet?

There are several other popular architectures such as GoogleNet, ResNet, and DenseNet which is superior to ALexNet and VGGNet in the image classification. The reason why we do not cover those architectures is because: (i) Considering the results of the CIFAR dataset in Kaggle, we can achieve very high accuracy (more than 90% in CIFAR-10) using VGGNet. (ii) Unlike ImageNet, CIFAR dataset is relatively small by the number of the images and small by the resolution. Hence, complex architectures such as resnet cannot give its full potential because small 32x32 RGB images do not have many features to be learned. Those architectures perhaps achieve a better accuracy, but only with a small margin [9]. (iii) The complexity of those architectures are high and training very deep network such as resnet takes much longer time [15]. With our limited resources and time, the complex architecture is not a good option.

4.3.4 Improving The Performance: Dropout, Batch Normalization, "The All Convolutional", and ELU

One of the many things to improve the performance is by reducing overfitting. To reduce overfitting, we will apply dropout and batch normalisation.

Dropout is a regularisation method where some of the neurons are "dropped-out" during training [16]. The neurons which dropped-out are switched off during the forward pass, and the weight updates are not carried out during the backwards pass. The number of neurons which dropped out are determined by some probability number. We use dropout probability of 0.25 after every max

pooling and 0.5 after every fully-connected layers. We pick 0.25 dropout for the convolutional layers because convolutional layer does not have many parameters. For that reason, dropout becomes ineffective. Indeed, dropout will regularize our network. So, we still will use dropout but keep the dropout probability low for the convolutional layer so that the network will not be "over-regularized". We try several dropout with various probability on the convolutional layers, and conclude that higher dropout probability only slows down the training and it seems that dropout does not prevent co-adaptation in the convolutional layers. The notion of dropout is to effectively train a network that acts like a sum of smaller networks. For the fully-connected layers, we choose 0.5 dropout probability because it results in equal probability distribution for the smaller networks that were created by dropping out neurons. And moreover, Srivastava and Hinton using 0.25 for the convolutional layers and 0.5 for the fully-connected layers in [16].

Batch Normalisation gives a different strategy to combat overfitting. It addresses a few problems during training, such as internal covariate shift and vanishing gradient [17]. It solves the internal covariate shift by normalising layer inputs. Batch normalisation is done as follows

$$\hat{u}_i = \frac{u_i - \mu_i}{\sqrt{\delta_i^2 + \epsilon}}. \tag{1}$$

The normalised values are scaled and shifted by

$$z_i = \gamma_i \hat{u}_i + \beta_i, \tag{2}$$

where γ and β scale and shift the normalised value.

To improve our classification performance, we will also try several approaches that CIFAR's leaderboards use. We will try an approach in [13]. Sprigenberg et al. replace max-pooling with a convolutional layer that acts like pooling. The authors argue only the spatial reduction that important for achieving good performance in CNN. Thus, one can replace pooling with a convolutional layer, ignoring that pooling's feature-wise nature makes the optimisation easier and the p-norm makes the representation more invariant.

We will also try an approach in [12] which employs ELU as the activation function. Both ELU and ReLU reduce the vanishing gradient problem, because unlike sigmoid and tanh, ELU and ReLU do not have saturating region. For that reason, we do not evaluate both sigmoid and tanh in our analysis; especially because we will train a deep neural network. ELUs have an advantage over ReLUs which it has negative values. Having negative value means that ELUs can shift the mean of the activations closer to zero and make the learning faster as the gradient is closer to the natural gradient.

There are other interesting approaches to improve our prediction performance such as Ben Graham's Fractional max-pooling [10] and spatially-sparse CNN [11]. However, we will not evaluate those strategies in our report.

4.4 Experimental Setup

We use TensorFlow 1.0 GPU version running on Amazon EC2 p2.xlarge instance. The p2.xlarge instance runs on four vCPUs Intel Xeon E5-2686, 61 GB of RAM, and one Nvidia Tesla k80 with 12GB of VRAM.

To train our model, we do not use mlp.data_providers. The mlp.data_providers uses the feed_dict system to provide the training data which is not efficient pipeline and can cause the GPU underutilised. Thus, we use Tensorflow's queue system to provide the training data during training.

To design our architecture, we will only evaluate the architecture performance on CIFAR-10. And when we have already created our final CNN architecture, we will evaluate the classification accuracy on both CIFAR-10 and CIFAR-100 dataset.

4.4.1 Implementation in Tensorflow

We heavily rely on the tensorflow module to build our CNN. We use *tf.nn.conv2d* for our convolutional layers, *tf.nn.max_pool* for max pooling layers, *tf.nn.relu* for ReLU activation functions, *tf.nn.elu* for ELU activation functions, *tf.nn.dropout* for droping out neurons, *tf.nn.l2_loss* for applying L2 regularisation, and *tf.matmul* for weight and input operation in the fully connected layers. For stochastic gradient descent with momentum, we use *tf.train.MomentumOptimizer*. To calculate the cross-entropy, we use *tf.nn.softmax_cross_entropy_with_logits*. And to create a variable, we use *tf.get_variable*. The *tf.get_variable* allows variable sharing across GPUs.

For batch normalisation, we use *tf.nn.batch_normalization*. We use *tf.nn.moments* with axis [0] for fully connected layers and axis $[0, 1, 2]$ for convolutional layers to evaluate batch's mean and variance.

4.4.2 Preparing The Dataset

We use the CIFAR dataset binary version to be fed into the queue system. The dataset has 6 different files, the training data named data_batch_1.bin, data_batch_2.bin, and so forth; and the test set named test_batch.bin. We rename our 5th training data into validation.bin for our validation data. In this way, we have 40,000 training images (data_batch_[1-4].bin), 10,000 validation images (validation.bin), and 10,000 test images (test_batch.bin). We apply a preprocessing image using *tf.nn.per_image_standarization* to standarize the image's RGB value.

5 Result and Discussion

5.1 Lessons Learned from AlexNet vs. VGGNet

5.1.1 Playing with AlexNet

We will explore and evaluate AlexNet which described on Table 2 to classify CIFAR dataset. Our AlexNet has two convolutional layers and two fully connected layers. Local response normalisation follows each convolutional layers, and max pooling follows every local response normalisation. The output of the last fully-connected layer is connected to a softmax which produces a distribution over the CIFAR-10/CIFAR-100 classes. It turns out that AlexNet can achieve 80.18% in accuracy on CIFAR-10. This accuracy is achieved without data augmentation.

Table 4: Accuracy and error AlexNet using different convolutional filter depth

Configuration	Error	Accuracy
Conv. Layers with filter depth 24 and 64	1.01	80.18%
Conv. Layers with filter depth 64 and 92	0.93	82.26%

Table 5: Accuracy and error VGGNet using different network depth after 50 epochs

Configuration	Error	Accuracy
9-layers VGGNet	1.01	86.34%
13-layers VGGNet	1.29	83.14%

We don't expect much by having only two convolutional layers in out network. Alex himself stated that the depth is important to obtain a good classification performance. The performance drops quite significantly when he removed any convolutional layer [3]. Yet, in our model, we only have two convolutional AlexNet's layer due to the spatial reduction cannot be done again using large AlexNet's filters size and stride. We think that it is useless to increase depth while the spatial information stays the same. So, the best we can do is to varying filter depth. Thus, we change the depth of our convolutional filters. The result is shown in Table 4. From the table, we know that deeper convolutional filter improves accuracy and reduces error. This is mainly because the deeper convolutional filter can represent more information when the spatial size reduced. Knowing that depth of the convolutional filter results in a better performance, we try always to use deep convolutional filters in the next experiment.

5.1.2 VGGNet and The Unstable Gradient Problem

Our VGGNet has a configuration as shown in Table 3. As stated in the previous section, we need to adjust the VGGNet to suit with CIFAR dataset. We end up with two different VGGNet configurations. The first one is a 9-layers VGGNet composed of six convolutional layers, three fully-connected layers, and max pooling follows every two convolutional layers. And the second one is a 13-layers VGGNet with ten convolutional layers, three fully-connected layers, and max pooling follows every two or three convolutional layers. In short, the first configuration is similar to VGGNet-B, while the second configuration is comparable to VGGNet-D.

The result of both configurations is shown in Table 5. Theoretically, a 13-layers VGGNet can perform better. But because the limited time and resource, we decided to stop at 50 epochs. And turns out that within such training time, 9-layers VGGNet performs better. Having a better classification accuracy is one reason to choose shallower a 9-layers VGGNet. Furthermore, another reason that drives us not to choose the deeper network is *the unstable gradient problem*. We found such problem when we trained a 13-layers VGG. The gradients somehow vanished when the network was trained using the exactly same learning hyperparameters with the 9-layers VGG. Then, to combat this problem, we reduced the learning rate until the gradients do not vanish anymore.

This resulting in a worse error and classification accuracy than a 9-layers VGG in 50 epochs due to the lower learning rate. Considering training deeper network is harder and our resource is limited, we then choose a 9-layers VGGNet for our VGG architecture to be compared to AlexNet.

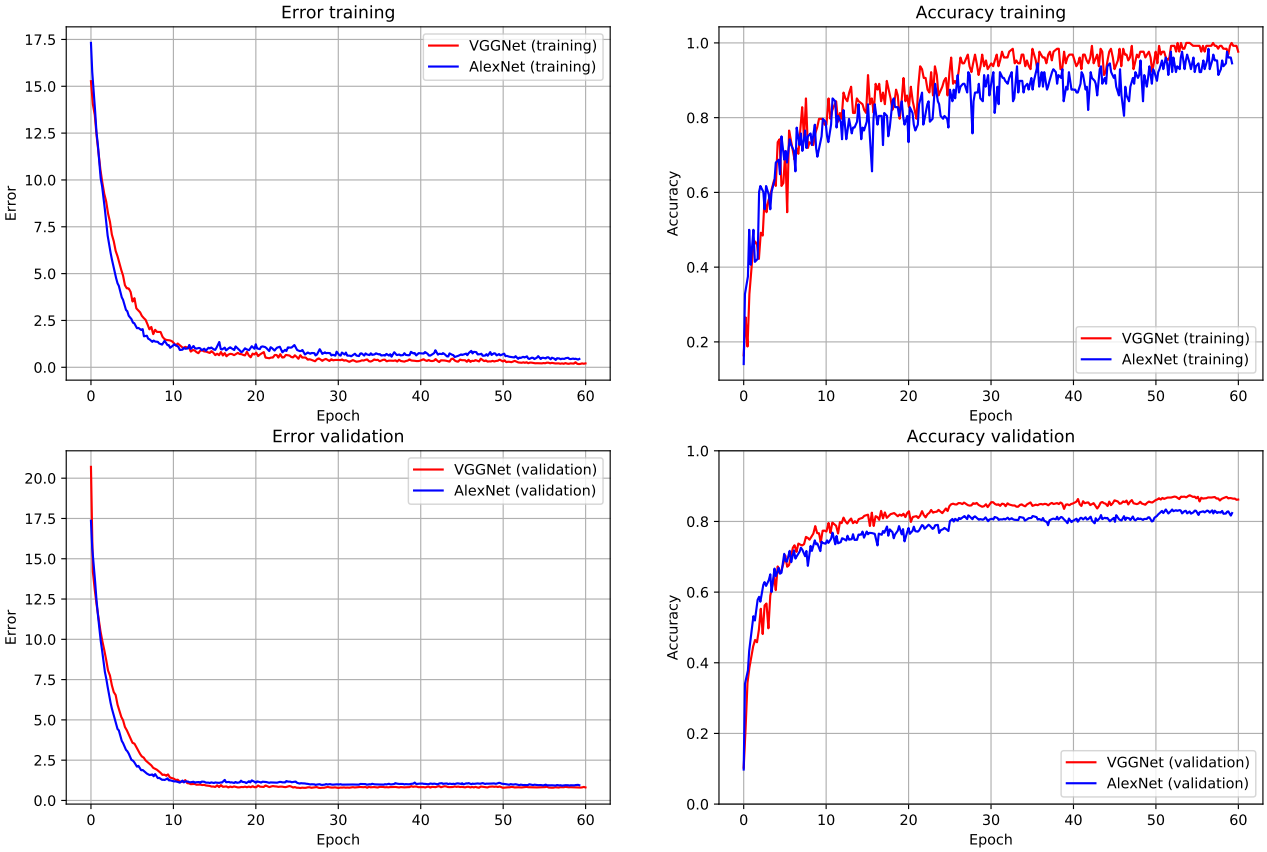

Figure 2: Comparing AlexNet and 9-layers VGGNet. Learning rate is set too high, but it has the best accuracy in 60 epochs comparing to the slower one. (The learning rate will be discussed in Section 5.4)

In general, VGGNet has a better performance comparing to AlexNet. The learning curve of both architectures is plotted in Figure 2. The obvious reasons why VGGNet has better classfication accuracy is that VGGNet is much deeper than AlexNet. Our VGGNet has 9-layers, while our AlexNet only has 5-layers with two convolutional layers. The large filter AlexNet used does not enable us to employ more convolutional layers, unless we use a different filter size and stride configuration. After all, this corresponds to what both papers stated [6] [3] that confirms the importance of depth in classification accuracy, or to be more specific: the importance of depth in visual representation. Finally, considering classification accuracy that better than AlexNet, **we choose the 9-layers VGGNet as our base architecture**, and keep improving it in the following section, so it can achieve a much better accuracy.

Table 6: Comparison of 9-layers VGGNet, 9-layers VGGNet + Dropout, and 9-layers VGGNet + Dropout + Batch Normalisation

Configuration	Error	Accuracy
9-layers VGGNet	1.01	86.34%
9-layers VGGNet + Dropout	0.55	88.45%
9-layers VGGNet + Dropout + BN	0.47	90.03%

5.2 Reducing Overfitting

We already have an adjusted 9 weighted layers VGGNet as our base architecture. As mentioned before, we will simply refer to this architecture as 9-layers VGGNet or just VGGNet. To combat overfitting, we will use dropout and/or batch normalisation. First, we will add dropout in our network and then we will add batch normalisation. Later, we will evaluate the performance of the regulated network.

We put dropout with probability 0.25 after every pooling layers and dropout with probability 0.5 after every fully-connected layers. The reason we pick this probability number has been explained in the previous section. In short, convolutional layer does not have many parameters. Therefore, it causes using high dropout probability will only slow down the training and will not prevent co-adaptation. We try to keep the dropout probability for the convolutional layers low, because dropout still gives a good regularisation since we see around 3% improvement with 0.25 dropout probability rather than without. We found that having probability closer to 0.5 for the convolutional layers becomes ineffective as we see no accuracy improvement and causes the learning much slower. We pick 0.5 as the dropout probability for the fully-connected layers because it results in equal probability distribution for the smaller networks that were create by dropping out neurons. Srivastava and Hinton himself use 0.25 dropout probability for the convolutional layers and 0.5 dropout probability for the fully-connected layers. We see that dropping out units prevents overfitting very well, but it also makes the training time significantly longer as the learning goes slower. As a result, applying dropout can increase the accuracy to 88.45% and reduce the error to 0.55.

We also try to implement batch normalisation to help dropout in combating overfitting. We put batch normalisation after every convolutional and every fully-connected layer; before its activation. Batch normalisation helps to preventing overfitting in a different way. During training, the distribution of inputs to certain hidden layer changes due to the distribution of the input to the hidden layer before also changed. This problem is called internal covariate shift [17]. The idea of batch normalisation is to whitening the input data of every hidden layers, not only in the input layers. Batch normalisation works by shifting inputs to zero-mean, unit variance, and making the inputs de-correlated. Consequently, it makes data comparable across features on a batch basis. It has been known that if the data is "whitened", the network will converge faster [4, 18]. As the result, we are able to boost the accuracy to 90.03% and lower the error to 0.47.

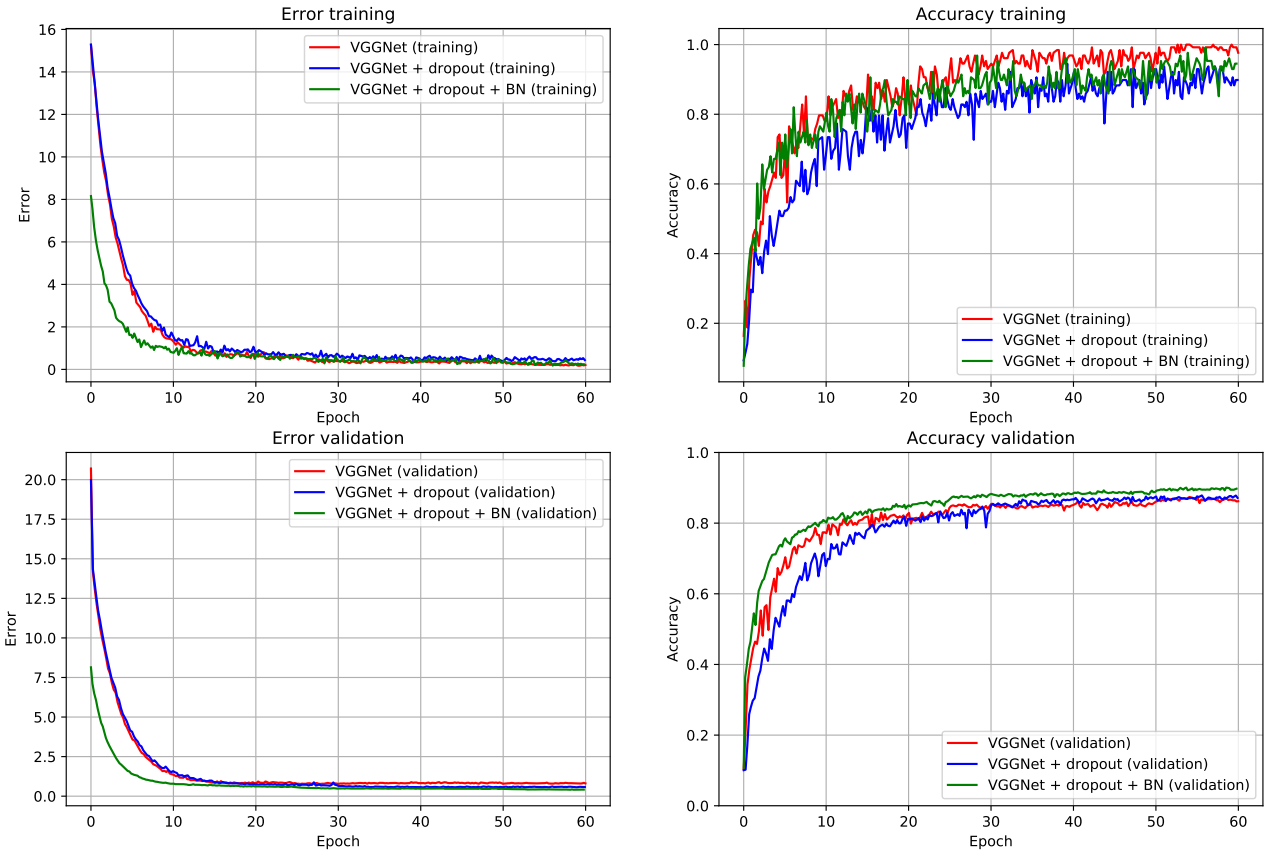

Figure 3: Comparing 9-layers VGGNet, 9-layers VGGNet with dropout, and 9-layers VGGNet with dropout and batch normalisation. Learning rate is set too high, but it has the best accuracy in 60 epochs comparing to the slower one. (The learning rate will be discussed in Section 5.4)

5.3 Exploring More

5.3.1 "The All Convolutional Layers"

The idea of the all convolutional layers is initiated by Springenberg et. al in [13]. We created the similar network by replacing max-pooling layers with convolutional layers with the size of the kernel and the stride correspond to max pooling's size and stride. The authors also implemented their method in CIFAR dataset and obtained accuracy of 90.92%. We try their method using a different architecture and achieve only 85.65%. The result is somewhat comparable, but slightly lower, to our CNN with max pooling. This contradicts to what authors state that the performance matches or slightly outperform the convolutional-max pooling configuration. Our view of this result is that: (i) we use a different architecture; thus it leads to a different result. (ii) The convolutional layers that act as pooling layers in the architecture used cannot learn the necessary invariances from our dataset as good as max-pooling. Therefore, because we cannot gain a better accuracy from replacing max pooling, we will keep max

Table 7: Accuracy and error VGGNet with convolutional-max pooling configuration and all convolutional layers

Configuration	Error	Accuracy
Convolutional + max pooling	1.01	86.34%
All convolutional layers	0.59	85.65%

pooling in our final architecture.

5.3.2 Trying ELU

In this subsection, we try a different activation function. We do not discuss Tanh or Sigmoid as it has already covered in the previous coursework. For training deep neural network, ReLU certainly has advantages over tanh and sigmoid that the probability of the gradient to vanish is smaller because it does not have saturating region.

The exponential linear unit (ELU) was introduced in [12]. Using ELU in our base architecture surprisingly boost the accuracy to 89.05%, improving almost 3% of from our base VGGNet classification accuracy. The main advantage of ELU over ReLU is that it has the negative value that allows pushing the mean of unit activation closer to zero. Mean shift closer to zero can speed up the learning process like batch normalisation does, because of a reduced bias shift effect.

However, applying ELU in our final architecture only gives a very little improvement. ELU improves the accuracy to 90.65% from 90.03% and reduces the error to 0.39 in our 9-layers VGGNet with dropout and batch normalisation. Although the improvement is not significant, we decide to use ELU as the activation function for our architecture. **We will refer our final architecture which composed of 9-layers VGGNet with dropout and batch normalisation, and uses ELU as its activation function as just 9-layers VGGNet**.

5.4 Finding A Better Learning Rate

All learning curve plot in Figure 2 and Figure 3 tell us that we use high learning rate. We set the same learning rate for the different architecture so we can easily compare to choose the best accuracy among them. We aware that higher learning rate leads to a worse value of convergence, but it will learn faster. And because we want to analyse the model after a certain number of epochs, we need a higher learning rate so that the model will settle faster. After all, we treat all model equally and we use high learning rate because of the time and resource constraint.

In this section, we present the learning curve plot of our final 9-layers VGGNet using several learning rates. We use 0.15, 0.1, and 0.5 learning rate for the evaluation. Our network is trained using SGD with momentum of 0.9. As we can see in Figure 4, higher learning rate (0.15) decays the error faster and achieves the high accuracy faster. Lower learning rate (0.05) shows the contrary. Considering the accuracy and error on the validation dataset, both 0.15 and 0.1 learning rates give the similar performance. However, we will pick 0.1 as our

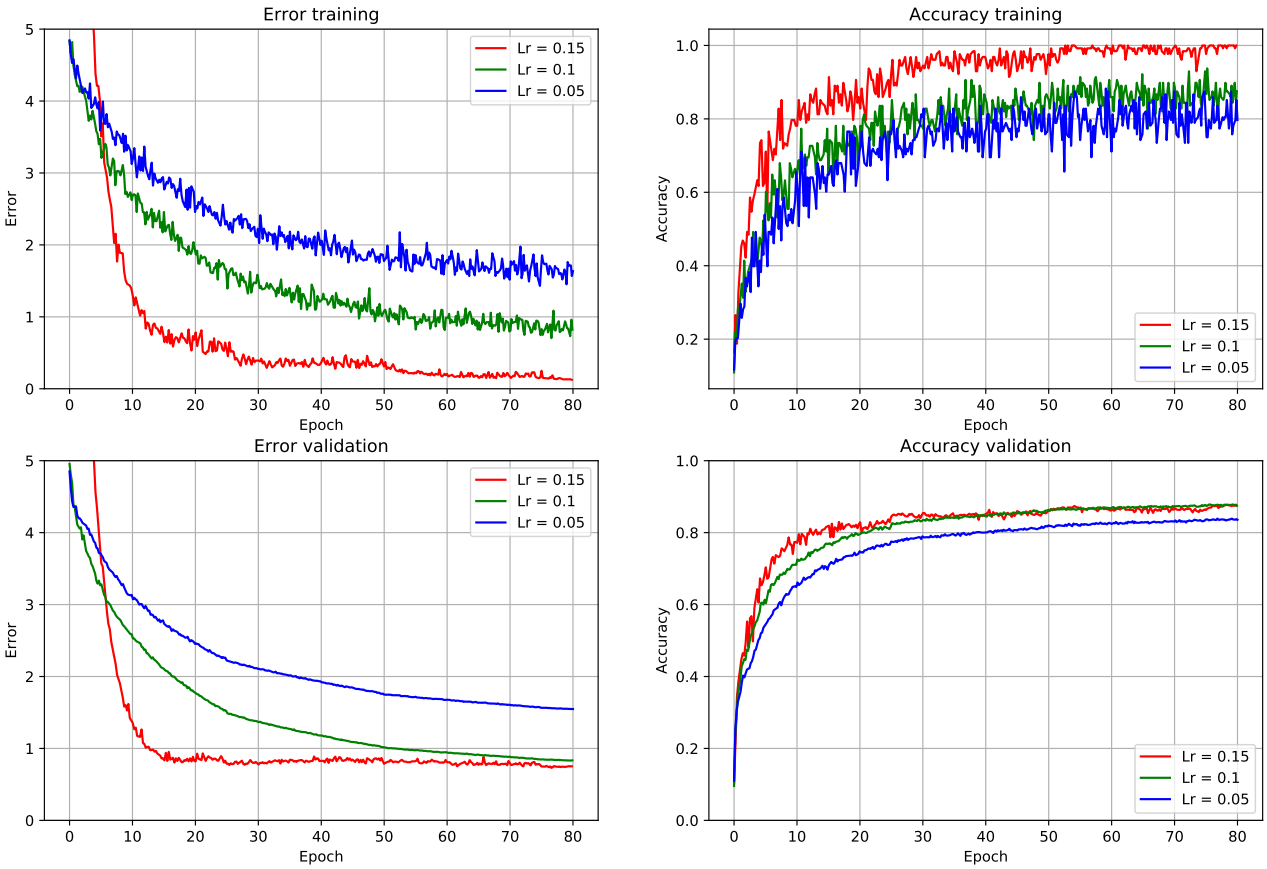

Figure 4: Our final 9-layers VGGNet architecure trained with various learning rate

learning rate as the learning curve shows a good learning curve, and it is the best practice to choose the good learning rate.

5.5 Classifying CIFAR-100

In this section, we present the CIFAR-100 classification result achieved by our 9-layers VGGNet. The classification performance on CIFAR-10 has already discussed in the previous section. Our 9-layer VGGNet architecture delivers accuracy at 58.79% on the test data. We actually expect the accuracy result more than 60% on CIFAR-100.

The learning curve is plotted in Figure. We try to increase the dropout probability but it did not help to reduce the overfit after about 50 epoch and cannot raise the accuracy. We have already applied batch normalisation in every convolution and fully-connected layers. It seems that we cannot reach the desired accuracy unless we build a deeper network with more hidden units for the fully connected layers. Moreover, we also need heavy data augmentation since CIFAR-100 dataset has fewer images per class.

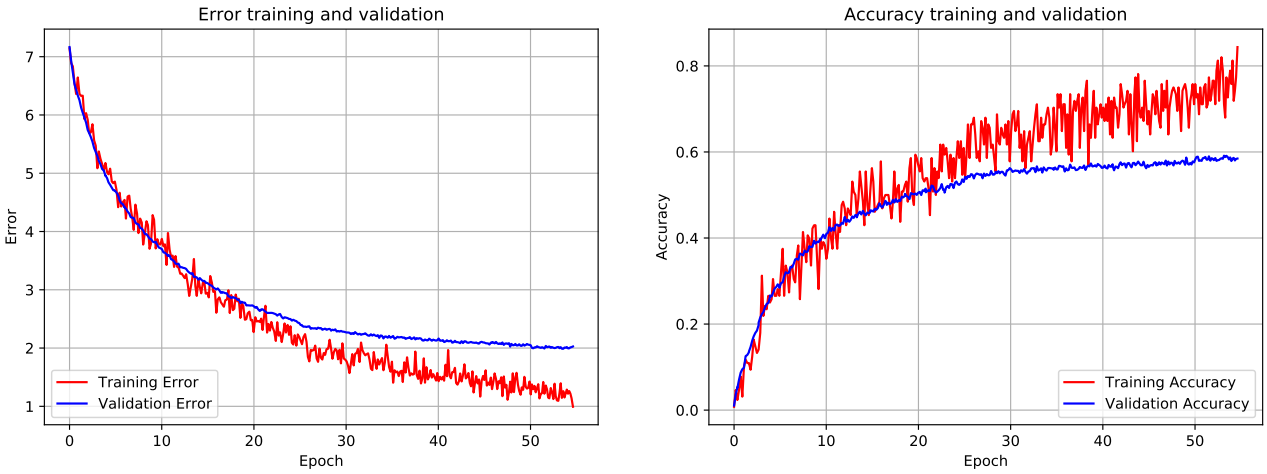

Figure 5: Learning curve classifying CIFAR-100

Table 8: Baseline and our final 9-layers VGGNet accuracy on test data

Model	CIFAR-10	CIFAR-100
Baseline	52.51%	24.02%
Our final 9-layers VGGNet	90.18%	57.39%

5.6 Comparing to The Baseline Model

We have already created a convolutional neural network with configuration as shown in Table 9. We use ELU as the activation function and SGD with momentum for the learning rule. We achieve validation accuracy of 90.65% using this architecture for CIFAR-10 and 58.79% for CIFAR-100. Using test data we achieved accuracy of 90.18% for CIFAR-10 and 57.39% for CIFAR-100. Our baseline has accuracy of 52.51% for CIFAR-10 and 24.02% for CIAR-100 on the test data. Our final CNN architecture *is obviously outperformed* the baseline. This is of course because a simple baseline cannot exploit spatial or temporal invariance in the dataset, while our convolutional neural network can.

6 Conclusion

In this coursework, we have already explored and evaluated the convolutional neural network for image classification using CIFAR dataset. We start by implementing AlexNet and VGGNet for our architecture, then evaluate both results, select the best out of those two architectures, and improve the performance. And although there are more advanced design such as ResNet and DenseNet, we believe that we will not gain much from its potential because CIFAR dataset is small and the images are small by resolution. Small resolution images mean that it does not has many features to be learned. We also think that using simple architecture such as AlexNet and VGG can give high classification accuracy on CIFAR dataset.

Table 9: Final configuration of our architecture

Configuration
Input (32x32 RGB image)
Convolutional layer. Depth = 64. Size = 3x3. with Batch Normalisation.
Convolutional layer. Depth = 64. Size = 3x3. with Batch Normalisation.
Max-pooling. Size = 2x2. Stride = 2.
Drop out 0.25
Convolutional layer. Depth = 128. Size = 3x3. with Batch Normalisation.
Convolutional layer. Depth = 128. Size = 3x3. with Batch Normalisation.
Max-pooling. Size = 2x2. Stride = 2.
Drop out 0.25
Convolutional layer. Depth = 256. Size = 3x3. with Batch Normalisation.
Convolutional layer. Depth = 256. Size = 3x3. with Batch Normalisation.
Convolutional layer. Depth = 256. Size = 3x3. with Batch Normalisation.
Max-pooling. Size = 2x2. Stride = 2.
Drop out 0.25
Fully-connected layer - 1024 units with Batch Normalisation.
Drop out 0.5
Fully-connected layer - 1024 units with Batch Normalisation.
Drop out 0.5
Fully-connected layer - 10/100 units
Softmax

Large convolutional and pooling filter make us cannot have the full potential from AlexNet. Using such design with CIFAR dataset, we only can use two convolutional layers. We don't expect much from our AlexNet-two-convolutional-layers performance. We try to improve the accuracy by varying the convolutional filter depth. We know that deeper convolutional filter gives a better representation from the squeezed spatial information. Yet, the performance is still significantly outperformed by VGGNet.

We also cannot have the full VGGNet potential because of the same reason. However, because VGGNet employs a smaller convolutional and max-pooling filter, we still can employ much more layers than AlexNet. We learned two lessons from training VGGNet. The first is the network depth is beneficial for classification accuracy. And the second is the training deeper neural network is harder. We faced vanishing gradient problem when we trained 13-layers VGGNet. We have to reduce the learning rate and it leads to have a worse result than the shallower 9-layers VGGNet with 50 epochs training. For that reason and limitation in our resource and time, we choose 9-layers VGGNet as our base architecture.

Then, we tried to improve the classification accuracy by reducing the overfitting. We applied dropout and batch normalisation to address this problem. We choose dropout with probability of 0.25 for our convolutional layers and 0.5 for our fully-connected layers. We found that higher dropout probability in convolutional layers lead to inefficiency as it slows down the training and does not prevent co-adaptation. Dropout suddenly becomes inefficient when regularizing convolutional layers because it does not have many parameters. But, with low dropout probability, the network is quite well-regularized. For the fully-connected layers, we choose dropout probability of 0.5 because it results in equal probability distribution for the "smaller network" caused by dropping out neurons. After all, Sristava and Hinton recommend 0.25 as the dropout probabilty for the convolutional layer and 0.5 as the dropout probability for the fully connected layers. Batch normalisation also help to reducing overfitting by addressing internal covariate shift problem. It applies the whitening data to every hidden layers input so the network converges faster. Employing both drop out and batch normalisation, we achieve the validation accuracy of 90.03%.

We then tried to implement some unconservative approach to improve the classification accuracy. We have tried to replace the pooling layer with convolutional layers that act like pooling. We do not see any improvement, so we keep pooling in our architecture. The reason why we do not have the improvement is because we use a different architecture with the paper and the convolutional layers that act as pooling layers somehow cannot learn the necessary invariant better than max-pooling. Then, we also tried ELU activation function to replace ReLU. ELU can help to push the mean of unit activation closer to zero, hence, it speeds up the learning process. ELU was able to slightly increase the classification performance in our architecture to 90.65%.

Lastly, we have already explored different notable architecture, different convolutional filter depth, various network depth, regularisation, substitute pooling layer, and different activation function. We did not try different learning rule because we stick to the paper which use SGD with momentum. We finally end up with a 9 layers weight VGGNet architecture composed of six convolutional layers with max pooling every two or three convolutional layers, followed by three fully connected layers before softmax. We employ dropout and batch nor-

malisation as the regularisation since it helps to prevent the overfitting problem very well. We also use ELU as the activation function. ELU performs better than ReLU because it has negative value and allows pushing the mean of unit activation closer to zero which results in faster learning process. This architecture achieves 90.65% validation accuracy for the CIFAR-10 dataset and 58.79% for the CIFAR-100 dataset. Using testing dataset, we achieve 90.18% on CIFAR-10 and 57.39% on CIFAR-100. Our result is successfully outperformed the baseline that achieve *only* 51.51% on CIFAR-10 and 24.02% on CIFAR-100. This of course shows that our convolutional neural network can learn invariance feature or exploit spatial invariance in the dataset much better than a simple feedforward neural network.

References

[1] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset, 2014.

[2] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[4] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[9] Rodrigo Benenson. Classification datasets results. http://rodrigob.github.io/. (Accessed on 03/12/2017).

[10] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[11] Benjamin Graham. Spatially-sparse convolutional neural networks. *arXiv preprint arXiv:1409.6070*, 2014.

[12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[13] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[14] Andrej Karpathy. Lessons learned from manually classifying cifar-10. http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/. (Accessed on 03/12/2017).

[15] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[16] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[18] Simon Wiesler and Hermann Ney. A convergence analysis of log-linear training. In *Advances in Neural Information Processing Systems*, pages 657–665, 2011.

7 Appendix

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					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Figure 6: VGGNet Configurations