Assignment 2: Cell type prediction using CNNs

Sanchit Goel
The University of Adelaide
Adelaide, South Australia, Australia
sanchit.goel@student.adelaide.edu.au

Abstract

Convolutional Neural Networks (CNNs) are probably the most widely used technique for computer vision problems. This project explores the performance of a custom CNN model designed using the findings of AlexNet and Very Deep Convolutional Network (VGG) against state-of-the-art CNN architectures such as AlexNet, VGG, InceptionNet and Residual Network (ResNet). The aim is to try and understand why these established architectures work so well and what makes them unique. It makes use of a a blood cell images dataset that consists of 17092 cell images from 8 different classes. The dataset is not highly imbalanced, which is why accuracy and F1 score are compared. Results show that ResNet-18 outperforms all the other models by reporting an accuracy of 97.19% and an F1 score of 97.14%. Each model is built using some novel method, and all these methods perform well. However, the findings show that ResNet-18 is the most robust model and is extremely quick compared to other models when it comes to learning image features.

1. Introduction

In 2009, a team of researchers from Princeton University, USA, released the ImageNet database [1]. Their goal was to create a comprehensive database that could be used by researchers to innovate new algorithms for solving a wide variety of computer vision challenges like object detection, image segmentation, image classification, and so on. With the release of ImageNet, Convolutional Neural Networks (CNNs) quickly gained attraction due to their outstanding performance on this benchmark. CNNs became the most used technique for computer vision tasks, since the convolution operation would aggregate information from the image much better than traditional Machine Learning (ML) algorithms. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) provided a competitive platform to further boost the incoming interest and research around CNNs.

Due to this sudden influx of interest, research in CNNs took off. AlexNet [2], introduced in 2012, was one of the first deep CNN architecture that introduced multiple layers. It achieved the best results by introducing new techniques like ReLU activation functions, dropout layers and max-pooling layers which helped address issues of overfitting and vanishing gradients. This sparked further interest in CNNs, from research institutes to private companies, everyone wanting to build better CNN architectures.

VGG [3], InceptionNet [4] and ResNet [5] followed AlexNet, all coming in successive years, showing the real capability of CNNs. VGG focused on simplifying the network by introducing smaller convolutional layers and by increasing the depth of the network, which improved feature extraction. InceptionNet (or GoogLeNet) introduced the inception module which stacked multiple convolutional and max-pooling layers to extract and concatenate different features. Finally, ResNet focused on solving the vanishing gradient issue in deep networks by introducing residual connections. These advancements are still relevant in recent literature and paved the way for research in many fields like robotics, computational biology, agriculture, and so on.

This project tries to understand the key differences between these architectures by using a database completely different from CIFAR10 or CIFAR1000 to compare the models better by training them on a dataset that hasn't been widely experimented with before from scratch. The detailed aim of this project are listed as follows:

1. To compare the performance of a custom CNN with State-Of-The-Art (SOTA) CNNs like AlexNet, VGG, InceptionNet and ResNet on a dataset different from CIFAR.

2. To analyse the performances and relate them to the differences in the architecture.

2. Method Description

This section deals with the dataset setup and describes the models that were tested.

2.1. Dataset and Preprocessing

For this project, the blood cells image dataset was used from Kaggle [6]. It consists of 17,092 images of individual normal cells acquired in the core laboratory at the Hospital Clinic of Barcelona. It consists of 8 classes: neutrophils, eosinophils, basophils, lymphocytes, monocytes, immature granulocytes (promyelocytes, myelocytes, and metamyelocytes), erythroblasts and platelets or thrombocytes. The dataset is slightly imbalanced which is why when modelling, a stratify split was used to split the classes evenly for the training, validation and test set. The training set consists of 70% of the entire dataset, while the validation and test set consist of 15% of the dataset each, adding up to 100%. The images are in JPGs and were labelled by clinical pathologists. The directory structure consisted of a folder for each of the labels with images for that label inside them. Using the directory structure, a CSV was created to specify the paths to the images and their labels. This CSV file was then used for creating the dataset class and the dataloaders.

2.2. Models

All the models use the convolution and the max-pooling operation. In the convolution operation, we slide a filter F through the image to compute a weighted average over the region (or pixels) the filter is slided on to get a smaller version of the image that condenses the information present in the original image. Based on the weights of the filter, we can decide what information to we want in our smaller version of the image. The max-pooling operation uses a filter as well that is slided over different regions of the image. However, we simply store the maximum pixel value observed in that region to get the smaller image.

The filter size determines the size of the output image. Bigger the filter, smaller the output image. In some cases, this might come across as an issue. For example, the task can be to aggregate information from a big region while not reducing the image size drastically. To overcome this, simply add a specific number of rows and columns of 0s in the input image based on the desired output. This increases the image size to overcome the issue of a smaller output when using a big filter. This is known as padding. Besides padding, in some cases we wouldn't want to slide the filter over the pixels by one step. We may want to slide the filter on alternate pixels, or maybe leave two pixels in between. This is called stride. We can determine the size of the output image given the input image size, filter size, padding and stride using the formula in eq 1.

$$
\begin{aligned}
W_O &= \frac{W_I + 2P - F}{S} + 1, \\
H_O &= \frac{H_I + 2P - F}{S} + 1, \\
D_O &= K
\end{aligned}
\tag{1}
$$

where:

W_O : Output image width

H_O : Output image height

D_O : Output image depth

W_I : Input image width

H_I : Input image height

P : Padding applied to the input

F : Filter size

S : Stride

K : Number of filters (output channels)

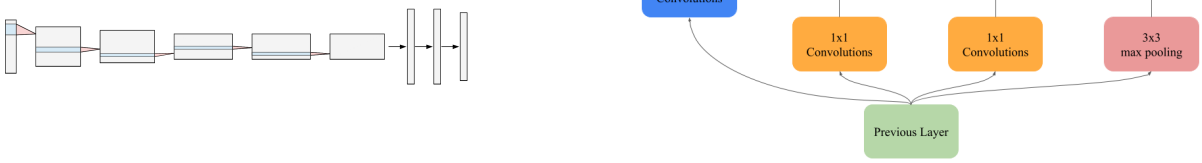
Depth means the number of channels. An image can have multiple channels, for example a coloured image has three channels, namely red, green and blue.

Layer Type	Config	Output
Input	–	224×224×3
Conv1	3→64, 3×3, p=1	224×224×64
	ReLU	224×224×64
	MaxPool 2×2	112×112×64
Conv2	64→128, 3×3, p=1	112×112×128
	ReLU	112×112×128
	MaxPool 2×2	56×56×128
Conv3	128→256, 3×3, p=1	56×56×256
	ReLU	56×56×256
	MaxPool 2×2	28×28×256
Conv4	256→512, 3×3, p=1	28×28×512
	ReLU	28×28×512
	MaxPool 2×2	14×14×512
	Dropout(0.4)	14×14×512
Classifier	Flatten	100,352
	Linear	2,048
	ReLU	2,048
	Dropout(0.5)	2,048
	Linear	8

Table 1. Custom CNN architecture. All convolutional layers use a stride of 1 and padding 1. Max-pooling layers use stride 2. The arrows show the change in the number of channels.

2.3. Custom CNN

Table 1 shows the architecture of the custom CNN used to compare with the SOTA models. This model was designed by keeping in mind some of the findings of AlexNet and VGG. Due to computational limits, the number of layers is less, but the depth of the image increases as it goes through the convolutional layers. This increases the number of features generated from the image, before flattening them and passing them into the linear layer. The layers are created making sure that the output dimensions can be processed by the next layer. Each convolution operating is followed by a max-pooling operation. Additionally, dropout has been used to regularise the CNN.



(a) AlexNet architecture

(b) A single Inception module

Figure 1. AlexNet architecture and the Inception module, showcasing the progress made in CNNs over the years.

2.3.1 AlexNet

AlexNet [2] was one of the first ground-breaking CNN architectures. It takes an input of 224x224x3 and includes eight layers, the first five being convolutional and the last three being fully connected layers. It was released in the year 2011 and outperformed all the other models in the ImageNet challenge by a huge margin in that year. It established that depth, even though increasing the complexity of the network, is key in understanding features present in the image better.

Besides showcasing the importance of depth, AlexNet was the first CNN to utilise ReLU activation function, shown in eq 2. ReLU activations soon became one of the most used activation functions in CNNs since they performed well and eliminated the issue of vanishing gradients. Being a linear operation, they significantly reduced training times as well. AlexNet also introduced dropout to regularise the network and prevent overfitting. They use dropout with a probability of 0.5, which means that when training, layers that used the dropout functionality would randomly turn off half of their neurons for generating output so that all the neurons can learn features efficiently and become more robust. Finally, AlexNet also incorporated the use of data augmentation to increase training size. These augmentations included random cropping, flipping and colour adjustments (to all channels in RGB). These techniques were so effective that they are even utilised today. Figure 1a shows the architecture of AlexNet, in the style introduced by AlexNet authors.

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2}$$

2.3.2 VGG

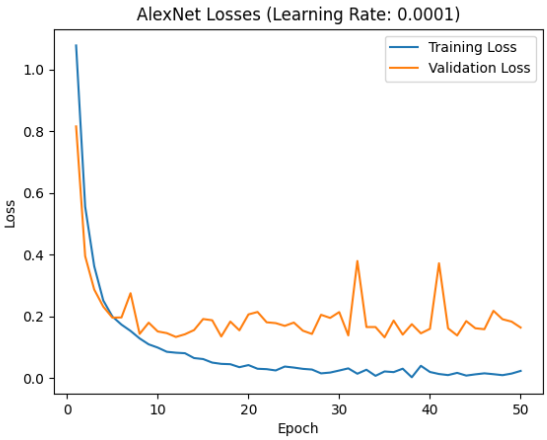
VGG [3] was released in 2014 with three versions, 13, 16 and 19 layered versions. It used smaller 3x3 convolutional layers followed by max-pooling layers. The authors experimentally found that 3x3 filters performed the best since they learnt the most information from the image. They also increased the depth based on the findings from AlexNet as stated above by using sixteen to nineteen layers. Like AlexNet, it take 224x224x3 image size as input. It came second in the ImageNet challenge in 2014 by achieving an accuracy of 92.7% in the 1000 classes dataset. This project uses the 13 layered version of VGG.

2.3.3 InceptionNet

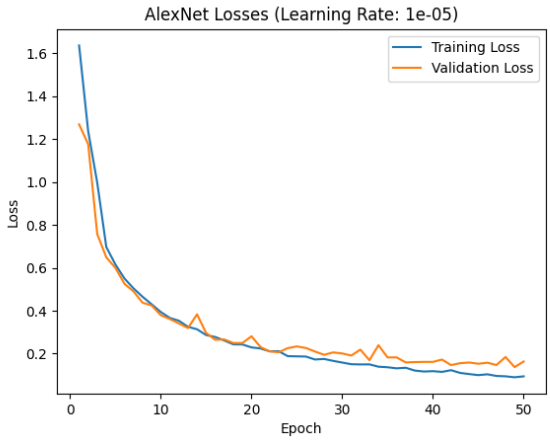
InceptionNet [4] was introduced by researchers at Google in the year 2014. This network was the winner of ImageNet in that year due to the novelty it introduced. Along with increasing the depth of the network, the authors introduced an inception module, shown in fig 1b. The inception module combines different outputs from different convolutional (1x1, 3x3 and 5x5) and max-pooling filters. This concatenates features generated by all these filters, thus incorporating much more information than a single convolutional operation. Stacking filters of the same size simply generates different kind of information from the same spatial region whereas using filters of different sizes for concatenating outputs includes information from the surrounding regions as well. They also reduced the number of parameters which made training fast. Unlike AlexNet and VGG, InceptionNet takes an input of size 229x229x3. This project uses InceptionNet v3 which includes 48 layers and is an improvised version of the original InceptionNet.

2.3.4 ResNet

ResNet [5] was introduced in 2015 and won the ImageNet challenge that year. It introduced residual blocks that implemented skip connections, a technique used in transformers [7] as well. Basically, deeper networks are prone to vanishing gradients and as the data flows through different



(a) Loss curve for learning rate 0.0001



(b) Loss curve for learning rate 0.00001

Figure 2. Loss curves used for hyperparameter optimisation

layers, information gained in earlier layers was easily lost. Using skip connections, input of a layer can be added to the output of the next 3 layers. To understand it better, we can say that ResNet consists of a sequence of residual blocks. Each block contains three convolutional layers and the input of a residual block is added to the output of the same residual block. Skip connections in reality can be added anywhere and after any amount of layers, which means that these residual blocks can be easily modified. ResNet takes 224x224x3 images as input, just like AlexNet and VGG. Through experimentation, the authors proposed four different versions of ResNet, having 18, 50, 101 and 152 layers. This project uses the 18 layered version.

2.4. Code Section

The code for the project can be found using this link: GitHub

3. Experiments and Results

This section deals with the experiments and the results of the project.

3.1. Experiments

The experiment design mainly included selecting models for the project, and choosing an optimal learning rate for training. Due to computational limits, the models selected were the smallest versions of the models introduced that are quick to train. Hyperparameter optimisation, which in this case was selecting the right learning rate is necessary to prevent overfitting and was done by looking at the training and validation loss curves of different learning rates for AlexNet trained for 50 epochs. This is because early stopping is implemented for training, so it is better to find the right learning rate for AlexNet which is the least robust model and should in theory take longer to converge. 50 epochs were used for this due to computational limits. The other models should in theory train faster with the same learning rate and therefore, simply stop training early. Learning rates tried included 0.001, 0.0001 and 0.00001.

Figure 2 shows the loss curves for learning rate 0.0001 and 0.00001. It is visible from the loss curves that the smaller learning rate converges better and the loss doesn't jump around much for it. All the models were trained for 100 epochs with a patience of 10 epochs for early stopping. The loss used was the multi-class cross entropy loss. Training was done using a P100 GPU. Adam optimiser [8] was used to adjust the learning rate and use momentum while training. Lastly, the weights of all the models were initialised randomly. This means that pre-trained weights were not used.

3.2. Results

Model	Accuracy	Precision	Recall	F1 Score	AUROC
Custom CNN	94.66	94.13	94.17	94.08	99.65
AlexNet	96.14	96.52	95.06	95.70	99.86
VGG-13	95.36	95.48	95.27	95.28	99.81
InceptionNet	95.87	95.55	95.99	95.70	99.86
ResNet-18	97.19	97.07	97.26	97.14	99.91

Table 2. Model Performance Comparison

Table 2 shows the metrics for all the models calculated on the test set (this is different from the validation set). The classes weren't highly imbalanced, so along with accuracy, let's compare F1 score as well. Precision, recall and AUROC are additionally reported for comparison. ResNet-18 seems to have the best result with an accuracy of 97.19% and F1 score of 97.14%. This means that it generalises

well compared to other models, having the best AUROC as well. Surprisingly, AlexNet has performed better than VGG-13 and InceptionNet. This can be because the number of classes is less and the data is not extremely complex, since most cell images have the cells in the centre. The custom CNN model performs good as well, even though not as good as rest of the models. Lastly. ResNet-18 has the best precision and recall, suggesting that it predicts a low number of false positives and false negatives.

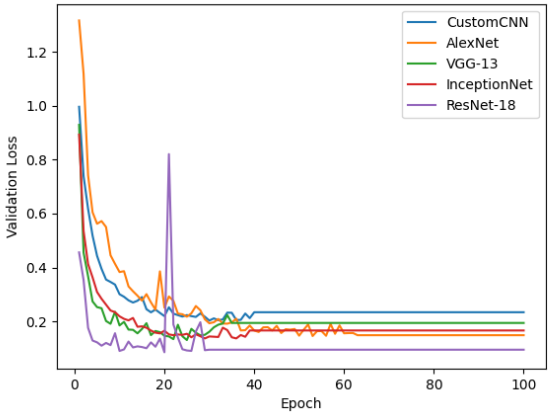


Figure 3. Validation Loss for different models over epochs. The straight line indicates that the model was early stopped.

Even though the results for all the models don't have a lot of difference, the key difference to observe is shown in figure 3. We can see that ResNet-18 stopped training earlier, which means that it was best at learning features in the dataset quickly. The straight line in the figure indicates that the model was simply early stopped, and the last recorded loss for that model was plotted. The custom CNN stopped training after 40 epochs, AlexNet stopped training after 63 epochs, VGG-13 after 35 epochs, InceptionNet after 39 epochs and finally ResNet-18 after 30 epochs. The custom CNN stops before AlexNet probably because of the use of max-pooling layers and dropout for regularisation. A con of InceptionNet and VGG-13 was that it took more time to train compared to the other models. Additionally, ResNet-18 could've even used a smaller patience or learning rate since the loss started jumping a lot, but the aim of the project was to compare all the models which required training using the same parameters.

4. Conclusion and Future Work

CNNs are a powerful variant of neural networks and an integral part of Computer Vision. They use the convolution operation to extract features from the image that can be used to predict different classes. This project aimed at exploring different CNN architectures and what they brought to the table. It used a cell image dataset that consisted of 8 different classes of cells. A custom CNN model was compared with AlexNet, VGG-13, InceptionNet and ResNet-18, showcasing that the SOTA models outperformed the custom CNN because of the extensive research that was put into them. From the results, it can be concluded that residual connections were a huge leap in deep learning, since they significantly improved the performance of CNNs and are also used in transformers.

Future work can explore cell classification by first using image segmentation to get the area of interest, i.e., the cell. Additionally, vision transformers [9] can also be explored to see how they fair against SOTA CNNs when it comes to image classification.

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009. 1

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks, 2017. 1, 3

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv (Cornell University)*, 2014. 1, 3

[4] Going deeper with convolutions. 2015. 1, 3

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. IEEE, 2016. 1, 3

[6] Acevedo Andrea, Merino Anna, Alferez Santiago, Molina Angel, Boldu Laura, and Rodelllar José. Blood cells dataset, 2022. 2

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv.org*, 2023. 3

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv (Cornell University)*, 2017. 4

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv (Cornell University)*, 2020. 5