ECS795P Coursework 3

Deeper Networks for Image Classification

Rob Clark [190821441], May 8, 2020

1 Introduction

This report examines a number of deep neural network architectures for the purposes of image classification. No attempt is made to match published accuracies for any of these architectures but instead their comparative classification power is compared when each has been trained in a similar way.

The software developed to perform these experiments was written in Python and uses the TensorFlow[1] framework developed by Google. Specifically it was developed with a TensorFlow version of at least 2.0 and extensively relies on the "Keras" API provided within the `tf.keras` package. The following sections may reference Python code and will assume an understanding and familiarity with Python, TensorFlow and Keras. A comprehensive guide to all of the above can be found in [1].

2 Data

2.1 MNIST

The MNIST dataset needs little introduction as it features as the exemplar dataset for almost any "getting started with deep learning" guide. A detailed description of the dataset and its sources can be found in [5]. In summary it is a collection of 60,000 single-channel images all of which are 28x28 pixels. Each image represents a decimal digit (0–9). There are an additional 10,000 digits in a separate test set. Pixel intensities are represented in the range 0–255. Each digit is represented equally in both the training and testing data.

Yann LeCun has compiled a list[2] of various classifiers and their accuracy vs. the MNIST test set which shows a number of approaches are able to achieve greater than 99.5% accuracy.

2.2 CIFAR-10

The CIFAR-10 dataset contains 60,000 images of 32x32 3-channel pixels, each belonging to one of ten image classes which are evenly represented. The dataset is split between 50,000 training images and 10,000 test set images. Example classes for the images are "cat", "horse" and "truck". More details are given in [4].

Baseline results[3] suggest an 82% test-set accuracy can be achieved without further augmentation of the training set.

[1] `https://www.tensorflow.org/`
[2] `http://yann.lecun.com/exdb/mnist/`
[3] `https://www.cs.toronto.edu/~kriz/cifar.html`

3 Architectures

3.1 Baseline convolutional networks (`CONV`, `DROP`)

To give a starting point to which more advanced architectures and approaches can be compared, a small and simple convolutional network following on from that shown in §5.1 of [1] will be used. The `CONV` network uses 3x3 convolutions with ReLU activation throughout. For a classification problem with K classes the network can be sketched as:

```
Conv[32] -> MaxPool[2] -> Conv[64] -> MaxPool[2] -> Conv[64]
         -> Dense[64] -> Softmax[K]
```

This network has approximately 93 thousand trainable parameters.

A counterpart to `CONV` is `DROP` which is identical apart from the addition of a *dropout* layer in an attempt to reduce over-fitting as proposed in [8]. This layer is placed between the fully-connected hidden layer and the K-neuron output layer with the probability of dropout set to 50%.

3.2 VGG

The network architectures described in [7] have become known as "VGG" after the publishing Visual Geometry Group at Oxford. The original paper outlined a number of variants (A, B, etc.) each of which had a large number of 3x3 convolutional layers alternating (dependent on the variant) with max-pooling layers. The most commonly encountered variant is often "VGG-16" which corresponds to the D variant described in the original paper, named after the number of weighted layers in the model.

The VGG networks were designed and tested on the ILSVRC-2012 ImageNet[4] dataset of images and so deal with larger images than the MNIST and CIFAR-10 datasets provide, the inputs used being cropped to 224x224 pixels.

The VGG architecture consists of stacked blocks of N 3x3 convolutional layers separated by max-pooling layers. The first block of convolutional layers output 64 channels of features, this number doubling per-block until a maximum of 512 features has been reached. After the final convolutional layer there are two fully-connected layers of 4,096 neurons using ReLU activations and then a final K-neuron softmax layer ($K = 1000$ for ImageNet).

We can sketch an example block of 2 layers of 64-filters each as:

```
Block[2 x 64] := Conv[64] -> Conv[64] -> MaxPool[2]
```

The paper notes that "The training was regularised by weight decay (the L2 penalty multiplier set to $5 \cdot 10^{-4}$) and dropout regularisation for the first two fully-connected layers (dropout ratio set to 0.5)" It's unclear whether the dropout is applied to the *input* or *output* of the fully-connected layers in the VGG paper. For the purposes of this report dropout will be applied to the *outputs* of the fully-connected hidden layers as suggested in a sample PyTorch implementation of VGG[5].

The `A` variant of the architecture could then be sketched as:

```
Block[1 x 64] -> Block[1 x 128] -> Block[2 x 256]
              -> Block[2 x 512] -> Block[2 x 512]
              -> Dense[4096, L2] -> Dropout[0.5]
              -> Dense[4096, L2] -> Dropout[0.5]
              -> Softmax[K]
```

[4] `http://www.image-net.org/challenges/LSVRC/2012/`
[5] `https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py`

The VGG approach to developing better models seems to be a question of how many layers can be possibly stacked together before the network becomes untrainable. The `A` variant has approximately 133 *million* trainable parameters when using a 224x224 input shape. The VGG networks are often referred to as large and taking a very long time to train to acceptable accuracy. During the running of these experiments the VGG models resisted all attempts to train using the `adam` optimiser and so the slower-to-converge `SGD` optimiser was used instead.

3.3 Network-in-network (`NIN`)

The Network-in-Network (NIN) architecture introduced in [6] extends the standard convolutional network to include "micro neural networks with more complex structures to abstract the data within the receptive field". The authors reported a state-of-the-art accuracy at the time of publishing on the CIFAR-10 test-set of 89.6%.

The authors suggest to "instantiate the micro neural network with a multilayer perceptron" and that "feature maps are obtained by sliding the micro networks over the input in a similar manner as CNN".

These additional sliding micro-networks can be implemented as layers of 1x1 convolutions after each layer of convolutions with a larger receptive field. Each block is referred to as an "MLP Convolutional Layer" or *mlpconv*.

```
mlpconv[3x3, 64] := Conv_3x3[64] -> Conv_1x1[64] -> Conv_1x1[64]
```

Additionally to this architectural change the authors also propose replacing the traditional finally fully-connected layers with "global average pooling" in an effort to reduce over-fitting and the number of trainable parameters in the network. In practice this requires the final convolutional layer to output the same number of feature channels as there are classes in the dataset (10 for both MNIST and CIFAR-10) and for the average value for each channel be used as input to a softmax activation.

Following on from the original paper the NIN networks used "all consist of three stacked mlpconv layers, and the mlpconv layers in all the experiments are followed by a spatial max pooling layer which downsamples the input image by a factor of two. As a regulariser, dropout is applied on the outputs of all but the last mlpconv layers". For example with N classes to predict:

```
mlpconv[5x5, 64] -> MaxPool[2] -> Dropout[0.5]
                 -> mlpconv[3x3, 128] -> MaxPool[2] -> Dropout[0.5]
                 -> mlpconv[3x3, 192]  # n.b. final 1x1 outputs N features
                 -> GlobalAvg -> SoftMax
```

Sadly the original paper states "The detailed settings of the parameters are provided in the supplementary materials" but this author could not locate said materials so the exact settings used in these experiments have been found through trial-and-error and reference to secondary online resources.

3.4 Residual Networks (`RES`)

The *residual networks* (ResNet) introduced in [3] seek to "reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions", this allows for significantly deeper networks but with fewer trainable parameters than architectures such as VGG. The implementation of ResNet for these experiments follows on from that detailed in §4.2 of [3] and from the implementation of ResNet published as a part of Keras.

The residual network consists of *residual blocks* each of which contains convolutional layers, then addition to the

block input before a final ReLU. Each convolutional layer is followed by batch normalisation (BN):

```
resblk := x ----> Conv[16] -> BN -> ReLU -> Conv[16] -> BN -> + -> ReLU
            \_____________________________________________/
```

The network is then constructed by stacking these residual blocks and every N blocks reducing the input size by half (achieved by using a stride of 2 for the next block's first convolutional layer) and doubling the number of features in following residual blocks. An example configuration of a residual network would be referred to as ResNet20 which means there are 20 trainable layers in the network, so excluding the initial and final layers there will be 9 residual blocks of 2 convolutional layers each. This report looks at ResNet20 for MNIST and CIFAR-10 and ResNet56 for CIFAR-10.

4 Scaling

The *native spatial resolution* of MNIST is 28x28 pixels and for CIFAR-10 it is 32x32. The VGG architectures were specifically designed to have an input shape of 224x224. Due to the 5 max-pooling layers (each halving the size of the input to the following layer) VGG will reduce down each input image from either dataset to a single 512-element vector (descriptor) to be used as input to the fully-connected layers. If we were to supply inputs of 64x64 (e.g. CIFAR-10 up-sampled by a factor of 2) then each image would result in a 1024-element input to the fully connected layers.

The performance of some configurations will be examined when the inputs are up-sampled by an integer factor to see whether too-small an input is resulting in a loss of information due to max-pooling effects. Given the cost of GPU compute time this will not be an exhaustive comparison.

5 Training and evaluation

Each configuration of dataset, architecture and scaling were trained in the following way. All pixel intensities in the range 0–255 were normalised into the range 0–1. The `adam` (adaptive momentum) optimiser was used in the first instance for each configuration with default parameters. The loss function used was *categorical crossentropy*. An initial run of 50 epochs were run for each configuration. If the loss was still reducing after each run of 50 epochs the training would be resumed for another run. Weight initialisation uses the Keras-default of *glorot-uniform* as proposed in [2]. A training/validation split of 85%/15% was selected to give feedback on the change in performance on unseen data throughout the training process.

Keras *callbacks*[6] were used to reduce the learning rate by half if no reduction in training-set loss was shown over 3 epochs (`ReduceLROnPlateau`) and stopping of training if no reduction in training-set loss was shown over 7 epochs (`EarlyStopping`). Therefore at least 2 attempts to make progress using a reduced learning rate will be made before finally giving up. The loss would be considered unchanged if it did not reduce by at least 0.0001.

For some combinations of data and architecture no progress was made at all in training with `adam` whereas `SGD` was able to make slow and steady progress. In these cases the default initial learning-rate of 0.01 was used with a momentum of 0.1.

Once the training was complete each configuration was evaluated against the previously unseen test set for that dataset. Within the results we will often look at *classification accuracy* as a percentage alongside the calculated loss. The `MNIST` and `CIFAR-10` datasets used were accessed using the `tf.keras.datasets` module.

[6] `https://www.tensorflow.org/api_docs/python/tf/keras/callbacks`

6 Results

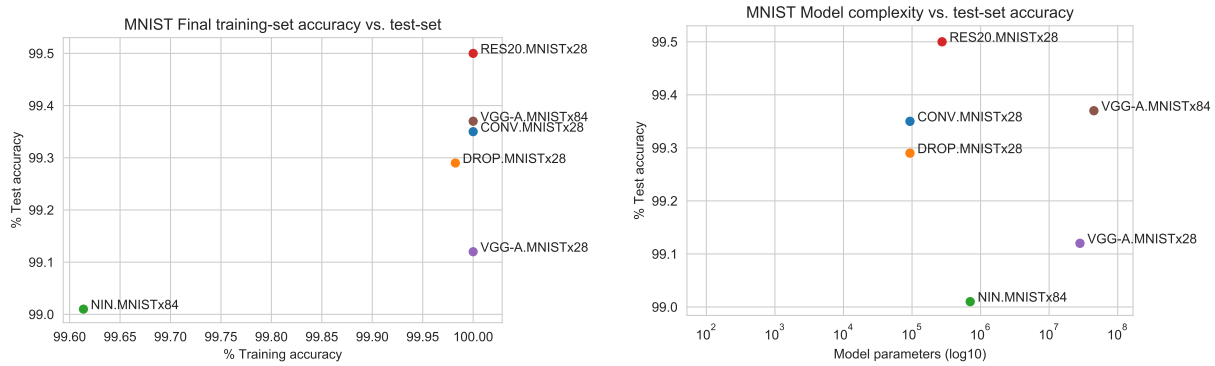
Presented below are the results for a range of architecture configurations trained against each dataset. Charts show the achieved training (in-sample) accuracy vsthe unseen (out-sample) test accuracy for each configuration. Additionally the complexity of each model (in terms of $\log_{10}$ of the number of trainable parameters) is shown vsthe achieved test-set accuracy. The final chart for each dataset shows the test-set accuracy broken down by class label for each configuration.

Detailed graphs of per-epoch training/validation accuracy for selected models are shown in Appendix A.

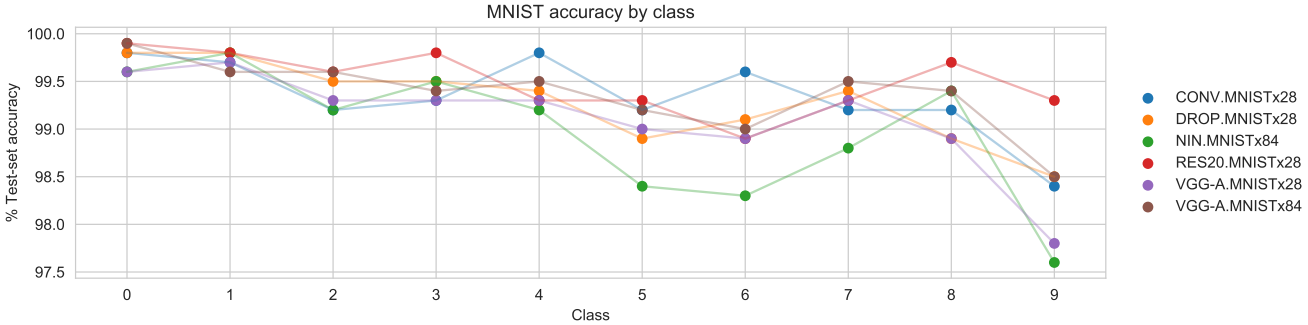
6.1 MNIST

NIN did very poorly on MNIST at 28x28 image size and so has been omitted, when the images were upscaled by a factor of 3 to 84x84 it performed much better but still inferior to the baseline models. This suggests either the NIN architecture is not well-suited for small input images, or that in the implementation of this architecture errors have been introduced. `VGG-A` fails to exceed the baseline model performance on the unscaled 28x28 images but shows notable improvement when using inputs upscaled by a factor of 3.

The `RES20` architecture achieves an improvement of 0.1% over `VGG-A` despite having fewer than 2% of the trainable parameters.

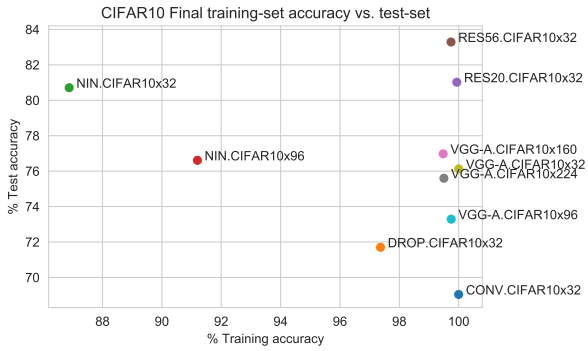


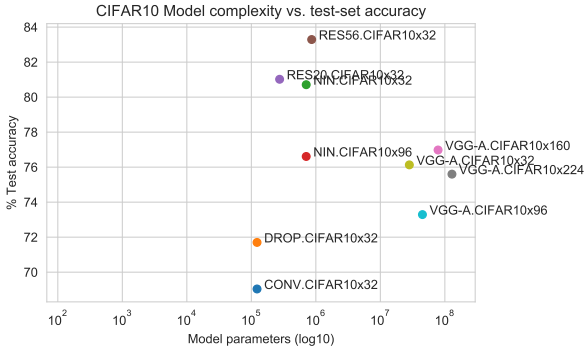
The `VGG-A` model was poorest for the digit "9" with a test-set accuracy of 97.8% which was most often confused for a "4". The `RES20` model is significantly better here.



6.2 CIFAR-10

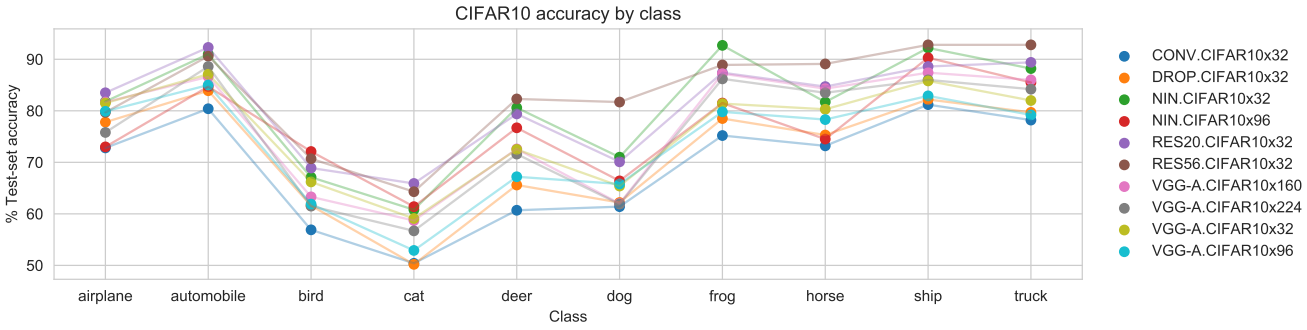
The ResNet architecture is the standout success on this dataset both in terms of achieved test-set accuracy but also delivering this without significantly more trainable parameters than Network-in-Network.





It is interesting to note that despite the `VGG-A` architecture being designed to work with images of size 224x224 that the achieved accuracy does not vary significantly with the scaling of the inputs. The images scaled by a factor of 5 (`x160`) stand out a little from the others, perhaps an example of a "goldilocks zone" in scaling: not too little and not too much.

With the MNIST data `NIN` performed poorly without upscaling and little better with. Although achieving lower training-set accuracies for CIFAR-10 than the various `VGG-A` configurations they show more consistency between training and test results. This is perhaps influenced/encouraged by the aggressive 70% dropout rate within the network. It is interesting to note the higher training-accuracy for the upscaled (`x96`) images but lower test accuracy when compared to the unscaled (`x32`) images.



The per-class "difficulty" appears quite consistent between models. The `NIN`-based models achieve good test-set accuracy for the majority of classes. `RES56` shows impressive improvement for "dog" when compared to other models.

The test-set accuracies reported for `RES20` and `RES56` are a close match for those given in §4.2 of [3].

7 Conclusions

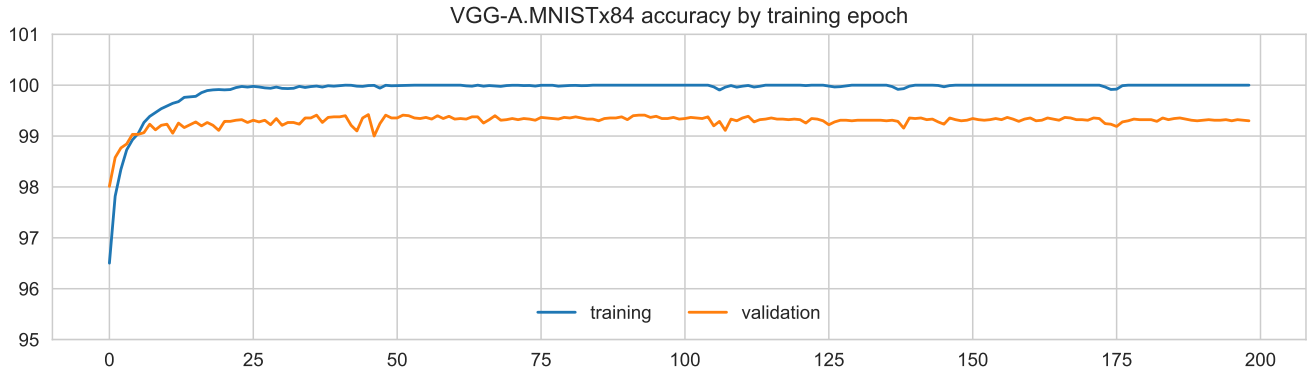
MNIST is very much a solved problem for convolutional nets: the baseline models trained quickly and accurately. For the additional accuracy the `VGG-A` architecture especially was hugely more expensive in terms of number of trainable parameters and therefore time to train, especially as in the author's experience these networks were resistant to convergence using the `adam` optimiser. Compared to the classification improvement seen using ResNet and the relatively lower computational cost VGG should be seen as a legacy blunt-instrument architecture.

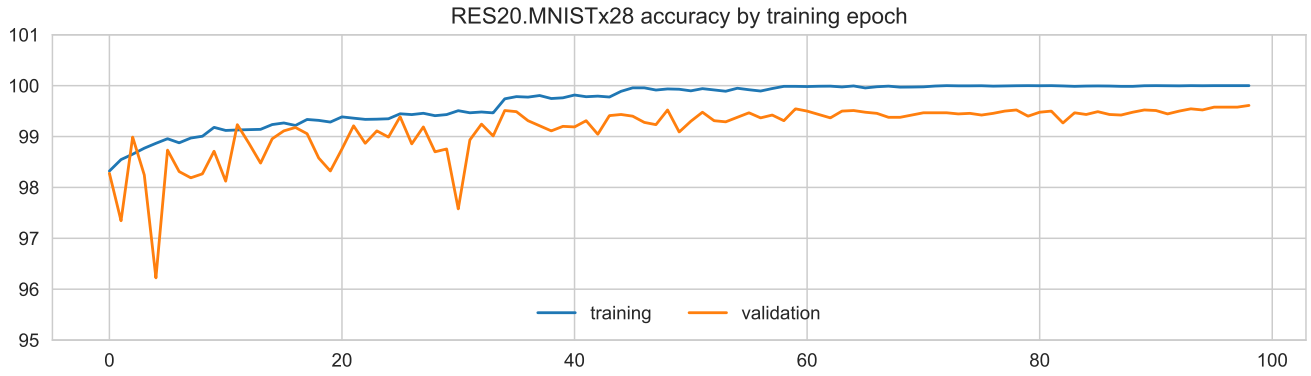
The `VGG-A` and `NIN` architectures are inherently sensitive to the size of the input images, resulting in higher train and test classification accuracies when inputs are upscaled by at least 3 times. Further work to strip unnecessary layers and pooling from these may yield better or equivalent accuracies with lower model complexity.

This reports experiments with ResNet on CIFAR-10 have matched the published accuracies closely. It is without doubt a significant improvement to earlier architectures.

A Accuracy achieved during training

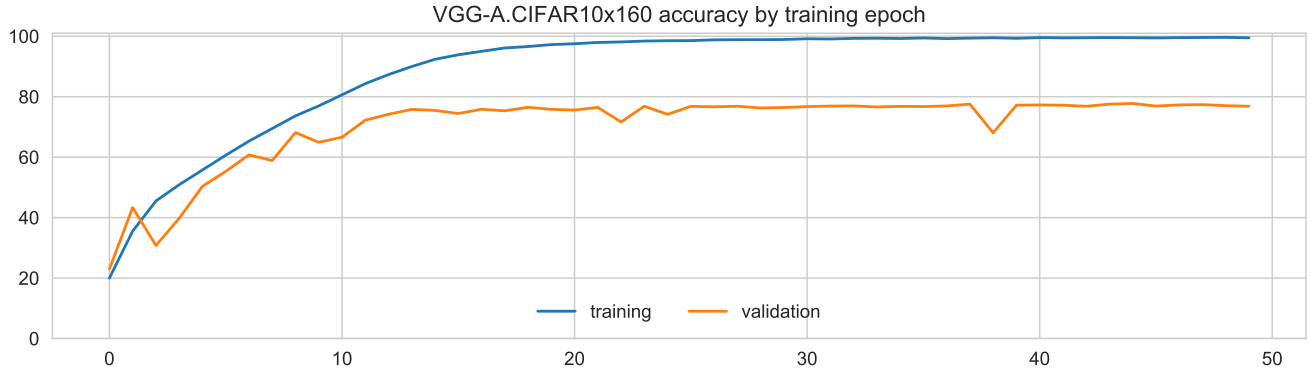
A.1 MNIST



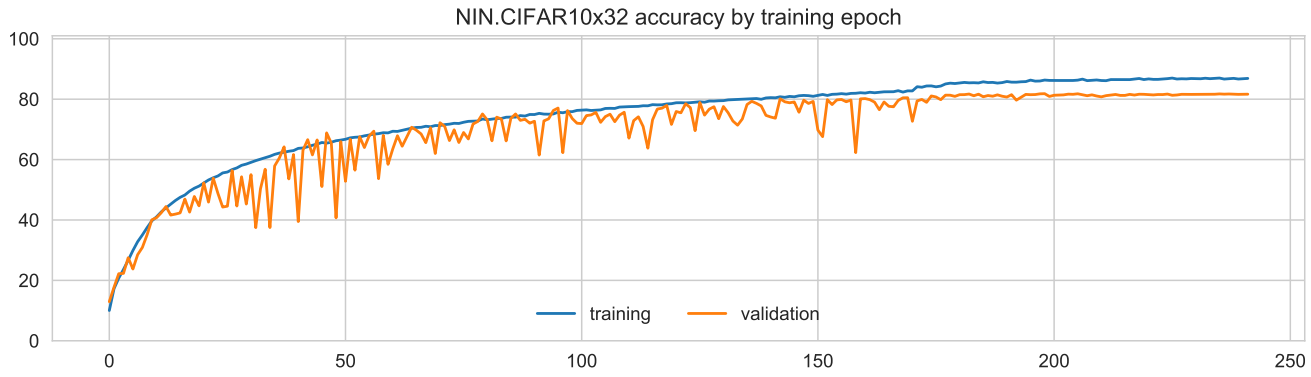


A.2 CIFAR-10

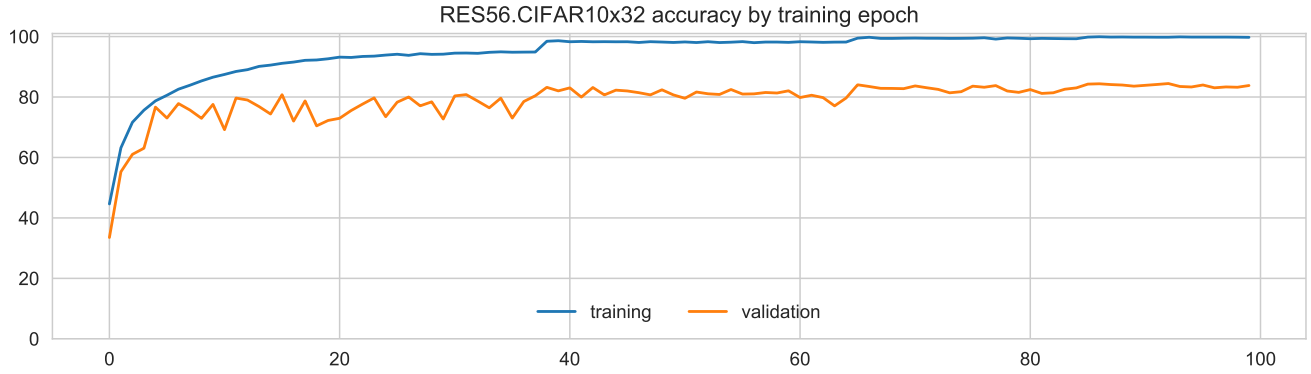
The `VGG-A` architecture achieves a high training accuracy rapidly (in epochs) but the validation accuracy plateaus.



The `NIN` architecture takes many more epochs to achieve a lower training accuracy than `VGG-A` but the validation accuracy tracks this much more closely.



The ResNet architecture provides us with the best for this dataset.



B Program usage

The program can be used from the command-prompt:

```
usage: deeper_networks.py [-h] [--epochs EPOCHS] [--restart] [--resume]
                          [--all-data] [--all-arch] [--sgd] [--mnist]
                          [--cifar10] [--scaling SCALING] [--conv] [--dropout]
                          [--vgg] [--nin] [--res20] [--res32]

optional arguments:
  -h, --help          show this help message and exit
  --epochs EPOCHS     Number of epochs to run
  --restart           Train ignoring any existing saved model
  --resume            Train an existing saved model some more
  --all-data          Run all datasets
  --all-arch          Run all architectures
  --sgd               Use SGD optimiser, default: Adam, (only for new or
                      restarting models)
  --mnist             Use MNIST dataset
  --cifar10           Use CIFAR-10 dataset
  --scaling SCALING   Upscale input images (only for new or restarting models)
```

```
--conv              Run baseline simple convnet
--dropout           Run baseline simple convnet with dropout
--vgg               Run VGG-A (11 layers) arch
--nin               Run Network-in-Network arch
--res20             Run 20-layer ResNet
--res32             Run 32-layer ResNet
--res56             Run 56-layer ResNet
```

Alternatively it can be used in a Jupyter Notebook style environment such as Google Colab:

```
import deeper_networks
deeper_networks.main([ "--drop", "--cifar", "--epochs", "50" ])
```

References

[1] François Chollet. *Deep Learning with Python*. Manning, November 2017.

[2] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[4] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[5] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[6] Min Lin, Qiang Chen, and Shuicheng Yan. Network In Network, 2013.

[7] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.