Project 3 - Modified MNIST

Find Digit with Largest MNIST Bounding Box

Stone Chen, 260654457, Yunwen Ji, 260738554, Linge Xie, 260660974

***Abstract*—In this project, we were asked to design and validate a supervised classification model to perform the Modified MNIST prediction. In particular, the model should be able to find the digit with the largest square MNIST bounding box in the image and classify it among number 0-9. To tackle this problem, we explored k-nearest Neighbors, support vector machine, and multi-layer perceptron as our baseline models. Furthermore, to achieve higher accuracy, we sought the help of deep learning, namely variant forms of Convolutional Neural Networks, including VGGNet [10], ResNet [16], as well as Inception [11]. During training of those deep learning models, we found that going deeper into our networks demonstrated significantly larger improvement in accuracy than going wider. In addition, using ensembles of different deep learning models further advanced our best model accuracy.**

I. Introduction

In this project, we were given a modified MNIST dataset, where each image contains multiple digits in the range of 0 to 9 in random backgrounds. We were asked to construct a supervised classification model to find the digit in each image with the largest square MNIST bounding box and classify its numerical value among 0 to 9. That is, our model must be able to describe both the size and numerical value of the digits in each image. We applied K-neighbors, support vector machines, and multi-layer perceptron but found their accuracy all saturated below 90%. In order to build a better model, we sought the help of deep learning, particularly variant forms of Convolutional Neural Networks that have achieved state-of-art accuracy on image classification problems. Those included VGG Network [10], Residual Network [16], and Inception Network [11] that have been applied to solve the original MINIST problem. We attempted to go deeper or wider in those models during training and discovered that going deeper resulted in substantially more improvement in terms of accuracy than going wider. After tuning each model's accuracy to saturation in our computational capacity, we furthered our model performance utilizing ensemble learning with our two best performing models with different prediction errors. This led to our best validation accuracy of 0.9779 [Table.I] which corresponded to a testing accuracy of 0.9787 on Kaggle public leaderboard.

II. Related Work

The task of recognizing a visual concept seems like a trivial for human, but it has been explored from the computer vision perspective as image classification problems in the past years. The approaches taken in those image classification tasks have become data-driven as big image dataset become available, like MNIST [7], CIFAR-10 [5], ImageNet [4] etc.. Variant forms of Convolutional Neural Networks have been presenting huge breakthrough in those dataset and they are also found in the core of many image recognition tasks today. Over the years, those networks have been designed to be deeper and deeper to deal with more complex problems, from the first 'deep' neural network VGG16 [10] in 2014, to residual [16] and inception [11] networks as well as their variant forms, which involves thousands of layers. In this paper, we worked on a dataset modified from MNIST [7], and we utilized some of the approaches mentioned above.

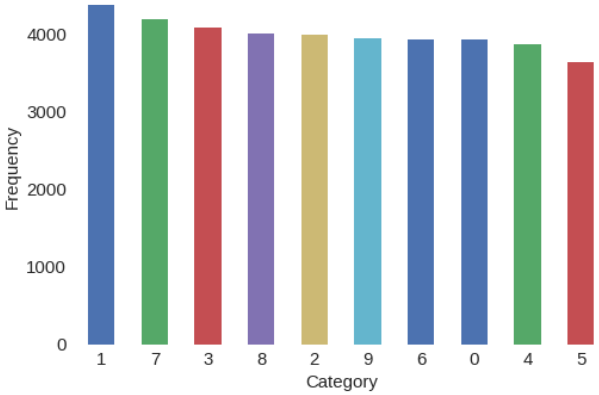
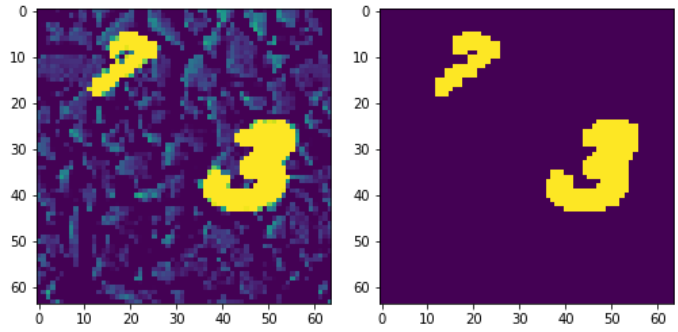


Fig. 1: Histogram of number of each label in the training data ordered by count.

III. Dataset and Setup

In the dataset we were given, there were multiple handwritten digits ranging from 0 to 9 in each grey-scale image in a random background. Each image sample was labeled with a number also in the range of 0 to 9, which corresponded to the numerical value of the digit with largest MNIST square bounding box in that image. There were approximately equal number of samples in each category [Fig.1] thus there was no need for adding bias in our training later.

Given there were noisy backgrounds at relatively low intensity in most of the image samples, we attempted to remove them in bulk with image binarization, thresholded at half of the max grey-scale intensity (127.5). Furthermore, additional intensity spikes were removed by performing morphological opening on the binarized image [Fig.2]. Specifically, an erosion was performed to remove spiking pixels while dilation followed to compensate for over-erosion on digits [1]. However, given the lesson from last project, where minimum processing led to better model accuracy, we compared the performance of a simple 1-layer Convolutional Neural Network on processed data to the unprocessed [Table.I]. Again,



(a) Original image (b) Preprocessed image

Fig. 2: (a) is the one of the original images in the training data set while (b) is the image after the removal of background noise.

we found that with no image cleaning, our model achieved much higher validation accuracy (0.9261 vs. 0.3111), despite that both model overfitted with such small complexity; thus we decided not to apply any cleaning processes to our samples.

Nevertheless, before inputting data to our model, we normalized our data by subtracting and dividing each sample by overall sample mean and standard deviation, respectively. For deep learning models, input data were expanded by an extra channel dimension in the shape of (64, 64, 1), otherwise the input data were flatten into vectors of shape (4096,1).

In addition, to prevent overfitting and increase generalization performance, we attempted to generate variations based on our current samples. By inspecting a number of existing samples, we came to a list of augmentations that could be done to benefit the training process, including rotation, horizontal and vertical shift, shear, as well as zoom. Those augmentation processes were chosen so that they did not alter the intrinsic properties of those digits; an counter example could be that a horizontal flip of number 6 produced a ρ shape digit that might confuse the model during training. To achieve such augmentations on our dataset, we utilized the Image Generator Class from Keras API [12] which applied those specified augmentations randomly during training time. The settings for the Image Data Generator are:

- rotation range=20
- width shift range=0.2
- height shift range=0.2
- shear range=0.5
- zoom range=(0.9,1.1)

IV. Proposed Approach

A. Baseline Model Training & Validation Setup

We mainly implemented SVM (Support Vector Machine), kNN (k-Nearest Neighbors) and MLP (Multi-Layer Perceptron) respectively as our baseline models with the packages in scikit-learn [8], which are important approaches in the field of multi-class classification. The two-dimensional images in the training set were transformed into one-dimensional vectors as input.

For each of the model, we utilized the GridSearchCV class from the sklearn.model_selection to exhaustively search for the optimal parameters in a given range. Further information about the parameter choosing will be specified in each of the following sections. In addition, the setting of 5-fold cross validation and l2 regularization in the class both provide the strength of reducing over-fitting.

B. Support Vector Machine

SVM, as a classical method for the classification [2], was implemented to predict the largest digit in the datasets.
By searching over the range of C, the regularization parameter, from 0.05 to 50, the value 50 gave us the maximum score of 0.8887 [TableI].

C. K-nearest Neighbors

The approach of kNN [3] makes use of the existing instances in the training set and attempts to predict the images in the test set by seeking out the k most similar points in the training set and choosing the class with the highest frequency. The choice of k can have profound effects on the performance of the model due to the bias-variance trade-off. Large k will be more generalizable but can lead to high bias thus low accuracy of the prediction.
During the training, we ran experiments on the range of k from 3 to 8. The 5-nearest neighbors classifier produced the most accurate prediction on the validation set [TableI].

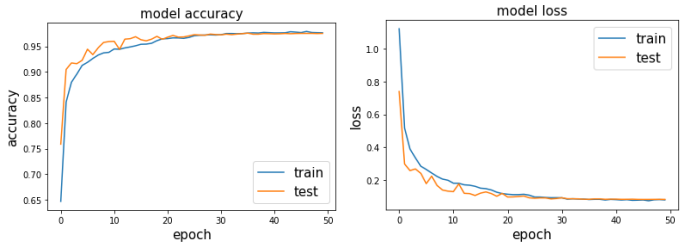
D. Multi-layer Perceptron

MLP, a class of feedforward neural network [18], has been implemented by the researchers who were exploring the digit recognition of MNIST dataset long ago and achieved remarkable success at that time [9]. We experimented on the parameter field of 3 different architectures: 3 hidden layers of 150 hidden units each, 3 hidden layers of 150, 200, 150 units each and one hidden layer of 300 hidden units. The model with 3 hidden layers of 150, 200, 150 units each results in the highest validation accuracy of 0.8855 [TableI] and the result also suggested that with the increasing number of hidden layers and hidden units per layer, the MLP model demonstrated great improvement.

E. Deep Learning Training & Validation Setup

Our implementation and validation of deep learning models were based on the Keras API [12]. Besides the data augmentation performed during training time, we also implemented two callbacks from Keras API [12] to help us control the learning rate and epochs as follows:

- ReduceLROnPlateau: this callback monitors the validation accuracy and if it does not increase for 3 epochs, it will reduce current learning rate by half for the following epochs. Minimum learning rate is set at $1e-6$.
- EarlyStopping: this callback monitors the validation loss and if it does not decrease for 5 epochs, training is stopped and the weights at last layer are saved.



(a) Training Time Accuracy Curve (b) Training Time Loss Curve

Fig. 3: Typical training curve of our training setup.

These two callbacks are added so that we did not have to explicitly tune our learning and decaying rate as well as the epoch for each model. The initial learning is the default learning rate for the optimizer (most of the time we used stochastic gradient descend and the default learning rate is 0.01) and the default maximum epoch is set at 60. With categorical cross-entropy, our typical training curves looked like that in Fig.3, where convergence occurred and training stopped before hitting 60 epochs. Thus we did not feel the need to add any dropout as regularization. Note that there were usually some fluctuations at the beginning of training that we attributed to a high initial learning rate. However we did not adjust the initial learning rate as it seemed to stabilize after reduction at later epochs. Besides, maintaining the initial learning rate at relatively high value prevented it from becoming too small after reduction at later epochs.

Furthermore, in the design of all our deep learning models, we inserted a batch normalization layer using Keras API [12] after all convolution layers as well as fully connected dense layers. This layer normalized the activated output of the previous layer at each batch during training before inputting it to the next layer. In this way, the input distribution at each step would remain the same so that each intermediate layer would not have to adapt to new distributions of input, which would significantly increase convergence time. In addition, normalizing activation of each layer also limited the expressive power of our networks which prevented overfitting to some extent [13].

In terms of our validation setup, we used the train and test split function from Scikit-learn [8] to split 33% of training data as validation set with random state 42. We were not able to perform 5-fold cross-validation for our deep learning model when it became very complex with over 5,000,000 parameters within reasonable time. However we can still obtain some confidence in our model's ability to generalize by comparing the training and validation accuracy.

Lastly, there were some hyper-parameters that were consistently used in our weighted layer, which we would like to specify here for all models in the following subsections:

- Padding = same (maintained image size before pooling since we already had small image to start with)
- Initialization = Xavier uniform [6] (Default initialization for convolutional layers in Keras [12]; we observed no issue in convergence.)
- Activation = ReLu (the only activation function used in the CNN we referred to [16, 11, 10]).

F. VGGNet Inspired CNN

VGG was first introduced by Simonyan and Zisserman in 2014 as a very deep neural network for large-scale image classification tasks. Its design in increasing the depth was straightforward, through stacking 3×3 convolutional layers on top of each other in increasing depth and reducing size by max pooling. Two fully connected layers and a softmax classifier followed to generate the final output [10]. Despite its design simplicity, a VGG16 (16 indicated to the number of weighted layers) was able to achieve around 90% accuracy on ImageNet [4].

Given our images were of dimension 64×64, which was limited to a maximum of 4 pooling of size (2,2), we designed a VGG12 [Appendix.Fig.2] and a VGG9, where both models contained approximately 15M parameters, but VGG12 went deeper whereas VGG9 went wider in their architectures. Specifically, VGG9 followed similar architecture of the first few layers in VGG16 [10], consisted of three modules of two (3, 3) convolutional layers followed by a (2,2) max pooling layer. On the other hand, VGG12 resembled the last few layers of the VGG16, where three convolutional layers were stacked; in particular, we inserted three modules of two (3,3) convolutional layers stacked on a (5,5) convolutional layer, followed by the same (2,2) max pooling layer. Note that all convolution strides were set to (1,1). We compared the performance of these two models of similar size and found VGG12 reached better validation accuracy, indicating that going deeper in our network would better describe the classification problem. More details were discussed in the result section but this finding set our direction in the design of our variant models.

G. ResNet Inspired CNN

Despite that going deeper seemed like the way to go for a better representation of our classification problem, we realized soon our VGG network's accuracy saturated regardless of the adjustment of any hyperparameters or addition of layers. High chances were that our network was suffering from the vanishing gradient problem, where weights in layers at the beginning of the network could not be updated properly due to very small gradient. Nevertheless, the residual network, since its introduction in 2015, had been proven to enable training of networks with thousands of layers with compelling performance. This type of network was able to deal with vanishing gradient problem by its auxiliary loss and identity shortcut connections [16]. Both features were in place to increase the magnitude of the gradient reaching early layers, either by directly increasing the gradient or skipping intermediate layers to pass the gradient to early layer.

For our purposes, we only utilized the identified shortcut connections to construct a 29-layer residual network ResNet29 with 12 modules, each of which was consisted of 2 stacked convolutions and a skip connection [Appendix.Fig.1]. Same choice of hyper-parameters in convolutional and pooling layers

used in the VGG9 were used here. To establish the identity shortcut connections, we utilized the addition layer in Keras API [12], which summed up the input from current module and the previous module before ouputting to the next module. In addition, when the filter size between two consecutive modules doubled as depth increased, we utilized the (1,1) convolutional layer to perform identify mapping of the output from previous module and increase filter size to that of the current module. This was a valid operation as the author *He* argued in his paper that stacking layers of identity mapping would not introduce negative effects on the network [16].

H. InceptionNet Inspired CNN

Another problem with going deeper was that this procedure might significantly increase the computational cost with no actual benefit, if the hyperparameters at each layer were not chosen properly. For example, in the image classification task, the variance in the location of information would make it a hard decision to choose a proper kernel size. A large kernel would miss local information whereas a small kernel would not capture global information. In this situation, stacking convolutional layers with various kernel sizes would not only be costly but also tiresome to experiment and come up with the best design.

Nevertheless, this problem was solved with the concept of inception network, where filters of multiple sizes could operate on the same level. Essentially, the network became 'wider', which however was in some way against our direction of going deeper, but we would like to explore this different scheme of going wider and compare its performance as well. Based on Inception V2 [14], we utilized its smart factorization method to construct a inception module of that shown in Fig.4. The two stacked 3×3 convolutions replaced the original 5×5 in that, if this 2-layer 3×3 convolution stack had C channels, then the stack was parametrized by 2(32C2) = 18C2 weights; in contrast, a single 5×5 layer contained 52C2 = 25C2 parameters. This was almost 40% more parameters, which as reported in this paper, was 2.78 times more expensive [14]. In terms of filter size, we used half of what was specified in the paper for this module. The grid size reduction module was replaced by simple max (2,2) pooling in our implementation. Note that we noticed a very slow increase in our training loss at the beginning, so we added two auxiliary softmax losses stemming from the intermediate layers. The final loss at the output layer was an average of three softmax losses. A partial view of our InceptionNet was shown in [Appendix.Fig.3]

I. Ensemble Learning

Difference during training of a deep neural network in many aspects will cause itself to make partially independent errors on the data, including differences in initialization, selection of mini-batches, hyper-parameters etc. Given that different models would make different prediction errors, the combination of those models would result in better performance than any single best models and add a bias that would counter the variance of a single model [15]. After obtaining the best models of VGG12 and ResNet29 [TableI], we assessed

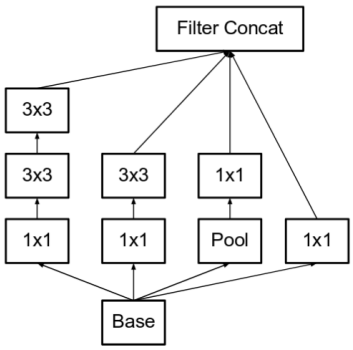


Fig. 4: Architecture of our inception module used in the InceptionNet

their prediction errors [Table.II] and decided to further our model performance by ensembling the two models through a weighted sum of their predicted probability for each sample and for each label. The weights assigned to each model were determined by explicit grid search implemented by ourselves. More details of this ensemble method were discussed in the results section.

V. RESULT

TABLE I: Validation accuracy of all trained models. Note for all models following 1-layer CNN, there is no image cleaning applied on the data. For the ensemble model, the number in the round brackets indicated their respective weights when calculating the weighted sum of prediction probability.

Model	**Validation accuracy**
1-layer CNN (with image cleaning)	0.3111
1-layer CNN (without image cleaning)	0.9261
SVM	0.8887
k-NN (k=5)	0.8857
MLP	0.8855
VGG12	0.9749
VGG9(wider)	0.9521
ResNet29	0.9698
InceptionNet	0.9529
ResNet29(0.4) & VGG12(0.6)	0.9779

A. Limited Performance in Baseline Models

For our baseline models, including SVM, k-nearest neighbors and MLP, they were all unable to reach over 90% accuracy, despite our efforts in tuning their parameters with explicit grid search. However, we expected this limitation as the expressive power of those model was quite restricted in the context of representing 2D image data. Besides, there were great loss of structural and spatial information when those 2D image data were flatten into vectors as input. Immediately, we saw that a 1-layer simple CNN could get to 0.9261 accuracy [Table.I] on the validation data. Although this small CNN was used for testing imaging cleaning and it was overfitted, we could not deny that a neural network already possessed a higher representation power than that of the baseline models.

B. Deeper vs. Wider

There seemed to be a trend in image classification tasks to go deeper, rather than wider [10, 16, 14] to achieve better model performance. At the beginning of our neural network training, we took this view with a grain of salt and decided to briefly investigated this in the context of our modified MNIST data. Through comparing our deep VGG12 with the wide VGG9, both of approximately 15M parameters, we observed much better performance in the deeper VGG12 model with validation accuracy of 0.9749 [Table.I]. A possible reason was that deeper a neural network like VGG12 would be able to capture hierarchical features in data, in which early layers learnt some basic features while the deep layers captured more abstract features in the data. This then became intuitive as incorporating more features of various forms would be beneficial than more features of the same kind in explicitly representing a dataset with a model.

The other way of increasing width by performing multiple convolutions of different filter size was also explored with the InceptionNet. Its model size was even larger than that of VGG12 and ResNet29, but its performance was not any higher. Chances were that there was still potential for fine-tuning in hyper-parameters and architectures in such a complex model which we implemented from scratch. However, the take-way from this model was that it might be more straightforward to see model improvement just by going deeper.

C. Best Single Model: VGG12

After training a VGG12 and reaching a validation accuracy of 0.9749, we proceeded to build ResNet and InceptionNet in the hope of achieving a higher accuracy. Unfortunately, after building the deepest (moderately wide) network which exhausted all our computational capacity, we were still unable to obtain anything exceeding our VGG12 performance. As we reflected on the training procedures, we realized that there were still much potential for fine-tuning those complex models in terms of optimizer and initialization, if we had the hardware to train and experiment hyper-parameters more efficiently. Nevertheless, we believed that the residual network and inception network would be able to surpass this single VGG model's baseline, with evidence from their relatively better performance on ImageNet [4].

D. Ensemble Learning

At the end of our training of single models, we obtained a VGG12 and a ResNet29 of validation accuracy 0.9749 and 0.9698, respectively. Given that the improvement of an ensemble from their single models was contingent on the difference between the prediction errors, we first investigated their prediction accuracy on the validation data. We found that despite that VGG12 had higher overall accuracy, it was relatively weaker in accurately predicting samples with label 8 than the ResNet29 [Table.II]. Provided with this small prediction difference, their ensemble demonstrated a slight but expected improvement on validation accuracy to 0.9779 [Table.I]. If ResNet29 could out-perform the VGG12 with regards to label 8, we would expect this ensemble to show even more improvement.

TABLE II: Validation accuracy of the two models used in ensemble sorted by validation labels. The high accuracy for the corresponding label was bolded.

Label ID	**VGG12** (0.9749)	**ResNet29** (0.9698)
0	**0.9944**	0.9873
1	**0.9780**	0.9760
2	**0.9762**	0.9686
3	**0.9810**	0.9740
4	**0.9717**	0.9694
5	**0.9708**	0.9613
6	**0.9796**	0.9749
7	**0.9568**	0.9553
8	0.9701	**0.9745**
9	**0.9746**	0.9569

VI. Discussion and Conclusion

A. VGG12 is not the Best Single Model

Even though our result indicated VGG12 resulted in the best performance, we have also repeatedly emphasized the potential of residual and inception network to perform better with more fine-tunings and also increases in depths. This would be a promising direction to continue into if time permitting, as that VGG model was intrinsically limited in depth because of vanishing gradient problem and generalization. In contrast, particular ResNet was immune to the vanishing gradient problem which permitted it to go a lot deeper, which was also the way we concluded to most improve our model. In terms of InceptionNet, it was definitely more costly, despite the benefit of incorporating filters of various sizes. Whether this would be able to produce any cost-effective benefit in representing a dataset would be highly dependent on the variance of information. In fact, now the state-of-art had explored the combination of the two, namely Inception-ResNet. [17]

B. Further Improvements on Ensemble Learning

Our ensemble only demonstrated a slight increase since the two models we combined were not of great difference in terms of their prediction errors. However to generate more different models, we would have trained the ResNet29 and VGG12 with different initializations and optimizers (however we were not able to because of limited capacity in hardware and time). The stochastic nature of the training process would produce different mappings from input to output. Given that there were 10 different class labels, there were high chance that every single model would have its strength and weakness in prediction. We believe that ensemble learning would be the best approach to our problem as it would integrate complementary strengths from the best single models.

Statement of Contribution

Linge Xie was in charge of building baseline models, generating tables and figures for the report, and writing introductory sections of the report. Sitong Chen and Yunwen Ji collaborated on building the deep learning models and writing rest of the report.

REFERENCES

[1] E.R. Dougherty. *An introduction to morphological image processing*. Tutorial texts in optical engineering. SPIE Optical Engineering Press, 1992.

[2] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning*. 1995, pp. 273–297.

[3] T. Cover and P. Hart. "Nearest Neighbor Pattern Classification". In: *IEEE Trans. Inf. Theor.* 13.1 (Sept. 2006), pp. 21–27. ISSN: 0018-9448. DOI: 10.1109/TIT.1967.1053964. URL: https://doi.org/10.1109/TIT.1967.1053964.

[4] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[5] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[6] 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*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

[7] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[8] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[9] Dan Claudiu Cirean et al. "Deep Big Multilayer Perceptrons For Digit Recognition". In: (Jan. 2012). DOI: 10.1007/978-3-642-35289-8_31.

[10] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[11] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[12] François Chollet et al. *Keras*. https://keras.io. 2015.

[13] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[14] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *CoRR* abs/1512.00567 (2015). arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016, pp. 257–258. ISBN: 0262035618, 9780262035613.

[16] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI: 10.1109/cvpr.2016.90.

[17] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *CoRR* abs/1602.07261 (2016). arXiv: 1602.07261. URL: http://arxiv.org/abs/1602.07261.

[18] H. Ramchoun et al. "Multilayer Perceptron: Architecture Optimization and Training with Mixed Activation Functions". In: *Proceedings of the 2Nd International Conference on Big Data, Cloud and Applications*. BDCA'17. Tetouan, Morocco: ACM, 2017, 71:1–71:6. ISBN: 978-1-4503-4852-2. DOI: 10.1145/3090354.3090427. URL: http://doi.acm.org/10.1145/3090354.3090427.

APPENDIX

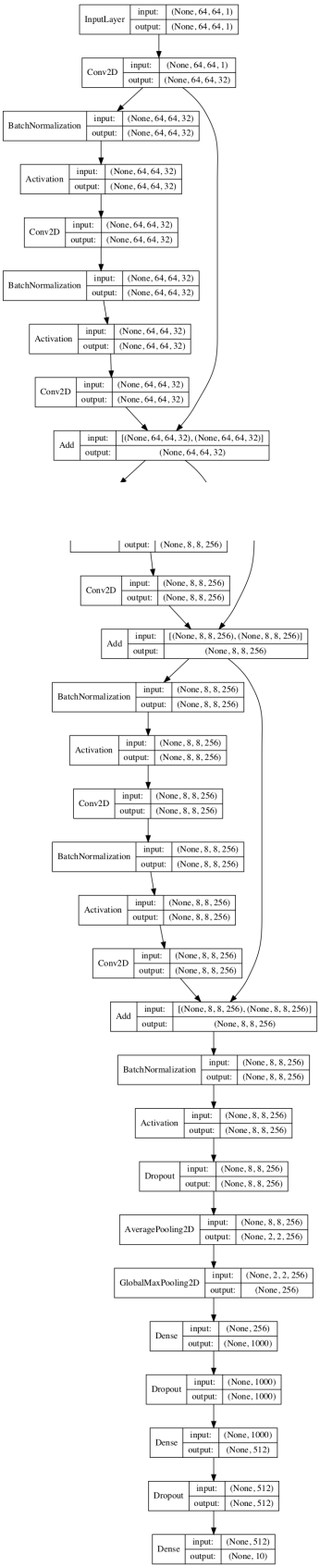


Fig. 1: Part of residual network's architecture shown above.

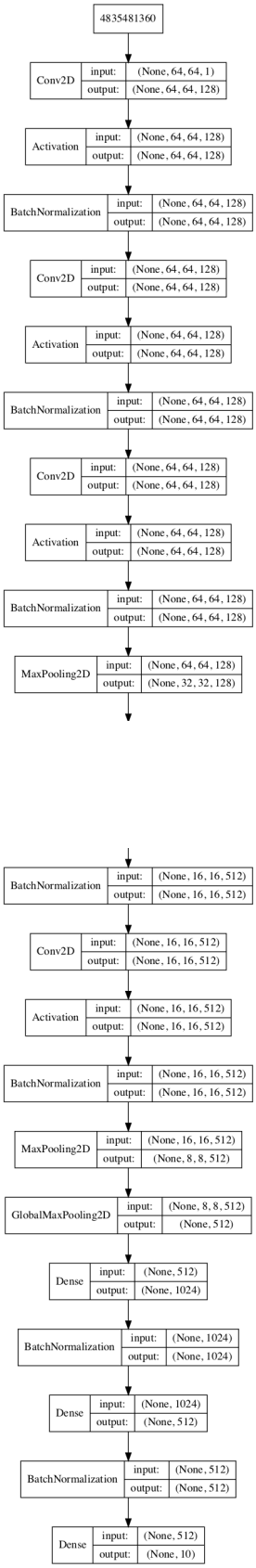


Fig. 2: Part of VGG12 network's architecture shown above.

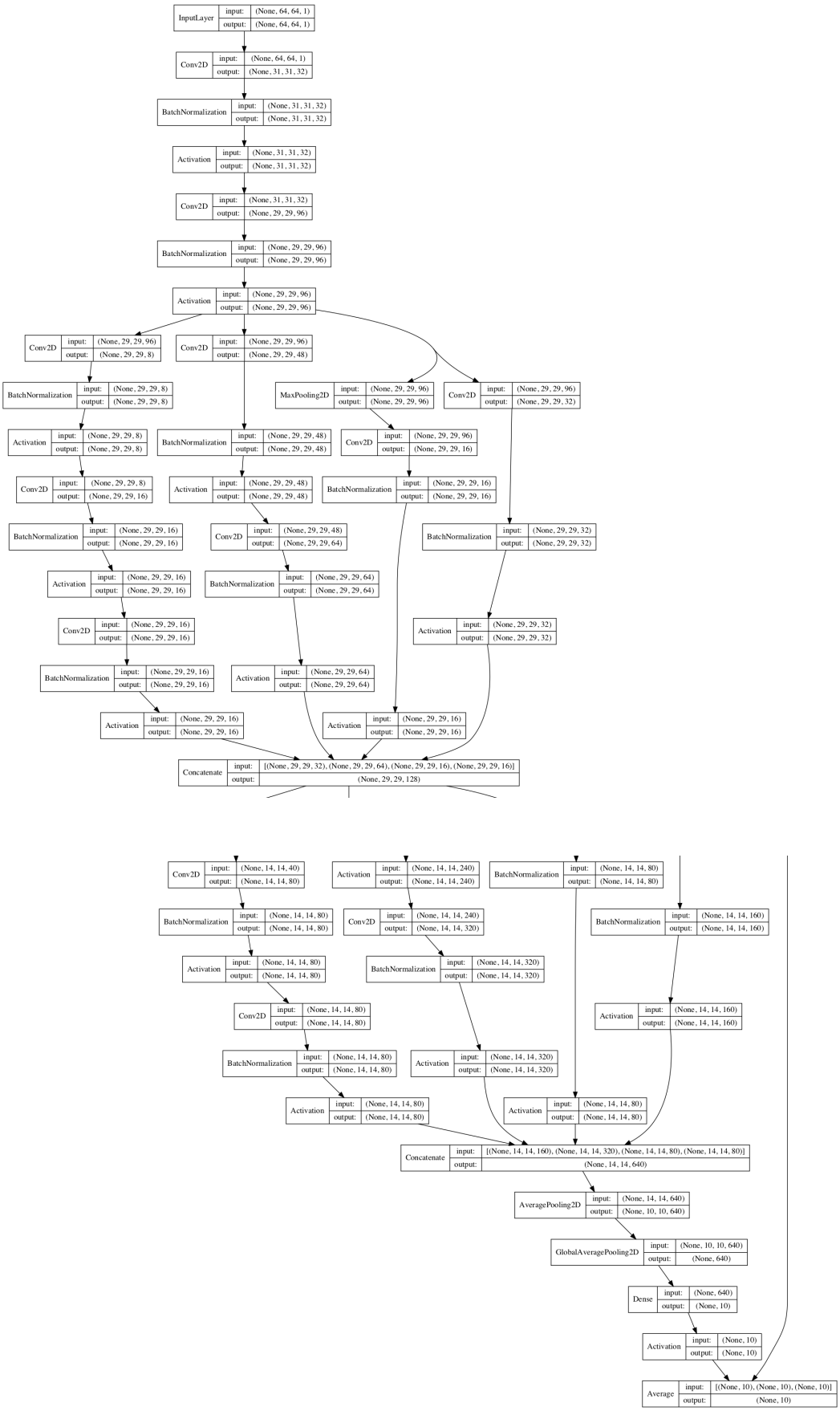


Fig. 3: Part of Inception network's architecture shown above.